HOLTEK

CAN BUS A/D Flash MCU

HT66F3370H

Revision: V1.00 Date: December 27, 2019

www.holtek.com

Table of Contents

Features

CPU Features

- Operating Voltage
  - f_{SYS}=8MHz: 2.2V~5.5V
  - f_{SYS}=12MHz: 2.7V~5.5V
  - f_{SYS}=16MHz: 3.3V~5.5V
- Up to 0.25μs instruction cycle with 16MHz system clock at V_{DD}=5V
- Power down and wake-up functions to reduce power consumption
- Oscillators
  - External High Speed Crystal – HXT
  - Internal High Speed 8/12/16MHz RC Oscillator – HIRC
  - External Low Speed 32.768kHz Crystal – LXT
  - Internal Low Speed 32kHz RC Oscillator– LIRC
- Fully integrated internal oscillators require no external components
- Multi-mode operation: FAST, SLOW, IDLE and SLEEP
- All instructions executed in 1~3 instruction cycles
- Table read instructions
- 115 powerful instructions
- 16-level subroutine nesting
- Bit manipulation instruction

Peripheral Features

- Flash Program Memory: 32K×16
- RAM Data Memory: 3072×8
- Ture EEPROM Memory: 1024×8
- In Application Programming function – IAP
- Watchdog Timer function
- Up to 58 bidirectional I/O lines
- Programmable I/O source current
- Software controlled 4-SCOM lines LCD driver with 1/2 bias
- Four external interrupt lines shared with I/O pins
- Multiple Timer Modules for time measurement, input capture, compare match output, PWM output function or single pulse output function
- Serial interfaces module – SPI or I^2C
- Single serial SPI interface – SPIA
- Multiple UART Interfaces for full duplex asynchronous communication
- Single CAN Bus interface

- CAN Module licensed from Bosch
  - Conforms to ISO11898-1, 2003
  - 32 Message Objects
  - Each Message Object has its own identifier mask
  - Programmable FIFO mode - concatenation of Message Objects
  - Maskable interrupt
  - Programmable loop-back mode for self-test operation
  - Support CAN Bus SOF (Start of Frame)
- Dual comparator functions
- Dual Time-Base functions for generation of fixed time interrupt signals
- Up to 16 external channels 12-bit resolution A/D converter with internal reference voltage V_R
- Integrated 16-bit Multiplier/Divider Unit – MDU
- Integrated 16-bit Cyclic Redundancy Check function – CRC
- Low voltage reset function – LVR
- Low voltage detect function – LVD
- Package types: 48/64-pin LQFP
- The OCDS EV, named HT66V3370H, AC/DC characteristics are guaranteed only at a temperature of 25˚C

General Description

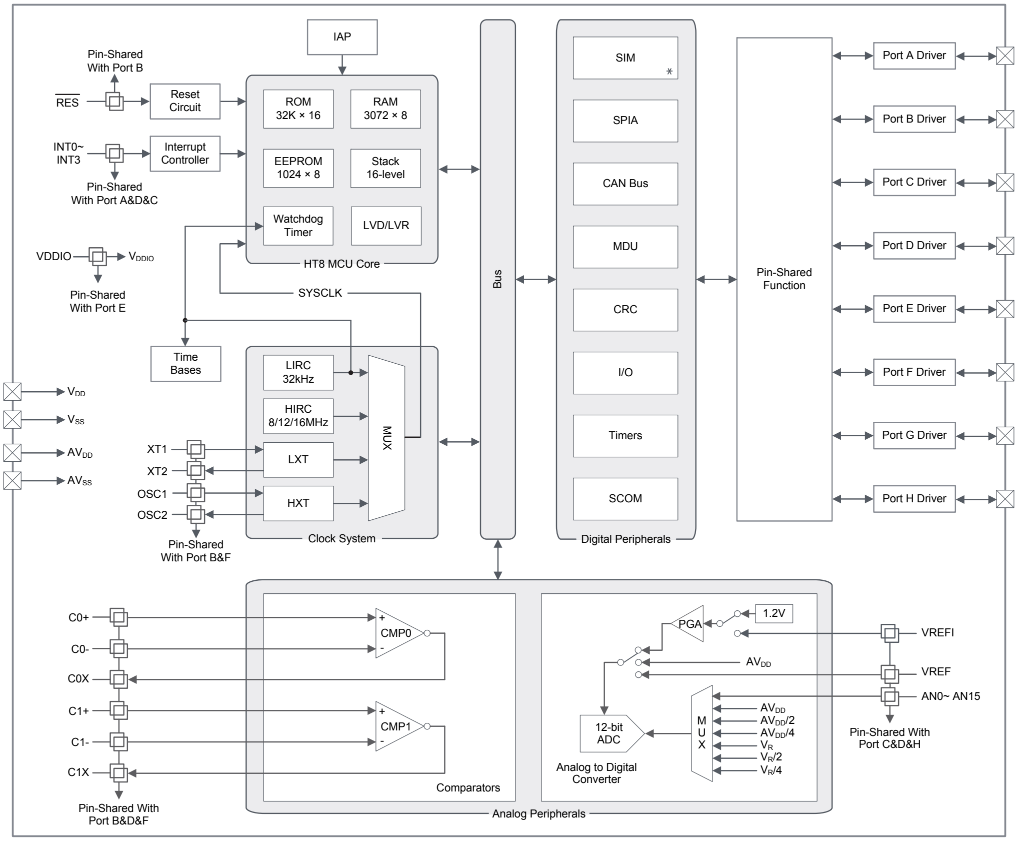
The HT66F3370H is a Flash Memory CAN Bus A/D 8-bit high performance RISC architecture microcontroller. Offering users the convenience of Flash Memory multi-programming features, the device also includes a wide range of functions and features. Other memory includes an area of RAM Data Memory as well as an area of true EEPROM memory for storage of non-volatile data such as serial numbers, calibration data etc.

Analog features include a multi-channel 12-bit A/D converter and dual comparator functions. Multiple and extremely flexible Timer Modules provide timing, pulse generation and PWM generation functions. Communication with the outside world includes a fully integrated CAN Bus, as well as SPI, I^2C and UART interface functions, popular interfaces which provide designers with a means of easy communication with external peripheral hardware. Protective features such as an internal Watchdog Timer, Low Voltage Reset and Low Voltage Detector coupled with excellent noise immunity and ESD protection ensure that reliable operation is maintained in hostile electrical environments.

A full choice of external and internal low and high oscillator functions are provided including fully integrated system oscillators which require no external components for their implementation. The ability to operate and switch dynamically between a range of operating modes using different clock sources gives users the ability to optimise microcontroller operation and minimise power consumption.

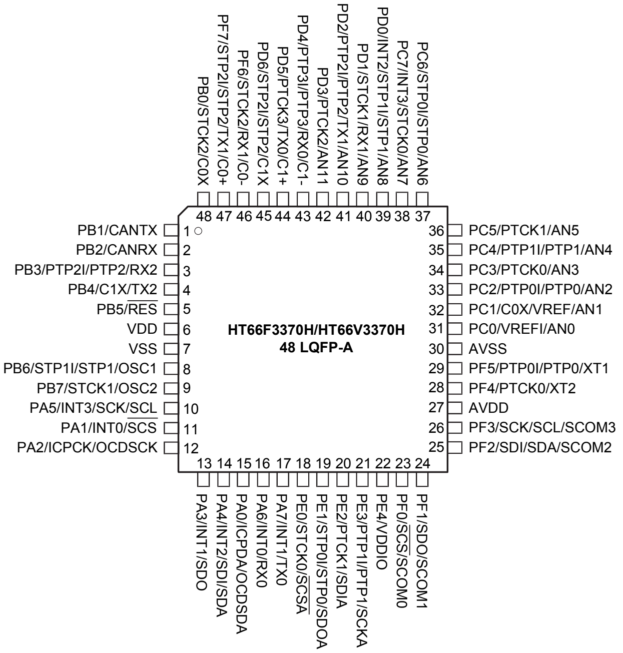
The inclusion of flexible I/O programming features, Time-Base functions along with many other features ensure that the device will find excellent use in a wide range of applications however the inclusion of a CAN bus especially opens up a wide range of automotive related applications.

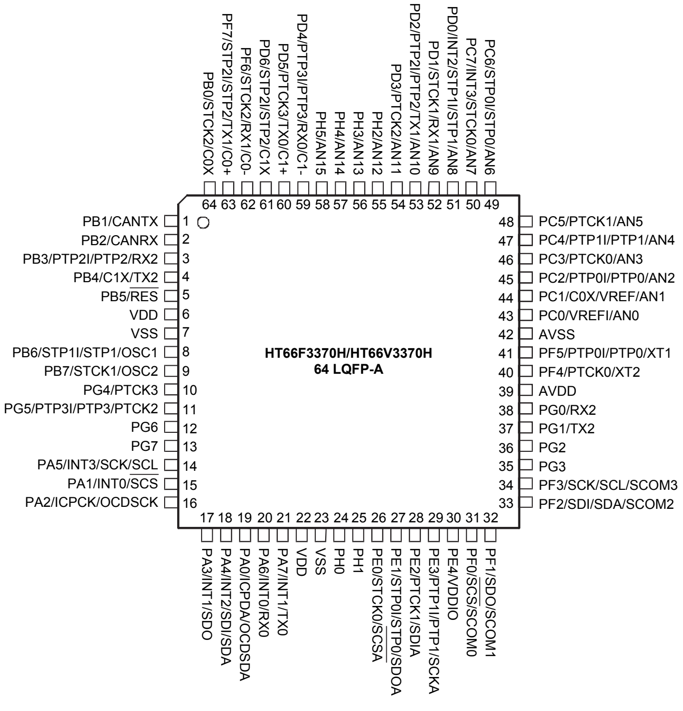
Block Diagram



: Pin-Shared Node   * : SIM including SPI & I²C

Pin Assignment





Note: 1. If the pin-shared pin functions have multiple outputs, the desired pin-shared function is determined by the corresponding software control bits.

2. The OCDSDA and OCDSCK pins are supplied as OCDS dedicated pins and as such only available for the HT66V3370H device which is the OCDS EV chip for the HT66F3370H device.

3. For the less pin count package type there will be unbounded pins which should be properly configured to avoid unwanted power consumption resulting from floating input conditions. Refer to the "Standby Current Considerations" and "Input/Output Ports" sections.

Pin Description

With the exception of the power pins and some relavant transformer control pins, all pins on the device can be referenced by their Port names, e.g. PA0, PA1 etc, which refer to the digital I/O function of the pins. However these Port pins are also shared with other function such as the Analog to Digital Converter, Timer Module pins etc. The function of each pin is listed in the following table, however the details behind how each pin is configured is contained in other sections of the datasheet. Note that where more than one package type exists the table will reflect the situation for the larger package type.

Pad Name	Function	OPT	I/T	O/T	Description
PA0/ICPDA/OCDSDA	PA0	PAWU PAPU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	ICPDA	—	ST	CMOS	ICP Data/Address pin
	OCDSDA	—	ST	CMOS	OCDS Data/Address pin, for EV chip only.
PA1/INT0/$\overline{\text{SCS}}$	PA1	PAWU PAPU PAS0	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	INT0	PAS0 INTEG INTC0 IFS2	ST	—	External Interrupt 0
	$\overline{\text{SCS}}$	PAS0 IFS2	ST	CMOS	SPI slave select
PA2/ICPCK/OCDSCK	PA2	PAWU PAPU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	ICPCK	—	ST	CMOS	ICP Clock pin
	OCDSCK	—	ST	—	OCDS Clock pin, for EV chip only.
PA3/INT1/SDO	PA3	PAWU PAPU PAS0	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	INT1	PAS0 INTEG INTC0 IFS2	ST	—	External Interrupt 1
	SDO	PAS0	—	CMOS	SPI data output
PA4/INT2/SDI/SDA	PA4	PAWU PAPU PAS1	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	INT2	PAS1 INTEG INTC3 IFS2	ST	—	External Interrupt 2
	SDI	PAS1 IFS2	ST	—	SPI data input
	SDA	PAS1 IFS2	ST	NMOS	I^{2}C data line
PA5/INT3/SCK/SCL	PA5	PAWU PAPU PAS1	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	INT3	PAS1 INTEG INTC3 IFS2	ST	—	External Interrupt 3
	SCK	PAS1 IFS2	ST	CMOS	SPI serial clock
	SCL	PAS1 IFS2	ST	NMOS	I^{2}C clock line

Pad Name	Function	OPT	I/T	O/T	Description
PA6/INT0/RX0	PA6	PAWU PAPU PAS1	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	INT0	PAS1 INTEG INTC0 IFS2	ST	—	External Interrupt 0
	RX0	PAS1 IFS3	ST	—	UART0 RX serial data input
PA7/INT1/TX0	PA7	PAWU PAPU PAS1	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	INT1	PAS1 INTEG INTC0 IFS2	ST	—	External Interrupt 1
	TX0	PAS1	—	CMOS	UART0 TX serial data output
PB0/STCK2/C0X	PB0	PBPU PBS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	STCK2	PBS0 IFS0	ST	—	STM2 clock input
	C0X	PBS0	—	CMOS	Comparator 0 output
PB1/CANTX	PB1	PBPU PBS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	CANTX	PBS0	—	CMOS	CAN TX data output
PB2/CANRX	PB2	PBPU PBS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	CANRX	PBS0	ST	—	CAN RX data input
PB3/PTP2I/PTP2/RX2	PB3	PBPU PBS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PTP2I	PBS0 IFS1	ST	—	PTM2 capture input
	PTP2	PBS0	—	CMOS	PTM2 output
	RX2	PBS0 IFS3	ST	—	UART2 RX serial data input
PB4/C1X/TX2	PB4	PBPU PBS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	C1X	PBS1	—	CMOS	Comparator 1 output
	TX2	PBS1	—	CMOS	UART2 TX serial data output
PB5/$\overline{\text{RES}}$	PB5	PBPU RSTC	ST	CMOS	General purpose I/O. Register enabled pull-up.
	$\overline{\text{RES}}$	RSTC	ST	—	External reset input
PB6/STP1I/STP1/OSC1	PB6	PBPU PBS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	STP1I	PBS1 IFS1	ST	—	STM1 capture input
	STP1	PBS1	—	CMOS	STM1 output
	OSC1	PBS1	HXT	—	HXT oscillator pin
PB7/STCK1/OSC2	PB7	PBPU PBS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	STCK1	PBS1 IFS0	ST	—	STM1 clock input
	OSC2	PBS1	—	HXT	HXT oscillator pin

Pad Name	Function	OPT	I/T	O/T	Description
PC0/VREFI/AN0	PC0	PCPU PCS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	VREFI	PCS0	AN	—	A/D Converter OPA input
	AN0	PCS0	AN	—	A/D Converter analog input
PC1/C0X/VREF/AN1	PC1	PCPU PCS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	C0X	PCS0	—	CMOS	Comparator 0 output
	VREF	PCS0	AN	—	A/D Converter reference voltage input
	AN1	PCS0	AN	—	A/D Converter analog input
PC2/PTP0I/PTP0/AN2	PC2	PCPU PCS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PTP0I	PCS0 IFS1	ST	—	PTM0 capture input
	PTP0	PCS0	—	CMOS	PTM0 output
	AN2	PCS0	AN	—	A/D Converter analog input
PC3/PTCK0/AN3	PC3	PCPU PCS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PTCK0	PCS0 IFS0	ST	—	PTM0 clock input
	AN3	PCS0	AN	—	A/D Converter analog input
PC4/PTP1I/PTP1/AN4	PC4	PCPU PCS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PTP1I	PCS1 IFS1	ST	—	PTM1 capture input
	PTP1	PCS1	—	CMOS	PTM1 output
	AN4	PCS1	AN	—	A/D Converter analog input
PC5/PTCK1/AN5	PC5	PCPU PCS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PTCK1	PCS1 IFS0	ST	—	PTM1 clock input
	AN5	PCS1	AN	—	A/D Converter analog input
PC6/STP0I/STP0/AN6	PC6	PCPU PCS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	STP0I	PCS1 IFS1	ST	—	STM0 capture input
	STP0	PCS1	—	CMOS	STM0 output
	AN6	PCS1	AN	—	A/D Converter analog input
PC7/INT3/STCK0/AN7	PC7	PCPU PCS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	INT3	PCS1 INTEG INTC3 IFS2	ST	—	External Interrupt 3
	STCK0	PCS1 IFS0	ST	—	STM0 clock input
	AN7	PCS1	AN	—	A/D Converter analog input

Pad Name	Function	OPT	I/T	O/T	Description
PD0/INT2/STP1I/STP1/ AN8	PD0	PDPU PDS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	INT2	PDS0 INTEG INTC3 IFS2	ST	—	External Interrupt 2
	STP1I	PDS0 IFS1	ST	—	STM1 capture input
	STP1	PDS0	—	CMOS	STM1 output
	AN8	PDS0	AN	—	A/D Converter analog input
PD1/STCK1/RX1/AN9	PD1	PDPU PDS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	STCK1	PDS0 IFS0	ST	—	STM1 clock input
	RX1	PDS0 IFS3	ST	—	UART1 RX serial data input
	AN9	PDS0	AN	—	A/D Converter analog input
PD2/PTP2I/PTP2/TX1/ AN10	PD2	PDPU PDS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PTP2I	PDS0 IFS1	ST	—	PTM2 capture input
	PTP2	PDS0	—	CMOS	PTM2 output
	TX1	PDS0	—	CMOS	UART1 TX serial data output
	AN10	PDS0	AN	—	A/D Converter analog input
PD3/PTCK2/AN11	PD3	PDPU PDS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PTCK2	PDS0 IFS0	ST	—	PTM2 clock input
	AN11	PDS0	AN	—	A/D Converter analog input
PD4/PTP3I/PTP3/RX0/C1-	PD4	PDPU PDS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PTP3I	PDS1 IFS1	ST	—	PTM3 capture input
	PTP3	PDS1	—	CMOS	PTM3 output
	RX0	PDS1 IFS3	ST	—	UART0 RX serial data input
	C1-	PDS1	AN	—	Comparator 1 negative input
PD5/PTCK3/TX0/C1+	PD5	PDPU PDS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PTCK3	PDS1 IFS0	ST	—	PTM3 clock input
	TX0	PDS1	—	CMOS	UART0 TX serial data output
	C1+	PDS1	AN	—	Comparator 1 positive input
PD6/STP2I/STP2/ C1X	PD6	PDPU PDS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	STP2I	PDS1 IFS1	ST	—	STM2 capture input
	STP2	PDS1	—	CMOS	STM2 output
	C1X	PDS1	—	CMOS	Comparator 1 output
PE0/STCK0/$\overline{\text{SCSA}}$	PE0	PEPU PES0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	STCK0	PES0 IFS0	ST	—	STM0 clock input
	$\overline{\text{SCSA}}$	PES0	ST	CMOS	SPIA slave select

Pad Name	Function	OPT	I/T	O/T	Description
PE1/STP0I/STP0/SDOA	PE1	PEPU PES0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	STP0I	PES0 IFS1	ST	—	STM0 capture input
	STP0	PES0	—	CMOS	STM0 output
	SDOA	PES0	—	CMOS	SPIA data output
PE2/PTCK1/SDIA	PE2	PEPU PES0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PTCK1	PES0 IFS0	ST	—	PTM1 clock input
	SDIA	PES0	ST	—	SPIA data input
PE3/PTP1I/PTP1/SCKA	PE3	PEPU PES0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PTP1I	PES0 IFS1	ST	—	PTM1 capture input
	PTP1	PES0	—	CMOS	PTM1 output
	SCKA	PES0	ST	CMOS	SPIA serial clock
PE4/VDDIO	PE4	PEPU PES1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	VDDIO	PES1 PMPS	PWR	—	PE0~PE3 pin power for level shift
PF0/$\overline{SCS}$/SCOM0	PF0	PFPU PFS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	$\overline{SCS}$	PFS0 IFS2	ST	CMOS	SPI slave select
	SCOM0	PFS0	—	CMOS	Software LCD COM output
PF1/SDO/SCOM1	PF1	PFPU PFS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	SDO	PFS0	—	CMOS	SPI data output
	SCOM1	PFS0	—	CMOS	Software LCD COM output
PF2/SDI/SDA/SCOM2	PF2	PFPU PFS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	SDI	PFS0 IFS2	ST	—	SPI data input
	SDA	PFS0 IFS2	ST	NMOS	I^2C data line
	SCOM2	PFS0	—	CMOS	Software LCD COM output
PF3/SCK/SCL/SCOM3	PF3	PFPU PFS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	SCK	PFS0 IFS2	ST	CMOS	SPI serial clock
	SCL	PFS0 IFS2	ST	NMOS	I^2C clock line
	SCOM3	PFS0	—	CMOS	Software LCD COM output
PF4/PTCK0/XT2	PF4	PFPU PFS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PTCK0	PFS1 IFS0	ST	—	PTM0 clock input
	XT2	PFS1	—	LXT	LXT oscillator pin

Pad Name	Function	OPT	I/T	O/T	Description
PF5/PTP0I/PTP0/XT1	PF5	PFPU PFS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PTP0I	PFS1 IFS1	ST	—	PTM0 capture input
	PTP0	PFS1	—	CMOS	PTM0 output
	XT1	PFS1	LXT	—	LXT oscillator pin
PF6/STCK2/RX1/C0-	PF6	PFPU PFS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	STCK2	PFS1 IFS0	ST	—	STM2 clock input
	RX1	PFS1 IFS3	ST	—	UART1 RX serial data input
	C0-	PFS1	AN	—	Comparator 0 negative input
PF7/STP2I/STP2/TX1/C0+	PF7	PFPU PFS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	STP2I	PFS1 IFS1	ST	—	STM2 capture input
	STP2	PFS1	—	CMOS	STM2 output
	TX1	PFS1	—	CMOS	UART1 TX serial data output
	C0+	PFS1	AN	—	Comparator 0 positive input
PG0/RX2	PG0	PGPU PGS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	RX2	PGS0 IFS3	ST	—	UART2 RX serial data input
PG1/TX2	PG1	PGPU PGS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	TX2	PGS0	—	CMOS	UART2 TX serial data output
PG2~PG3	PGn	PGPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
PG4/PTCK3	PG4	PGPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PTCK3	IFS0	ST	—	PTM3 clock input
PG5/PTP3I/PTP3/PTCK2	PG5	PGPU PGS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PTP3I	PGS1 IFS1	ST	—	PTM3 capture input
	PTP3	PGS1	—	CMOS	PTM3 output
	PTCK2	PGS1 IFS0	ST	—	PTM2 clock input
PG6	PG6	PGPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
PG7	PG7	PGPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
PH0~PH1	PHn	PHPU PHS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
PH2/AN12	PH2	PHPU PHS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	AN12	PHS0	AN	—	A/D Converter analog input
PH3/AN13	PH3	PHPU PHS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	AN13	PHS0	AN	—	A/D Converter analog input
PH4/AN14	PH4	PHPU PHS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	AN14	PHS1	AN	—	A/D Converter analog input
PH5/AN15	PH5	PHPU PHS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	AN15	PHS1	AN	—	A/D Converter analog input

Pad Name	Function	OPT	I/T	O/T	Description
VDD	VDD	—	PWR	—	Positive power supply
VSS	VSS	—	PWR	—	Negative power supply, ground.
AVDD	AVDD	—	PWR	—	Analog positive power supply
AVSS	AVSS	—	PWR	—	Analog negative power supply, ground.

Legend: I/T: Input type	O/T: Output type
OPT: Optional by register option	ST: Schmitt Trigger input
CMOS: CMOS output	NMOS: NMOS output
HXT: High frequency crystal oscillator	LXT: Low frequency crystal oscillator
AN: Analog signal	PWR: Power

Absolute Maximum Ratings

Supply Voltage	V_{SS}-0.3V to V_{SS}+6.0V
Input Voltage	V_{SS}-0.3V to V_{DD}+0.3V
Storage Temperature	-50°C to 125°C
Operating Temperature	-40°C to 125°C
I_{OL} Total	80mA
I_{OH} Total	-80mA
Total Power Dissipation	500mW

Note: These are stress ratings only. Stresses exceeding the range specified under "Absolute Maximum Ratings" may cause substantial damage to the device. Functional operation of this device at other conditions beyond those listed in the specification is not implied and prolonged exposure to extreme conditions may affect device reliability.

D.C. Characteristics

For data in the following tables, note that factors such as oscillator type, operating voltage, operating frequency, pin load conditions, temperature and program instruction type, etc., can all exert an influence on the measured values.

Operating Voltage Characteristics

Ta=-40°C~125°C, unless otherwise specified.

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_{DD}	Operating Voltage – HXT	f_{SYS}=8MHz, Ta=-40°C~85°C	2.2	—	5.5	V
		f_{SYS}=8MHz	2.3	—	5.5	
		f_{SYS}=12MHz	2.7	—	5.5	
		f_{SYS}=16MHz, Ta=-40°C~105°C	3.3	—	5.5	
		f_{SYS}=16MHz	3.7	—	5.5	
	Operating Voltage – HIRC	f_{SYS}=8MHz	2.2	—	5.5	V
		f_{SYS}=12MHz	2.7	—	5.5	
		f_{SYS}=16MHz	3.3	—	5.5	
	Operating Voltage – LXT	f_{SYS}=32768Hz	2.2	—	5.5	V
	Operating Voltage – LIRC	f_{SYS}=32kHz	2.2	—	5.5	V

Standby Current Characteristics

Ta=25°C, unless otherwise specified.

Symbol	Standby Mode	Test Conditions		Min.	Typ.	Max.	Max. 85°C	Max. 105°C	Max. 125°C	Unit
		V_{DD}	Conditions							
I_{STB}	SLEEP Mode	2.2V	WDT off	—	0.13	0.18	3.30	7.20	15.30	μA
		3V		—	0.14	0.19	3.50	7.90	15.30	
		5V		—	0.21	0.50	4.50	9.50	20.00	
		2.2V	WDT on	—	1.2	2.4	3.3	6.7	17.5	
		3V		—	1.5	3.0	4.2	8.7	18.6	
		5V		—	3.0	5.0	6.5	11.8	21.6	
	IDLE0 Mode – LIRC	2.2V	f_{SUB} on	—	2.4	4.0	4.6	8.0	18.0	μA
		3V		—	3.0	5.0	5.7	10.0	20.0	
		5V		—	5	10	11	16	26	
	IDLE0 Mode – LXT	2.2V	f_{SUB} on	—	2.4	4.0	4.6	8.0	18.0	μA
		3V		—	3.0	5.0	5.7	10.0	20.0	
		5V		—	5	10	11	16	26	
	IDLE1 Mode – HIRC	2.2V	f_{SUB} on, f_{SYS}=8MHz	—	0.3	0.6	0.8	0.8	1.2	mA
		3V		—	0.5	0.8	1.2	1.2	1.6	
		5V		—	1.0	1.6	1.8	1.8	2.2	
		2.7V	f_{SUB} on, f_{SYS}=12MHz	—	0.4	0.6	0.8	0.8	1.0	
		3V		—	0.6	0.8	1.0	1.0	1.4	
		5V		—	1.00	1.20	1.44	1.44	1.80	
		3.3V	f_{SUB} on, f_{SYS}=16MHz	—	1.2	1.5	2.0	2.0	2.4	
		5V		—	1.5	3.0	3.5	3.5	4.0	
	IDLE1 Mode – HXT	2.2V	f_{SUB} on, f_{SYS}=8MHz Ta=-40°C~85°C	—	0.3	0.4	0.5	—	—	mA
		3V	f_{SUB} on, f_{SYS}=8MHz	—	0.4	0.8	1.2	1.2	1.6	
		5V		—	1.0	2.0	2.2	2.2	2.4	
		2.7V	f_{SUB} on, f_{SYS}=12MHz	—	0.4	0.8	1.0	1.2	1.6	
		3V		—	0.6	1.2	1.4	1.4	1.8	
		5V		—	1.0	1.8	2.0	2.0	2.2	
		3.3V	f_{SUB} on, f_{SYS}=16MHz	—	1.2	1.6	1.9	1.9	2.4	
		5V		—	1.6	2.2	2.6	2.6	4.0	

Note: When using the characteristic table data, the following notes should be taken into consideration:

1. Any digital inputs are setup in a non-floating condition.
2. All measurements are taken under conditions of no load and with all peripherals in an off state.
3. There are no DC current paths.
4. All Standby Current values are taken after a HALT instruction execution thus stopping all instruction execution.

Operating Current Characteristics

Ta=-40°C~125°C

Symbol	Operating Mode	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
I_{DD}	SLOW Mode – LIRC	2.2V	f_{SYS}=32kHz	—	15	45	μA
		3V		—	16	55	
		5V		—	24	65	
	SLOW Mode – LXT	2.2V	f_{SYS}=32768Hz	—	15.5	45.5	μA
		3V		—	17	56	
		5V		—	25	67	
	FAST Mode – HIRC	2.2V	f_{SYS}=8MHz	—	0.5	1.2	mA
		3V		—	1.0	2.0	
		5V		—	2.0	3.0	
		2.7V	f_{SYS}=12MHz	—	1.2	2.2	mA
		3V		—	1.50	2.75	
		5V		—	3.0	4.5	
		3.3V	f_{SYS}=16MHz	—	3.2	4.8	mA
		5V		—	4.5	7.0	
	FAST Mode – HXT	2.2V	f_{SYS}=8MHz, Ta=-40°C~85°C	—	0.5	1.2	mA
		3V	f_{SYS}=8MHz	—	1.0	2.0	
		5V		—	2.0	3.0	
		2.7V	f_{SYS}=12MHz	—	1.2	2.2	mA
		3V		—	1.50	2.75	
		5V		—	3.0	4.5	
		3.3V	f_{SYS}=16MHz	—	3.2	4.8	mA
		5V		—	4.5	7.0	

Note: When using the characteristic table data, the following notes should be taken into consideration:

1. Any digital inputs are setup in a non-floating condition.
2. All measurements are taken under conditions of no load and with all peripherals in an off state.
3. There are no DC current paths.
4. All Operating Current values are measured using a continuous NOP instruction program loop.

A.C. Characteristics

For data in the following tables, note that factors such as oscillator type, operating voltage, operating frequency and temperature etc., can all exert an influence on the measured values.

High Speed Internal Oscillator – HIRC – Frequency Accuracy

During the program writing operation the writer will trim the HIRC oscillator at a user selected HIRC frequency and user selected voltage of either 3V or 5V.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Temp.				
f_{HIRC}	8MHz Writer Trimmed HIRC Frequency	3V/5V	25°C	-1%	8	+1%	MHz
			-40°C~85°C	-2%	8	+2%	
			-40°C~105°C	-4%	8	+4%	
			-40°C~125°C	-5%	8	+5%	
		2.2V~5.5V	25°C	-2.5%	8	+2.5%	
			-40°C~85°C	-3%	8	+3%	
			-40°C~105°C	-5%	8	+5%	
			-40°C~125°C	-5%	8	+5%	
	12MHz Writer Trimmed HIRC Frequency	3V/5V	25°C	-1%	12	+1%	MHz
			-40°C~85°C	-2%	12	+2%	
			-40°C~105°C	-4%	12	+4%	
			-40°C~125°C	-5%	12	+5%	
		2.7V~5.5V	25°C	-2.5%	12	+2.5%	
			-40°C~85°C	-3%	12	+3%	
			-40°C~105°C	-5%	12	+5%	
			-40°C~125°C	-5%	12	+5%	
	16MHz Writer Trimmed HIRC Frequency	5V	25°C	-1%	16	+1%	MHz
			-40°C~85°C	-2%	16	+2%	
			-40°C~105°C	-4%	16	+4%	
			-40°C~125°C	-5%	16	+5%	
		3.3V~5.5V	25°C	-2.5%	16	+2.5%	
			-40°C~85°C	-3%	16	+3%	
			-40°C~105°C	-5%	16	+5%	
			-40°C~125°C	-5%	16	+5%	

Note: 1. The 3V/5V values for V_{DD} are provided as these are the two selectable fixed voltages at which the HIRC frequency is trimmed by the writer.

2. The row below the 3V/5V trim voltage row is provided to show the values for the full V_{DD} range operating voltage. It is recommended that the trim voltage is fixed at 3V for application voltage ranges from 2.2V to 3.6V and fixed at 5V for application voltage ranges from 3.3V to 5.5V.

3. The minimum and maximum tolerance values provided in the table are only for the frequency at which the writer trims the HIRC oscillator. After trimming at this chosen specific frequency any change in HIRC oscillator frequency using the oscillator register control bits by the application program will give a frequency tolerance to within ±20%.

Low Speed Internal Oscillator Characteristics – LIRC

Ta=-40°C~125°C, unless otherwise specified.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Temp.				
f_{LIRC}	LIRC Frequency	2.2V~5.5V	25°C	-5%	32	+5%	kHz
			-40°C~85°C	-10%	32	+10%	
			-40°C~105°C	-12%	32	+12%	
			-40°C~125°C	-13%	32	+13%	
t_{START}	LIRC Start Up Time	—	—	—	—	100	μs

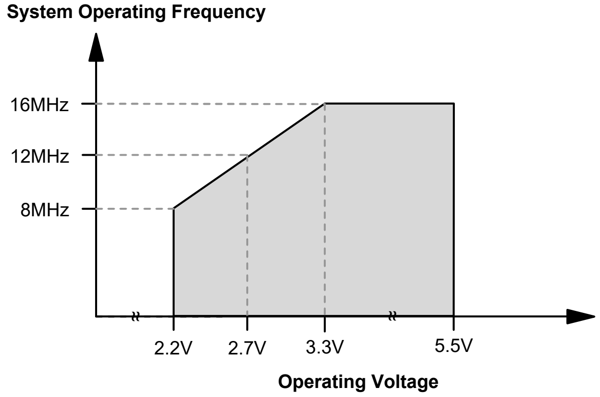
Low Speed Crystal Oscillator Characteristics – LXT

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
f_{LXT}	Oscillator Frequency	2.2V~5.5V	—	—	32768	—	Hz
Duty Cycle	Duty Cycle	—	—	40	—	60	%
t_{START}	Start Up Time	—	—	—	—	1000	ms
R_{NEG}	Negative Resistance	2.2V	—	3×ESR	—	—	Ω

*: C1, C2 and R_P are external components. C1=C2=10pF. R_P=10MΩ. C_L=7pF, ESR=30kΩ.

Operating Frequency Characteristic Curves



System Start Up Time Characteristics

Ta=-40°C~125°C

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
t_{SST}	System Start-up Time Wake-up from Condition where f_{SYS} is Off	$f_{SYS}=f_H\sim f_H/64$, $f_H=f_{HXT}$	—	128	—	t_{HXT}
		$f_{SYS}=f_H\sim f_H/64$, $f_H=f_{HIRC}$	—	16	—	t_{HIRC}
		$f_{SYS}=f_{SUB}=f_{LXT}$	—	1024	—	t_{LXT}
		$f_{SYS}=f_{SUB}=f_{LIRC}$	—	2	—	t_{LIRC}
	System Start-up Time Wake-up from Condition where f_{SYS} is On	$f_{SYS}=f_H\sim f_H/64$, $f_H=f_{HXT}$ or f_{HIRC}	—	2	—	t_H
		$f_{SYS}=f_{SUB}=f_{LXT}$ or f_{LIRC}	—	2	—	t_{SUB}
	System Speed Switch Time FAST to SLOW Mode or SLOW to FAST Mode	f_{HXT} switches from off → on	—	1024	—	t_{HXT}
		f_{HIRC} switches from off → on	—	16	—	t_{HIRC}
		f_{LXT} switches from off → on	—	1024	—	t_{LXT}

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
t_{RSTD}	System Reset Delay Time Reset Source from Power-on Reset or LVR Hardware Reset	RR_{POR}=5V/ms	42	48	54	ms
	System Reset Delay Time LVRC/WDTC/RSTC Software Reset	—				
	System Reset Delay Time Reset Source from WDT Overflow or Reset pin Reset	—	14	16	18	ms
t_{SRESET}	Minimum Software Reset Pulse Width to Reset	—	45	90	120	μs

Notes: 1. For the System Start-up time values, whether f_{SYS} is on or off depends upon the mode type and the chosen f_{SYS} system oscillator. Details are provided in the System Operating Modes section.

2. The time units, shown by the symbols t_{HIRC}. are the inverse of the corresponding frequency values as provided in the frequency tables. For example $t_{HIRC}=1/f_{HIRC}$, $t_{SYS}=1/f_{SYS}$ etc.

3. If the LIRC is used as the system clock and if it is off when in the SLEEP Mode, then an additional LIRC start up time, t_{START}, as provided in the LIRC frequency table, must be added to the t_{SST} time in the table above.

4. The System Speed Switch Time is effectively the time taken for the newly activated oscillator to start up.

Input/Output (without Multi-power) Characteristics

Ta = -40°C~125°C, unless otherwise specified.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{IL}	Input Low Voltage for I/O Ports (Except PE0~PE3 & $\overline{\text{RES}}$ Pin)	5V	—	0	—	1.5	V
		—	—	0	—	$0.2V_{DD}$	
	Input Low Voltage for External Reset Pin	—	—	0	—	$0.3V_{DD}$	
V_{IH}	Input High Voltage for I/O Ports (Except PE0~PE3 & $\overline{\text{RES}}$ Pin)	5V	—	3.5	—	5.0	V
		—	—	$0.8V_{DD}$	—	V_{DD}	
	Input High Voltage for External Reset Pin	—	—	$0.9V_{DD}$	—	V_{DD}	
I_{OL}	Sink Current for I/O Pins (Except PE0~PE3 pins)	3V	$V_{OL}=0.1V_{DD}$	16	32	—	mA
		5V		32	65	—	
I_{OH}	Source Current for I/O Pins (Except PE0~PE3 pins)	3V	$V_{OH}=0.9V_{DD}$, SLEDCn[m+1:m]=00, n=0, 1, 2 or 3; m=0, 2, 4 or 6	-0.7	-1.5	—	mA
		5V		-1.5	-2.9	—	
		3V	$V_{OH}=0.9V_{DD}$, SLEDCn[m+1:m]=01, n=0, 1, 2 or 3, m=0, 2, 4 or 6	-1.3	-2.5	—	
		5V		-2.5	-5.1	—	
		3V	$V_{OH}=0.9V_{DD}$, SLEDCn[m+1:m]=10, n=0, 1, 2 or 3, m=0, 2, 4 or 6	-1.8	-3.6	—	
		5V		-3.6	-7.3	—	
		3V	$V_{OH}=0.9V_{DD}$, SLEDCn[m+1:m]=11, n=0, 1, 2 or 3, m=0, 2, 4 or 6	-4	-8	—	
		5V		-8	-16	—	
R_{PH}	Pull-high Resistance for I/O Ports (Note)	3V	Ta=-40°C~105°C	20	60	100	kΩ
		5V		10	30	50	
		3V	—	20	60	120	
		5V		10	30	60	
I_{LEAK}	Input Leakage Current	5V	$V_{IN}=V_{DD}$ or $V_{IN}=V_{SS}$	—	—	±1	μA

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
t_{TPI}	TM Capture Input Minimum Pulse Width	—	—	0.3	—	—	μs
t_{TCK}	TM Clock Input Minimum Pulse Width	—	—	0.3	—	—	μs
t_{INT}	Interrupt Input Pin Minimum Pulse Width	—	—	10	—	—	μs
t_{RES}	External Reset Pin Minimum Pulse Width	—	—	10	—	—	μs

Note: The R_{PH} internal pull-high resistance value is calculated by connecting to ground and enabling the input pin with a pull-high resistor and then measuring the pin current at the specified supply voltage level. Dividing the voltage by this measured current provides the R_{PH} value.

Input/Output (with Multi-power) Characteristics

Ta=-40°C~125°C, unless otherwise specified.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD0}	V_{DD} Power Supply for PE0~PE3 Pins	—	—	2.2	5.0	5.5	V
V_{DDIO}	V_{DDIO} Power Supply for PE0~PE3 Pins	—	—	2.2	—	V_{DD}	V
V_{IL}	Input Low Voltage for PE0~PE3 pins	5V	Pin power=V_{DD} or V_{DDIO}, V_{DDIO}=V_{DD}	0	—	1.5	V
		—	Pin power=V_{DD} or V_{DDIO}	0	—	0.2 (V_{DD}/V_{DDIO})	
V_{IH}	Input High Voltage for PE0~PE3 pins	5V	Pin power=V_{DD} or V_{DDIO}, V_{DDIO}=V_{DD}	3.5	—	5.0	V
		—	Pin power=V_{DD} or V_{DDIO}	0.8 (V_{DD}/V_{DDIO})	—	V_{DD}/V_{DDIO}	
I_{OL}	Sink Current for PE0~PE3 pins	3V	V_{OL}=0.1(V_{DD}/V_{DDIO}) V_{DDIO}=V_{DD}	16	32	—	mA
		5V	V_{OL}=0.1(V_{DD}/V_{DDIO}) V_{DDIO}=V_{DD}	32	65	—	mA
			V_{OL}=0.1V_{DDIO}, V_{DDIO}=3V	20	40	—	mA
I_{OH}	Source Current for PE0~PE3 pins	3V	V_{OH}=0.9(V_{DD}/V_{DDIO}), V_{DDIO}=V_{DD}, SLEDCn[m+1, m]=00B (n=0, 1, 2, 3; m=0, 2, 4, 6)	-0.7	-1.5	—	mA
		5V	V_{OH}=0.9(V_{DD}/V_{DDIO}), V_{DDIO}=V_{DD}, SLEDCn[m+1, m]=00B (n=0, 1, 2, 3; m=0, 2, 4, 6)	-1.5	-2.9	—	mA
			V_{OH}=0.9V_{DDIO}, V_{DDIO}=3V, SLEDCn[m+1, m]=00B (n=0, 1, 2, 3; m=0, 2, 4, 6)	-0.40	-0.85	—	mA
		3V	V_{OH}=0.9(V_{DD}/V_{DDIO}), V_{DDIO}=V_{DD}, SLEDCn[m+1, m]=01B (n=0, 1, 2, 3; m=0, 2, 4, 6)	-1.3	-2.5	—	mA
		5V	V_{OH}=0.9(V_{DD}/V_{DDIO}), V_{DDIO}=V_{DD}, SLEDCn[m+1, m]=01B (n=0, 1, 2, 3; m=0, 2, 4, 6)	-2.5	-5.1	—	mA
			V_{OH}=0.9V_{DDIO}, V_{DDIO}=3V, SLEDCn[m+1, m]=01B (n=0, 1, 2, 3; m=0, 2, 4, 6)	-0.70	-1.35	—	mA

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
I_{OH}	Source Current for PE0~PE3 pins	3V	V_{OH}=0.9(V_{DD}/V_{DDIO}), V_{DDIO}=V_{DD}, SLEDCn[m+1, m]=10B (n=0, 1, 2, 3; m=0, 2, 4, 6)	-1.8	-3.6	—	mA
		5V	V_{OH}=0.9(V_{DD}/V_{DDIO}), V_{DDIO}=V_{DD}, SLEDCn[m+1, m]=10B (n=0, 1, 2, 3; m=0, 2, 4, 6)	-3.6	-7.3	—	mA
			V_{OH}=0.9V_{DDIO}, V_{DDIO}=3V, SLEDCn[m+1, m]=10B (n=0, 1, 2, 3; m=0, 2, 4, 6)	-0.95	-1.90	—	mA
		3V	V_{OH}=0.9(V_{DD}/V_{DDIO}), V_{DDIO}=V_{DD}, SLEDCn[m+1, m]=11B (n=0, 1, 2, 3; m=0, 2, 4, 6)	-4	-8	—	mA
		5V	V_{OH}=0.9(V_{DD}/V_{DDIO}), V_{DDIO}=V_{DD}, SLEDCn[m+1, m]=11B (n=0, 1, 2, 3; m=0, 2, 4, 6)	-8	-16	—	mA
			V_{OH}=0.9V_{DDIO}, V_{DDIO}=3V, SLEDCn[m+1, m]=11B (n=0, 1, 2, 3; m=0, 2, 4, 6)	-2.5	-5.0	—	mA
R_{PH}	Pull-high Resistance for PE0~PE3 Pins (Note)	3V	Pin power = V_{DD} or V_{DDIO} V_{DDIO}=V_{DD}, Ta=-40°C~105°C	20	60	100	kΩ
		5V	Pin power = V_{DD} or V_{DDIO} V_{DDIO}=V_{DD}, Ta=-40°C~105°C	10	30	50	kΩ
			V_{DDIO}=3V, Ta=-40°C~105°C	36	110	180	kΩ
		3V	Pin power = V_{DD} or V_{DDIO} V_{DDIO}=V_{DD}	20	60	110	kΩ
		5V	Pin power = V_{DD} or V_{DDIO} V_{DDIO}=V_{DD}	10	30	60	kΩ
			V_{DDIO}=3V	36	110	180	kΩ
I_{LEAK}	Input Leakage Current for PE0~PE3 Pins	5V	V_{IN}=V_{SS} or V_{IN}=V_{DD} or V_{DDIO}	—	—	±1	μA

Note: The R_{PH} internal pull-high resistance value is calculated by connecting to ground and enabling the input pin with a pull-high resistor and then measuring the pin current at the specified supply voltage level. Dividing the voltage by this measured current provides the R_{PH} value.

Memory Characteristics

Ta=-40°C~125°C, unless otherwise specified.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{RW}	V_{DD} for Read/Write	—	—	V_{DDmin}	—	V_{DDmax}	V
Flash Program Memory/Data EEPROM Memory							
t_{DEW}	Erase/Write Cycle Time – Flash Program Memory	—	—	—	2	3	ms
	Write Cycle Time – Data EEPROM Memory	—	—	—	4	6	
I_{DDPGM}	Programming/Erase Current on V_{DD}	—	—	—	—	5	mA
E_P	Cell Endurance – Flash Program Memory	—	—	10K	—	—	E/W
	Cell Endurance – Data EEPROM Memory	—	—	100K	—	—	E/W
t_{RETD}	ROM Data Retention Time	—	Ta=25°C	—	40	—	Year
RAM Data Memory							
V_{DR}	RAM Data Retention Voltage	—	Device in SLEEP Mode	1	—	—	V

LVD/LVR Electrical Characteristics

Ta=-40°C~125°C, unless otherwise specified.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{LVR}	Low Voltage Reset Voltage	—	LVR enable, voltage select 2.1V Ta=-40°C~85°C	- 5%	2.1	+ 5%	V
			LVR enable, voltage select 2.1V Ta=-40°C~105°C	- 6%	2.1	+ 6%	
			LVR enable, voltage select 2.1V	- 7%	2.1	+ 7%	
			LVR enable, voltage select 2.55V Ta=-40°C~85°C	- 5%	2.55	+ 5%	
			LVR enable, voltage select 2.55V Ta=-40°C~105°C	- 6%	2.55	+ 6%	
			LVR enable, voltage select 2.55V	- 7%	2.55	+ 7%	
			LVR enable, voltage select 3.15V Ta=-40°C~85°C	- 5%	3.15	+ 5%	
			LVR enable, voltage select 3.15V Ta=-40°C~105°C	- 6%	3.15	+ 6%	
			LVR enable, voltage select 3.15V	- 7%	3.15	+ 7%	
			LVR enable, voltage select 3.8V Ta=-40°C~85°C	- 5%	3.8	+ 5%	
			LVR enable, voltage select 3.8V Ta=-40°C~105°C	- 6%	3.8	+ 6%	
			LVR enable, voltage select 3.8V	- 7%	3.8	+ 7%	
V_{LVD}	Low Voltage Detection Voltage	—	LVD enable, voltage select 2.0V Ta=-40°C~85°C	- 5%	2.0	+ 5%	V
			LVD enable, voltage select 2.0V Ta=-40°C~105°C	- 6%	2.0	+ 6%	
			LVD enable, voltage select 2.0V	- 7%	2.0	+ 7%	
			LVD enable, voltage select 2.2V Ta=-40°C~85°C	- 5%	2.2	- 5%	
			LVD enable, voltage select 2.2V Ta=-40°C~105°C	- 6%	2.2	+ 6%	
			LVD enable, voltage select 2.2V	- 7%	2.2	+ 7%	
			LVD enable, voltage select 2.4V Ta=-40°C~85°C	- 5%	2.4	+ 5%	
			LVD enable, voltage select 2.4V Ta=-40°C~105°C	- 6%	2.4	+ 6%	
			LVD enable, voltage select 2.4V	- 7%	2.4	+ 7%	
			LVD enable, voltage select 2.7V Ta=-40°C~85°C	- 5%	2.7	+ 5%	
			LVD enable, voltage select 2.7V Ta=-40°C~105°C	- 6%	2.7	+ 6%	
			LVD enable, voltage select 2.7V	- 7%	2.7	+ 7%	
			LVD enable, voltage select 3.0V Ta=-40°C~85°C	- 5%	3.0	+ 5%	
			LVD enable, voltage select 3.0V Ta=-40°C~105°C	- 6%	3.0	+ 6%	
			LVD enable, voltage select 3.0V	- 7%	3.0	+ 7%	
			LVD enable, voltage select 3.3V Ta=-40°C~85°C	- 5%	3.3	+ 5%	
			LVD enable, voltage select 3.3V Ta=-40°C~105°C	- 6%	3.3	+ 6%	
			LVD enable, voltage select 3.3V	- 7%	3.3	+ 7%	

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{LVD}	Low Voltage Detection Voltage	—	LVD enable, voltage select 3.6V Ta=-40°C~85°C	- 5%	3.6	+ 5%	V
			LVD enable, voltage select 3.6V Ta=-40°C~105°C	- 6%	3.6	+ 6%	
			LVD enable, voltage select 3.6V	- 7%	3.6	+ 7%	
			LVD enable, voltage select 4.0V Ta=-40°C~85°C	- 5%	4.0	+ 5%	
			LVD enable, voltage select 4.0V Ta=-40°C~105°C	- 6%	4.0	+ 6%	
			LVD enable, voltage select 4.0V	- 7%	4.0	+ 7%	
$I_{LVRLVDBG}$	Operating Current	3V	LVD enable, LVR enable, VBGEN=0	—	—	20	μA
		5V	LVD enable, LVR enable, VBGEN=0	—	20	25	
		3V	LVD enable, LVR enable, VBGEN=1	—	—	25	
		5V	LVD enable, LVR enable, VBGEN=1	—	25	30	
t_{LVDS}	LVDO Stable Time	—	For LVR enable, VBGEN=0, LVD off → on, Ta=-40°C~85°C	—	—	18	μs
			For LVR enable, VBGEN=0, LVD off → on, Ta=-40°C~105°C	—	—	20	
			For LVR enable, VBGEN=0, LVD off → on	—	—	25	
t_{LVR}	Minimum Low Voltage Width to Reset	—	—	120	240	480	μs
t_{LVD}	Minimum Low Voltage Width to Interrupt	—	—	60	120	240	μs
I_{LVR}	Additional Current for LVR Enable	—	LVD disable, VBGEN=0	—	—	24	μA

A/D Converter Electrical Characteristics

Ta=-40°C~125°C, unless otherwise specified.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
AV_{DD}	Operating Voltage	—	—	2.2	—	5.5	V
V_{ADI}	Input Voltage	—	—	0	—	V_{REF}	V
V_{REF}	Reference Voltage	—	—	2	—	AV_{DD}	V
N_R	Resolution	—	—	—	—	12	Bit
DNL	Differential Non-linearity	—	t_{ADCK}=0.5μs, V_{REF}=AV_{DD}, Ta=-40°C~85°C	-3	—	3	LSB
			t_{ADCK}=0.5μs, V_{REF}=AV_{DD}, Ta=-40°C~105°C	-4	—	4	
			t_{ADCK}=0.5μs, V_{REF}=AV_{DD}	-5	—	5	
INL	Integral Non-linearity	—	t_{ADCK}=0.5μs, V_{REF}=AV_{DD}, Ta=-40°C~85°C	-4	—	4	LSB
			t_{ADCK}=0.5μs, V_{REF}=AV_{DD}, Ta=-40°C~105°C	-5	—	5	
			t_{ADCK}=0.5μs, V_{REF}=AV_{DD}	-11	—	11	

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
I_{ADC}	Additional Current for ADC Enable	2.2V	No load (t_{ADCK}=0.5μs)	—	300	420	μA
		3V	No load (t_{ADCK}=0.5μs)	—	340	500	μA
		5V	No load (t_{ADCK}=0.5μs)	—	500	700	μA
t_{ADCK}	Clock Period	—	—	0.5	—	10	μs
t_{ON2ST}	ADC On-to-Start Time	—	—	4	—	—	μs
t_{ADS}	Sampling Time	—	—	—	4	—	t_{ADCK}
t_{ADC}	Conversion Time (Include ADC Sample and Hold Time)	—	—	—	16	—	t_{ADCK}
GERR	A/D Conversion Gain Error	—	$V_{REF}=AV_{DD}$	-4	—	4	LSB
OSRR	A/D Conversion Offset Error	—	$V_{REF}=AV_{DD}$	-4	—	4	LSB
I_{PGA}	Additional Current for PGA Enable	3V	No load	—	300	450	μA
		5V	No load	—	400	550	μA
V_{CM}	PGA Common Mode Voltage Range	3V	—	V_{SS}+0.1	—	V_{DD}-1.4	V
		5V		V_{SS}+0.1	—	V_{DD}-1.4	
V_{IR}	PGA Input Voltage Range	3V	Gain=1, PGAIS=0, Relative gain, gain error < ±5%	V_{SS}+0.1	—	V_{DD}-1.4	V
		5V		V_{SS}+0.1	—	V_{DD}-1.4	
V_{OR}	PGA Maximum Output Voltage Range	3V	—	V_{SS}+0.1	—	V_{DD}-0.1	V
		5V	—	V_{SS}+0.1	—	V_{DD}-0.1	V
V_{VR}	Fix Voltage Output of PGA	2.7V~ 5.5V	$V_{RI}=V_{BGREF}$ (PGAIS=1)	-1.5%	2	+1.5%	V
		3.2V~ 5.5V	Ta=-40°C~105°C, $V_{RI}=V_{BGREF}$ (PGAIS=1)	-1.5%	3	+1.5%	V
		4.2V~ 5.5V	Ta=-40°C~105°C, $V_{RI}=V_{BGREF}$ (PGAIS=1)	-1.5%	4	+1.5%	V

Reference Voltage Characteristics

Ta=-40°C~125°C, unless otherwise specified.

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{BGREF}	Bandgap Reference Voltage	—	Ta=25°C	-1%	1.2	+1%	V
			Ta=-40°C~85°C	-2%	1.2	+2%	
			Ta=-40°C~105°C	-2%	1.2	+2%	
			—	-2%	1.2	+2%	
I_{BGREF}	Operating Current	5.5V	Ta=-40°C~85°C	—	25	40	μA
			Ta=-40°C~105°C	—	25	40	
			—	—	25	50	
PSRR	Power Supply Rejection Ratio	—	Ta=25°C, $V_{RIPPLE}=1V_{P-P}$, f_{RIPPLE}=100Hz	75	—	—	dB
En	Output Noise	—	Ta=25°C, no load current, f=0.1Hz~10Hz	—	300	—	$μV_{RMS}$
I_{DRV}	Buffer Driving Capacity	—	ΔV_{BGREF}=-1%	1	—	—	mA
I_{SD}	Shutdown Current	—	VBGREN=0	—	—	0.1	μA
t_{START}	Startup Time	2.2V~5.5V	Ta=25°C	—	—	400	μs

Note: 1. All the above parameters are measured under no load conditions unless otherwise described.
2. A 0.1μF ceramic capacitor should be connected between VDD and GND.
3. The V_{BGREF} voltage is used as the A/D converter PGA input.

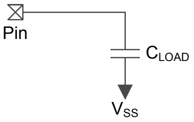
Comparator Characteristics

Ta=-40°C~125°C, unless otherwise specified.
All measurement is under CMPn positive input voltage = (V_{DD}-1.4)/2 and remain constant

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	Operating Voltage	—	—	2.2	—	5.5	V
I_{CMP}	Additional Current for Comparator Enable	3V	CNVTn[1:0]=00B, Ta=-40°C~85°C	—	1	5	μA
			CNVTn[1:0]=00B, Ta=-40°C~105°C	—	1	5	
			CNVTn[1:0]=00B	—	1	7	
		5V	CNVTn[1:0]=00B, Ta=-40°C~85°C	—	1	5	
			CNVTn[1:0]=00B, Ta=-40°C~105°C	—	1	5	
			CNVTn[1:0]=00B	—	1	9	
		3V	CNVTn[1:0]=01B	—	—	30	
		5V		—	14	30	
		3V	CNVTn[1:0]=10B	—	—	65	
		5V		—	36	65	
		3V	CNVTn[1:0]=11B	—	—	110	
		5V		—	58	110	
V_{OS}	Input Offset Voltage	3V	Without calibration, (CnOF[4:0]=10000B), CNVTn[1:0]=00B	-10	—	10	mV
		5V					
		3V	With calibration, CNVTn[1:0]=00B	-4	—	4	
		5V					
V_{CM}	Common Mode Voltage Range	3V	CNVTn[1:0]=00B	0.21	—	V_{DD}-1.0	V
		5V		0.23	—	V_{DD}-1.0	
		3V	CNVTn[1:0]=01B	0.38	—	V_{DD}-1.0	V
		5V		0.43	—	V_{DD}-1.0	
		3V	CNVTn[1:0]=10B	0.55	—	V_{DD}-1.0	V
		5V		0.63	—	V_{DD}-1.0	
		3V	CNVTn[1:0]=11B	0.70	—	V_{DD}-1.0	V
		5V		0.80	—	V_{DD}-1.0	
A_{OL}	Open Loop Gain	3V	CNVTn[1:0]=00B	60	—	—	dB
		5V	CNVTn[1:0]=00B	60	80	—	
V_{HYS}	Hysteresis	3V	CNVTn[1:0]=00B	10	—	30	mV
		5V	CNVTn[1:0]=00B	10	24	30	
t_{RP}	Response Time (Note2)	3V/5V	With 10mV overdrive, CNVTn[1:0]=00B	—	25	40	μs
			With 100mV overdrive, CNVTn[1:0]=00B	—	20	40	
		3V/5V	With 10mV overdrive, CNVTn[1:0]=01B	—	1.5	4.0	
			With 100mV overdrive, CNVTn[1:0]=01B	—	1.2	3.0	
		3V/5V	With 10mV overdrive, CNVTn[1:0]=10B	—	0.8	2.0	
			With 100mV overdrive, CNVTn[1:0]=10B	—	0.5	1.5	
		3V/5V	With 10mV overdrive, CNVTn[1:0]=11B	—	0.7	1.5	
			With 100mV overdrive, CNVTn[1:0]=11B	—	0.3	1.0	

Note: 1. The input offset voltage should first be calibrated when the comparator operates with the compared threshold voltage level lower than 250mV. Otherwise, the input offset voltage will be out of specification.

2. Load Condition: C_{LOAD}=50pF



Load Condition

Software Controlled LCD Driver Electrical Characteristics

Ta=-40°C~125°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
I_{BIAS}	V_{DD}/2 Bias Current for LCD	3V	ISEL[1:0] = 00B	10.5	15.0	50.0	μA
		5V		17.5	25.0	60.0	
		3V	ISEL[1:0] = 01B	21	30	70	
		5V		35	50	85	
		3V	ISEL[1:0] = 10B	42	60	100	
		5V		70	100	150	
		3V	ISEL[1:0] = 11B	82.6	118.0	153.4	
		5V		140	200	260	
V_{SCOM}	V_{DD}/2 Voltage for LCD SCOM Port	2.2V~5.5V	No load	0.475V_{DD}	0.500V_{DD}	0.525V_{DD}	V

CAN Module Electrical Characteristics

D.C Characteristics

Ta=-40°C~125°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	Operating Voltage (Crystal OSC)	—	f_{CAN}=8MHz	2.2	—	5.5	V
		—	f_{CAN}=16MHz	3.3	—	5.5	

A.C Characteristics

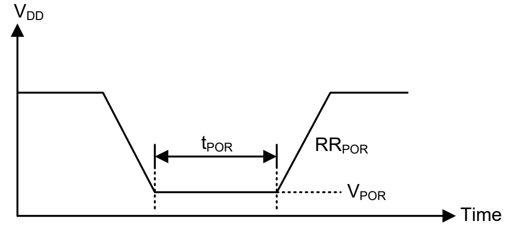
Ta=-40°C~125°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
f_{CAN}	System Clock (Crystal OSC)	2.2V~5.5V	—	—	8	—	MHz
		3.3V~5.5V		—	16	—	
f_{MCLK}	Memory Clock(Crystal OSC)	2.2V~5.5V	—	—	8	—	MHz
		3.3V~5.5V		—	16	—	

Power-on Reset Characteristics

Ta=-40°C~125°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{POR}	V_{DD} Start Voltage to Ensure Power-on Reset	—	—	—	—	100	mV
RR_{POR}	V_{DD} Rising Rate to Ensure Power-on Reset	—	—	0.035	—	—	V/ms
t_{POR}	Minimum Time for V_{DD} Stays at V_{POR} to Ensure Power-on Reset	—	—	1	—	—	ms



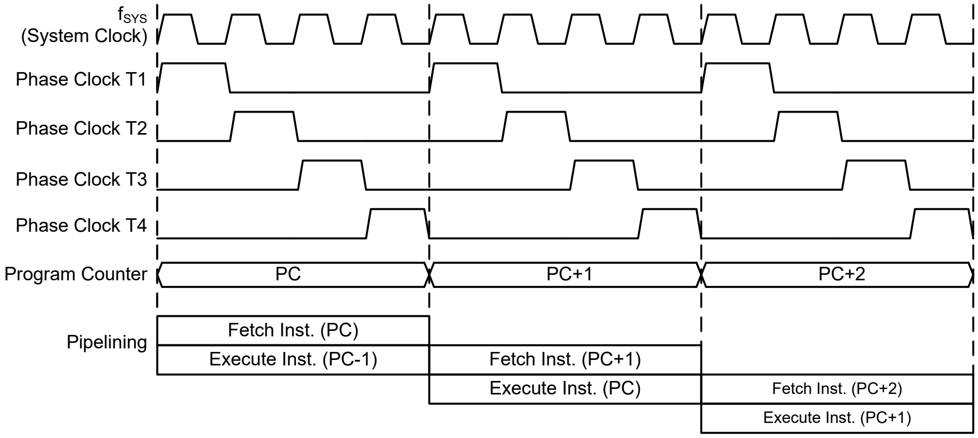
System Architecture

A key factor in the high-performance features of the range of microcontrollers is attributed to their internal system architecture. The device takes advantage of the usual features found within RISC microcontrollers providing increased speed of operation and enhanced performance. The pipelining scheme is implemented in such a way that instruction fetching and instruction execution are overlapped, hence instructions are effectively executed in one or two cycles for most of the standard or extended instructions respectively. The exceptions to this are branch or call instructions which need one more cycle. An 8-bit wide ALU is used in practically all instruction set operations, which carries out arithmetic operations, logic operations, rotation, increment, decrement, branch decisions, etc. The internal data path is simplified by moving data through the Accumulator and the ALU. Certain internal registers are implemented in the Data Memory and can be directly or indirectly addressed. The simple addressing methods of these registers along with additional architectural features ensure that a minimum of external components is required to provide a functional I/O and A/D control system with maximum reliability and flexibility. This makes the device suitable for low-cost, high-volume production for controller applications.

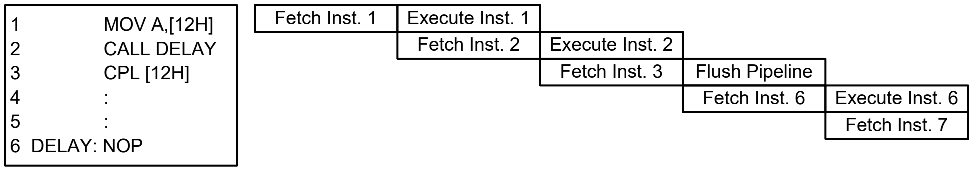
Clocking and Pipelining

The main system clock, derived from either a HXT, LXT, HIRC or LIRC oscillator is subdivided into four internally generated non-overlapping clocks, T1~T4. The Program Counter is incremented at the beginning of the T1 clock during which time a new instruction is fetched. The remaining T2~T4 clocks carry out the decoding and execution functions. In this way, one T1~T4 clock cycle forms one instruction cycle. Although the fetching and execution of instructions takes place in consecutive instruction cycles, the pipelining structure of the microcontroller ensures that instructions are effectively executed in one instruction cycle. The exception to this are instructions where the contents of the Program Counter are changed, such as subroutine calls or jumps, in which case the instruction will take one more instruction cycle to execute.

For instructions involving branches, such as jump or call instructions, two machine cycles are required to complete instruction execution. An extra cycle is required as the program takes one cycle to first obtain the actual jump or call address and then another cycle to actually execute the branch. The requirement for this extra cycle should be taken into account by programmers in timing sensitive applications.



System Clocking and Pipelining


```
1         MOV A,[12H]
2         CALL DELAY
3         CPL [12H]
4         :
5         :
6  DELAY: NOP
```

Instruction Fetching

Program Counter

During program execution, the Program Counter is used to keep track of the address of the next instruction to be executed. It is automatically incremented by one each time an instruction is executed except for instructions, such as "JMP" or "CALL" that demands a jump to a non-consecutive Program Memory address. For the device whose memory capacity is greater than 8K words the Program Memory address may be located in a certain program memory bank which is selected by the program memory bank pointer bits, PBPn. Only the lower 8 bits, known as the Program Counter Low Register, are directly addressable by the application program.

When executing instructions requiring jumps to non-consecutive addresses such as a jump instruction, a subroutine call, interrupt or reset, etc., the microcontroller manages program control by loading the required address into the Program Counter. For conditional skip instructions, once the condition has been met, the next instruction, which has already been fetched during the present instruction execution, is discarded and a dummy cycle takes its place while the correct instruction is obtained.

Program Counter	
Program Counter High Byte	PCL Register
PBP1~PBP0, PC12~PC8	PCL7~PCL0

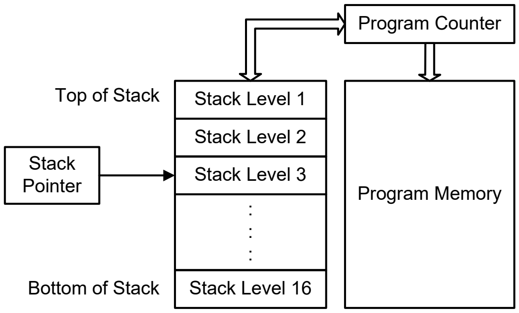
Program Counter

The lower byte of the Program Counter, known as the Program Counter Low register or PCL, is available for program control and is a readable and writeable register. By transferring data directly into this register, a short program jump can be executed directly. However, as only this low byte is available for manipulation, the jumps are limited to the present page of memory, that is 256 locations. When such program jumps are executed it should also be noted that a dummy cycle will be inserted. Manipulating the PCL register may cause program branching, so an extra cycle is needed to pre-fetch.

Stack

This is a special part of the memory which is used to save the contents of the Program Counter only. The stack is organised into 16 levels and is neither part of the data nor part of the program space, and is neither readable nor writeable. The activated level is indexed by the Stack Pointer, and is neither readable nor writeable. At a subroutine call or interrupt acknowledge signal, the contents of the Program Counter are pushed onto the stack. At the end of a subroutine or an interrupt routine, signaled by a return instruction, RET or RETI, the Program Counter is restored to its previous value from the stack. After a device reset, the Stack Pointer will point to the top of the stack.

If the stack is full and an enabled interrupt takes place, the interrupt request flag will be recorded but the acknowledge signal will be inhibited. When the Stack Pointer is decremented, by RET or RETI, the interrupt will be serviced. This feature prevents stack overflow allowing the programmer to use the structure more easily. However, when the stack is full, a CALL subroutine instruction can still be executed which will result in a stack overflow. Precautions should be taken to avoid such cases which might cause unpredictable program branching. If the stack is overflow, the first Program Counter save in the stack will be lost.



Arithmetic and Logic Unit – ALU

The arithmetic-logic unit or ALU is a critical area of the microcontroller that carries out arithmetic and logic operations of the instruction set. Connected to the main microcontroller data bus, the ALU receives related instruction codes and performs the required arithmetic or logical operations after which the result will be placed in the specified register. As these ALU calculation or operations may result in carry, borrow or other status changes, the status register will be correspondingly updated to reflect these changes. The ALU supports the following functions:

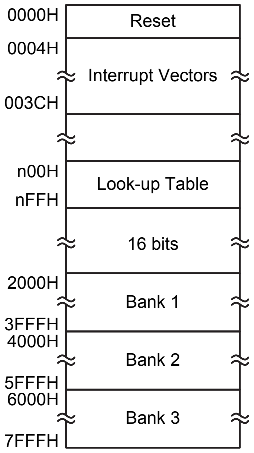
- Arithmetic operations:
  ADD, ADDM, ADC, ADCM, SUB, SUBM, SBC, SBCM, DAA,
  LADD, LADDM, LADC, LADCM, LSUB, LSUBM, LSBC, LSBCM, LDAA
- Logic operations:
  AND, OR, XOR, ANDM, ORM, XORM, CPL, CPLA,
  LAND, LANDM, LOR, LORM, LXOR, LXORM, LCPL, LCPLA
- Rotation:
  RRA, RR, RRCA, RRC, RLA, RL, RLCA, RLC,
  LRR, LRRA, LRRCA, LRRC, LRLA, LRL, LRLCA, LRLC
- Increment and Decrement:
  INCA, INC, DECA, DEC,
  LINCA, LINC, LDECA, LDEC
- Branch decision:
  JMP, SZ, SZA, SNZ, SIZ, SDZ, SIZA, SDZA, CALL, RET, RETI,
  LSNZ, LSZ, LSZA, LSIZ, LSIZA, LSDZ, LSDZA

Flash Program Memory

The Program Memory is the location where the user code or program is stored. For this device the Program Memory is Flash type, which means it can be programmed and re-programmed a large number of times, allowing the user the convenience of code modification on the same device. By using the appropriate programming tools, the Flash device offer users the flexibility to conveniently debug and develop their applications while also offering a means of field programming and updating.

Structure

The Program Memory has a capacity of 32K×16 bits. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries. Table data, which can be setup in any location within the Program Memory, is addressed by a separate table pointer register.



Program Memory Structure

Special Vectors

Within the Program Memory, certain locations are reserved for the reset and interrupts. The location 000H is reserved for use by the device reset for program initialisation. After a device reset is initiated, the program will jump to this location and begin execution.

Look-up Table

Any location within the Program Memory can be defined as a look-up table where programmers can store fixed data. To use the look-up table, the table pointer must first be setup by placing the address of the look up data to be retrieved in the table pointer register, TBLP and TBHP. These registers define the total address of the look-up table.

After setting up the table pointer, the table data can be retrieved from the Program Memory using the corresponding table read instruction such as "TABRD [m]" or "TABRDL [m]" respectively when the memory [m] is located in sector 0. If the memory [m] is located in other sectors, the data can be retrieved from the program memory using the corresponding extended table read instruction such as "LTABRD [m]" or "LTABRDL [m]" respectively. When the instruction is executed, the lower order table byte from the Program Memory will be transferred to the user defined Data Memory register [m] as specified in the instruction. The higher order table data byte from the Program Memory will be transferred to the TBLH special register.

The accompanying diagram illustrates the addressing data flow of the look-up table.

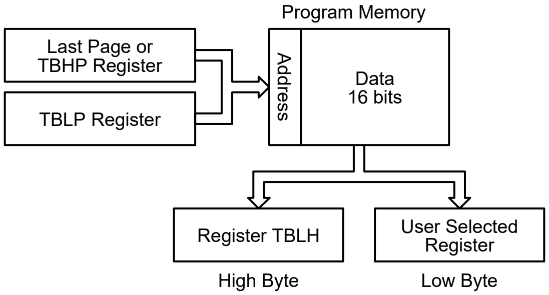


Table Program Example

The following example shows how the table pointer and table data is defined and retrieved from the microcontroller. This example uses raw table data located in the Program Memory which is stored there using the ORG statement. The value at this ORG statement is "1F00H" which is located in ROM Bank 3 and refers to the start address of the last page within the 32K words Program Memory of the device. The table pointer low byte register is setup here to have an initial value of "06H". This will ensure that the first data read from the data table will be at the Program Memory address "7F06H" or 6 locations after the start of the last page. Note that the value for the table pointer is referenced to the first address specified by TBLP and TBHP if the "TABRD [m]" or "LTABRD [m]" instruction is being used. The high byte of the table data which in this case is equal to zero will be transferred to the TBLH register automatically when the "TABRD [m]" or "LTABRD [m]" instruction is executed.

Because the TBLH register is a read/write register and can be restored, care should be taken to ensure its protection if both the main routine and Interrupt Service Routine use table read instructions. If using the table read instructions, the Interrupt Service Routines may change the value of the TBLH and subsequently cause errors if used again by the main routine. As a rule it is recommended that simultaneous use of the table read instructions should be avoided. However, in situations where simultaneous use cannot be avoided, the interrupts should be disabled prior to the execution of any main routine table-read instructions. Note that all table related instructions require two instruction cycles to complete their operation.

Table Read Program Example

```
rombank3 code3
ds .section 'data'
tempreg1 db ?      ; temporary register #1
tempreg2 db ?      ; temporary register #2
:
:
code0 .section 'code'
mov a,06h          ; initialise low table pointer - note that this address is referenced
mov tblp,a         ; to the last page or the page that tbhp pointed
mov a,7Fh          ; initialise high table pointer
mov tbhp,a         ; It is not necessary to set tbhp register if executing "tabrdl"
                   ; instruction
:
:
tabrd tempreg1     ; transfers value in table referenced by table pointer data at program
                   ; memory address " 7F06H" transferred to tempreg1 and TBLH
dec tblp           ; reduce value of table pointer by one
tabrd tempreg2     ; transfers value in table referenced by table pointer
                   ; data at program memory address "7F05H" transferred to
```

```
                  ; tempreg2 and TBLH in this example the data "1AH" is
                  ; transferred to tempreg1 and data "0FH" to register tempreg2
:
:
code3 .section 'code'
org 1F00h         ; sets initial address of program memory
dc 00Ah, 00Bh, 00Ch, 00Dh, 00Eh, 00Fh, 01Ah, 01Bh
:
```

In Circuit Programming – ICP

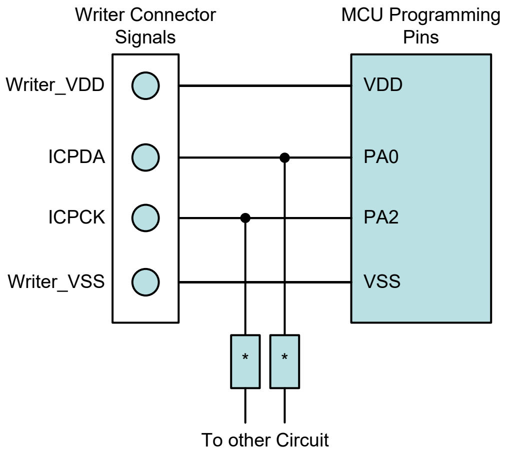
The provision of Flash type Program Memory provides the user with a means of convenient and easy upgrades and modifications to their programs on the same device. As an additional convenience, a means of programming the microcontroller in-circuit has provided using a 4-pin interface. This provides manufacturers with the possibility of manufacturing their circuit boards complete with a programmed or un-programmed microcontroller, and then programming or upgrading the program at a later stage. This enables product manufacturers to easily keep their manufactured products supplied with the latest program releases without removal and re-insertion of the device.

The Flash MCU to Writer Programming Pin correspondence table is as follows:

Writer Pins	MCU Programming Pins	Pin Description
ICPDA	PA0	Programming serial data/address
ICPCK	PA2	Programming clock
VDD	VDD	Power supply
VSS	VSS	Ground

The Program Memory can be programmed serially in-circuit using this 4-wire interface. Data is downloaded and uploaded serially on a single pin with an additional line for the clock. Two additional lines are required for the power supply. The technical details regarding the in-circuit programming of the device are beyond the scope of this document and will be supplied in supplementary literature.

During the programming process, taking control of the ICPDA and ICPCK pins for data and clock programming purposes. The user must there take care to ensure that no other outputs are connected to these two pins.



Note: * may be resistor or capacitor. The resistance of * must be greater than 1kΩ or the capacitance of * must be less than 1nF.

On-Chip Debug Support – OCDS

An EV chip exists for the purposes of device emulation. This EV chip device also provides an "On-Chip Debug" function to debug the device during the development process. The EV chip and the actual MCU device are almost functionally compatible except for the "On-Chip Debug" function. Users can use the EV chip device to emulate the real chip device behavior by connecting the OCDSDA and OCDSCK pins to the HT-IDE development tools. The OCDSDA pin is the OCDS Data/Address input/output pin while the OCDSCK pin is the OCDS clock input pin. When users use the EV chip for debugging, other functions which are shared with the OCDSDA and OCDSCK pins in the actual MCU device will have no effect in the EV chip. However, the two OCDS pins which are pin-shared with the ICP programming pins are still used as the Flash Memory programming pins for ICP. For a more detailed OCDS description, refer to the corresponding document.

e-Link Pins	EV Chip Pins	Pin Description
OCDSDA	OCDSDA	On-chip debug support data/address input/output
OCDSCK	OCDSCK	On-chip debug support clock input
VDD	VDD	Power supply
VSS	VSS	Ground

In Application Programming – IAP

Flash type Program Memory provides the user with a means of convenient and easy upgrades and modifications to their programs on the same device. The provision of IAP function offers users the convenience of Flash Memory multi-programming features. The convenience of the IAP function is that it can execute the updated program procedure using its internal firmware, without requiring an external Program Writer or PC. In addition, the IAP interface can also be any type of communication protocol, such as UART, using I/O pins. Regarding the internal firmware, the user can select versions provided by Holtek or create their own. The following section illustrates the procedures regarding how to implement the IAP firmware.

Flash Memory Read/Write Size

The flash memory Erase and Write operations are carried out in a page format while the Read operation is carried out in a word format. The page size and write buffer size are both assigned with a capacity of 64 words. Note that the Erase operation should be executed before the Write operation is executed.

When the Flash Memory Erase/Write Function is successfully enabled, the CFWEN bit will be set high. When the CFWEN bit is set high, the data can be written into the write buffer. The FWT bit is used to initiate the write process and then indicate the write operation status. This bit is set high by application programs to initiate a write process and will be cleared by hardware if the write process is finished.

The Read operation can be carried out by executing a specific read procedure. The FRDEN bit is used to enable the read function and the FRD bit is used to initiate the read process by application programs and then indicate the read operation status. When the read process is finished, this bit will be cleared by hardware.

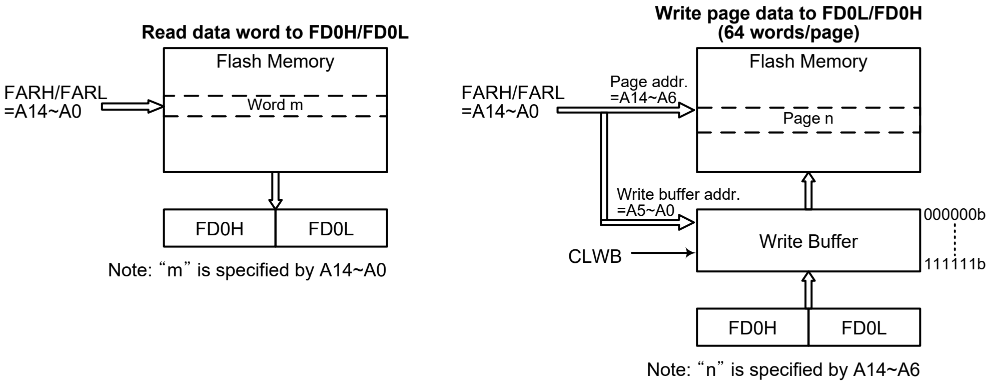
Operations	Format
Erase	64 words/page
Write	64 words/time
Read	1 word/time
Note: Page size = Write buffer size = 64 words.	

IAP Read/Write Format

Erase Page	FARH	FARL [7:6]	FARL [5:0]
0	0000 0000	00	xx xxxx
1	0000 0000	01	xx xxxx
2	0000 0000	10	xx xxxx
3	0000 0000	11	xx xxxx
4	0000 0001	00	xx xxxx
: :	: :	: :	: :
510	0111 1111	10	xx xxxx
511	0111 1111	11	xx xxxx

"x": don't care

Erase Page Number and Selection



Flash Memory IAP Read/Write Structure

Write Buffer

The write buffer is used to store the written data temporarily when executing the write operation. The Write Buffer can be filled with written data after the Flash Memory Erase/Write Function has been successfully enabled by executing the Flash Memory Erase/Write Function Enable procedure. The write buffer can be cleared by configuring the CLWB bit in the FC2 register. The CLWB bit can be set high to enable the Clear Write Buffer procedure. When the procedure is finished this bit will be cleared to low by the hardware. It is recommended that the write buffer should be cleared by setting the CLWB bit high before the write buffer is used for the first time or when the data in the write buffer is updated.

The write buffer size is 64 words corresponding to a page. The write buffer address is mapped to a specific flash memory page specified by the memory address bits, A14~A6. The data written into the FD0L and FD0H registers will be loaded into the write buffer. When data is written into the high byte data register, FD0H, it will result in the data stored in the high and low byte data registers both being written into the write buffer. It will also cause the flash memory address to be incremented by one, after which the new address will be loaded into the FARH and FARL address registers. When the flash memory address reaches the page boundary, 111111b of a page with 64 words, the address will now not be incremented but will stop at the last address of the page. At this point a new page address should be specified for any other erase/write operations.

After a write process is finished, the write buffer will automatically be cleared by the hardware. Note that the write buffer should be cleared manually by the application program when the data written into the flash memory is incorrect in the data verification step. The data should again be written into the write buffer after the write buffer has been cleared when the data is found to be incorrect during the data verification step.

IAP Flash Program Memory Registers

There are two address registers, four 16-bit data registers and three control registers. The address and data register pairs are located in Sector 0 while the control registers are located in Sector 1. Read and Write operations to the Flash memory are carried out using 16-bit data operations using the address and data registers and the control register. Several registers control the overall operation of the internal Flash Program Memory. The address registers are named FARL and FARH, the data registers are named FDnL and FDnH and the control registers are named FC0, FC1 and FC2. As the address and data register pairs are located in Sector 0, they can be directly accessed in the same way as any other Special Function Register. The control registers, being located in Sector 1, can be addressed directly only using the corresponding extended instructions or can be read from or written to indirectly using the MP1H/MP1L or MP2H/MP2L Memory Pointer pairs and Indirect Addressing Register, IAR1 or IAR2.

Register Name	Bit							
	7	6	5	4	3	2	1	0
FC0	CFWEN	FMOD2	FMOD1	FMOD0	FWPEN	FWT	FRDEN	FRD
FC1	D7	D6	D5	D4	D3	D2	D1	D0
FC2	—	—	—	—	—	—	—	CLWB
FARL	A7	A6	A5	A4	A3	A2	A1	A0
FARH	—	A14	A13	A12	A11	A10	A9	A8
FD0L	D7	D6	D5	D4	D3	D2	D1	D0
FD0H	D15	D14	D13	D12	D11	D10	D9	D8
FD1L	D7	D6	D5	D4	D3	D2	D1	D0
FD1H	D15	D14	D13	D12	D11	D10	D9	D8
FD2L	D7	D6	D5	D4	D3	D2	D1	D0
FD2H	D15	D14	D13	D12	D11	D10	D9	D8
FD3L	D7	D6	D5	D4	D3	D2	D1	D0
FD3H	D15	D14	D13	D12	D11	D10	D9	D8

IAP Registers List

- **FARL Register**

Bit	7	6	5	4	3	2	1	0
Name	A7	A6	A5	A4	A3	A2	A1	A0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 Flash Memory Address bit 7 ~ bit 0

- **FARH Register**

Bit	7	6	5	4	3	2	1	0
Name	—	A14	A13	A12	A11	A10	A9	A8
R/W	—	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	0	0	0	0	0	0	0

Bit 7 Unimplemented, read as “0”

Bit 6~0 Flash Memory Address bit 14 ~ bit 8

- **FD0L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 The first Flash Memory data word bit 7 ~ bit 0

Note that data written into the low byte data register FD0L will only be stored in the FD0L register and not loaded into the lower 8-bit write buffer.

- **FD0H Register**

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 The first Flash Memory data word bit 15 ~ bit 8

Note that when 8-bit data is written into the high byte data register FD0H, the whole 16-bit of data stored in the FD0H and FD0L registers will simultaneously be loaded into the 16-bit write buffer after which the contents of the Flash memory address register pair, FARH and FARL, will be incremented by one.

- **FD1L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 The second Flash Memory data word bit 7 ~ bit 0

- **FD1H Register**

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 The second Flash Memory data word bit 15 ~ bit 8

- **FD2L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 The third Flash Memory data word bit 7 ~ bit 0

- **FD2H Register**

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 The third Flash Memory data word bit 15 ~ bit 8

- **FD3L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 The fourth Flash Memory data word bit 7 ~ bit 0

- **FD3H Register**

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 The fourth Flash Memory data word bit 15 ~ bit 8

- **FC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	CFWEN	FMOD2	FMOD1	FMOD0	FWPEN	FWT	FRDEN	FRD
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **CFWEN**: Flash Memory Erase/Write function enable control
0: Flash memory erase/write function is disabled
1: Flash memory erase/write function has been successfully enabled

When this bit is cleared to 0 by application program, the Flash memory erase/write function is disabled. Note that this bit cannot be set high by application programs. Writing "1" into this bit results in no action. This bit is used to indicate the Flash memory erase/write function status. When this bit is set to 1 by the hardware, it means that the Flash memory erase/write function is enabled successfully. Otherwise, the Flash memory erase/write function is disabled if the bit is zero.

Bit 6~4 **FMOD2~FMOD0**: Flash memory Mode selection
000: Write Mode
001: Page erase Mode
010: Reserved
011: Read Mode
100: Reserved
101: Reserved
110: Flash memory Erase/Write function Enable Mode
111: Reserved

These bits are used to select the Flash Memory operation modes. Note that the "Flash memory Erase/Write function Enable Mode" should first be successfully enabled before the Erase or Write Flash memory operation is executed.

Bit 3 **FWPEN**: Flash memory Erase/Write function enable procedure Trigger
0: Erase/Write function enable procedure is not triggered or procedure timer times out
1: Erase/Write function enable procedure is triggered and procedure timer starts to count

This bit is used to activate the flash memory Erase/Write function enable procedure and an internal timer. It is set by the application programs and then cleared by the hardware when the internal timer times out. The correct patterns must be written into the FD1L/FD1H, FD2L/FD2H and FD3L/FD3H register pairs respectively as soon as possible after the FWPEN bit is set high.

Bit 2 **FWT**: Flash memory write initiate control
0: Do not initiate Flash memory write or indicating that a Flash memory write process has completed
1: Initiate Flash memory write process
This bit is set by software and cleared by the hardware when the Flash memory write process has completed.

Bit 1 **FRDEN**: Flash memory read enable control
0: Flash memory read disable
1: Flash memory read enable
This is the Flash memory Read Enable Bit which must be set high before any Flash memory read operations are carried out. Clearing this bit to zero will inhibit Flash memory read operations.

Bit 0 **FRD**: Flash memory read initiate control
0: Do not initiate Flash memory read or indicating that a Flash memory read process has completed
1: Initiate Flash memory read process
This bit is set by software and cleared by the hardware when the Flash memory read process has completed.

Note: 1. The FWT, FRDEN and FRD bits cannot be set to “1” at the same time with a single instruction.
2. Ensure that the f_{SUB} clock is stable before executing the erase or write operation.
3. Note that the CPU will be stopped when a read, erase or write operation is successfully activated.
4. Ensure that the read, erase or write operation is totally complete before executing other operations.

- **FC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: Chip Reset Pattern
When a specific value of “55H” is written into this register, a reset signal will be generated to reset the whole chip.

- **FC2 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	—	CLWB
R/W	—	—	—	—	—	—	—	R/W
POR	—	—	—	—	—	—	—	0

Bit 7~1 Unimplemented, read as “0”

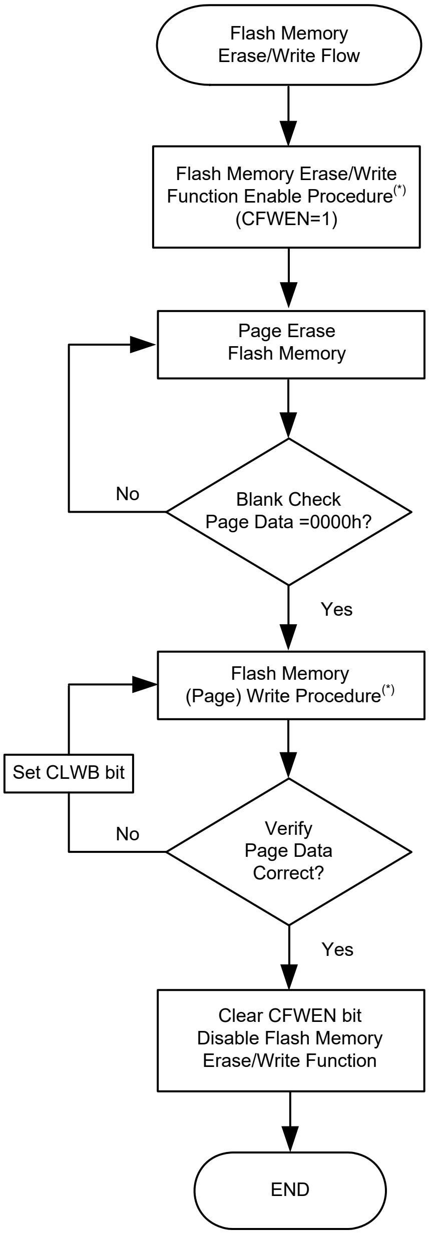
Bit 0 **CLWB**: Flash memory Write Buffer Clear control
0: Do not initiate a Write Buffer Clear process or indicating that a Write Buffer Clear process has completed.
1: Initiate Write Buffer Clear process
This bit is set by software and cleared by hardware when the Write Buffer Clear process has completed.

Flash Memory Erase/Write Flow

It is important to understand the Flash memory Erase/Write flow before the Flash memory contents are updated. Users can refer to the corresponding operation procedures when developing their IAP program to ensure that the flash memory contents are correctly updated.

Flash Memory Erase/Write Flow Descriptions:

1. Activate the "Flash Memory Erase/Write function enable procedure" first. When the Flash Memory Erase/Write function is successfully enabled, the CFWEN bit in the FC0 register will automatically be set high by hardware. After this, Erase or Write operations can be executed on the Flash memory. Refer to the "Flash Memory Erase/Write Function Enable Procedure" for details.
2. Configure the flash memory address to select the desired erase page and then erase this page.
3. Execute a Blank Check operation to ensure whether the page erase operation is successful or not. The "TABRD" instruction should be executed to read the flash memory contents and to check if the contents is 0000h or not. If the flash memory page erase operation fails, users should go back to Step 2 and execute the page erase operation again.
4. Write data into the specific page. Refer to the "Flash Memory Write Procedure" for details.
5. Execute the "TABRD" instruction to read the flash memory contents and check if the written data is correct or not. If the data read from the flash memory is different from the written data, it means that the page write operation has failed. The CLWB bit should be set high to clear the write buffer and then write the data into the specific page again if the write operation has failed.
6. Clear the CFWEN bit to disable the Flash Memory Erase/Write function enable mode if the current page Erase and Write operations are complete if no more pages need to be erased or written.



Flash Memory Erase/Write Flow

Note: The Flash Memory Erase/Write Function Enable procedure and Flash Memory Write procedure will be described in the following sections.

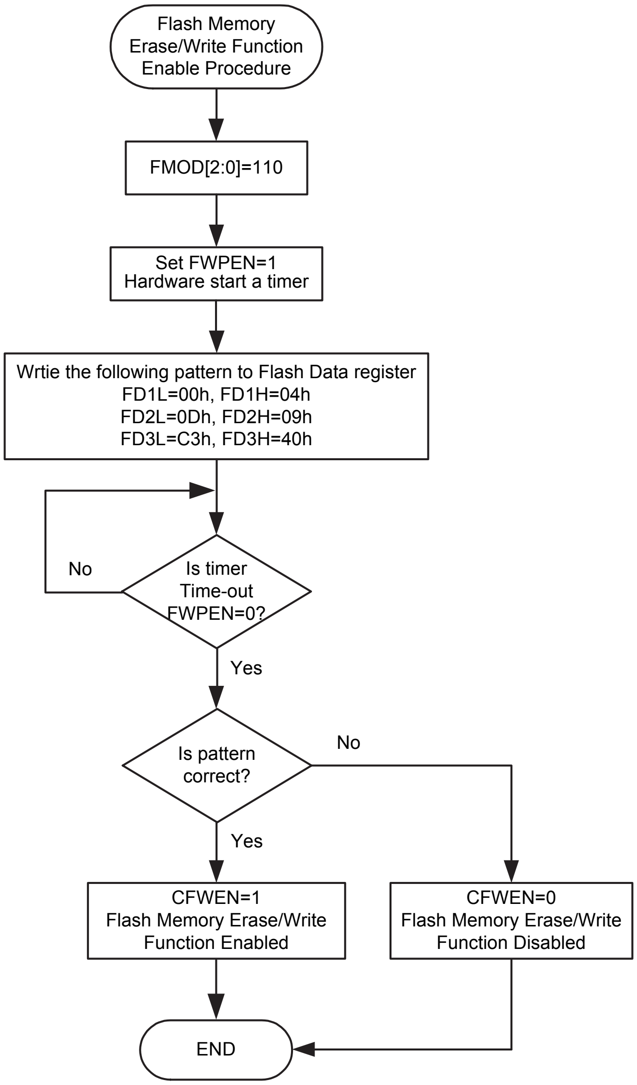
Flash Memory Erase/Write Function Enable Procedure

The Flash Memory Erase/Write Function Enable Mode is specially designed to prevent the flash memory contents from being wrongly modified. In order to allow users to change the Flash memory data using the IAP control registers, users must first enable the Flash memory Erase/Write function.

Flash Memory Erase/Write Function Enable Procedure Description

1. Write data "110" to the FMOD [2:0] bits in the FC0 register to select the Flash Memory Erase/Write Function Enable Mode.
2. Set the FWPEN bit in the FC0 register to "1" to activate the Flash Memory Erase/Write Function. This will also activate an internal timer.
3. Write the correct data pattern into the Flash data registers, FD1L~FD3L and FD1H~FD3H, as soon as possible after the BWT bit is set high. The enable Flash memory erase/write function data pattern is 00H, 0DH, C3H, 04H, 09H and 40H corresponding to the FD1L~FD3L and FD1H~FD3H registers respectively.
4. Once the timer has timed out, the BWT bit will automatically be cleared to 0 by hardware regardless of the input data pattern.
5. If the written data pattern is incorrect, the Flash memory erase/write function will not be enabled successfully and the above steps should be repeated. If the written data pattern is correct, the Flash memory erase/write function will be enabled successfully.
6. Once the Flash memory erase/write function is enabled, the Flash memory contents can be updated by executing the page erase and write operations using the IAP control registers.

To disable the Flash memory erase/write function, the CFWEN bit in the FC0 register can be cleared. There is no need to execute the above procedure.



Flash Memory Erase/Write Function Enable Procedure

Flash Memory Write Procedure

After the Flash memory erase/write function has been successfully enabled as the CFWEN bit is set high, the data to be written into the flash memory can be loaded into the write buffer. The selected flash memory page data should be erased by properly configuring the IAP control registers before the data write procedure is executed.

The write buffer size is 64 words, known as a page, whose address is mapped to a specific flash memory page specified by the memory address bits, A14~A6. It is important to ensure that the page where the write buffer data is located is the same one which the memory address bits, A14~A6, specify.

Flash Memory Consecutive Write Description

The maximum amount of write data is 64 words for each write operation. The write buffer address will be automatically incremented by one when consecutive write operations are executed. The start address of a specific page should first be written into the FARL and FARH registers. Then the data word should first be written into the FD0L register and then the FD0H register. At the same time the write buffer address will be incremented by one and then the next data word can be written into the FD0L and FD0H registers for the next address without modifying the address register pair, FARH and FARL. When the write buffer address reaches the page boundary the address will not be further incremented but will stop at the last address of the page.

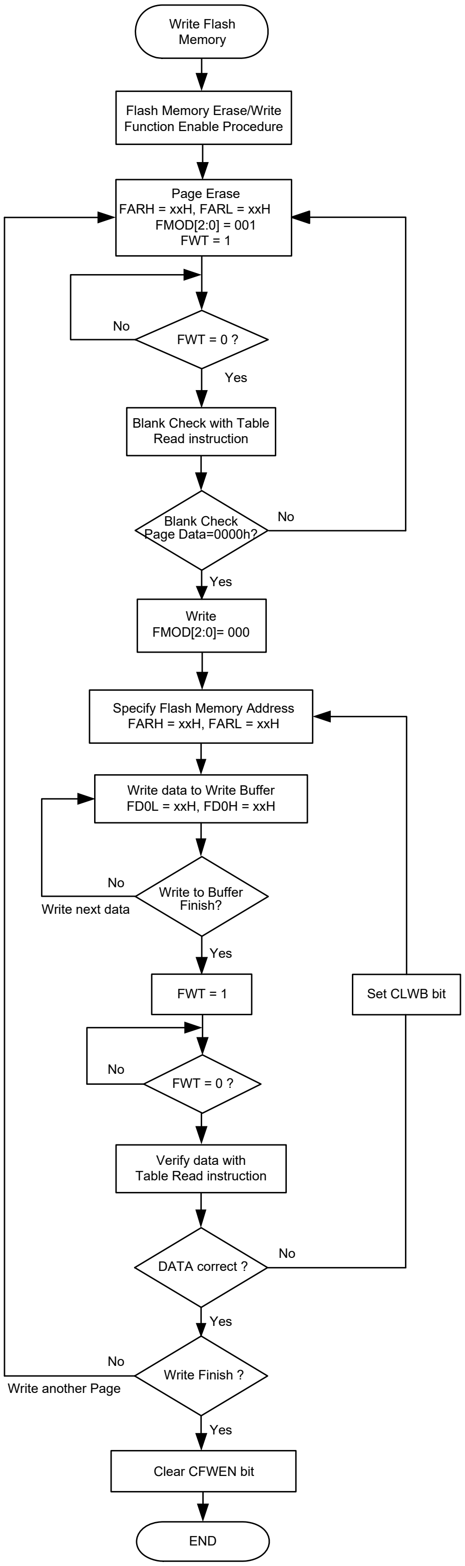
1. Activate the "Flash Memory Erase/Write function enable procedure". Check the CFWEN bit value and then execute the erase/write operations if the CFWEN bit is set high. Refer to the "Flash Memory Erase/Write function enable procedure" for more details.
2. Set the FMOD field to "001" to select the erase operation. Set the FWT bit high to erase the desired page which is specified by the FARH and FARL registers. Wait until the FWT bit goes low.
3. Execute a Blank Check operation using the table read instruction to ensure that the erase operation has successfully completed.

   Go to step 2 if the erase operation is not successful.

   Go to step 4 if the erase operation is successful.
4. Set the FMOD field to "000" to select the write operation.
5. Setup the desired start address in the FARH and FARL registers. Write the desired data words consecutively into the FD0L and FD0H registers within a page as specified by their consecutive addresses. The maximum written data number is 64 words.
6. Set the FWT bit high to write the data words from the write buffer to the flash memory. Wait until the FWT bit goes low.
7. Verify the data using the table read instruction to ensure that the write operation has successfully completed.

   If the write operation has not successfully completed, set the CLWB bit high to clear the write buffer and then go to step 5.

   Go to step 8 if the write operation is successful.
8. Clear the CFWEN bit low to disable the Flash memory erase/write function.



Flash Memory Consecutive Write Procedure

Note: 1. When the erase or write operation is successfully activated, all CPU operations will temporarily cease.
2. It will take a typical time of 2.2ms for the FWT bit state changing from high to low.

Flash Memory Non-Consecutive Write Description

The main difference between Flash Memory Consecutive and Non-Consecutive Write operations is whether the data words to be written are located in consecutive addresses or not. If the data to be written is not located in consecutive addresses the desired address should be re-assigned after a data word is successfully written into the Flash Memory.

A two data word non-consecutive write operation is taken as an example here and described as follows:

1. Activate the "Flash Memory Erase/Write function enable procedure". Check the CFWEN bit value and then execute the erase/write operation if the CFWEN bit is set high. Refer to the "Flash Memory Erase/Write function enable procedure" for more details.
2. Set the FMOD field to "001" to select the erase operation. Set the FWT bit high to erase the desired page which is specified by the FARH and FARL registers. Wait until the FWT bit goes low.
3. Execute a Blank Check operation using the table read instruction to ensure that the erase operation has successfully completed.

   Go to step 2 if the erase operation is not successful.

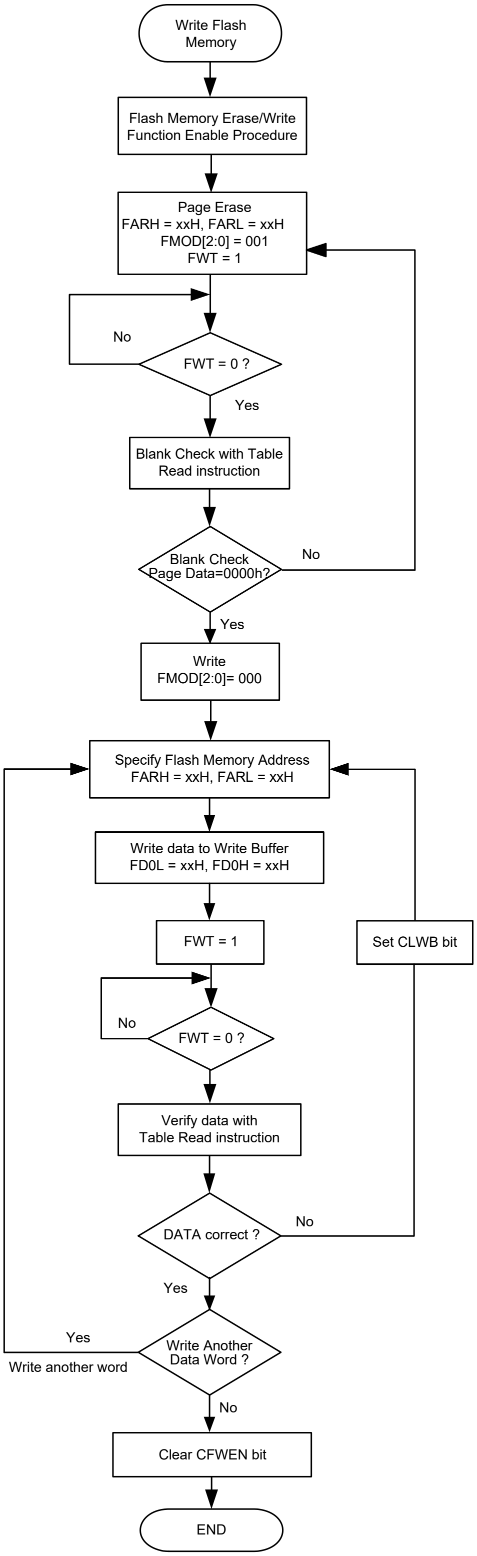
   Go to step 4 if the erase operation is successful.
4. Set the FMOD field to "000" to select the write operation.
5. Setup the desired address ADDR1 in the FARH and FRARL registers. Write the desired data word DATA1 first into the FD0L register and then into the FD0H register.
6. Set the FWT bit high to transfer the data word from the write buffer to the flash memory. Wait until the FWT bit goes low.
7. Verify the data using the table read instruction to ensure that the write operation has successfully completed.

   If the write operation has not successfully completed, set the CLWB bit high to clear the write buffer and then go to step 5.

   Go to step 8 if the write operation is successful.
8. Setup the desired address ADDR2 in the FARH and FRARL registers. Write the desired data word DATA2 first into the FD0L register and then into the FD0H register.
9. Set the FWT bit high to transfer the data word from the write buffer to the flash memory. Wait until the FWT bit goes low.
10. Verify the data using the table read instruction to ensure that the write operation has successfully completed.

    If the write operation has not successfully completed, set the CLWB bit high to clear the write buffer and then go to step 8.

    Go to step 11 if the write operation is successful.
11. Clear the CFWEN bit low to disable the Flash memory erase/write function.



Flash Memory Non-Consecutive Write Procedure

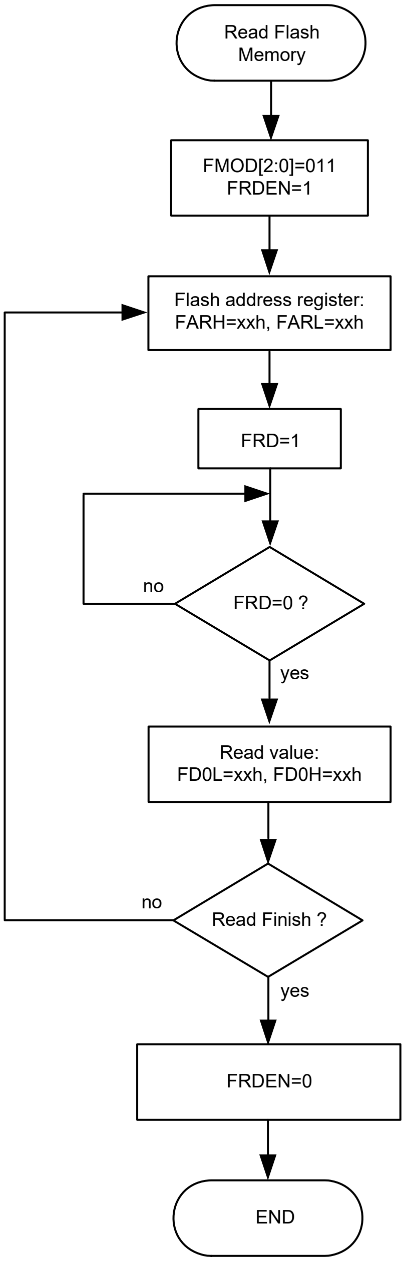
Note: 1. When the erase or write operation is successfully activated, all CPU operations will temporarily cease.
2. It will take a typical time of 2.2ms for the FWT bit state changing from high to low.

Important Points to Note for Flash Memory Write Operations

1. The "Flash Memory Erase/Write Function Enable Procedure" must be successfully activated before the Flash Memory erase/write operation is executed.
2. The Flash Memory erase operation is executed to erase a whole page.
3. The whole write buffer data will be written into the flash memory in a page format. The corresponding address cannot exceed the page boundary.
4. After the data is written into the flash memory the flash memory contents must be read out using the table read instruction, TABRD, and checked if it is correct or not. If the data written into the flash memory is incorrect, the write buffer should be cleared by setting the CLWB bit high and then writing the data again into the write buffer. Then activate a write operation on the same flash memory page without erasing it. The data check, buffer clear and data re-write steps should be repeatedly executed until the data written into the flash memory is correct.
5. The system frequency should be setup to the maximum application frequency when data write and data check operations are executed using the IAP function.

Flash Memory Read Procedure

To activate the Flash Memory Read procedure, the FMOD field should be set to "011" to select the flash memory read mode and the FRDEN bit should be set high to enable the read function. The desired flash memory address should be written into the FARH and FARL registers and then the FRD bit should be set high. After this the flash memory read operation will be activated. The data stored in the specified address can be read from the data registers, FD0H and FD0L, when the FRD bit goes low. There is no need to first activate the Flash Memory Erase/Write Function Enable Procedure before the flash memory read operation is executed.



Flash Memory Read Procedure

Note: 1. When the read operation is successfully activated, all CPU operations will temporarily cease.
2. It will take a typical time of three instruction cycles for the FRD bit state changing from high to low.

Data Memory

The Data Memory is a volatile area of 8-bit wide RAM internal memory and is the location where temporary information is stored.

Categorized into two types, the first of these is an area of RAM where special function registers are located. These registers have fixed locations and are necessary for correct operation of the device. Many of these registers can be read from and written to directly under program control, however, some remain protected from user manipulation. The second area of Data Memory is reserved for general purpose use. All locations within this area are read and write accessible under program control.

Switching between the different Data Memory sectors is achieved by properly setting the Memory Pointers to correct value.

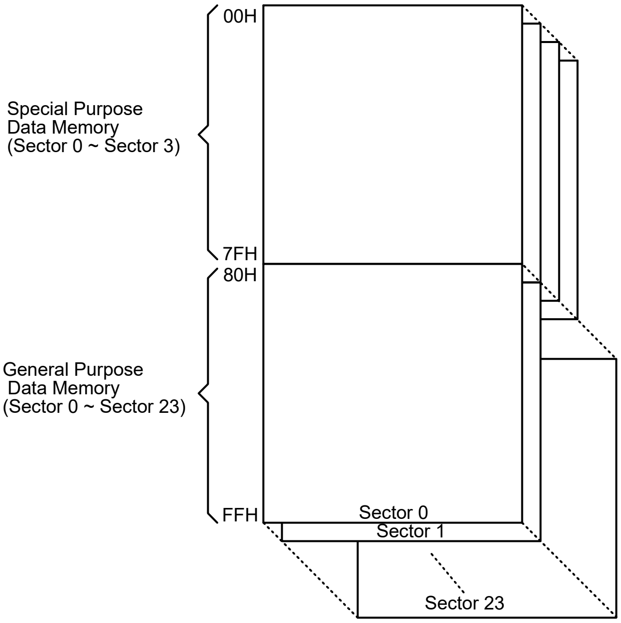
Structure

The Data Memory is subdivided into several sectors, all of which are implemented in 8-bit wide Memory. Each of the Data memoru sectors is categorized into two types, the Special Purpose Data Memory and the General Purpose Data Memory.

The address range of the Special Purpose Data Memory for the device is from 00H to 7FH while the General Purpose Data Memoru address range is from 80H to FFH.

Special Purpose Data Memory	General Purpose Data Memory	
Available Sectors	Capacity	Sector: Address
0, 1, 2, 3	3072×8	0: 80H~FFH 1: 80H~FFH : 23: 80H~FFH

Data Memory Summary



Data Memory Structure

Data Memory Addressing

For device that supports the extended instructions, there is no Bank Pointer for Data Memory. The Bank Pointer, PBP, is only available for Program Memory. For Data Memory the desired Sector is pointed by the MP1H or MP2H register and the certain Data Memory address in the selected sector is specified by the MP1L or MP2L register when using indirect addressing access.

Direct Addressing can be used in all sectors using the corresponding instruction which can address all available data memory space. For the accessed data memory which is located in any data memory sectors except sector 0, the extended instructions can be used to access the data memory instead of using the indirect addressing access. The main difference between standard instructions and extended instructions is that the data memory address “m” in the extended instructions can be 13 valid bits for the device, the high byte indicates a sector and the low byte indicates a specific address.

General Purpose Data Memory

All microcontroller programs require an area of read/write memory where temporary data can be stored and retrieved for use later. It is this area of RAM memory that is known as General Purpose Data Memory. This area of Data Memory is fully accessible by the user programing for both reading and writing operations. By using the bit operation instructions individual bits can be set or reset under program control giving the user a large range of flexibility for bit manipulation in the Data Memory.

Special Purpose Data Memory

This area of Data Memory is where registers, necessary for the correct operation of the microcontroller, are stored. Most of the registers are both readable and writeable but some are protected and are readable only, the details of which are located under the relevant Special Function Register section. Note that for locations that are unused, any read instruction to these addresses will return the value “00H”.

	Sector 0	Sector 1	Sector 2	Sector 3
00H	IAR0	CRCCR	CTRLRL	TREQR1L
01H	MP0	CRCIN	▒	TREQR1H
02H	IAR1	CRCDL	STATRL	TREQR2L
03H	MP1L	CRCDH	▒	TREQR2H
04H	MP1H	▒	ERRCNTL	▒
05H	ACC	▒	ERRCNTH	▒
06H	PCL	IECC	BTRL	▒
07H	TBLP	▒	BTRH	▒
08H	TBLH	▒	INTRL	▒
09H	TBHP	▒	INTRH	▒
0AH	STATUS	▒	TESTRL	▒
0BH	PBP	U2SR	▒	▒
0CH	IAR2	U2CR1	BRPERL	▒
0DH	MP2L	U2CR2	▒	▒
0EH	MP2H	TXR_RXR2	▒	▒
0FH	RSTFC	BRG2	▒	▒
10H	INTC0	U0SR	IF1CREQL	NEWDT1L
11H	INTC1	U0CR1	IF1CREQH	NEWDT1H
12H	INTC2	U0CR2	IF1CMSKL	NEWDT2L
13H	INTC3	TXR_RXR0	▒	NEWDT2H
14H	PA	BRG0	IF1MSK1L	▒
15H	PAC	U1SR	IF1MSK1H	▒
16H	PAPU	U1CR1	IF1MSK2L	▒
17H	PAWU	U1CR2	IF1MSK2H	▒
18H	PB	TXR_RXR1	IF1ARB1L	▒
19H	PBC	BRG1	IF1ARB1H	▒
1AH	PBPU	PTM1C0	IF1ARB2L	▒
1BH	PC	PTM1C1	IF1ARB2H	▒
1CH	PCC	PTM1DL	IF1MCTRL	▒
1DH	PCPU	PTM1DH	IF1MCTRH	▒
1EH	PD	PTM1AL	IF1DTA1L	▒
1FH	PDC	PTM1AH	IF1DTA1H	▒
20H	PDPU	PTM1RPL	IF1DTA2L	INTPND1L
21H	PE	PTM1RPH	IF1DTA2H	INTPND1H
22H	PEC	PTM2C0	IF1DTB1L	INTPND2L
23H	PEPU	PTM2C1	IF1DTB1H	INTPND2H
24H	PF	PTM2DL	IF1DTB2L	▒
25H	PFC	PTM2DH	IF1DTB2H	▒
26H	PFPU	PTM2AL	▨	▒
27H	PG	PTM2AH	▨	▒
28H	PGC	PTM2RPL	▨	▒
29H	PGPU	PTM2RPH	▨	▒
2AH	PH	PTM3C0	▨	▒
2BH	PHC	PTM3C1	▨	▒
2CH	PHPU	PTM3DL	▨	▒
2DH	PMPS	PTM3DH	▨	▒
2EH	RSTC	PTM3AL	▨	▒
2FH	VBGRC	PTM3AH	▨	▒
30H	MFI0	PTM3RPL	▨	MSGVAL1L
31H	MFI1	PTM3RPH	▨	MSGVAL1H
32H	MFI2	STM1C0	▨	MSGVAL2L
33H	MFI3	STM1C1	▒	MSGVAL2H
34H	MFI4	STM1DL	▨	▒
35H	MFI5	STM1DH	▨	▒
36H	MFI6	STM1AL	▨	▒
37H	MFI7	STM1AH	▨	▒
38H	▒	STM1RP	CRLL	▒
39H	INTEG	STM2C0	CRLH	▒
3AH	SCC	STM2C1	CRHL	▒
3BH	HIRCC	STM2DL	CRHH	▒
3CH	HXTC	STM2DH	▒	▒
3DH	LXTC	STM2AL	▒	▒
3EH	WDTC	STM2AH	▒	▒
3FH	LVRC	STM2RP	▒	CANCFG

	Sector 0	Sector 1	Sector 2
40H	LVDC	EEC	IF2CREQL
41H	EEAL	▒	IF2CREQH
42H	EEAH	▒	IF2CMSKL
43H	EED	FC0	▒
44H	CP0C	FC1	IF2MSK1L
45H	CP1C	FC2	IF2MSK1H
46H	PTM0C0	▒	IF2MSK2L
47H	PTM0C1	▒	IF2MSK2H
48H	PTM0DL	IFS0	IF2ARB1L
49H	PTM0DH	IFS1	IF2ARB1H
4AH	PTM0AL	IFS2	IF2ARB2L
4BH	PTM0AH	IFS3	IF2ARB2H
4CH	PTM0RPL	▒	IF2MCTRL
4DH	PTM0RPH	PAS0	IF2MCTRH
4EH	STM0C0	PAS1	IF2DTA1L
4FH	STM0C1	PBS0	IF2DTA1H
50H	STM0DL	PBS1	IF2DTA2L
51H	STM0DH	PCS0	IF2DTA2H
52H	STM0AL	PCS1	IF2DTB1L
53H	STM0AH	PDS0	IF2DTB1H
54H	STM0RP	PDS1	IF2DTB2L
55H	SLEDC0	PES0	IF2DTB2H
56H	SLEDC1	PES1	▒
57H	SLEDC2	PFS0	▒
58H	SLEDC3	PFS1	▒
59H	MDUWR0	PGS0	▒
5AH	MDUWR1	PGS1	▒
5BH	MDUWR2	PHS0	▒
5CH	MDUWR3	PHS1	▒
5DH	MDUWR4	▒	▒
5EH	MDUWR5	▒	▒
5FH	MDUWCTRL	▒	▒
60H	CP0VOS	▒	▒
61H	CP1VOS	▒	▒
62H	▒	▒	▒
63H	PSC0R	▒	▒
64H	TB0C	▒	▒
65H	TB1C	▒	▒
66H	PSC1R	▒	▒
67H	SADOL	▒	▒
68H	SADOH	▒	▒
69H	SADC0	▒	▒
6AH	SADC1	▒	▒
6BH	SADC2	▒	▒
6CH	SIMC0	▒	▒
6DH	SIMC1	▒	▒
6EH	SIMD	▒	▒
6FH	SIMA/SIMC2	▒	▒
70H	SIMTOC	▒	▒
71H	SPIAC0	▒	▒
72H	SPIAC1	▒	▒
73H	SPIAD	▒	▒
74H	FARL	▒	▒
75H	FARH	▒	▒
76H	FD0L	▒	▒
77H	FD0H	▒	▒
78H	FD1L	▒	▒
79H	FD1H	▒	▒
7AH	FD2L	▒	▒
7BH	FD2H	▒	▒
7CH	FD3L	▒	▒
7DH	FD3H	▒	▒
7EH	▒	▒	▒
7FH	SCOMC	▒	▒

▒ : Unused, read as 00H

▨ : Reserved, cannot be changed

Special Purpose Data Memory

Special Function Register Description

Most of the Special Function Register details will be described in the relevant functional sections, however several registers require a separate description in this section.

Indirect Addressing Registers – IAR0, IAR1, IAR2

The Indirect Addressing Registers, IAR0, IAR1 and IAR2, although having their locations in normal RAM register space, do not actually physically exist as normal registers. The method of indirect addressing for RAM data manipulation uses these Indirect Addressing Registers and Memory Pointers, in contrast to direct memory addressing, where the actual memory address is specified. Actions on the IAR0, IAR1 and IAR2 registers will result in no actual read or write operation to these registers but rather to the memory location specified by their corresponding Memory Pointers, MP0, MP1L/MP1H or MP2L/MP2H. Acting as a pair, IAR0 and MP0 can together access data only from Sector 0 while the IAR1 register together with the MP1L/MP1H register pair and IAR2 register together with the MP2L/MP2H register pair can access data from any Data Memory Sector. As the Indirect Addressing Registers are not physically implemented, reading the Indirect Addressing Registers will return a result of “00H” and writing to the registers will result in no operation.

Memory Pointers – MP0, MP1L, MP1H, MP2L, MP2H

Five Memory Pointers, known as MP0, MP1L, MP1H, MP2L, MP2H, are provided. These Memory Pointers are physically implemented in the Data Memory and can be manipulated in the same way as normal registers providing a convenient way with which to address and track data. When any operation to the relevant Indirect Addressing Registers is carried out, the actual address that the microcontroller is directed to is the address specified by the related Memory Pointer. MP0, together with Indirect Addressing Register, IAR0, are used to access data from Sector 0, while MP1L/MP1H together with IAR1 and MP2L/MP2H together with IAR2 are used to access data from all sectors according to the corresponding MP1H or MP2H register. Direct Addressing can be used in all sectors using the corresponding instruction which can address all available data memory space.

The following example shows how to clear a section of four Data Memory locations already defined as locations adres1 to adres4.

Indirect Addressing Program Example 1

```
data .section ´data´
adres1   db ?
adres2   db ?
adres3   db ?
adres4   db ?
block    db ?
code .section at 0 ´code´
org 00h
start:
     mov a, 04h               ; setup size of block
     mov block, a
     mov a, offset adres1     ; Accumulator loaded with first RAM address
     mov mp0, a               ; setup memory pointer with first RAM address
loop:
     clr IAR0                 ; clear the data at address defined by MP0
     inc mp0                  ; increment memory pointer
     sdz block                ; check if last memory location has been cleared
     jmp loop
continue:
```

Indirect Addressing Program Example 2

```
data .section ´data´
adres1   db ?
adres2   db ?
adres3   db ?
adres4   db ?
block    db ?
code .section at 0 ´code´
org 00h
start:
     mov a, 04h              ; setup size of block
     mov block, a
     mov a, 01h              ; setup the memory sector
     mov mp1h, a
     mov a, offset adres1    ; Accumulator loaded with first RAM address
     mov mp1l, a             ; setup memory pointer with first RAM address
loop:
     clr IAR1                ; clear the data at address defined by MP1L
     inc mp1l                ; increment memory pointer MP1L
     sdz block               ; check if last memory location has been cleared
     jmp loop
continue:
```

The important point to note here is that in the example shown above, no reference is made to specific Data Memory addresses.

Direct Addressing Program Example using extended instructions

```
data .section ´data´
temp  db ?
code .section at 0 ´code´
org 00h
start:
     lmov a, [m]          ; move [m] data to acc
     lsub a, [m+1]        ; compare [m] and [m+1] data
     snz  c               ; [m]>[m+1]?
     jmp  continue        ; no
     lmov a, [m]          ; yes, exchange [m] and [m+1] data
     mov  temp, a
     lmov a, [m+1]
     lmov [m], a
     mov  a, temp
     lmov [m+1], a
continue:
```

Note: Here “m” is a data memory address located in any data memory sectors. For example, m=1F0H, it indicates address 0F0H in Sector 1.

Program Memory Bank Pointer – PBP

For the device the Program Memory is divided into several banks. Selecting the required Program Memory area is achieved using the Program Memory Bank Pointer, PBP. The PBP register should be properly configured before the device executes the "Branch" operation using the "JMP" or "CALL" instruction. After that a jump to a non-consecutive Program Memory address which is located in a certain bank selected by the program memory bank pointer bits will occur.

- **PBP Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	PBP1	PBP0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~2 **D7~D2**: General data bits and can be read or written

Bit 1~0 **PBP1~PBP0**: Program Memory Bank Point bit 1 ~ bit 0
00: Bank 0
01: Bank 1
10: Bank 2
11: Bank 3

Accumulator – ACC

The Accumulator is central to the operation of any microcontroller and is closely related with operations carried out by the ALU. The Accumulator is the place where all intermediate results from the ALU are stored. Without the Accumulator it would be necessary to write the result of each calculation or logical operation such as addition, subtraction, shift, etc., to the Data Memory resulting in higher programming and timing overheads. Data transfer operations usually involve the temporary storage function of the Accumulator; for example, when transferring data between one user-defined register and another, it is necessary to do this by passing the data through the Accumulator as no direct transfer between two registers is permitted.

Program Counter Low Register – PCL

To provide additional program control functions, the low byte of the Program Counter is made accessible to programmers by locating it within the Special Purpose area of the Data Memory. By manipulating this register, direct jumps to other program locations are easily implemented. Loading a value directly into this PCL register will cause a jump to the specified Program Memory location, however, as the register is only 8-bit wide, only jumps within the current Program Memory page are permitted. When such operations are used, note that a dummy cycle will be inserted.

Look-up Table Registers – TBLP, TBHP, TBLH

These three special function registers are used to control operation of the look-up table which is stored in the Program Memory. TBLP and TBHP are the table pointers and indicate the location where the table data is located. Their value must be setup before any table read commands are executed. Their value can be changed, for example using the "INC" or "DEC" instructions, allowing for easy table data pointing and reading. TBLH is the location where the high order byte of the table data is stored after a table read data instruction has been executed. Note that the lower order table data byte is transferred to a user defined location.

Status Register – STATUS

This 8-bit register contains the SC flag, CZ flag, zero flag (Z), carry flag (C), auxiliary carry flag (AC), overflow flag (OV), power down flag (PDF), and watchdog time-out flag (TO). These arithmetic/logical operation and system management flags are used to record the status and operation of the microcontroller.

With the exception of the TO and PDF flags, bits in the status register can be altered by instructions like most other registers. Any data written into the status register will not change the TO or PDF flag. In addition, operations related to the status register may give different results due to the different instruction operations. The TO flag can be affected only by a system power-up, a WDT time-out or by executing the "CLR WDT" or "HALT" instruction. The PDF flag is affected only by executing the "HALT" or "CLR WDT" instruction or during a system power-up.

The Z, OV, AC, C, SC and CZ flags generally reflect the status of the latest operations.

- C is set if an operation results in a carry during an addition operation or if a borrow does not take place during a subtraction operation; otherwise C is cleared. C is also affected by a rotate through carry instruction.
- AC is set if an operation results in a carry out of the low nibbles in addition, or no borrow from the high nibble into the low nibble in subtraction; otherwise AC is cleared.
- Z is set if the result of an arithmetic or logical operation is zero; otherwise Z is cleared.
- OV is set if an operation results in a carry into the highest-order bit but not a carry out of the highest-order bit, or vice versa; otherwise OV is cleared.
- PDF is cleared by a system power-up or executing the "CLR WDT" instruction. PDF is set by executing the "HALT" instruction.
- TO is cleared by a system power-up or executing the "CLR WDT" or "HALT" instruction. TO is set by a WDT time-out.
- CZ is the operational result of different flags for different instructions. Refer to register definitions for more details.
- SC is the result of the "XOR" operation which is performed by the OV flag and the MSB of the current instruction operation result.

In addition, on entering an interrupt sequence or executing a subroutine call, the status register will not be pushed onto the stack automatically. If the contents of the status registers are important and if the subroutine can corrupt the status register, precautions must be taken to correctly save it.

- **STATUS Register**

Bit	7	6	5	4	3	2	1	0
Name	SC	CZ	TO	PDF	OV	Z	AC	C
R/W	R/W	R/W	R	R	R/W	R/W	R/W	R/W
POR	x	x	0	0	x	x	x	x

"x": unknown

Bit 7 **SC**: The result of the "XOR" operation which is performed by the OV flag and the MSB of the instruction operation result.

Bit 6 **CZ**: The operational result of different flags for different instructions

For SUB/SUBM/LSUB/LSUBM instructions, the CZ flag is equal to the Z flag.

For SBC/SBCM/LSBC/LSBCM instructions, the CZ flag is the "AND" operation result which is performed by the previous operation CZ flag and current operation zero flag.

For other instructions, the CZ flag will not be affected.

Bit 5 **TO**: Watchdog Time-Out Flag
0: After power up or executing the "CLR WDT" or "HALT" instruction
1: A watchdog time-out occurred.

Bit 4 **PDF**: Power Down Flag
0: After power up or executing the "CLR WDT" instruction
1: By executing the "HALT" instruction

Bit 3 **OV**: Overflow Flag
0: No overflow
1: An operation results in a carry into the highest-order bit but not a carry out of the highest-order bit or vice versa.

Bit 2 **Z**: Zero Flag
0: The result of an arithmetic or logical operation is not zero
1: The result of an arithmetic or logical operation is zero

Bit 1 **AC**: Auxiliary flag
0: No auxiliary carry
1: An operation results in a carry out of the low nibbles in addition, or no borrow from the high nibble into the low nibble in subtraction

Bit 0 **C**: Carry Flag
0: No carry-out
1: An operation results in a carry during an addition operation or if a borrow does not take place during a subtraction operation

The "C" flag is also affected by a rotate through carry instruction.

EEPROM Data Memory

This device contains an area of internal EEPROM Data Memory. EEPROM is by its nature a non-volatile form of re-programmable memory, with data retention even when its power supply is removed. By incorporating this kind of data memory, a whole new host of application possibilities are made available to the designer. The availability of EEPROM storage allows information such as product identification numbers, calibration values, specific user data, system setup data or other product information to be stored directly within the product microcontroller. The process of reading and writing data to the EEPROM memory has been reduced to a very trivial affair.

EEPROM Data Memory Structure

The EEPROM Data Memory capacity is 1024×8 bits for the device. Unlike the Program Memory and RAM Data Memory, the EEPROM Data Memory is not directly mapped into memory space and is therefore not directly addressable in the same way as the other types of memory. Read and Write operations to the EEPROM are carried out in single byte operations using two address and a data register in Sector 0 and a single control register in Sector 1.

EEPROM Registers

Four registers control the overall operation of the internal EEPROM Data Memory. These are the address registers, EEAL and EEAH, the data register, EED and a single control register, EEC. As the EEAL/EEAH and EED registers are located in Sector 0, they can be directly accessed in the same was as any other Special Function Register. The EEC register however, being located in Sector 1, can only be read from or written to indirectly using the MP1L/MP1H or MP2L/MP2H Memory Pointer and Indirect Addressing Register, IAR1/IAR2. Because the EEC control register is located at address 40H in Sector 1, the MP1L or MP2L Memory Pointer must first be set to the value 40H and the MP1H or MP2H Memory Pointer high byte set to the value, 01H, before any operations on the EEC register are executed.

Register Name	Bit							
	7	6	5	4	3	2	1	0
EEAL	EEAL7	EEAL6	EEAL5	EEAL4	EEAL3	EEAL2	EEAL1	EEAL0
EEAH	—	—	—	—	—	—	EEAH1	EEAH0
EED	EED7	EED6	EED5	EED4	EED3	EED2	EED1	EED0
EEC	—	—	—	—	EEWREN	EEWR	EERDEN	EERD

EEPROM Register List

- **EEAL Register**

Bit	7	6	5	4	3	2	1	0
Name	EEAL7	EEAL6	EEAL5	EEAL4	EEAL3	EEAL2	EEAL1	EEAL0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **EEAL7~EEAL0**: Data EEPROM low byte address bit 7 ~ bit 0

- **EEAH Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	EEAH1	EEAH0
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as “0”

Bit 1~0 **EEAH1~EEAH0**: Data EEPROM high byte address bit 1 ~ bit 0

- **EED Register**

Bit	7	6	5	4	3	2	1	0
Name	EED7	EED6	EED5	EED4	EED3	EED2	EED1	EED0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **EED7~EED0**: Data EEPROM Data bit 7 ~ bit 0

- **EEC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	EEWREN	EEWR	EERDEN	EERD
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	0	0	0

Bit 7~4 Unimplemented, read as "0"

Bit 3 **EEWREN**: Data EEPROM Write Enable
0: Disable
1: Enable

This is the Data EEPROM Write Enable Bit which must be set high before Data EEPROM write operations are carried out. Clearing this bit to zero will inhibit Data EEPROM write operations.

Bit 2 **EEWR**: EEPROM Write Control
0: Write cycle has finished
1: Activate a write cycle

This is the Data EEPROM Write Control Bit and when set high by the application program will activate a write cycle. This bit will be automatically reset to zero by the hardware after the write cycle has finished. Setting this bit high will have no effect if the EEWREN has not first been set high.

Bit 1 **EERDEN**: Data EEPROM Read Enable
0: Disable
1: Enable

This is the Data EEPROM Read Enable Bit which must be set high before Data EEPROM read operations are carried out. Clearing this bit to zero will inhibit Data EEPROM read operations.

Bit 0 **EERD**: EEPROM Read Control
0: Read cycle has finished
1: Activate a read cycle

This is the Data EEPROM Read Control Bit and when set high by the application program will activate a read cycle. This bit will be automatically reset to zero by the hardware after the read cycle has finished. Setting this bit high will have no effect if the EERDEN bit has not first been set high.

Note: 1. The EEWREN, EEWR, EERDEN and EERD cannot be set high at the same time in one instruction. The EEWR and EERD cannot be set high at the same time.

2. Ensure that the f_{SUB} clock is stable before executing the write operation.

3. Ensure that the write operation is totally complete before changing the EEC register content.

Reading Data from the EEPROM

To read data from the EEPROM, the EEPROM address of the data to be read must first be placed in the EEAL and EEAH registers. Then the read enable bit, EERDEN, in the EEC register must be set high to enable the read function. If the EERD bit in the EEC register is now set high, a read cycle will be initiated. Setting the EERD bit high will not initiate a read operation if the EERDEN bit has not been set. When the read cycle terminates, the EERD bit will be automatically cleared to zero, after which the data can be read from the EED register. The data will remain in the EED register until another read or write operation is executed. The application program can poll the EERD bit to determine when the data is valid for reading.

Writing Data to the EEPROM

The EEPROM address of the data to be written must first be placed in the EEAL and EEAH registers and the data placed in the EED register. To write data to the EEPROM, the write enable bit, EEWREN, in the EEC register must first be set high to enable the write function. After this, the EEWR bit in the EEC register must be immediately set high to initiate a write cycle. These two instructions must be executed in two consecutive instruction cycles. The global interrupt bit EMI should also first be cleared before implementing any write operations, and then set again after the write cycle has started. Note that setting the EEWR bit high will not initiate a write cycle if the EEWREN bit has not been set. As the EEPROM write cycle is controlled using an internal timer whose operation is asynchronous to microcontroller system clock, a certain time will elapse before the data will have been written into the EEPROM. Detecting when the write cycle has finished can be implemented either by polling the EEWR bit in the EEC register or by using the EEPROM interrupt. When the write cycle terminates, the EEWR bit will be automatically cleared to zero by the microcontroller, informing the user that the data has been written to the EEPROM. The application program can therefore poll the EEWR bit to determine when the write cycle has ended.

Write Protection

Protection against inadvertent write operation is provided in several ways. After the device is powered-on the Write Enable bit in the control register will be cleared preventing any write operations. Also at power-on the Memory Pointer high byte register, MP1H or MP2H, will be reset to zero, which means that Data Memory Sector 0 will be selected. As the EEPROM control register is located in Sector 1, this adds a further measure of protection against spurious write operations. During normal program operation, ensuring that the Write Enable bit in the control register is cleared will safeguard against incorrect write operations.

EEPROM Interrupt

The EEPROM write interrupt is generated when an EEPROM write cycle has ended. The EEPROM interrupt must first be enabled by setting the DEE bit in the relevant interrupt register. However as the EEPROM is contained within a Multi-function Interrupt, the associated multi-function interrupt enable bit must also be set. When an EEPROM write cycle ends, the DEF request flag and its associated multi-function interrupt request flag will both be set. If the global, EEPROM and Multi-function interrupts are enabled and the stack is not full, a jump to the associated Multi-function Interrupt vector will take place. When the interrupt is serviced only the Multi-function interrupt flag will be automatically reset, the EEPROM interrupt flag must be manually reset by the application program. More details can be obtained in the Interrupt section.

Programming Considerations

Care must be taken that data is not inadvertently written to the EEPROM. Protection can be enhanced by ensuring that the Write Enable bit is normally cleared to zero when not writing. Also the Memory Pointer high byte register, MP1H or MP2H, could be normally cleared to zero as this would inhibit access to Sector 1 where the EEPROM control register exist. Although certainly not necessary, consideration might be given in the application program to the checking of the validity of new write data by a simple read back process.

When writing data the EEWR bit must be set high immediately after the EEWREN bit has been set high, to ensure the write cycle executes correctly. The global interrupt bit EMI should also be cleared before a write cycle is executed and then re-enabled after the write cycle starts. Note that the device should not enter the IDLE or SLEEP mode until the EEPROM read or write operation is totally complete. Otherwise, the EEPROM read or write operation will fail.

Programming Examples

Reading data from the EEPROM – polling method

```
MOV  A, EEPROM_ADRES      ; user defined address
MOV  EEA, A
MOV  A, 40H               ; setup memory pointer MP1L
MOV  MP1L, A              ; MP1L points to EEC register
MOV  A, 01H               ; setup memory pointer MP1H
MOV  MP1H, A
SET  IAR1.1               ; set EERDEN bit, enable read operations
SET  IAR1.0               ; start Read Cycle - set EERD bit
BACK:
SZ   IAR1.0               ; check for read cycle end
JMP  BACK
CLR  IAR1                 ; disable EEPROM read if no more read operations are required
CLR  MP1H
MOV  A, EED               ; move read data to register
MOV  READ_DATA, A
```

Note: For each read operation, the address register should be re-specified followed by setting the EERD bit high to activate a read cycle even if the target address is consecutive.

Writing Data to the EEPROM – polling method

```
MOV  A, EEPROM_ADRES      ; user defined address
MOV  EEA, A
MOV  A, EEPROM_DATA       ; user defined data
MOV  EED, A
MOV  A, 040H              ; setup memory pointer MP1L
MOV  MP1L, A              ; MP1L points to EEC register
MOV  A, 01H               ; setup memory pointer MP1H
MOV  MP1H, A
CLR  EMI
SET  IAR1.3               ; set EEWREN bit, enable write operations
SET  IAR1.2               ; start Write Cycle - set EEWR bit – executed immediately
                          ; after set EEWREN bit
SET  EMI
BACK:
SZ   IAR1.2               ; check for write cycle end
JMP  BACK
CLR  MP1H
```

Oscillators

Various oscillator options offer the user a wide range of functions according to their various application requirements. The flexible features of the oscillator functions ensure that the best optimisation can be achieved in terms of speed and power saving. Oscillator selections and operation are selected through relevant control registers.

Oscillator Overview

In addition to being the source of the main system clock the oscillators also provide clock sources for the Watchdog Timer and Time Base Interrupts. External oscillators requiring some external components as well as fully integrated internal oscillators, requiring no external components, are provided to form a wide range of both fast and slow system oscillators. All oscillator options are selected through the registers. The higher frequency oscillators provide higher performance but carry with it the disadvantage of higher power requirements, while the opposite is of course true for the lower frequency oscillators. With the capability of dynamically switching between fast and slow system clock, the device has the flexibility to optimize the performance/power ratio, a feature especially important in power sensitive portable applications.

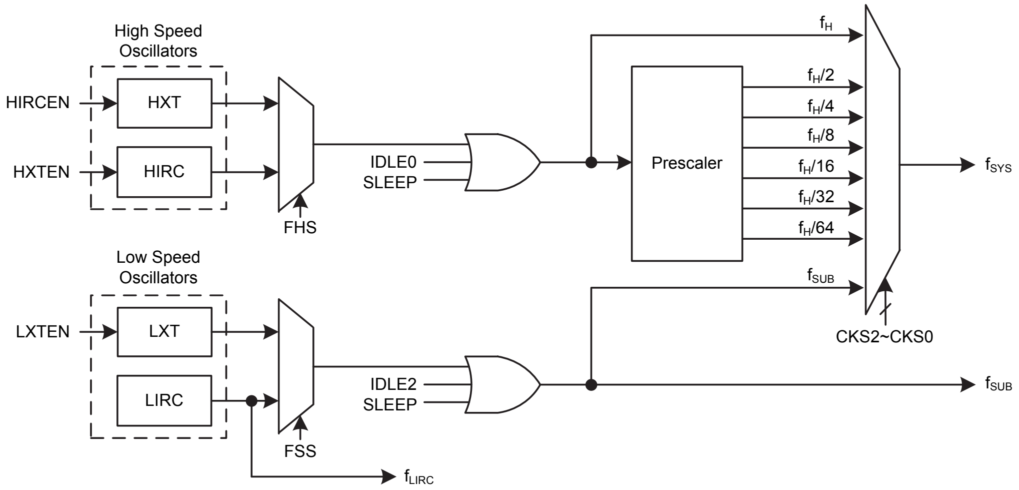
Type	Name	Freq.	Pins
External High Speed Crystal	HXT	400kHz~16MHz	OSC1/OSC2
Internal High Speed RC	HIRC	8/12/16MHz	—
External Low Speed Crystal	LXT	32.768kHz	XT1/XT2
Internal Low Speed RC	LIRC	32kHz	—

Oscillator Types

System Clock Configurations

There are four methods of generating the system clock, two high speed oscillators and two low speed oscillators. The two high speed oscillators are the external crystal/ceramic oscillator, HXT, and the internal 8/12/16MHz RC oscillator, HIRC. The two low speed oscillators are the internal 32kHz RC oscillator, LIRC, and the external 32.768kHz crystal oscillator, LXT. Selecting whether the low or high speed oscillator is used as the system oscillator is implemented using the CKS2 ~ CKS0 bits in the SCC register and the system clock can be dynamically selected.

The actual source clock used for the low speed oscillators is chosen via the FSS bit in the SCC register while for the high speed oscillator the source clock is selected by the FHS bit in the SCC register. The frequency of the slow speed or high speed system clock is determined using the CKS2~CKS0 bits in the SCC register. Note that two oscillator selections must be made namely one high speed and one low speed system oscillators. It is not possible to choose a no-oscillator selection for either the high or low speed oscillator.

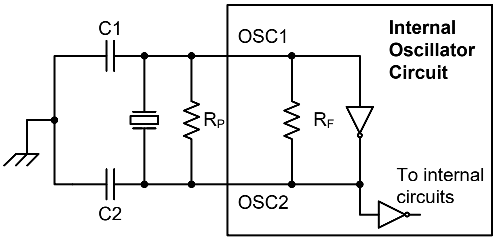


System Clock Configurations

External Crystal/Ceramic Oscillator – HXT

The External Crystal/Ceramic System Oscillator is the high frequency oscillator. For most crystal oscillator configurations, the simple connection of a crystal across OSC1 and OSC2 will create the necessary phase shift and feedback for oscillation, without requiring external capacitors. However, for some crystal types and frequencies, to ensure oscillation, it may be necessary to add two small value capacitors, C1 and C2. Using a ceramic resonator will usually require two small value capacitors, C1 and C2, to be connected as shown for oscillation to occur. The values of C1 and C2 should be selected in consultation with the crystal or resonator manufacturer's specification.

For oscillator stability and to minimise the effects of noise and crosstalk, it is important to ensure that the crystal and any associated resistors and capacitors along with interconnecting lines are all located as close to the MCU as possible.



Note: 1. R_P is normally not required. C1 and C2 are required.
2. Although not shown OSC1/OSC2 pins have a parasitic capacitance of around 7pF.

Crystal/Resonator Oscillator

HXT Oscillator C1 and C2 Values		
Crystal Frequency	C1	C2
16MHz	0pF	0pF
12MHz	0pF	0pF
8MHz	0pF	0pF
4MHz	0pF	0pF
1MHz	100pF	100pF
Note: C1 and C2 values are for guidance only.		

Crystal Recommended Capacitor Values

Internal RC Oscillator – HIRC

The internal RC oscillator is a fully integrated system oscillator requiring no external components. The internal RC oscillator has a fixed frequency of 8/12/16MHz. Device trimming during the manufacturing process and the inclusion of internal frequency compensation circuits are used to ensure that the influence of the power supply voltage, temperature and process variations on the oscillation frequency are minimised. As a result, at a power supply of 3V or 5V and at a temperature of 25°C degrees, the selected trimmed oscillation frequency will have a tolerance within 1%. Note that if this internal system clock is selected, it requires no external pins for its operation.

External 32.768kHz Crystal Oscillator – LXT

The External 32.768kHz Crystal System Oscillator is one of the low frequency oscillator choices, which is selected via a software control bit, FSS. This clock source has a fixed frequency of 32.768kHz and requires a 32.768kHz crystal to be connected between pins XT1 and XT2. The external resistor and capacitor components connected to the 32.768kHz crystal are necessary to provide oscillation. For applications where precise frequencies are essential, these components may be required to provide frequency compensation due to different crystal manufacturing tolerances. After the LXT oscillator is enabled by setting the LXTEN bit to 1, there is a time delay associated with the LXT oscillator waiting for it to start-up.

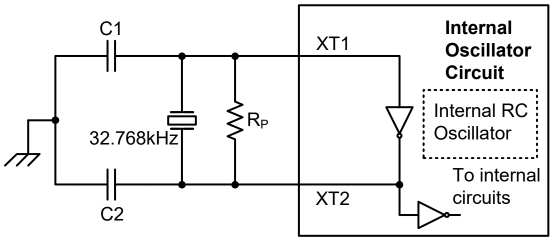
When the microcontroller enters the SLEEP or IDLE Mode, the system clock is switched off to stop microcontroller activity and to conserve power. However, in many microcontroller applications it may be necessary to keep the internal timers operational even when the microcontroller is in the SLEEP or IDLE Mode. To do this, another clock, independent of the system clock, must be provided.

However, for some crystals, to ensure oscillation and accurate frequency generation, it is necessary to add two small value external capacitors, C1 and C2. The exact values of C1 and C2 should be selected in consultation with the crystal or resonator manufacturer's specification. The external parallel feedback resistor, Rp, is required.

The pin-shared software control bits determine if the XT1/XT2 pins are used for the LXT oscillator or as I/O or other pin-shared functional pins.

- If the LXT oscillator is not used for any clock source, the XT1/XT2 pins can be used as normal I/O or other pin-shared functional pins.
- If the LXT oscillator is used for any clock source, the 32.768kHz crystal should be connected to the XT1/XT2 pins.

For oscillator stability and to minimise the effects of noise and crosstalk, it is important to ensure that the crystal and any associated resistors and capacitors along with interconnecting lines are all located as close to the MCU as possible.



Note: 1. R_P, C1 and C2 are required.
2. Although not shown pins have a parasitic capacitance of around 7pF.

External LXT Oscillator

Internal 32kHz Oscillator – LIRC

The Internal 32kHz System Oscillator is one of the low frequency oscillator choices, which is selected via a software control bit, FSS. It is a fully integrated RC oscillator with a typical frequency of 32kHz at full voltage range, requiring no external components for its implementation. Device trimming during the manufacturing process and the inclusion of internal frequency compensation circuits are used to ensure that the influence of the power supply voltage, temperature and process variations on the oscillation frequency are minimised.

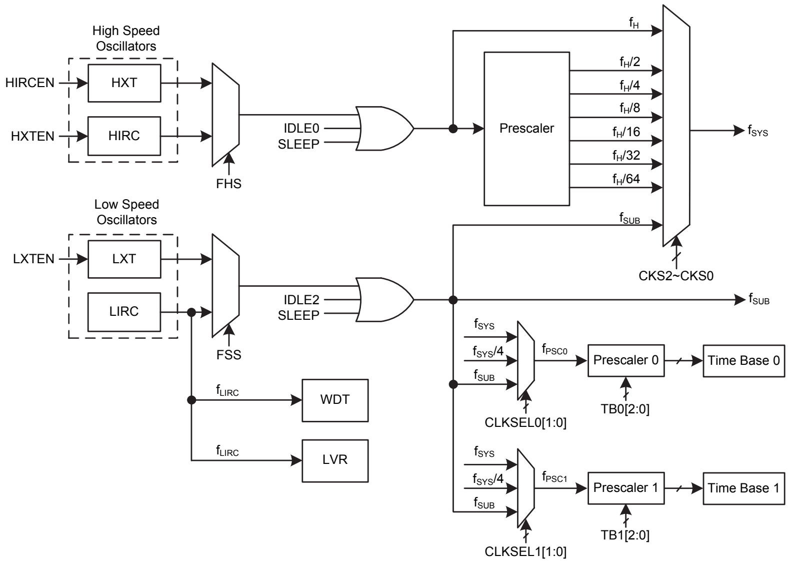
Operating Modes and System Clocks

Present day applications require that their microcontrollers have high performance but often still demand that they consume as little power as possible, conflicting requirements that are especially true in battery powered portable applications. The fast clocks required for high performance will by their nature increase current consumption and of course vice versa, lower speed clocks reduce current consumption. As both high and low speed clock sources are provided the means to switch between them dynamically, the user can optimise the operation of their microcontroller to achieve the best performance/power ratio.

System Clocks

The device has different clock sources for both the CPU and peripheral function operation. By providing the user with a wide range of clock selections using register programming, a clock system can be configured to obtain maximum application performance.

The main system clock, can come from either a high frequency, f_H, or low frequency, f_{SUB}, source, and is selected using the CKS2~CKS0 bits in the SCC register. The high speed system clock is sourced from an HXT or HIRC oscillator, selected via configuring the FHS bit in the SCC register. The low speed system clock source can be sourced from the internal clock f_{SUB}. If f_{SUB} is selected then it can be sourced from the LXT or LIRC oscillator. The other choice, which is a divided version of the high speed system oscillator has a range of $f_H/2$~$f_H/64$.



Device Clock Configurations

Note: When the system clock source f_{SYS} is switched to f_{SUB} from f_H, the high speed oscillator can be stopped to conserve the power or continue to oscillate to provide the clock source, f_H~$f_H/64$, for peripheral circuit to use, which is determined by configuring the corresponding high speed oscillator enable control bit.

System Operation Modes

There are six different modes of operation for the microcontroller, each one with its own special characteristics and which can be chosen according to the specific performance and power requirements of the application. There are two modes allowing normal operation of the microcontroller, the FAST Mode and SLOW Mode. The remaining four modes, the SLEEP, IDLE0, IDLE1 and IDLE2 Mode are used when the microcontroller CPU is switched off to conserve power.

Operation Mode	CPU	Register Setting			f_{SYS}	f_H	f_{SUB}	f_{LIRC}
		FHIDEN	FSIDEN	CKS2~CKS0				
FAST	On	x	x	000~110	f_H~f_H/64	On	On	On
SLOW	On	x	x	111	f_{SUB}	On/Off [(1)]	On	On
IDLE0	Off	0	1	000~110	Off	Off	On	On
				111	On			
IDLE1	Off	1	1	xxx	On	On	On	On
IDLE2	Off	1	0	000~110	On	On	Off	On
				111	Off			
SLEEP	Off	0	0	xxx	Off	Off	Off	On/Off [(2)]

"x": don't care

Note: 1. The f_H clock will be switched on or off by configuring the corresponding oscillator enable bit in the SLOW mode.

2. The f_{LIRC} clock can be switched on or off which is controlled by the WDT function being enabled or disabled in the SLEEP mode.

FAST Mode

This is one of the main operating modes where the microcontroller has all of its functions operational and where the system clock is provided by one of the high speed oscillators. This mode operates allowing the microcontroller to operate normally with a clock source which will come from the HXT or HIRC oscillator. The high speed oscillator will however first be divided by a ratio ranging from 1 to 64, the actual ratio being selected by the CKS2~CKS0 bits in the SCC register. Although a high speed oscillator is used, running the microcontroller at a divided clock ratio reduces the operating current.

SLOW Mode

This is also a mode where the microcontroller operates normally although now with a slower speed clock source. The clock source used will be from f_{SUB}. The f_{SUB} clock is derived from the LXT or LIRC oscillator.

SLEEP Mode

The SLEEP Mode is entered when a HALT instruction is executed and when the FHIDEN and FSIDEN bit are low. In the SLEEP mode the CPU will be stopped. However the f_{LIRC} clock can still continue to operate if the WDT function is enabled by the WDTC register.

IDLE0 Mode

The IDLE0 Mode is entered when a HALT instruction is executed and when the FHIDEN bit in the SCC register is low and the FSIDEN bit in the SCC register is high. In the IDLE0 Mode the CPU will be switched off but the low speed oscillator will be turned on to drive some peripheral functions.

IDLE1 Mode

The IDLE1 Mode is entered when a HALT instruction is executed and when the FHIDEN bit in the SCC register is high and the FSIDEN bit in the SCC register is high. In the IDLE1 Mode the CPU will be switched off but both the high and low speed oscillators will be turned on to provide a clock source to keep some peripheral functions operational.

IDLE2 Mode

The IDLE2 Mode is entered when a HALT instruction is executed and when the FHIDEN bit in the SCC register is high and the FSIDEN bit in the SCC register is low. In the IDLE2 Mode the CPU will be switched off but the high speed oscillator will be turned on to provide a clock source to keep some peripheral functions operational.

Control Registers

The registers, SCC, HIRCC, HXTC and LXTC, are used to control the system clock and the corresponding oscillator configurations.

Register Name	Bit							
	7	6	5	4	3	2	1	0
SCC	CKS2	CKS1	CKS0	—	FHS	FSS	FHIDEN	FSIDEN
HIRCC	—	—	—	—	HIRC1	HIRC0	HIRCF	HIRCEN
HXTC	—	—	—	—	—	HXTM	HXTF	HXTEN
LXTC	—	—	—	—	—	—	LXTF	LXTEN

System Operating Mode Control Registers List

- **SCC Register**

Bit	7	6	5	4	3	2	1	0
Name	CKS2	CKS1	CKS0	—	FHS	FSS	FHIDEN	FSIDEN
R/W	R/W	R/W	R/W	—	R/W	R/W	R/W	R/W
POR	0	0	0	—	0	0	0	0

Bit 7~5 **CKS2~CKS0**: System clock selection
000: f_H
001: $f_H/2$
010: $f_H/4$
011: $f_H/8$
100: $f_H/16$
101: $f_H/32$
110: $f_H/64$
111: f_{SUB}

These three bits are used to select which clock is used as the system clock source. In addition to the system clock source directly derived from f_H or f_{SUB}, a divided version of the high speed system oscillator can also be chosen as the system clock source.

Bit 4 Unimplemented, read as “0”

Bit 3 **FHS**: High Frequency clock selection
0: HIRC
1: HXT

Bit 2 **FSS**: Low Frequency clock selection
0: LIRC
1: LXT

Bit 1 **FHIDEN**: High Frequency oscillator control when CPU is switched off
0: Disable
1: Enable

This bit is used to control whether the high speed oscillator is activated or stopped when the CPU is switched off by executing an “HALT” instruction.

Bit 0 **FSIDEN**: Low Frequency oscillator control when CPU is switched off
0: Disable
1: Enable

This bit is used to control whether the low speed oscillator is activated or stopped when the CPU is switched off by executing an “HALT” instruction.

- **HIRCC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	HIRC1	HIRC0	HIRCF	HIRCEN
R/W	—	—	—	—	R/W	R/W	R	R/W
POR	—	—	—	—	0	0	0	1

Bit 7~4 Unimplemented, read as "0"

Bit 3~2 **HIRC1~HIRC0**: HIRC Frequency selection
00: 8MHz
01: 12MHz
10: 16MHz
11: 8MHz

When the HIRC oscillator is enabled or the HIRC frequency selection is changed by application program, the clock frequency will automatically be changed after the HIRCF flag is set to 1.

Bit 1 **HIRCF**: HIRC oscillator stable flag
0: HIRC unstable
1: HIRC stable

This bit is used to indicate whether the HIRC oscillator is stable or not. When the HIRCEN bit is set to 1 to enable the HIRC oscillator or the HIRC frequency selection is changed by application program, the HIRCF bit will first be cleared to 0 and then set to 1 after the HIRC oscillator is stable.

Bit 0 **HIRCEN**: HIRC oscillator enable control
0: Disable
1: Enable

- **HXTC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	HXTM	HXTF	HXTEN
R/W	—	—	—	—	—	R/W	R	R/W
POR	—	—	—	—	—	0	0	0

Bit 7~3 Unimplemented, read as "0"

Bit 2 **HXTM**: HXT mode selection
0: HXT frequency ≤ 10 MHz
1: HXT frequency >10 MHz

This bit is used to select the HXT oscillator operating mode. Note that this bit must be properly configured before the HXT is enabled. When the OSC1 and OSC2 pins are enabled and the HXTEN bit is set to 1 to enable the HXT oscillator, it is invalid to change the value of this bit. Otherwise, this bit value can be changed with no operation on the HXT function.

Bit 1 **HXTF**: HXT oscillator stable flag
0: HXT unstable
1: HXT stable

This bit is used to indicate whether the HXT oscillator is stable or not. When the HXTEN bit is set to 1 to enable the HXT oscillator, the HXTF bit will first be cleared to 0 and then set to 1 after the HXT oscillator is stable.

Bit 0 **HXTEN**: HXT oscillator enable control
0: Disable
1: Enable

- **LXTC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	LXTF	LXTEN
R/W	—	—	—	—	—	—	R	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as “0”

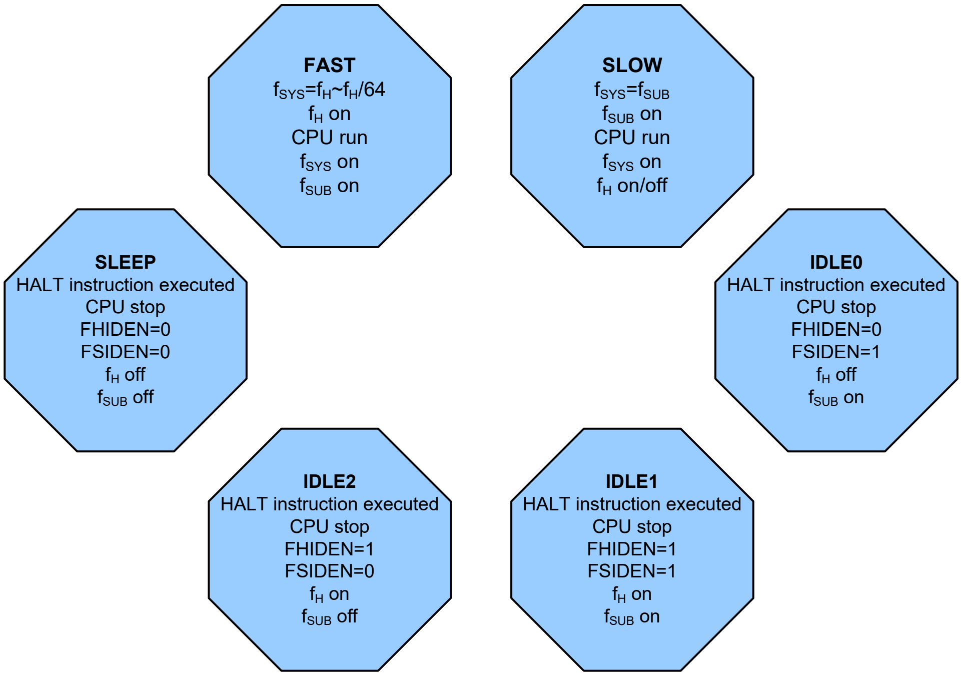
Bit 1 **LXTF**: LXT oscillator stable flag
0: LXT unstable
1: LXT stable
This bit is used to indicate whether the LXT oscillator is stable or not. When the LXTEN bit is set to 1 to enable the LXT oscillator, the LXTF bit will first be cleared to 0 and then set to 1 after the LXT oscillator is stable.

Bit 0 **LXTEN**: LXT oscillator enable control
0: Disable
1: Enable

Operating Mode Switching

The device can switch between operating modes dynamically allowing the user to select the best performance/power ratio for the present task in hand. In this way microcontroller operations that do not require high performance can be executed using slower clocks thus requiring less operating current and prolonging battery life in portable applications.

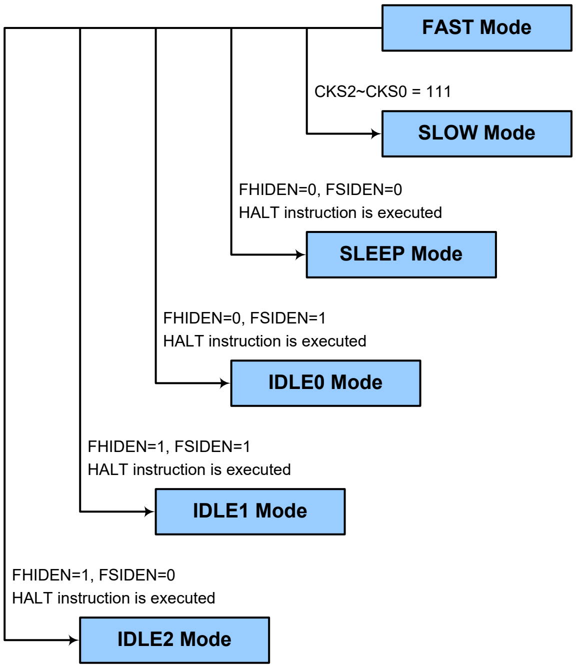
In simple terms, Mode Switching between the FAST Mode and SLOW Mode is executed using the CKS2~CKS0 bits in the SCC register while Mode Switching from the FAST/SLOW Modes to the SLEEP/IDLE Modes is executed via the HALT instruction. When a HALT instruction is executed, whether the device enters the IDLE Mode or the SLEEP Mode is determined by the condition of the FHIDEN and FSIDEN bits in the SCC register.



FAST Mode to SLOW Mode Switching

When running in the FAST Mode, which uses the high speed system oscillator, and therefore consumes more power, the system clock can switch to run in the SLOW Mode by setting the CKS2~CKS0 bits to “111” in the SCC register. This will then use the low speed system oscillator which will consume less power. Users may decide to do this for certain operations which do not require high performance and can subsequently reduce power consumption.

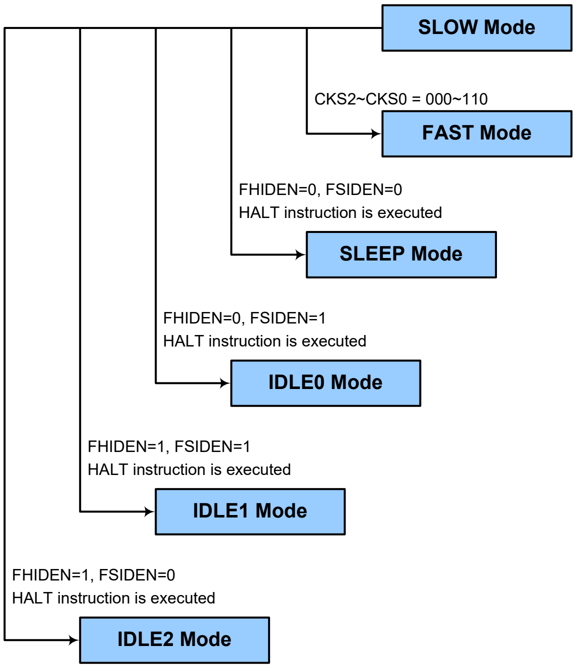
The SLOW Mode is sourced from the LXT or LIRC oscillator and therefore requires the selected oscillator to be stable before full mode switching occurs.



SLOW Mode to FAST Mode Switching

In SLOW mode the system clock is derived from f_{SUB}. When system clock is switched back to the FAST mode from f_{SUB}, the CKS2~CKS0 bits should be set to "000"~"110" and then the system clock will respectively be switched to f_H~f_H/64.

However, if f_H is not used in SLOW mode and thus switched off, it will take some time to re-oscillate and stabilise when switching to the FAST mode from the SLOW Mode. This is monitored using the HXTF bit in the HXTC register or the HIRCF bit in the HIRCC register. The time duration required for the high speed system oscillator stabilization is specified in the relevant characteristics.



Entering the SLEEP Mode

There is only one way for the device to enter the SLEEP Mode and that is to execute the "HALT" instruction in the application program with both the FHIDEN and FSIDEN bits in the SCC register equal to "0". In this mode all the clocks and functions will be switched off except the WDT function. When this instruction is executed under the conditions described above, the following will occur:

- The system clock will be stopped and the application program will stop at the "HALT" instruction.
- The Data Memory contents and registers will maintain their present condition.
- The I/O ports will maintain their present conditions.
- In the status register, the Power Down flag PDF will be set, and WDT timeout flag TO will be cleared.
- The WDT will be cleared and resume counting if the WDT function is enabled by the WDTC register.

Entering the IDLE0 Mode

There is only one way for the device to enter the IDLE0 Mode and that is to execute the “HALT” instruction in the application program with the FHIDEN bit in the SCC register equal to “0” and the FSIDEN bit in the SCC register equal to “1”. When this instruction is executed under the conditions described above, the following will occur:

- The f_H clock will be stopped and the application program will stop at the “HALT” instruction, but the f_{SUB} clock will be on.
- The Data Memory contents and registers will maintain their present condition.
- The I/O ports will maintain their present conditions.
- In the status register, the Power Down flag PDF will be set, and WDT timeout flag TO will be cleared.
- The WDT will be cleared and resume counting if the WDT function is enabled by the WDTC register.

Entering the IDLE1 Mode

There is only one way for the device to enter the IDLE1 Mode and that is to execute the “HALT” instruction in the application program with both the FHIDEN and FSIDEN bits in the SCC register equal to “1”. When this instruction is executed under the conditions described above, the following will occur:

- The f_H and f_{SUB} clocks will be on but the application program will stop at the “HALT” instruction.
- The Data Memory contents and registers will maintain their present condition.
- The I/O ports will maintain their present conditions.
- In the status register, the Power Down flag PDF will be set, and WDT timeout flag TO will be cleared.
- The WDT will be cleared and resume counting if the WDT function is enabled by the WDTC register.

Entering the IDLE2 Mode

There is only one way for the device to enter the IDLE2 Mode and that is to execute the “HALT” instruction in the application program with the FHIDEN bit in the SCC register equal to “1” and the FSIDEN bit in the SCC register equal to “0”. When this instruction is executed under the conditions described above, the following will occur:

- The f_H clock will be on but the f_{SUB} clock will be off and the application program will stop at the “HALT” instruction.
- The Data Memory contents and registers will maintain their present condition.
- The I/O ports will maintain their present conditions.
- In the status register, the Power Down flag PDF will be set, and WDT timeout flag TO will be cleared.
- The WDT will be cleared and resume counting if the WDT function is enabled by the WDTC register.

Standby Current Considerations

As the main reason for entering the SLEEP or IDLE Mode is to keep the current consumption of the device to as low a value as possible, perhaps only in the order of several micro-amps except in the IDLE1 and IDLE2 Mode, there are other considerations which must also be taken into account by the circuit designer if the power consumption is to be minimised. Special attention must be made to the I/O pins on the device. All high-impedance input pins must be connected to either a fixed high or low level as any floating input pins could create internal oscillations and result in increased current consumption. This also applies to devices which have different package types, as there may be unbonbed pins. These must either be setup as outputs or if setup as inputs must have pull-high resistors connected.

Care must also be taken with the loads, which are connected to I/O pins, which are setup as outputs. These should be placed in a condition in which minimum current is drawn or connected only to external circuits that do not draw current, such as other CMOS inputs. Also note that additional standby current will also be required if the LIRC oscillator has enabled.

In the IDLE1 and IDLE 2 Mode the high speed oscillator is on, if the peripheral function clock source is derived from the high speed oscillator, the additional standby current will also be perhaps in the order of several hundred micro-amps.

Wake-up

To minimise power consumption the device can enter the SLEEP or any IDLE Mode, where the CPU will be switched off. However, when the device is woken up again, it will take a considerable time for the original system oscillator to restart, stablise and allow normal operation to resume.

After the system enters the SLEEP or IDLE Mode, it can be woken up from one of various sources listed as follows:

- An external falling edge on Port A
- A external reset
- A system interrupt
- A WDT overflow

If the system is woken up by an external reset, the device will experience a full system reset. However, if the device is woken up by a WDT overflow, a Watchdog Timer reset will be initiated. Although both of these wake-up methods will initiate a reset operation, the actual source of the wake-up can be determined by examining the TO and PDF flags. The PDF flag is cleared by a system power-up or executing the clear Watchdog Timer instructions and is set when executing the "HALT" instruction. The TO flag is set if a WDT time-out occurs and causes a wake-up that only resets the Program Counter and Stack Pointer, other flags remain in their original status.

Each pin on Port A can be setup using the PAWU register to permit a negative transition on the pin to wake up the system. When a Port A pin wake-up occurs, the program will resume execution at the instruction following the "HALT" instruction. If the system is woken up by an interrupt, then two possible situations may occur. The first is where the related interrupt is disabled or the interrupt is enabled but the stack is full, in which case the program will resume execution at the instruction following the "HALT" instruction. In this situation, the interrupt which woke up the device will not be immediately serviced, but will rather be serviced later when the related interrupt is finally enabled or when a stack level becomes free. The other situation is where the related interrupt is enabled and the stack is not full, in which case the regular interrupt response takes place. If an interrupt request flag is set high before entering the SLEEP or IDLE Mode, the wake-up function of the related interrupt will be disabled.

Watchdog Timer

The Watchdog Timer is provided to prevent program malfunctions or sequences from jumping to unknown locations, due to certain uncontrollable external events such as electrical noise.

Watchdog Timer Clock Source

The Watchdog Timer clock source is provided by the internal clock, f_{LIRC}, which is in turn supplied by the LIRC oscillator. The Watchdog Timer source clock is then subdivided by a ratio of 2^8 to 2^{18} to give longer timeouts, the actual value being chosen using the WS2~WS0 bits in the WDTC register.

Watchdog Timer Control Register

A single register, WDTC, controls the required timeout period as well as the enable/disable and reset MCU operation.

- **WDTC Register**

Bit	7	6	5	4	3	2	1	0
Name	WE4	WE3	WE2	WE1	WE0	WS2	WS1	WS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	1	0	1	0	0	1	1

Bit 7~3 **WE4~WE0**: WDT function software control
10101: Disable
01010: Enable
Others: Reset MCU
When these bits are changed by the environmental noise or software setting to reset the microcontroller, the reset operation will be activated after a delay time, t_{SRESET}, and the WRF bit in the RSTFC register will be set high.

Bit 2~0 **WS2~WS0**: WDT time-out period selection
000: $2^8/f_{LIRC}$
001: $2^{10}/f_{LIRC}$
010: $2^{12}/f_{LIRC}$
011: $2^{14}/f_{LIRC}$
100: $2^{15}/f_{LIRC}$
101: $2^{16}/f_{LIRC}$
110: $2^{17}/f_{LIRC}$
111: $2^{18}/f_{LIRC}$
These three bits determine the division ratio of the watchdog timer source clock, which in turn determines the time-out period.

- **RSTFC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	RSTF	LVRF	LRF	WRF
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	x	0	0

"x": unknown

Bit 7~4 Unimplemented, read as "0"

Bit 3 **RSTF**: Reset control register software reset flag
Described elsewhere

Bit 2 **LVRF**: LVR function reset flag
Described elsewhere

Bit 1 **LRF**: LVR Control Register Software Reset Flag
Described elsewhere

Bit 0 **WRF**: WDT Control Register Software Reset Flag
0: Not occur
1: Occurred
This bit is set high by the WDT Control register software reset and cleared by the application program. Note that this bit can only be cleared to zero by the application program.

Watchdog Timer Operation

The Watchdog Timer operates by providing a device reset when its timer overflows. This means that in the application program and during normal operation the user has to strategically clear the Watchdog Timer before it overflows to prevent the Watchdog Timer from executing a reset. This is done using the clear watchdog instruction. If the program malfunctions for whatever reason, jumps to an unknown location, or enters an endless loop, the clear instruction will not be executed in the correct manner, in which case the Watchdog Timer will overflow and reset the device. There are five bits, WE4~WE0, in the WDTC register to offer the enable/disable control and reset control of the Watchdog Timer. The WDT function will be disabled when the WE4~WE0 bits are set to a value of 10101B while the WDT function will be enabled if the WE4~WE0 bits are equal to 01010B. If the WE4~WE0 bits are set to any other values, other than 01010B and 10101B, it will reset the device after a delay time, t_{SRESET}. After power on these bits will have a value of 01010B.

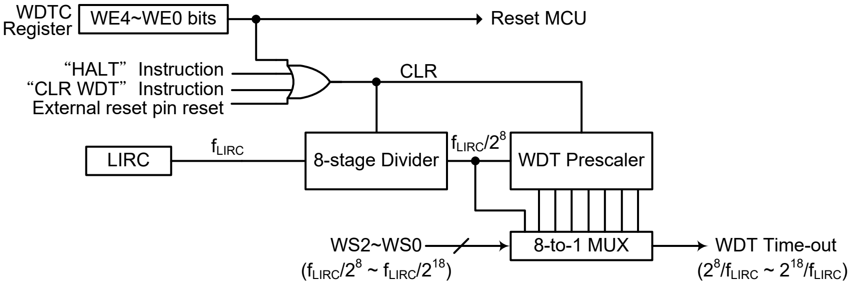
WE4~WE0 Bits	WDT Function
10101B	Disable
01010B	Enable
Any other values	Reset MCU

Watchdog Timer Enable/Disable Control

Under normal program operation, a Watchdog Timer time-out will initialise a device reset and set the status bit TO. However, if the system is in the SLEEP or IDLE Mode, when a Watchdog Timer time-out occurs, the TO bit in the status register will be set and only the Program Counter and Stack Pointer will be reset. Four methods can be adopted to clear the contents of the Watchdog Timer. The first is a WDT reset, which means a certain value except 01010B and 10101B written into the WE4~WE0 bit filed, the second is using the Watchdog Timer software clear instruction and the third is via a HALT instruction. The last is an external hardware reset, which means a low level on the external reset pin if the external reset pin exists by the RSTC register.

There is only one method of using software instruction to clear the Watchdog Timer. That is to use the single "CLR WDT" instruction to clear the WDT.

The maximum time out period is when the 2^{18} division ratio is selected. As an example, with a 32kHz LIRC oscillator as its source clock, this will give a maximum watchdog period of around 8 seconds for the 2^{18} division ratio, and a minimum timeout of 8ms for the 2^8 division ration.



Watchdog Timer

Reset and Initialisation

A reset function is a fundamental part of any microcontroller ensuring that the device can be set to some predetermined condition irrespective of outside parameters. The most important reset condition is after power is first applied to the microcontroller. In this case, internal circuitry will ensure that the microcontroller, after a short delay, will be in a well-defined state and ready to execute the first program instruction. After this power-on reset, certain important internal registers will be set to defined states before the program commences. One of these registers is the Program Counter, which will be reset to zero forcing the microcontroller to begin program execution from the lowest Program Memory address.

In addition to the power-on reset, situations may arise where it is necessary to forcefully apply a reset condition when the microcontroller is already running, the $\overline{\text{RES}}$ line is forcefully pulled low. In such a case, known as a normal operation reset, some of the microcontroller registers remain unchanged allowing the microcontroller to preceed with normal operation after the reset line is allowed to return high.

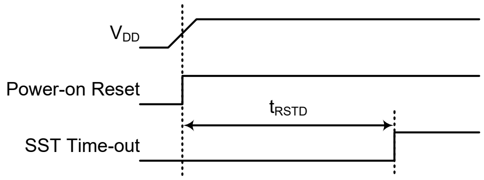
The Watchdog Timer overflow is one of many reset types and will reset the microcontroller. Another reset exists in the form of a Low Voltage Reset, LVR, where a full reset, similar to the $\overline{\text{RES}}$ reset is implemented in situations where the power supply voltage falls below a certain threshold. Another type of reset is when the Watchdog Timer overflows and resets the microcontroller. All types of reset operations result in different register conditions being setup.

Reset Functions

There are several ways in which a microcontroller reset can occur, through events occurring both internally and externally.

Power-on Reset

The most fundamental and unavoidable reset is the one that occurs after power is first applied to the microcontroller. As well as ensuring that the Program Memory begins execution from the first memory address, a power-on reset also ensures that certain other registers are preset to known conditions. All the I/O port and port control registers will power up in a high condition ensuring that all I/O ports will be first set to inputs.

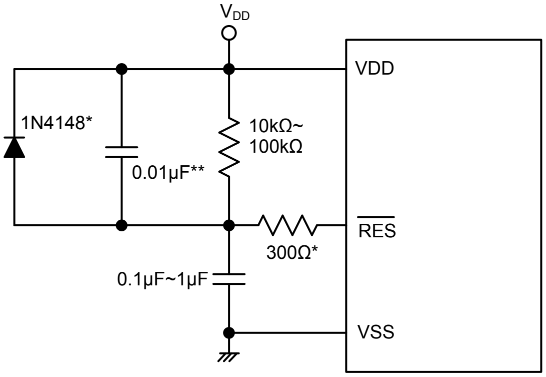


Power-On Reset Timing Chart

$\overline{\text{RES}}$ Pin Reset

As the reset pin is shared with I/O pins, the reset function must be selected using a control register, RSTC. Although the microcontroller has an internal RC reset function, if the V_{DD} power supply rise time is not fast enough or does not stabilise quickly at power-on, the internal reset function may be incapable of providing proper reset operation. For this reason it is recommended that an external RC network is connected to the $\overline{\text{RES}}$ pin, whose additional time delay will ensure that the $\overline{\text{RES}}$ pin remains low for an extended period to allow the power supply to stabilise. During this time delay, normal operation of the microcontroller will be inhibited. After the $\overline{\text{RES}}$ line reaches a certain voltage value, the reset delay time, t_{RSTD}, is invoked to provide an extea delay time after which the microcontroller will begin normal operation. The abbreviation SST in the figures stands for System

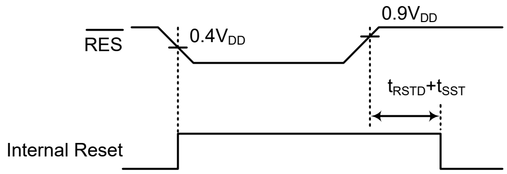
Start-up Time. For most applications a resistor connected between VDD and the $\overline{\text{RES}}$ line and a capacitor connected betweeb VSS and the $\overline{\text{RES}}$ pin will provide a suitable external reset circuit. Any wiring connected to the $\overline{\text{RES}}$ pin should be kept as short as possible to minimise any stray noise interference. For applications that operate within an environment where more noise is present the Enhanced Reset Circuit shown is recommended.



Note: "*" It is recommended that this component is added for added ESD protection.
"**" It is recommended that this component is added in environments where power line noise is significant.

External $\overline{\text{RES}}$ Circuit

Pulling the $\overline{\text{RES}}$ pin low using external hardware will also execute a device reset. In this case, as in the case of other resets, the Progran Counter will reset to zero and program execution initiated from this point.



$\overline{\text{RES}}$ Reset Timing Chart

There is an internal reset control register, RSTC, which is used to select the external $\overline{\text{RES}}$ pin function and provide a reset when the device operates abnormally due to the environmental noise interference. If the content of the RSTC register is set to any value other than 01010101B or 10101010B, it will reset the device after a delay time, t_{SRESET}. After power on the register will have a value of 01010101B.

RSTC7 ~ RSTC0 Bits	Reset Function
01010101B	PB5
10101010B	$\overline{\text{RES}}$
Any other value	Reset MCU

Internal Reset Function Control

- **RSTC Register**

Bit	7	6	5	4	3	2	1	0
Name	RSTC7	RSTC6	RSTC5	RSTC4	RSTC3	RSTC2	RSTC1	RSTC0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	1	0	1	0	1	0	1

Bit 7~0 **RSTC7~RSTC0**: Reset function control
01010101: PB5
10101010: $\overline{\text{RES}}$ pin
Other values: Reset MCU

If these bits are changed due to adverse environmental conditions, the microcontroller will be reset. The reset operation will be activated after a delay time, t_{SRESET}, and the RSTF bit in the RSTFC register will be set to 1.

All resets will reset this register to POR value except the WDT time out hardware warm reset. Note that if the register is set to 10101010 to select the $\overline{\text{RES}}$ pin, this configuration has higher priority than other related pin-shared controls.

- **RSTFC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	RSTF	LVRF	LRF	WRF
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	x	0	0

“x”: unknown

Bit 7~4 Unimplemented, read as “0”

Bit 3 **RSTF**: Reset control register software reset flag
0: Not occurred
1: Occurred

This bit is set to 1 by the RSTC control register software reset and cleared by the application program. Note that this bit can only be cleared to 0 by the application program.

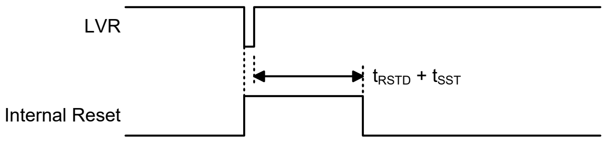
Bit 2 **LVRF**: LVR function reset flag
Described elsewhere

Bit 1 **LRF**: LVR control register software reset flag
Described elsewhere

Bit 0 **WRF**: WDT control register software reset flag
Described elsewhere

Low Voltage Reset – LVR

The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device. The LVR function is always enabled with a specific LVR voltage V_{LVR}. If the supply voltage of the device drops to within a range of 0.9V~V_{LVR} such as might occur when changing the battery, the LVR will automatically reset the device internally and the LVRF bit in the RSTFC register will also be set high. For a valid LVR signal, a low supply voltage, i.e., a voltage in the range between 0.9V~V_{LVR} must exist for a time greater than that specified by t_{LVR} in the LVD/LVR Electrical Characteristics. If the low supply voltage state does not exceed this value, the LVR will ignore the low supply voltage and will not perform a reset function. The actual V_{LVR} value can be selected by the LVS7~LVS0 bits in the LVRC register. If the LVS7~LVS0 bits are changed to some certain values by the environmental noise or software setting, the LVR will reset the device after a delay time, t_{SRESET}. When this happens, the LRF bit in the RSTFC register will be set high. After power on the register will have the value of 01010101B. Note that the LVR function will be automatically disabled when the device enters the IDLE/SLEEP mode.



Low Voltage Reset Timing Chart

- **LVRC Register**

Bit	7	6	5	4	3	2	1	0
Name	LVS7	LVS6	LVS5	LVS4	LVS3	LVS2	LVS1	LVS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	1	0	1	0	1	0	1

Bit 7~0 **LVS7~LVS0**: LVR voltage select
01010101: 2.1V
00110011: 2.55V
10011001: 3.15V
10101010: 3.8V
Other values: MCU reset (register is reset to POR value).

When an actual low voltage condition occurs, as specified by one of the four defined LVR voltage values above, an MCU reset will be generated. The reset operation will be activated after the low voltage condition keeps more than a t_{LVR} time. In this situation the register contents will remain the same after such a reset occurs.

Any register value, other than the four defined LVR values above, will also result in the generation of an MCU reset. The reset operation will be activated after a delay time, t_{SRESET}. However in this situation the register contents will be reset to the POR value.

- **RSTFC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	RSTF	LVRF	LRF	WRF
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	x	0	0

"x": unknown

Bit 7~4 Unimplemented, read as "0"

Bit 3 **RSTF**: Reset control register software reset flag
Described elsewhere

Bit 2 **LVRF**: LVR function reset flag
0: Not occur
1: Occurred

This bit is set high when a specific Low Voltage Reset situation condition occurs. This bit can only be cleared to zero by the application program.

Bit 1 **LRF**: LVR control register software reset flag
0: Not occur
1: Occurred

This bit is set high if the LVRC register contains any non-defined LVR voltage register values. This in effect acts like a software-reset function. This bit can only be cleared to zero by the application program.

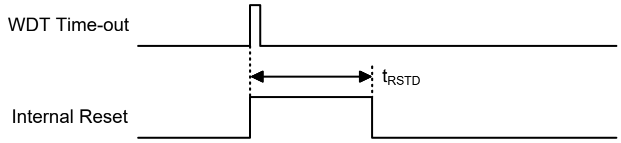
Bit 0 **WRF**: WDT Control register software reset flag
Described elsewhere

In Application Programming Reset

The device contains an IAP function, therefore an IAP reset exists, which is caused by writing data 55H to FC1 register.

Watchdog Time-out Reset during Normal Operation

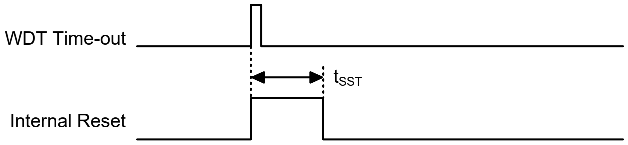
The Watchdog time-out Reset during normal operation is the same as LVR reset except that the Watchdog time-out flag TO will be set high.



WDT Time-out Reset during Normal Operation Timing Chart

Watchdog Time-out Reset during SLEEP or IDLE Mode

The Watchdog time-out Reset during SLEEP or IDLE Mode is a little different from other kinds of reset. Most of the conditions remain unchanged except that the Program Counter and the Stack Pointer will be cleared to zero and the TO flag will be set high. Refer to the System Start Up Time Characteristics for t_{SST} details.



WDT Time-out Reset during Sleep or IDLE Mode Timing Chart

Reset Initial Conditions

The different types of reset described affect the reset flags in different ways. These flags, known as PDF and TO are located in the status register and are controlled by various microcontroller operations, such as the SLEEP or IDLE Mode function or Watchdog Timer. The reset flags are shown in the table:

TO	PDF	Reset Conditions
0	0	Power-on reset
u	u	$\overline{\text{RES}}$ or LVR reset during FAST or SLOW Mode operation
1	u	WDT time-out reset during FAST or SLOW Mode operation
1	1	WDT time-out reset during IDLE or SLEEP Mode operation

"u" stands for unchanged

The following table indicates the way in which the various components of the microcontroller are affected after a power-on reset occurs.

Item	Condition after Reset
Program Counter	Reset to zero
Interrupts	All interrupts will be disabled
WDT, Time Bases	Clear after reset, WDT begins counting
Timer Modules	Timer Modules will be turned off
Input/Output Ports	I/O ports will be setup as inputs
Stack Pointer	Stack Pointer will point to the top of the stack

The different kinds of resets all affect the internal registers of the microcontroller in different ways. To ensure reliable continuation of normal program execution after a reset occurs, it is important to know what condition the microcontroller is in after a particular reset occurs. The following table describes how each type of reset affects each of the microcontroller internal registers. Note that where more than one package type exists the table will reflect the situation for the larger package type.

Register	Reset (Power On)	RESB Reset (Normal Operation)	LVR Reset (Normal Operation)	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
IAR0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MP0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IAR1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MP1L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MP1H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCL	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBHP	-xxx xxxx	-uuu uuuu	-uuu uuuu	-uuu uuuu	-uuu uuuu
STATUS	xx00 xxxx	uuuu uuuu	uuuu uuuu	xx1u uuuu	uu11 uuuu
PBP	---- --00	---- --00	---- --00	---- --00	---- --uu
IAR2	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MP2L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MP2H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
RSTFC	---- 0x00	---- uuuu	---- u1uu	---- uuuu	---- uuuu
INTC0	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
INTC1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
INTC2	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
INTC3	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PA	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAPU	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PAWU	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PB	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBPU	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PCC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PCPU	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PD	-111 1111	-111 1111	-111 1111	-111 1111	-uuu uuuu
PDC	-111 1111	-111 1111	-111 1111	-111 1111	-uuu uuuu
PDPU	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
PE	---1 1111	---1 1111	---1 1111	---1 1111	---u uuuu
PEC	---1 1111	---1 1111	---1 1111	---1 1111	---u uuuu
PEPU	---0 0000	---0 0000	---0 0000	---0 0000	---u uuuu
PF	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PFC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PFPU	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PG	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PGC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PGPU	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PH	--11 1111	--11 1111	--11 1111	--11 1111	--uu uuuu

Register	Reset (Power On)	RESB Reset (Normal Operation)	LVR Reset (Normal Operation)	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
PHC	--11 1111	--11 1111	--11 1111	--11 1111	--uu uuuu
PHPU	--00 0000	--00 0000	--00 0000	--00 0000	--uu uuuu
PMPS	---- --00	---- --00	---- --00	---- --00	---- --uu
RSTC	0101 0101	0101 0101	0101 0101	0101 0101	uuuu uuuu
VBGRC	---- ---0	---- ---0	---- ---0	---- ---0	---- ---u
MFI0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MFI1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MFI2	--00 --00	--00 --00	--00 --00	--00 --00	--uu --uu
MFI3	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MFI4	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MFI5	-000 -000	-000 -000	-000 -000	-000 -000	-uuu -uuu
MFI6	--00 --00	--00 --00	--00 --00	--00 --00	--uu --uu
MFI7	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
INTEG	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
SCC	000- 0000	000- 0000	000- 0000	000- 0000	uuu- uuuu
HIRCC	---- 0001	---- 0001	---- 0001	---- 0001	---- uuuu
HXTC	---- -000	---- -000	---- -000	---- -000	---- -uuu
LXTC	---- --00	---- --00	---- --00	---- --00	---- --uu
WDTC	0101 0011	0101 0011	0101 0011	0101 0011	uuuu uuuu
LVRC	0101 0101	0101 0101	0101 0101	0101 0101	uuuu uuuu
LVDC	--00 0000	--00 0000	--00 0000	--00 0000	--uu uuuu
EEAL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
EEAH	---- --00	---- --00	---- --00	---- --00	---- --uu
EED	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
CP0C	-000 00--	-000 00--	-000 00--	-000 00--	-uuu uu--
CP1C	-000 00--	-000 00--	-000 00--	-000 00--	-uuu uu--
PTM0C0	0000 0---	0000 0---	0000 0---	0000 0---	uuuu u---
PTM0C1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM0DL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM0DH	---- --00	---- --00	---- --00	---- --00	---- --uu
PTM0AL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM0AH	---- --00	---- --00	---- --00	---- --00	---- --uu
PTM0RPL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM0RPH	---- --00	---- --00	---- --00	---- --00	---- --uu
STM0C0	0000 0---	0000 0---	0000 0---	0000 0---	uuuu u---
STM0C1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM0DL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM0DH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM0AL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM0AH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM0RP	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
SLEDC0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
SLEDC1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
SLEDC2	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
SLEDC3	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MDUWR0	xxxx xxxx	0000 0000	0000 0000	0000 0000	uuuu uuuu
MDUWR1	xxxx xxxx	0000 0000	0000 0000	0000 0000	uuuu uuuu
MDUWR2	xxxx xxxx	0000 0000	0000 0000	0000 0000	uuuu uuuu
MDUWR3	xxxx xxxx	0000 0000	0000 0000	0000 0000	uuuu uuuu

Register	Reset (Power On)	RESB Reset (Normal Operation)	LVR Reset (Normal Operation)	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
MDUWR4	xxxx xxxx	0000 0000	0000 0000	0000 0000	uuuu uuuu
MDUWR5	xxxx xxxx	0000 0000	0000 0000	0000 0000	uuuu uuuu
MDUWCTRL	00-- ----	00-- ----	00-- ----	00-- ----	uu-- ----
CP0VOS	-001 0000	-001 0000	-001 0000	-001 0000	-uuu uuuu
CP1VOS	-001 0000	-001 0000	-001 0000	-001 0000	-uuu uuuu
PSC0R	---- --00	---- --00	---- --00	---- --00	---- --uu
TB0C	0--- -000	0--- -000	0--- -000	0--- -000	u--- -uuu
TB1C	0--- -000	0--- -000	0--- -000	0--- -000	u--- -uuu
PSC1R	---- --00	---- --00	---- --00	---- --00	---- --uu
SADOL	xxxx ----	xxxx ----	xxxx ----	xxxx ----	uuuu ---- (ADRFS=0)
					uuuu uuuu (ADRFS=1)
SADOH	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu (ADRFS=0)
					---- uuuu (ADRFS=1)
SADC0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
SADC1	0000 -000	0000 -000	0000 -000	0000 -000	uuuu -uuu
SADC2	0--0 0000	0--0 0000	0--0 0000	0--0 0000	u--u uuuu
SIMC0	111- 0000	111- 0000	111- 0000	111- 0000	uuu- uuuu
SIMC1	1000 0001	1000 0001	1000 0001	1000 0001	uuuu uuuu
SIMD	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
SIMA/SIMC2	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
SIMTOC	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
SPIAC0	111- --00	111- --00	111- --00	111- --00	uuu- --uu
SPIAC1	--00 0000	--00 0000	--00 0000	--00 0000	--uu uuuu
SPIAD	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
FARL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FARH	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
FD0L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FD0H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FD1L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FD1H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FD2L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FD2H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FD3L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FD3H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
SCOMC	-000 ----	-000 ----	-000 ----	-000 ----	-uuu ----
CRCCR	---- ---0	---- ---0	---- ---0	---- ---0	---- ---u
CRCIN	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
CRCDL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
CRCDH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IECC	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
U2SR	0000 1011	0000 1011	0000 1011	0000 1011	uuuu uuuu
U2CR1	0000 00x0	0000 00x0	0000 00x0	0000 00x0	uuuu uuuu
U2CR2	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TXR_RXR2	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
BRG2	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
U0SR	0000 1011	0000 1011	0000 1011	0000 1011	uuuu uuuu

Register	Reset (Power On)	RESB Reset (Normal Operation)	LVR Reset (Normal Operation)	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
U0CR1	0000 00x0	0000 00x0	0000 00x0	0000 00x0	uuuu uuuu
U0CR2	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TXR_RXR0	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
BRG0	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
U1SR	0000 1011	0000 1011	0000 1011	0000 1011	uuuu uuuu
U1CR1	0000 00x0	0000 00x0	0000 00x0	0000 00x0	uuuu uuuu
U1CR2	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TXR_RXR1	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
BRG1	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
PTM1C0	0000 0---	0000 0---	0000 0---	0000 0---	uuuu u---
PTM1C1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM1DL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM1DH	---- --00	---- --00	---- --00	---- --00	---- --uu
PTM1AL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM1AH	---- --00	---- --00	---- --00	---- --00	---- --uu
PTM1RPL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM1RPH	---- --00	---- --00	---- --00	---- --00	---- --uu
PTM2C0	0000 0---	0000 0---	0000 0---	0000 0---	uuuu u---
PTM2C1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM2DL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM2DH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM2AL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM2AH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM2RPL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM2RPH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM3C0	0000 0---	0000 0---	0000 0---	0000 0---	uuuu u---
PTM3C1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM3DL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM3DH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM3AL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM3AH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM3RPL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PTM3RPH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM1C0	0000 0---	0000 0---	0000 0---	0000 0---	uuuu u---
STM1C1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM1DL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM1DH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM1AL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM1AH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM1RP	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM2C0	0000 0---	0000 0---	0000 0---	0000 0---	uuuu u---
STM2C1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM2DL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM2DH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM2AL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM2AH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
STM2RP	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
EEC	---- 0000	---- 0000	---- 0000	---- 0000	---- uuuu
FC0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu

Register	Reset (Power On)	RESB Reset (Normal Operation)	LVR Reset (Normal Operation)	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
FC1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
FC2	---- ---0	---- ---0	---- ---0	---- ---0	---- ---u
IFS0	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
IFS1	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
IFS2	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
IFS3	---- -000	---- -000	---- -000	---- -000	---- -uuu
PAS0	00-- 00--	00-- 00--	00-- 00--	00-- 00--	uu-- uu--
PAS1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PBS0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PBS1	0000 --00	0000 --00	0000 --00	0000 --00	uuuu --uu
PCS0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PCS1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PDS0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PDS1	--00 0000	--00 0000	--00 0000	--00 0000	--uu uuuu
PES0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PES1	---- --00	---- --00	---- --00	---- --00	---- --uu
PFS0	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PFS1	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PGS0	---- 0000	---- 0000	---- 0000	---- 0000	---- uuuu
PGS1	0000 00--	0000 00--	0000 00--	0000 00--	uuuu uu--
PHS0	0000 ----	0000 ----	0000 ----	0000 ----	uuuu ----
PHS1	---- 0000	---- 0000	---- 0000	---- 0000	---- uuuu
CTRLRL	000- 0001	000- 0001	000- 0001	000- 0001	uuu- uuuu
STATRL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
ERRCNTL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
ERRCNTH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
BTRL	0000 0001	0000 0001	0000 0001	0000 0001	uuuu uuuu
BTRH	-010 0011	-010 0011	-010 0011	-010 0011	-uuu uuuu
INTRL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
INTRH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TESTRL	x000 00--	x000 00--	x000 00--	x000 00--	uuuu uu--
BRPERL	---- 0000	---- 0000	---- 0000	---- 0000	---- uuuu
IF1CREQL	--00 0001	--00 0001	--00 0001	--00 0001	--uu uuuu
IF1CREQH	0--- ----	0--- ----	0--- ----	0--- ----	u--- ----
IF1CMSKL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF1MSK1L	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
IF1MSK1H	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
IF1MSK2L	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
IF1MSK2H	11-1 1111	11-1 1111	11-1 1111	11-1 1111	uu-u uuuu
IF1ARB1L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF1ARB1H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF1ARB2L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF1ARB2H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF1MCTRL	0--- 0000	0--- 0000	0--- 0000	0--- 0000	u--- uuuu
IF1MCTRH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF1DTA1L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF1DTA1H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF1DTA2L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF1DTA2H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu

Register	Reset (Power On)	RESB Reset (Normal Operation)	LVR Reset (Normal Operation)	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
IF1DTB1L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF1DTB1H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF1DTB2L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF1DTB2H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
CRLL	0010 0111	0010 0111	0010 0111	0010 0111	uuuu uuuu
CRLH	0000 0010	0000 0010	0000 0010	0000 0010	uuuu uuuu
CRHL	0000 0101	0000 0101	0000 0101	0000 0101	uuuu uuuu
CRHH	0010 0001	0010 0001	0010 0001	0010 0001	uuuu uuuu
IF2CREQL	--00 0001	--00 0001	--00 0001	--00 0001	--uu uuuu
IF2CREQH	0--- ----	0--- ----	0--- ----	0--- ----	u--- ----
IF2CMSKL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF2MSK1L	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
IF2MSK1H	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
IF2MSK2L	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
IF2MSK2H	11-1 1111	11-1 1111	11-1 1111	11-1 1111	uu-u uuuu
IF2ARB1L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF2ARB1H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF2ARB2L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF2ARB2H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF2MCTRL	0--- 0000	0--- 0000	0--- 0000	0--- 0000	u--- uuuu
IF2MCTRH	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF2DTA1L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF2DTA1H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF2DTA2L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF2DTA2H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF2DTB1L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF2DTB1H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF2DTB2L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
IF2DTB2H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TREQR1L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TREQR1H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TREQR2L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TREQR2H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
NEWDT1L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
NEWDT1H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
NEWDT2L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
NEWDT2H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
INTPND1L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
INTPND1H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
INTPND2L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
INTPND2H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MSGVAL1L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MSGVAL1H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MSGVAL2L	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
MSGVAL2H	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
CANCFG	10-0 0000	10-0 0000	10-0 0000	10-0 0000	uu-u uuuu

Note: "u" stands for unchanged
"x" stands for unknown
"-" stands for unimplemented

Input/Output Ports

The microcontrollers offer considerable flexibility on their I/O ports. With the input or output designation of every pin fully under user program control, pull-high selections for all ports and wake-up selections on certain pins, the user is provided with an I/O structure to meet the needs of a wide range of application possibilities.

The device provides bidirectional input/output lines labeled with port names PA~PH. These I/O ports are mapped to the RAM Data Memory with specific addresses as shown in the Special Purpose Data Memory table. All of these I/O ports can be used for input and output operations. For input operation, these ports are non-latching, which means the inputs must be ready at the T2 rising edge of instruction "MOV A, [m]", where m denotes the port address. For output operation, all the data is latched and remains unchanged until the output latch is rewritten.

Register Name	Bit							
	7	6	5	4	3	2	1	0
PA	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
PAC	PAC7	PAC6	PAC5	PAC4	PAC3	PAC2	PAC1	PAC0
PAPU	PAPU7	PAPU6	PAPU5	PAPU4	PAPU3	PAPU2	PAPU1	PAPU0
PAWU	PAWU7	PAWU6	PAWU5	PAWU4	PAWU3	PAWU2	PAWU1	PAWU0
PB	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
PBC	PBC7	PBC6	PBC5	PBC4	PBC3	PBC2	PBC1	PBC0
PBPU	PBPU7	PBPU6	PBPU5	PBPU4	PBPU3	PBPU2	PBPU1	PBPU0
PC	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
PCC	PCC7	PCC6	PCC5	PCC4	PCC3	PCC2	PCC1	PCC0
PCPU	PCPU7	PCPU6	PCPU5	PCPU4	PCPU3	PCPU2	PCPU1	PCPU0
PD	—	PD6	PD5	PD4	PD3	PD2	PD1	PD0
PDC	—	PDC6	PDC5	PDC4	PDC3	PDC2	PDC1	PDC0
PDPU	—	PDPU6	PDPU5	PDPU4	PDPU3	PDPU2	PDPU1	PDPU0
PE	—	—	—	PE4	PE3	PE2	PE1	PE0
PEC	—	—	—	PEC4	PEC3	PEC2	PEC1	PEC0
PEPU	—	—	—	PEPU4	PEPU3	PEPU2	PEPU1	PEPU0
PF	PF7	PF6	PF5	PF4	PF3	PF2	PF1	PF0
PFC	PFC7	PFC6	PFC5	PFC4	PFC3	PFC2	PFC1	PFC0
PFPU	PFPU7	PFPU6	PFPU5	PFPU4	PFPU3	PFPU2	PFPU1	PFPU0
PG	PG7	PG6	PG5	PG4	PG3	PG2	PG1	PG0
PGC	PGC7	PGC6	PGC5	PGC4	PGC3	PGC2	PGC1	PGC0
PGPU	PGPU7	PGPU6	PGPU5	PGPU4	PGPU3	PGPU2	PGPU1	PGPU0
PH	—	—	PH5	PH4	PH3	PH2	PH1	PH0
PHC	—	—	PHC5	PHC4	PHC3	PHC2	PHC1	PHC0
PHPU	—	—	PHPU5	PHPU4	PHPU3	PHPU2	PHPU1	PHPU0

"—": Unimplemented

I/O Logic Function Register List

Pull-high Resistors

Many product applications require pull-high resistors for their switch inputs usually requiring the use of an external resistor. To eliminate the need for these external resistors, all I/O pins, when configured as an digital input have the capability of being connected to an internal pull-high resistor. These pull-high resistors are selected using the relavant pull-high control registers, and are implemented using weak PMOS transistors. Note that the pull-high resistor can be controlled by the relevant pull-high control register only when the pin-shared functional pin is selected as a digital input or NMOS output. Otherwise, the pull-high resistors cannot be enabled.

- **PxPU Register**

Bit	7	6	5	4	3	2	1	0
Name	PxPU7	PxPU6	PxPU5	PxPU4	PxPU3	PxPU2	PxPU1	PxPU0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

PxPUn: I/O Port x Pin Pull-high Function Control
0: Disable
1: Enable
The PxPUn bit is used to control the pin pull-high function.Here the “x” can be A, B, C, D, E, F, G and H. However, the actual available bits for each I/O port may be different.

Port A Wake-up

The HALT instruction forces the microcontroller into the SLEEP or IDLE Mode which preserves power, a feature that is important for battery and other low-power applications. Various methods exist to wake-up the microcontroller, one of which is to change the logic condition on one of the Port A pins from high to low. This function is especially suitable for applications that can be woken up via external switches. Each pin on Port A can be selected individually to have this wake-up feature using the PAWU register. Note that the wake-up function can be controlled by the wake-up control registers only when the pin is selected as a general purpose input and the MCU enters the Power down mode.

- **PAWU Register**

Bit	7	6	5	4	3	2	1	0
Name	PAWU7	PAWU6	PAWU5	PAWU4	PAWU3	PAWU2	PAWU1	PAWU0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

PAWUn: Port A Pin Wake-up Control
0: Disable
1: Enable

I/O Port Control Registers

Each I/O port has its own control register which controls the input/output configuration. With this control register, each CMOS output or input can be reconfigured dynamically under software control. Each pin of the I/O ports is directly mapped to a bit in its associated port control register. For the I/O pin to function as an input, the corresponding bit of the control register must be written as a “1”. This will then allow the logic state of the input pin to be directly read by instructions. When the corresponding bit of the control register is written as a “0”, the I/O pin will be setup as a CMOS output. If the pin is currently setup as an output, instructions can still be used to read the output register. However, it should be noted that the program will in fact only read the status of the output data latch and not the actual logic status of the output pin.

- **PxC Register**

Bit	7	6	5	4	3	2	1	0
Name	PxC7	PxC5	PxC5	PxC4	PxC3	PxC2	PxC1	PxC0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	1	1	1	1	1	1	1

PxCn: I/O Port x Pin Type Selection
0: Output
1: Input
The PxCn bit is used to control the pin type selection. Here the “x” can be A, B, C, D, E, F, G and H. However, the actual available bits for each I/O port may be different.

I/O Port Source Current Control

The device supports different source current driving capability for each I/O port. With the corresponding selection registers, SLEDCn, each I/O port can support four levels of the source current driving capability. These source current bits are available when the corresponding pin is configured as a CMOS output. Otherwise, these select bits have no effect. Users should refer to the Input/Output Characteristics section to select the desired source current for different applications.

Register Name	Bit							
	7	6	5	4	3	2	1	0
SLEDC0	SLEDC07	SLEDC06	SLEDC05	SLEDC04	SLEDC03	SLEDC02	SLEDC01	SLEDC00
SLEDC1	SLEDC17	SLEDC16	SLEDC15	SLEDC14	SLEDC13	SLEDC12	SLEDC11	SLEDC10
SLEDC2	SLEDC27	SLEDC26	SLEDC25	SLEDC24	SLEDC23	SLEDC22	SLEDC21	SLEDC20
SLEDC3	SLEDC37	SLEDC36	SLEDC35	SLEDC34	SLEDC33	SLEDC32	SLEDC31	SLEDC30

I/O Port Source Current Control Register List

- **SLEDC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	SLEDC07	SLEDC06	SLEDC05	SLEDC04	SLEDC03	SLEDC02	SLEDC01	SLEDC00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **SLEDC07~SLEDC06**: PB7~PB4 Source Current Selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3 (max.)

Bit 5~4 **SLEDC05~SLEDC04**: PB3~PB0 Source Current Selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3 (max.)

Bit 3~2 **SLEDC03~SLEDC02**: PA7~PA4 Source Current Selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3 (max.)

Bit 1~0 **SLEDC01~SLEDC00**: PA3~PA0 Source Current Selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3 (max.)

- **SLEDC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	SLEDC17	SLEDC16	SLEDC15	SLEDC14	SLEDC13	SLEDC12	SLEDC11	SLEDC10
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **SLEDC17~SLEDC16**: PD6~PD4 Source Current Selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3 (max.)

Bit 5~4 **SLEDC15~SLEDC14**: PD3~PD0 Source Current Selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3 (max.)

Bit 3~2 **SLEDC13~SLEDC12**: PC7~PC4 Source Current Selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3 (max.)

Bit 1~0 **SLEDC11~SLEDC10**: PC3~PC0 Source Current Selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3 (max.)

- **SLEDC2 Register**

Bit	7	6	5	4	3	2	1	0
Name	SLEDC27	SLEDC26	SLEDC25	SLEDC24	SLEDC23	SLEDC22	SLEDC21	SLEDC20
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **SLEDC27~SLEDC26**: PF7~PF4 Source Current Selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3 (max.)

Bit 5~4 **SLEDC25~SLEDC24**: PF3~PF0 Source Current Selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3 (max.)

Bit 3~2 **SLEDC23~SLEDC22**: PE4 Source Current Selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3 (max.)

Bit 1~0 **SLEDC21~SLEDC20**: PE3~PE0 Source Current Selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3 (max.)

- **SLEDC3 Register**

Bit	7	6	5	4	3	2	1	0
Name	SLEDC37	SLEDC36	SLEDC35	SLEDC34	SLEDC33	SLEDC32	SLEDC31	SLEDC30
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **SLEDC37~SLEDC36**: PH5~PH4 Source Current Selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3 (max.)

Bit 5~4 **SLEDC35~SLEDC34**: PH3~PH0 Source Current Selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3 (max.)

Bit 3~2 **SLEDC33~SLEDC32**: PG7~PG4 Source Current Selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3 (max.)

Bit 1~0 **SLEDC31~SLEDC30**: PG3~PG0 Source Current Selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3 (max.)

I/O Port Power Source Control

The device supports different I/O port power source selections for PE3~PE0. The port power can come from either the power pin VDD or VDDIO which is determined using the PMPS1~PMPS0 bits in the PMPS register. The VDDIO power pin function should first be selected using the corresponding pin-shared function selection bits if the port power is supposed to come from the VDDIO pin. An important point to know is that the input power voltage on the VDDIO pin should be equal to or less than the device supply power voltage when the VDDIO pin is selected as the port power supply pin.

- **PMPS Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	PMPS1	PMPS0
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as “0”

Bit 1~0 **PMPS1~PMPS0**: PE3~PE0 pin power source selection
0x: VDD
1x: VDDIO

Pin-shared Functions

The flexibility of the microcontroller range is greatly enhanced by the use of pins that have more than one function. Limited numbers of pins can force serious design constraints on designers but by supplying pins with multi-functions, many of these difficulties can be overcome. For these pins, the desired function of the multi-function I/O pins is selected by a series of registers via the application program control.

Pin-shared Function Selection Registers

The limited number of supplied pins in a package can impose restrictions on the amount of functions a certain device can contain. However by allowing the same pins to share several different functions and providing a means of function selection, a wide range of different functions can be incorporated into even relatively small package sizes. The device includes Port “x” output function Selection register “n”, labeled as PxSn, and Input Function Selection register, labeled as IFSn, which can select the desired functions of the multi-function pin-shared pins.

The most important point to note is to make sure that the desired pin-shared function is properly selected and also deselected. For most pin-shared functions, to select the desired pin-shared function, the pin-shared function should first be correctly selected using the corresponding pin-shared control register. After that the corresponding peripheral functional setting should be configured and then the peripheral function can be enabled. However, special point must be noted for some digital input pins, such as INTn, xTCKn, etc, which share the same pin-shared control configuration with their corresponding general purpose I/O functions when setting the relevant functions, in addition to the necessary pin-shared control and peripheral functional setup aforementioned, they must also be setup as input by setting the corresponding bit in the I/O port control register. To correctly deselect the pin-shared function, the peripheral function should first be disabled and then the corresponding pin-shared function control register can be modified to select other pin-shared functions.

Register Name	Bit							
	7	6	5	4	3	2	1	0
PAS0	PAS07	PAS06	—	—	PAS03	PAS02	—	—
PAS1	PAS17	PAS16	PAS15	PAS14	PAS13	PAS12	PAS11	PAS10
PBS0	PBS07	PBS06	PBS05	PBS04	PBS03	PBS02	PBS01	PBS00
PBS1	PBS17	PBS16	PBS15	PBS14	—	—	PBS11	PBS10
PCS0	PCS07	PCS06	PCS05	PCS04	PCS03	PCS02	PCS01	PCS00
PCS1	PCS17	PCS16	PCS15	PCS14	PCS13	PCS12	PCS11	PCS10
PDS0	PDS07	PDS06	PDS05	PDS04	PDS03	PDS02	PDS01	PDS00
PDS1	—	—	PDS15	PDS14	PDS13	PDS12	PDS11	PDS10
PES0	PES07	PES06	PES05	PES04	PES03	PES02	PES01	PES00
PES1	—	—	—	—	—	—	PES11	PES10
PFS0	PFS07	PFS06	PFS05	PFS04	PFS03	PFS02	PFS01	PFS00
PFS1	PFS17	PFS16	PFS15	PFS14	PFS13	PFS12	PFS11	PFS10
PGS0	—	—	—	—	PGS03	PGS02	PGS01	PGS00
PGS1	D7	D6	D5	D4	PGS13	PGS12	—	—
PHS0	PHS07	PHS06	PHS05	PHS04	—	—	—	—
PHS1	—	—	—	—	PHS13	PHS12	PHS11	PHS10
IFS0	—	PTCK3PS	PTCK2PS	PTCK1PS	PTCK0PS	STCK2PS	STCK1PS	STCK0PS
IFS1	—	PTP3IPS	PTP2IPS	PTP1IPS	PTP0IPS	STP2IPS	STP1IPS	STP0IPS
IFS2	—	SCSBPS	SDISDAPS	SCKSCLPS	INT3PS	INT2PS	INT1PS	INT0PS
IFS3	—	—	—	—	—	RX2PS	RX1PS	RX0PS

Pin-shared Function Selection Registers List

- **PAS0 Register**

Bit	7	6	5	4	3	2	1	0
Name	PAS07	PAS06	—	—	PAS03	PAS02	—	—
R/W	R/W	R/W	—	—	R/W	R/W	—	—
POR	0	0	—	—	0	0	—	—

Bit 7~6 **PAS07~PAS06**: PA3 pin-shared function selection
00/01/10: PA3/INT1
11: SDO

Bit 5~4 Unimplemented, read as "0"

Bit 3~2 **PAS03~PAS02**: PA1 pin-shared function selection
00/01/10: PA1/INT0
11: $\overline{\text{SCS}}$

Bit 1~0 Unimplemented, read as "0"

- **PAS1 Register**

Bit	7	6	5	4	3	2	1	0
Name	PAS17	PAS16	PAS15	PAS14	PAS13	PAS12	PAS11	PAS10
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PAS17~PAS16**: PA7 pin-shared function selection
00/01/10: PA7/INT1
11: TX0

Bit 5~4 **PAS15~PAS14**: PA6 pin-shared function selection
00/01/10: PA6/INT0
11: RX0

Bit 3~2 **PAS13~PAS12**: PA5 pin-shared function selection
00/01/10: PA5/INT3
11: SCK/SCL

Bit 1~0 **PAS11~PAS10**: PA4 pin-shared function selection
00/01/10: PA4/INT2
11: SDI/SDA

- **PBS0 Register**

Bit	7	6	5	4	3	2	1	0
Name	PBS07	PBS06	PBS05	PBS04	PBS03	PBS02	PBS01	PBS00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PBS07~PBS06**: PB3 pin-shared function selection
00/01: PB3/PTP2I
10: RX2
11: PTP2

Bit 5~4 **PBS05~PBS04**: PB2 pin-shared function selection
00/01/10: PB2
11: CANRX

Bit 3~2 **PBS03~PBS02**: PB1 pin-shared function selection
00/01/10: PB1
11: CANTX

Bit 1~0 **PBS01~PBS00**: PB0 pin-shared function selection
00/01/10: PB0/STCK2
11: C0X

- **PBS1 Register**

Bit	7	6	5	4	3	2	1	0
Name	PBS17	PBS16	PBS15	PBS14	—	—	PBS11	PBS10
R/W	R/W	R/W	R/W	R/W	—	—	R/W	R/W
POR	0	0	0	0	—	—	0	0

Bit 7~6 **PBS17~PBS16**: PB7 pin-shared function selection
00/01/10: PB7/STCK1
11: OSC2

Bit 5~4 **PBS15~PBS14**: PB6 pin-shared function selection
00/01: PB6/STP1I
10: STP1
11: OSC1

Bit 3~2 Unimplmented, read as "0"

Bit 1~0 **PBS11~PBS10**: PB4 pin-shared function selection
00/01: PB4
10: TX2
11: C1X

- **PCS0 Register**

Bit	7	6	5	4	3	2	1	0
Name	PCS07	PCS06	PCS05	PCS04	PCS03	PCS02	PCS01	PCS00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PCS07~PCS06**: PC3 pin-shared function selection
00/01/10: PC3/PTCK0
11: AN3

Bit 5~4 **PCS05~PCS04**: PC2 pin-shared function selection
00/01: PC2/PTP0I
10: PTP0
11: AN2

Bit 3~2 **PCS03~PCS02**: PC1 pin-shared function selection
00: PC1
01: C0X
10: VREF
11: AN1

Bit 1~0 **PCS01~PCS00**: PC0 pin-shared function selection
00/01: PC0
10: VREFI
11: AN0

- **PCS1 Register**

Bit	7	6	5	4	3	2	1	0
Name	PCS17	PCS16	PCS15	PCS14	PCS13	PCS12	PCS11	PCS10
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PCS17~PCS16**: PC7 pin-shared function selection
00/01/10: PC7/INT3/STCK0
11: AN7

Bit 5~4 **PCS15~PCS14**: PC6 pin-shared function selection
00/01: PC6/STP0I
10: STP0
11: AN6

Bit 3~2 **PCS13~PCS12**: PC5 pin-shared function selection
00/01/10: PC5/PTCK1
11: AN5

Bit 1~0 **PCS11~PCS10**: PC4 pin-shared function selection
00/01: PC4/PTP1I
10: PTP1
11: AN4

- **PDS0 Register**

Bit	7	6	5	4	3	2	1	0
Name	PDS07	PDS06	PDS05	PDS04	PDS03	PDS02	PDS01	PDS00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PDS07~PDS06**: PD3 pin-shared function selection
00/01/10: PD3/PTCK2
11: AN11

Bit 5~4 **PDS05~PDS04**: PD2 pin-shared function selection
00: PD2/PTP2I
01: PTP2
10: TX1
11: AN10

Bit 3~2 **PDS03~PDS02**: PD1 pin-shared function selection
00/01: PD1/STCK1
10: RX1
11: AN9

Bit 1~0 **PDS01~PDS00**: PD0 pin-shared function selection
00/01: PD0/INT2/STP1I
10: STP1
11: AN8

- **PDS1 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	PDS15	PDS14	PDS13	PDS12	PDS11	PDS10
R/W	—	—	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	—	0	0	0	0	0	0

Bit 7~6 Unimplemented, read as “0”

Bit 5~4 **PDS15~PDS14**: PD6 pin-shared function selection
00/01: PD6/STP2I
10: STP2
11: C1X

Bit 3~2 **PDS13~PDS12**: PD5 pin-shared function selection
00/01: PD5/PTCK3
10: TX0
11: C1+

Bit 1~0 **PDS11~PDS10**: PD4 pin-shared function selection
00: PD4/PTP3I
01: RX0
10: PTP3
11: C1-

- **PES0 Register**

Bit	7	6	5	4	3	2	1	0
Name	PES07	PES06	PES05	PES04	PES03	PES02	PES01	PES00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PES07~PES06**: PE3 pin-shared function selection
00/01: PE3/PTP1I
10: PTP1
11: SCKA

Bit 5~4 **PES05~PES04**: PE2 pin-shared function selection
00/01/10: PE2/PTCK1
11: SDIA

Bit 3~2 **PES03~PES02**: PE1 pin-shared function selection
00/01: PE1/STP0I
10: STP0
11: SDOA

Bit 1~0 **PES01~PES00**: PE0 pin-shared function selection
00/01/10: PE0/STCK0
11: $\overline{\text{SCSA}}$

- **PES1 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	PES11	PES10
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as “0”

Bit 1~0 **PES11~PES10**: PE4 pin-shared function selection
00/01/10: PE4
11: VDDIO

- **PFS0 Register**

Bit	7	6	5	4	3	2	1	0
Name	PFS07	PFS06	PFS05	PFS04	PFS03	PFS02	PFS01	PFS00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PFS07~PFS06**: PF3 pin-shared function selection
00/01: PF3
10: SCK/SCL
11: SCOM3

Bit 5~4 **PFS05~PFS04**: PF2 pin-shared function selection
00/01: PF2
10: SDI/SDA
11: SCOM2

Bit 3~2 **PFS03~PFS02**: PF1 pin-shared function selection
00/01: PF1
10: SDO
11: SCOM1

Bit 1~0 **PFS01~PFS00**: PF0 pin-shared function selection
00/01: PF0
10: $\overline{\text{SCS}}$
11: SCOM0

- **PFS1 Register**

Bit	7	6	5	4	3	2	1	0
Name	PFS17	PFS16	PFS15	PFS14	PFS13	PFS12	PFS11	PFS10
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PFS17~PFS16**: PF7 pin-shared function selection
00: PF7/STP2I
01: TX1
10: STP2
11: C0+

Bit 5~4 **PFS15~PFS14**: PF6 pin-shared function selection
00/01: PF6/STCK2
10: RX1
11: C0-

Bit 3~2 **PFS13~PFS12**: PF5 pin-shared function selection
00/01: PF5/PTP0I
10: PTP0
11: XT1

Bit 1~0 **PFS11~PFS10**: PF4 pin-shared function selection
00/01/10: PF4/PTCK0
11: XT2

- **PGS0 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	PGS03	PGS02	PGS01	PGS00
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	0	0	0

Bit 7~4 Unimplemented, read as “0”

Bit 3~2 **PGS03~PGS02**: PG1 pin-shared function selection
00/01/10: PG1
11: TX2

Bit 1~0 **PGS01~PGS00**: PG0 pin-shared function selection
00/01/10: PG0
11: RX2

- **PGS1 Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	PGS13	PGS12	—	—
R/W	R/W	R/W	R/W	R/W	R/W	R/W	—	—
POR	0	0	0	0	0	0	—	—

Bit 7~4 **D7~D4**: Reserved, must be fixed at “0000B”

Bit 3~2 **PGS13~PGS12**: PG5 pin-shared function selection
00/01/10: PG5/PTP3I/PTCK2
11: PTP3

Bit 1~0 Unimplemented, read as “0”

- **PHS0 Register**

Bit	7	6	5	4	3	2	1	0
Name	PHS07	PHS06	PHS05	PHS04	—	—	—	—
R/W	R/W	R/W	R/W	R/W	—	—	—	—
POR	0	0	0	0	—	—	—	—

Bit 7~6 **PHS07~PHS06**: PH3 pin-shared function selection
00/01/10: PH3
11: AN13

Bit 5~4 **PHS05~PHS04**: PH2 pin-shared function selection
00/01/10: PH2
11: AN12

Bit 3~0 Unimplemented, read as “0”

- **PHS1 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	PHS13	PHS12	PHS11	PHS10
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	0	0	0

Bit 7~4 Unimplemented, read as “0”

Bit 3~2 **PHS13~PHS12**: PH5 pin-shared function selection
00/01/10: PH5
11: AN15

Bit 1~0 **PHS11~PHS10**: PH4 pin-shared function selection
00/01/10: PH4
11: AN14

- **IFS0 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	PTCK3PS	PTCK2PS	PTCK1PS	PTCK0PS	STCK2PS	STCK1PS	STCK0PS
R/W	—	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	0	0	0	0	0	0	0

Bit 7 Unimplemented, read as “0”

Bit 6 **PTCK3PS**: PTCK3 input source pin selection
0: PD5
1: PG4

Bit 5 **PTCK2PS**: PTCK2 input source pin selection
0: PD3
1: PG5

Bit 4 **PTCK1PS**: PTCK1 input source pin selection
0: PC5
1: PE2

Bit 3 **PTCK0PS**: PTCK0 input source pin selection
0: PC3
1: PF4

Bit 2 **STCK2PS**: STCK2 input source pin selection
0: PF6
1: PB0

Bit 1 **STCK1PS**: STCK1 input source pin selection
0: PD1
1: PB7

Bit 0 **STCK0PS**: STCK0 input source pin selection
0: PC7
1: PE0

- **IFS1 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	PTP3IPS	PTP2IPS	PTP1IPS	PTP0IPS	STP2IPS	STP1IPS	STP0IPS
R/W	—	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	0	0	0	0	0	0	0

Bit 7 Unimplemented, read as “0”

Bit 6 **PTP3IPS**: PTP3I input source pin selection
0: PD4
1: PG5

Bit 5 **PTP2IPS**: PTP2I input source pin selection
0: PD2
1: PB3

Bit 4 **PTP1IPS**: PTP1I input source pin selection
0: PC4
1: PE3

Bit 3 **PTP0IPS**: PTP0I input source pin selection
0: PC2
1: PF5

Bit 2 **STP2IPS**: STP2I input source pin selection
0: PD6
1: PF7

Bit 1 **STP1IPS**: STP1I input source pin selection
0: PD0
1: PB6

Bit 0 **STP0IPS**: STP0I input source pin selection
0: PC6
1: PE1

- **IFS2 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	SCSBPS	SDISDAPS	SCKSCLPS	INT3PS	INT2PS	INT1PS	INT0PS
R/W	—	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	0	0	0	0	0	0	0

Bit 7 Unimplemented, read as “0”

Bit 6 **SCSBPS**: $\overline{\text{SCS}}$ input source pin selection
0: PA1
1: PF0

Bit 5 **SDISDAPS**: SDI/SDA input source pin selection
0: PA4
1: PF2

Bit 4 **SCKSCLPS**: SCK/SCL input source pin selection
0: PA5
1: PF3

Bit 3 **INT3PS**: INT3 input source pin selection
0: PA5
1: PC7

Bit 2 **INT2PS**: INT2 input source pin selection
0: PA4
1: PD0

Bit 1 **INT1PS**: INT1 input source pin selection
0: PA3
1: PA7

Bit 0 **INT0PS**: INT0 input source pin selection
0: PA1
1: PA6

- **IFS3 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	RX2PS	RX1PS	RX0PS
R/W	—	—	—	—	—	R/W	R/W	R/W
POR	—	—	—	—	—	0	0	0

Bit 7~3 Unimplemented, read as “0”

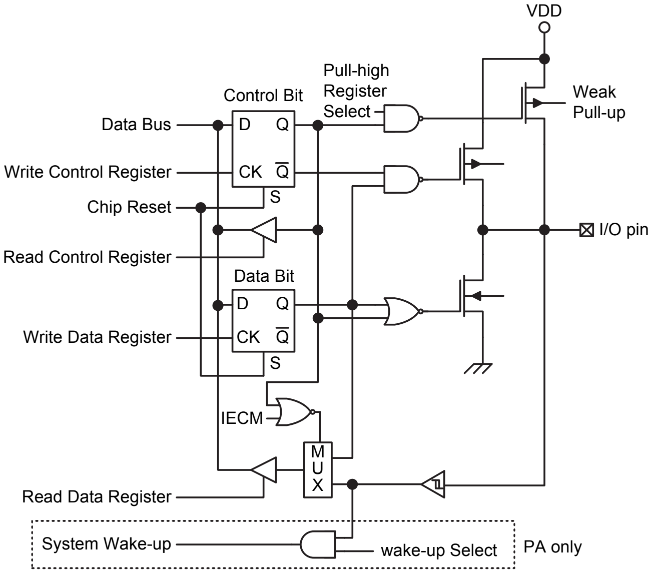
Bit 2 **RX2PS**: RX2 input source pin selection
0: PB3
1: PG0

Bit 1 **RX1PS**: RX1 input source pin selection
0: PD1
1: PF6

Bit 0 **RX0PS**: RX0 input source pin selection
0: PA6
1: PD4

I/O Pin Structures

The accompanying diagram illustrates the internal structures of the I/O logic function. As the exact logical construction of the I/O pin will differ from this diagram, it is supplied as a guide only to assist with the functional understanding of the logic function I/O pins. The wide range of pin-shared structures does not permit all types to be shown.



Logic Function Input/Output Structure

READ PORT Function

The READ PORT function is used to manage the reading of the output data from the data latch or I/O pin, which is specially designed for the IEC60730 self-diagnostic test on the I/O function and A/D paths. There is a register, IECC, which is used to control the READ PORT function. If the READ PORT function is disabled, the pin function will operate as the selected pin-shared function. When a specific data pattern, “11001010”, is written into the IECC register, the internal signal named IECM will be set high to enable the READ PORT function. If the READ PORT function is enabled, the value on the corresponding pins will be passed to the accumulator ACC when the read port instruction “mov acc, Px” is executed where the “x” stands for the corresponding I/O port name.

- **IECC Register**

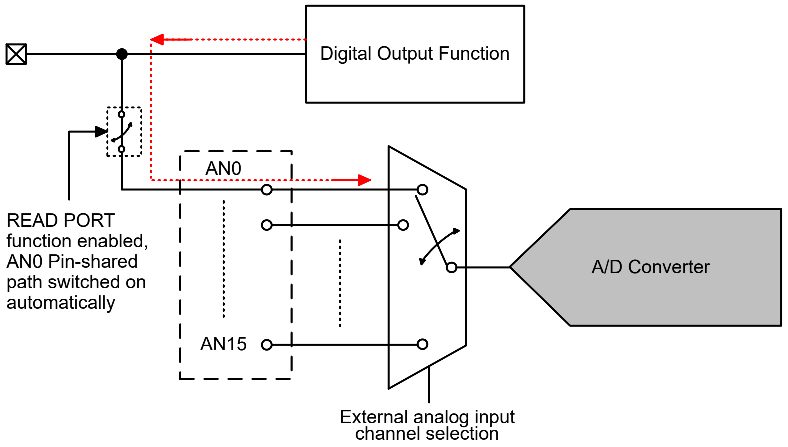
Bit	7	6	5	4	3	2	1	0
Name	IECS7	IECS6	IECS5	IECS4	IECS3	IECS2	IECS1	IECS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **IECS7~IECS0**: READ PORT function enable control bit 7~bit 0
11001010: IECM=1 – READ PORT function is enabled
Others: IECM=0 – READ PORT function is disabled

READ PORT Function	Disabled		Enabled	
Port Control Register Bit – PxC.n	1	0	1	0
I/O Function	Pin value	Data latch value	Pin value	
Digital Input Function				
Digital Output Function (except I^{2}C SDA/SCL)	0			
I^{2}C SDA/SCL	Pin value			
Analog Function	0			

Note: The value on the above table is the content of the ACC register after "mov a, Px" instruction is executed where "x" means the relevant port name.

The additional function of the READ PORT mode is to check the A/D path. When the READ PORT function is disabled, the A/D path from the external pin to the internal analog input will be switched off if the A/D input pin function is not selected by the corresponding selection bits. For the MCU with A/D converter channels, such as A/D AN15~AN0, the desired A/D channel can be switched on by properly configuring the external analog input channel selection bits in the A/D Control Register together with the corresponding analog input pin function is selected. However, the additional function of the READ PORT mode is to force the A/D path to be switched on. For example, when the AN0 is selected as the analog input channel as the READ PORT function is enabled, the AN0 analog input path will be switched on even if the AN0 analog input pin function is not selected. In this way, the AN0 analog input path can be examined by internally connecting the digital output on this shared pin with the AN0 analog input pin switch and then converting the corresponding digital data without any external analog input voltage connected.


Digital Output Function
AN0
AN15
A/D Converter
READ PORT function enabled, AN0 Pin-shared path switched on automatically
External analog input channel selection

A/D Channel Input Path Internal Connection

Programming Considerations

Within the user program, one of the things first to consider is port initialisation. After a reset, all of the I/O data and port control registers will be set to high. This means that all I/O pins will be defaulted to an input state, the level of which depends on the other connected circuitry and whether pull-high selections have been chosen. If the port control registers are then programmed to setup some pins as outputs, these output pins will have an initial high output value unless the associated port data registers are first programmed. Selecting which pins are inputs and which are outputs can be achieved byte-wide by loading the correct values into the appropriate port control register or by programming individual bits in the port control register using the "SET [m].i" and "CLR [m].i" instructions. Note that when using these bit control instructions, a read-modify-write operation takes place. The microcontroller must first read in the data on the entire port, modify it to the required new bit values and then rewrite this data back to the output ports.

Port A has the additional capability of providing wake-up functions. When the device is in the SLEEP or IDLE Mode, various methods are available to wake the device up. One of these is a high to low transition of any of the Port A pins. Single or multiple pins on Port A can be setup to have this function.

Timer Modules – TM

One of the most fundamental functions in any microcontroller devices is the ability to control and measure time. To implement time related functions the device includes several Timer Modules, generally abbreviated to the name TM. The TMs are multi-purpose timing units and serve to provide operations such as Timer/Counter, Input Capture, Compare Match Output and Single Pulse Output as well as being the functional unit for the generation of PWM signals. Each of the TMs has two interrupts. The addition of input and output pins for each TM ensures that users are provided with timing units with a wide and flexible range of features.

The common features of the different TM types are described here with more detailed information provided in the individual Standard and Periodic TM sections.

Introduction

The device contains several TMs and each individual TM can be categorised as a certain type, namely Standard Type TM or Periodic Type TM. Although similar in nature, the different TM types vary in their feature complexity. The common features to all of the Standard and Periodic type TMs will be described in this section and the detailed operation regarding each of the TM types will be described in separate sections. The main features and differences between the two types of TMs are summarised in the accompanying table.

TM Function	STM	PTM
Timer/Counter	√	√
Input Capture	√	√
Compare Match Output	√	√
PWM Output	√	√
Single Pulse Output	√	√
PWM Alignment	Edge	Edge
PWM Adjustment Period & Duty	Duty or Period	Duty or Period

TM Function Summary

TM Operation

The different types of TM offer a diverse range of functions, from simple timing operations to PWM signal generation. The key to understanding how the TM operates is to see it in terms of a free running count-up counter whose value is then compared with the value of pre-programmed internal comparators. When the free running count-up counter has the same value as the pre-programmed comparator, known as a compare match situation, a TM interrupt signal will be generated which can clear the counter and perhaps also change the condition of the TM output pin. The internal TM counter is driven by a user selectable clock source, which can be an internal clock or an external pin.

TM Clock Source

The clock source which drives the main counter in each TM can originate from various sources. The selection of the required clock source is implemented using the xTnCK2~xTnCK0 bits in the xTMn control registers, where “x” stands for S or P type TM and “n” stands for the specific TM serial number. The clock source can be a ratio of the system clock, f_{SYS}, or the internal high clock, f_H, the f_{SUB} clock source or the external xTCKn pin. The xTCKn pin clock source is used to allow an external signal to drive the TM as an external clock source for event counting.

TM Interrupts

The Standard or Periodic type TM has two internal interrupt, one for each of the internal comparator A or comparator P, which generate a TM interrupt when a compare match condition occurs. When a TM interrupt is generated, it can be used to clear the counter and also to change the state of the TM output pin.

TM External Pins

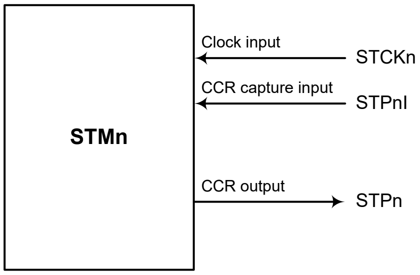
Each of the TMs, irrespective of what type, has two TM input pins, with the label xTCKn and xTPnI respectively. The xTMn input pin, xTCKn, is essentially a clock source for the xTMn and is selected using the xTnCK2~xTnCK0 bits in the xTMnC0 register. This external TM input pin allows an external clock source to drive the internal TM. The xTCKn input pin can be chosen to have either a rising or falling active edge. The STCKn and PTCKn pins are also used as the external trigger input pin in single pulse output mode for the STMn and PTMn respectively.

The other xTM input pin, STPnI or PTPnI, is the capture input whose active edge can be a rising edge, a falling edge or both rising and falling edges and the active edge transition type is selected using the STnIO1~STnIO0 or PTnIO1~PTnIO0 bits in the STMnC1 or PTMnC1 register respectively. There is another capture input, PTCKn, for PTMn capture input mode, which can be used as the external trigger input source except the PTPnI pin.

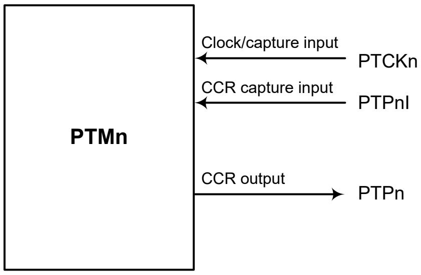
The TMs each have one output pin, xTPn. The TM output pin can be selected using the corresponding pin-shared function selection bits described in the Pin-shared Function section. When the TM is in the Compare Match Output Mode, these pins can be controlled by the TM to switch to a high or low level or to toggle when a compare match situation occurs. The external xTPn output pin is also the pin where the TM generates the PWM output waveform. As the TM output pins are pin-shared with other functions, the TM output function must first be setup using relevant pin-shared function selection register. The details of the pin-shared function selection are described in the pin-shared function section.

STM		PTM	
Input	Output	Input	Output
STCK0, STP0I STCK1, STP1I STCK2, STP2I	STP0 STP1 STP2	PTCK0, PTP0I PTCK1, PTP1I PTCK2, PTP2I PTCK3, PTP3I	PTP0 PTP1 PTP2 PTP3

TM External Pins



STM Function Pin Block Diagram (n = 0~2)

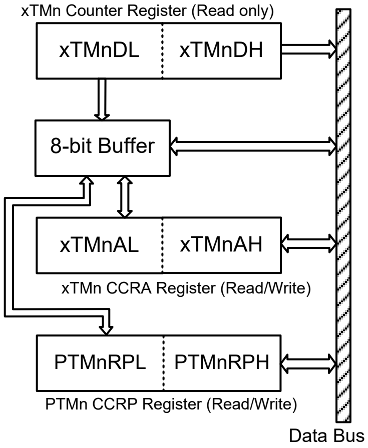


PTM Function Pin Block Diagram (n = 0~3)

Programming Considerations

The TM Counter Registers and the Capture/Compare CCRA and CCRP registers, all have a low and high byte structure. The high bytes can be directly accessed, but as the low bytes can only be accessed via an internal 8-bit buffer, reading or writing to these register pairs must be carried out in a specific way. The important point to note is that data transfer to and from the 8-bit buffer and its related low byte only takes place when a write or read operation to its corresponding high byte is executed.

As the CCRA and CCRP registers are implemented in the way shown in the following diagram and accessing these register pairs is carried out in a specific way as described above, it is recommended to use the "MOV" instruction to access the CCRA and CCRP low byte registers, named xTMnAL and PTMnRPL, using the following access procedures. Accessing the CCRA or CCRP low byte registers without following these access procedures will result in unpredictable values.


xTMn Counter Register (Read only)
xTMnDL
xTMnDH
8-bit Buffer
xTMnAL
xTMnAH
xTMn CCRA Register (Read/Write)
PTMnRPL
PTMnRPH
PTMn CCRP Register (Read/Write)
Data Bus

The following steps show the read and write procedures:

- Writing Data to CCRA or CCRP
  - ♦ Step 1. Write data to Low Byte xTMnAL or PTMnRPL
    – note that here data is only written to the 8-bit buffer.

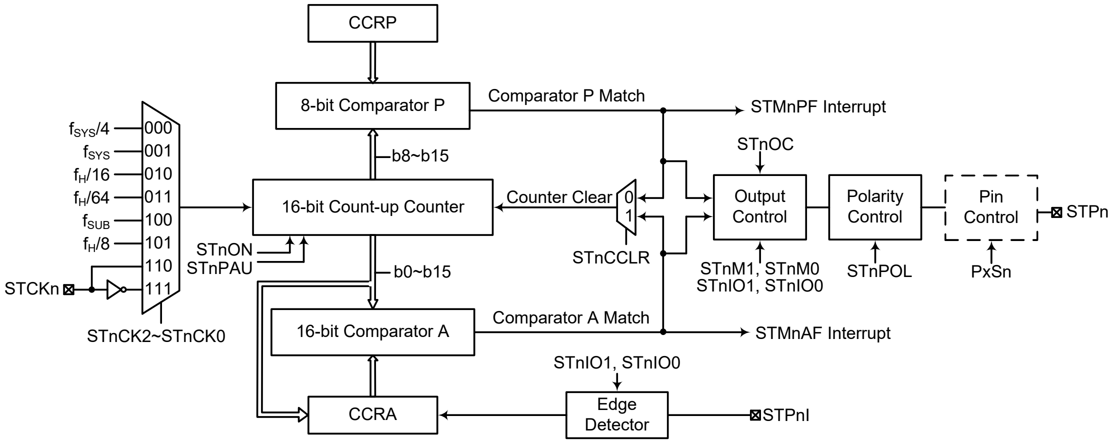
- Step 2. Write data to High Byte xTMnAH or PTMnRPH
  – here data is written directly to the high byte registers and simultaneously data is latched from the 8-bit buffer to the Low Byte registers.

- Reading Data from the Counter Registers and CCRA or CCRP
  - Step 1. Read data from the High Byte xTMnDH, xTMnAH or PTMnRPH
    – here data is read directly from the High Byte registers and simultaneously data is latched from the Low Byte register into the 8-bit buffer.
  - Step 2. Read data from the Low Byte xTMnDL, xTMnAL or PTMnRPL
    – this step reads data from the 8-bit buffer.

Standard Type TM – STM

The Standard Type TM contains five operating modes, which are Compare Match Output, Timer/Event Counter, Capture Input, Single Pulse Output and PWM Output modes. The Standard TM can also be controlled with two external input pins and can drive one external output pin.

STM Core	STM Input Pin	STM Output Pin
16-bit STM (STM0, STM1, STM2)	STCK0, STP0I STCK1, STP1I STCK2, STP2I	STP0 STP1 STP2



Standard Type TM Block Diagram (n = 0~2)

Standard TM Operation

The size of Standard TM is 16-bit wide and its core is a 16-bit count-up counter which is driven by a user selectable internal or external clock source. There are also two internal comparators with the names, Comparator A and Comparator P. These comparators will compare the value in the counter with CCRP and CCRA registers. The CCRP comparator is 8-bit wide whose value is compared the with highest 8 bits in the counter while the CCRA is the sixteen bits and therefore compares all counter bits.

The only way of changing the value of the 16-bit counter using the application program, is to clear the counter by changing the STnON bit from low to high. The counter will also be cleared automatically by a counter overflow or a compare match with one of its associated comparators. When these conditions occur, a STM interrupt signal will also usually be generated. The Standard Type TM can operate in a number of different operational modes, can be driven by different clock sources including an input pin and can also control an output pin. All operating setup conditions are selected using relevant internal registers.

Standard Type TM Register Description

Overall operation of the Standard TM is controlled using a series of registers. A read only register pair exists to store the internal counter 16-bit value, while a read/write register pair exists to store the internal 16-bit CCRA value. The STMnRP register is used to store the 8-bit CCRP value. The remaining two registers are control registers which setup the different operating and control modes.

Register Name	Bit							
	7	6	5	4	3	2	1	0
STMnC0	STnPAU	STnCK2	STnCK1	STnCK0	STnON	—	—	—
STMnC1	STnM1	STnM0	STnIO1	STnIO0	STnOC	STnPOL	STnDPX	STnCCLR
STMnDL	D7	D6	D5	D4	D3	D2	D1	D0
STMnDH	D15	D14	D13	D12	D11	D10	D9	D8
STMnAL	D7	D6	D5	D4	D3	D2	D1	D0
STMnAH	D15	D14	D13	D12	D11	D10	D9	D8
STMnRP	D7	D6	D5	D4	D3	D2	D1	D0

16-bit Standard TM Registers List (n = 0~2)

- **STMnDL Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 STMn Counter Low Byte Register bit 7 ~ bit 0
STMn 16-bit Counter bit 7 ~ bit 0

- **STMnDH Register**

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 STMn Counter High Byte Register bit 7 ~ bit 0
STMn 16-bit Counter bit 15 ~ bit 8

- **STMnAL Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 STMn CCRA Low Byte Register bit 7 ~ bit 0
STMn 16-bit CCRA bit 7 ~ bit 0

- **STMnAH Register**

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 STMn CCRA High Byte Register bit 7 ~ bit 0
STMn 16-bit CCRA bit 15 ~ bit 8

- **STMnRP Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: STMn CCRP 8-bit register, compared with the STMn counter bit 15~bit 8

Comparator P match period =
0: 65536 STMn clocks
1~255: (1~255) × 256 STMn clocks

These eight bits are used to setup the value on the internal CCRP 8-bit register, which are then compared with the internal counter's highest eight bits. The result of this comparison can be selected to clear the internal counter if the STnCCLR bit is set to zero. Setting the STnCCLR bit to zero ensures that a compare match with the CCRP values will reset the internal counter. As the CCRP bits are only compared with the highest eight counter bits, the compare values exist in 256 clock cycle multiples. Clearing all eight bits to zero is in effect allowing the counter to overflow at its maximum value.

- **STMnC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	STnPAU	STnCK2	STnCK1	STnCK0	STnON	—	—	—
R/W	R/W	R/W	R/W	R/W	R/W	—	—	—
POR	0	0	0	0	0	—	—	—

Bit 7 **STnPAU**: STMn Counter Pause control
0: Run
1: Pause

The counter can be paused by setting this bit high. Clearing the bit to zero restores normal counter operation. When in a Pause condition the STMn will remain powered up and continue to consume power. The counter will retain its residual value when this bit changes from low to high and resume counting from this value when the bit changes to a low value again.

Bit 6~4 **STnCK2~STnCK0**: Select STMn Counter clock
000: $f_{SYS}/4$
001: f_{SYS}
010: $f_H/16$
011: $f_H/64$
100: f_{SUB}
101: f_{SUB}
110: STCKn rising edge clock
111: STCKn falling edge clock

These three bits are used to select the clock source for the STMn. The external pin clock source can be chosen to be active on the rising or falling edge. The clock source f_{SYS} is the system clock, while f_H and f_{SUB} are other internal clocks, the details of which can be found in the oscillator section.

Bit 3 **STnON**: STMn Counter On/Off control
0: Off
1: On

This bit controls the overall on/off function of the STMn. Setting the bit high enables the counter to run while clearing the bit disables the STMn. Clearing this bit to zero will stop the counter from counting and turn off the STMn which will reduce its power consumption. When the bit changes state from low to high the internal counter value will be reset to zero, however when the bit changes from high to low, the internal counter will retain its residual value until the bit returns high again. If the STMn is in the Compare Match Output Mode then the STMn output pin will be reset to its initial condition, as specified by the STnOC bit, when the STnON bit changes from low to high.

Bit 2~0 Unimplemented, read as "0"

- **STMnC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	STnM1	STnM0	STnIO1	STnIO0	STnOC	STnPOL	STnDPX	STnCCLR
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **STnM1~STnM0**: Select STMn Operating Mode
00: Compare Match Output Mode
01: Capture Input Mode
10: PWM Output Mode or Single Pulse Output Mode
11: Timer/Counter Mode

These bits setup the required operating mode for the STMn. To ensure reliable operation the STMn should be switched off before any changes are made to the STnM1 and STnM0 bits. In the Timer/Counter Mode, the STMn output pin state is undefined.

Bit 5~4 **STnIO1~STnIO0**: Select STMn external pin STPn function

Compare Match Output Mode
00: No change
01: Output low
10: Output high
11: Toggle output

PWM Output Mode/Single Pulse Output Mode
00: PWM output inactive state
01: PWM output active state
10: PWM output
11: Single Pulse Output

Capture Input Mode
00: Input capture at rising edge of STPnI
01: Input capture at falling edge of STPnI
10: Input capture at rising/falling edge of STPnI
11: Input capture disabled

Timer/Counter Mode
Unused

These two bits are used to determine how the STMn output pin changes state when a certain condition is reached. The function that these bits select depends upon in which mode the STMn is running.

In the Compare Match Output Mode, the STnIO1 and STnIO0 bits determine how the STMn output pin changes state when a compare match occurs from the Comparator A. The TM output pin can be setup to switch high, switch low or to toggle its present state when a compare match occurs from the Comparator A. When the bits are both zero, then no change will take place on the output. The initial value of the STMn output pin should be setup using the STnOC bit in the STMnC1 register. Note that the output level requested by the STnIO1 and STnIO0 bits must be different from the initial value setup using the STnOC bit otherwise no change will occur on the STMn output pin when a compare match occurs. After the STMn output pin changes state, it can be reset to its initial level by changing the level of the STnON bit from low to high.

In the PWM Output Mode, the STnIO1 and STnIO0 bits determine how the STMn output pin changes state when a certain compare match condition occurs. The PWM output function is modified by changing these two bits. It is necessary to only change the values of the STnIO1 and STnIO0 bits only after the STMn has been switched off. Unpredictable PWM outputs will occur if the STnIO1 and STnIO0 bits are changed when the STMn is running.

Bit 3 **STnOC**: STMn STPn Output control

Compare Match Output Mode
0: Initial low
1: Initial high

PWM Output Mode/Single Pulse Output Mode
0: Active low
1: Active high

This is the output control bit for the STMn output pin. Its operation depends upon whether STMn is being used in the Compare Match Output Mode or in the PWM Output Mode/Single Pulse Output Mode. It has no effect if the STMn is in the Timer/Counter Mode. In the Compare Match Output Mode it determines the logic level of the STMn output pin before a compare match occurs. In the PWM output Mode it determines if the PWM signal is active high or active low. In the Single Pulse Output Mode it determines the logic level of the STMn output pin when the STnON bit changes from low to high.

Bit 2 **STnPOL**: STMn STPn Output polarity control
0: Non-invert
1: Invert

This bit controls the polarity of the STPn output pin. When the bit is set high the STMn output pin will be inverted and not inverted when the bit is zero. It has no effect if the STMn is in the Timer/Counter Mode.

Bit 1 **STnDPX**: STMn PWM duty/period control
0: CCRP – period; CCRA – duty
1: CCRP – duty; CCRA – period

This bit determines which of the CCRA and CCRP registers are used for period and duty control of the PWM waveform.

Bit 0 **STnCCLR**: STMn Counter Clear condition selection
0: Comparator P match
1: Comparator A match

This bit is used to select the method which clears the counter. Remember that the Standard TM contains two comparators, Comparator A and Comparator P, either of which can be selected to clear the internal counter. With the STnCCLR bit set high, the counter will be cleared when a compare match occurs from the Comparator A. When the bit is low, the counter will be cleared when a compare match occurs from the Comparator P or with a counter overflow. A counter overflow clearing method can only be implemented if the CCRP bits are all cleared to zero. The STnCCLR bit is not used in the PWM Output, Single Pulse Output or Capture Input Mode.

Standard Type TM Operation Modes

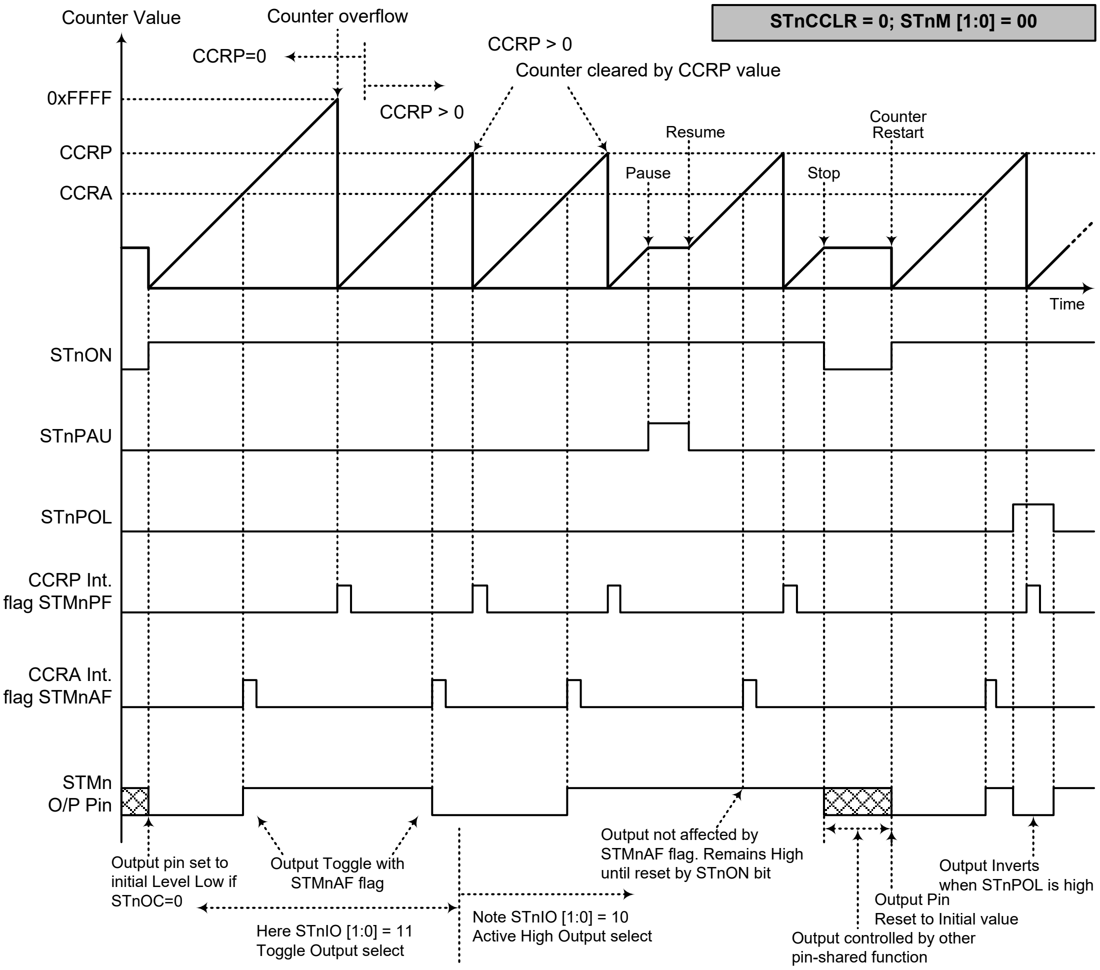
The Standard Type TM can operate in one of five operating modes, Compare Match Output Mode, PWM Output Mode, Single Pulse Output Mode, Capture Input Mode or Timer/Counter Mode. The operating mode is selected using the STnM1 and STnM0 bits in the STMnC1 register.

Compare Match Output Mode

To select this mode, bits STnM1 and STnM0 in the STMnC1 register, should be set to 00 respectively. In this mode once the counter is enabled and running it can be cleared by three methods. These are a counter overflow, a compare match from Comparator A and a compare match from Comparator P. When the STnCCLR bit is low, there are two ways in which the counter can be cleared. One is when a compare match from Comparator P, the other is when the CCRP bits are all zero which allows the counter to overflow. Here both STMnAF and STMnPF interrupt request flags for Comparator A and Comparator P respectively, will both be generated.

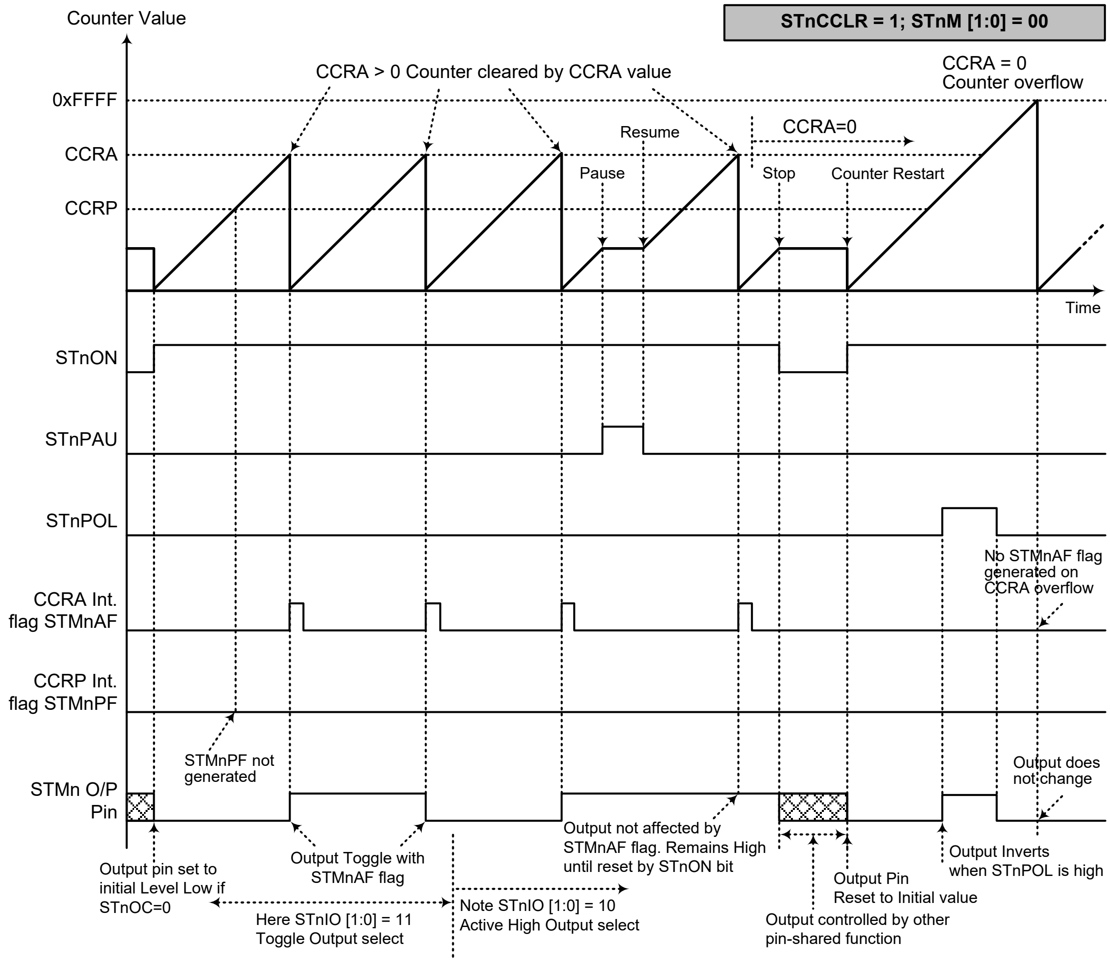
If the STnCCLR bit in the STMnC1 register is high then the counter will be cleared when a compare match occurs from Comparator A. However, here only the STMnAF interrupt request flag will be generated even if the value of the CCRP bits is less than that of the CCRA registers. Therefore when STnCCLR is high no STMnPF interrupt request flag will be generated. In the Compare Match Output Mode, the CCRA can not be set to "0".

As the name of the mode suggests, after a comparison is made, the STMn output pin, will change state. The STMn output pin condition however only changes state when a STMnAF interrupt request flag is generated after a compare match occurs from Comparator A. The STMnPF interrupt request flag, generated from a compare match occurs from Comparator P, will have no effect on the STMn output pin. The way in which the STMn output pin changes state are determined by the condition of the STnIO1 and STnIO0 bits in the STMnC1 register. The STMn output pin can be selected using the STnIO1 and STnIO0 bits to go high, to go low or to toggle from its present condition when a compare match occurs from Comparator A. The initial condition of the STMn output pin, which is setup after the STnON bit changes from low to high, is setup using the STnOC bit. Note that if the STnIO1 and STnIO0 bits are zero then no pin change will take place.


STnCCLR = 0; STnM [1:0] = 00
Counter Value
Counter overflow
CCRP=0
CCRP > 0
CCRP > 0
Counter cleared by CCRP value
0xFFFF
CCRP
CCRA
Resume
Counter Restart
Pause
Stop
Time
STnON
STnPAU
STnPOL
CCRP Int. flag STMnPF
CCRA Int. flag STMnAF
STMn O/P Pin
Output pin set to initial Level Low if STnOC=0
Output Toggle with STMnAF flag
Here STnIO [1:0] = 11 Toggle Output select
Note STnIO [1:0] = 10 Active High Output select
Output not affected by STMnAF flag. Remains High until reset by STnON bit
Output controlled by other pin-shared function
Output Pin Reset to Initial value
Output Inverts when STnPOL is high

Compare Match Output Mode – STnCCLR = 0

Note: 1. With STnCCLR=0 a Comparator P match will clear the counter
2. The STMn output pin is controlled only by the STMnAF flag
3. The output pin is reset to its initial state by a STnON bit rising edge
4. n=0~2


STnCCLR = 1; STnM [1:0] = 00
Counter Value
0xFFFF
CCRA
CCRP
CCRA > 0 Counter cleared by CCRA value
Resume
Pause
CCRA=0
Stop
Counter Restart
CCRA = 0
Counter overflow
Time
STnON
STnPAU
STnPOL
CCRA Int. flag STMnAF
No STMnAF flag generated on CCRA overflow
CCRP Int. flag STMnPF
STMnPF not generated
STMn O/P Pin
Output does not change
Output not affected by STMnAF flag. Remains High until reset by STnON bit
Output pin set to initial Level Low if STnOC=0
Output Toggle with STMnAF flag
Here STnIO [1:0] = 11 Toggle Output select
Note STnIO [1:0] = 10 Active High Output select
Output Pin Reset to Initial value
Output controlled by other pin-shared function
Output Inverts when STnPOL is high

Compare Match Output Mode – STnCCLR = 1

Note: 1. With STnCCLR=1 a Comparator A match will clear the counter
2. The STMn output pin is controlled only by the STMnAF flag
3. The output pin is reset to its initial state by a STnON bit rising edge
4. A STMnPF flag is not generated when STnCCLR=1
5. n=0~2

Timer/Counter Mode

To select this mode, bits STnM1 and STnM0 in the STMnC1 register should be set to 11 respectively. The Timer/Counter Mode operates in an identical way to the Compare Match Output Mode generating the same interrupt flags. The exception is that in the Timer/Counter Mode the STMn output pin is not used. Therefore the above description and Timing Diagrams for the Compare Match Output Mode can be used to understand its function. As the STMn output pin is not used in this mode, the pin can be used as a normal I/O pin or other pin-shared function.

PWM Output Mode

To select this mode, bits STnM1 and STnM0 in the STMnC1 register should be set to 10 respectively and also the STnIO1 and STnIO0 bits should be set to 10 respectively. The PWM function within the STMn is useful for applications which require functions such as motor control, heating control, illumination control etc. By providing a signal of fixed frequency but of varying duty cycle on the STMn output pin, a square wave AC waveform can be generated with varying equivalent DC RMS values.

As both the period and duty cycle of the PWM waveform can be controlled, the choice of generated waveform is extremely flexible. In the PWM Output Mode, the STnCCLR bit has no effect as the PWM period. Both of the CCRA and CCRP registers are used to generate the PWM waveform, one register is used to clear the internal counter and thus control the PWM waveform frequency, while the other one is used to control the duty cycle. Which register is used to control either frequency or duty cycle is determined using the STnDPX bit in the STMnC1 register. The PWM waveform frequency and duty cycle can therefore be controlled by the values in the CCRA and CCRP registers.

An interrupt flag, one for each of the CCRA and CCRP, will be generated when a compare match occurs from either Comparator A or Comparator P. The STnOC bit in the STMnC1 register is used to select the required polarity of the PWM waveform while the two STnIO1 and STnIO0 bits are used to enable the PWM output or to force the STMn output pin to a fixed high or low level. The STnPOL bit is used to reverse the polarity of the PWM output waveform.

- **16-bit STMn, PWM Output Mode, Edge-aligned Mode, STnDPX=0**

CCRP	1~255	0
Period	CCRP × 256	65536
Duty	CCRA	

If f_{SYS}=16MHz, STMn clock source is f_{SYS}/4, CCRP=2 and CCRA=128,

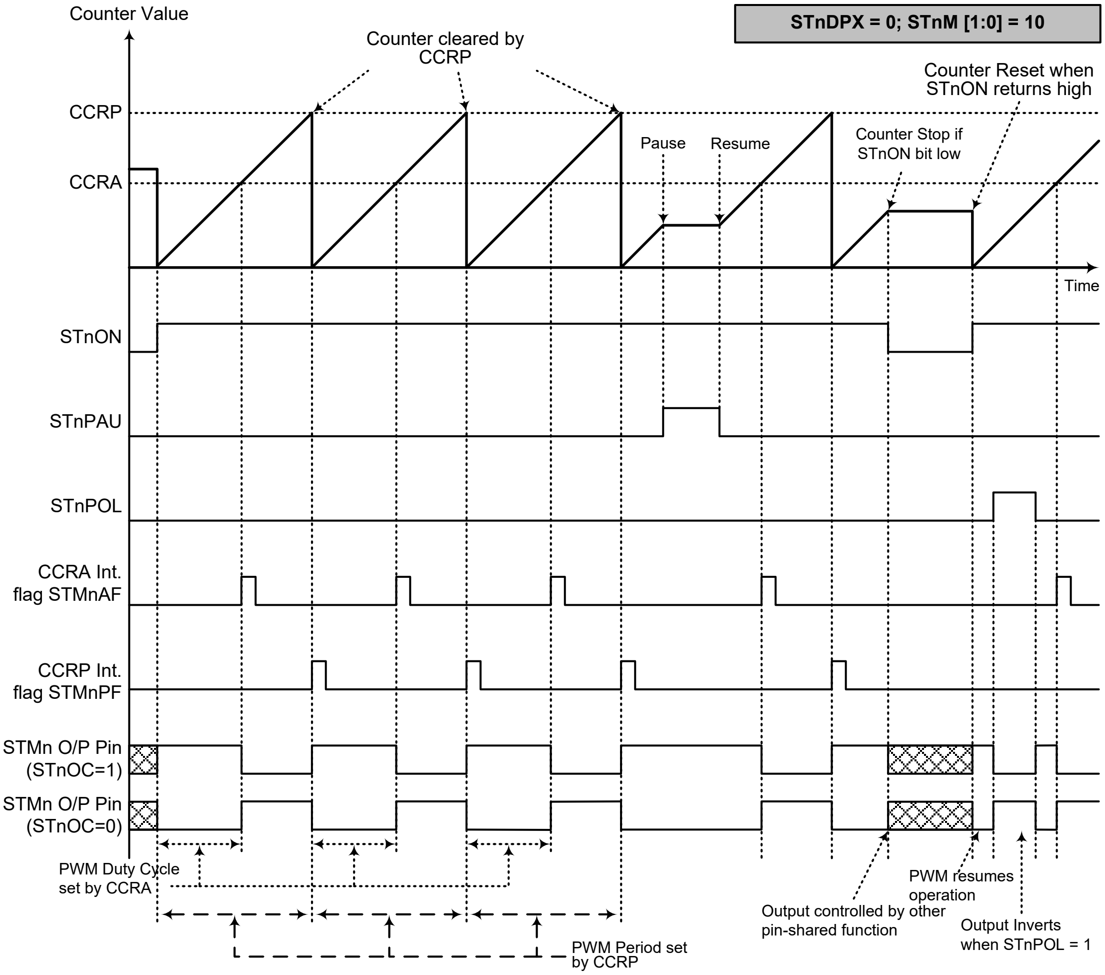
The STMn PWM output frequency=(f_{SYS}/4)/(2×256)=f_{SYS}/2048=8kHz, duty=128/(2×256)=25%.

If the Duty value defined by the CCRA register is equal to or greater than the Period value, then the PWM output duty is 100%.

- **16-bit STMn, PWM Output Mode, Edge-aligned Mode, STnDPX=1**

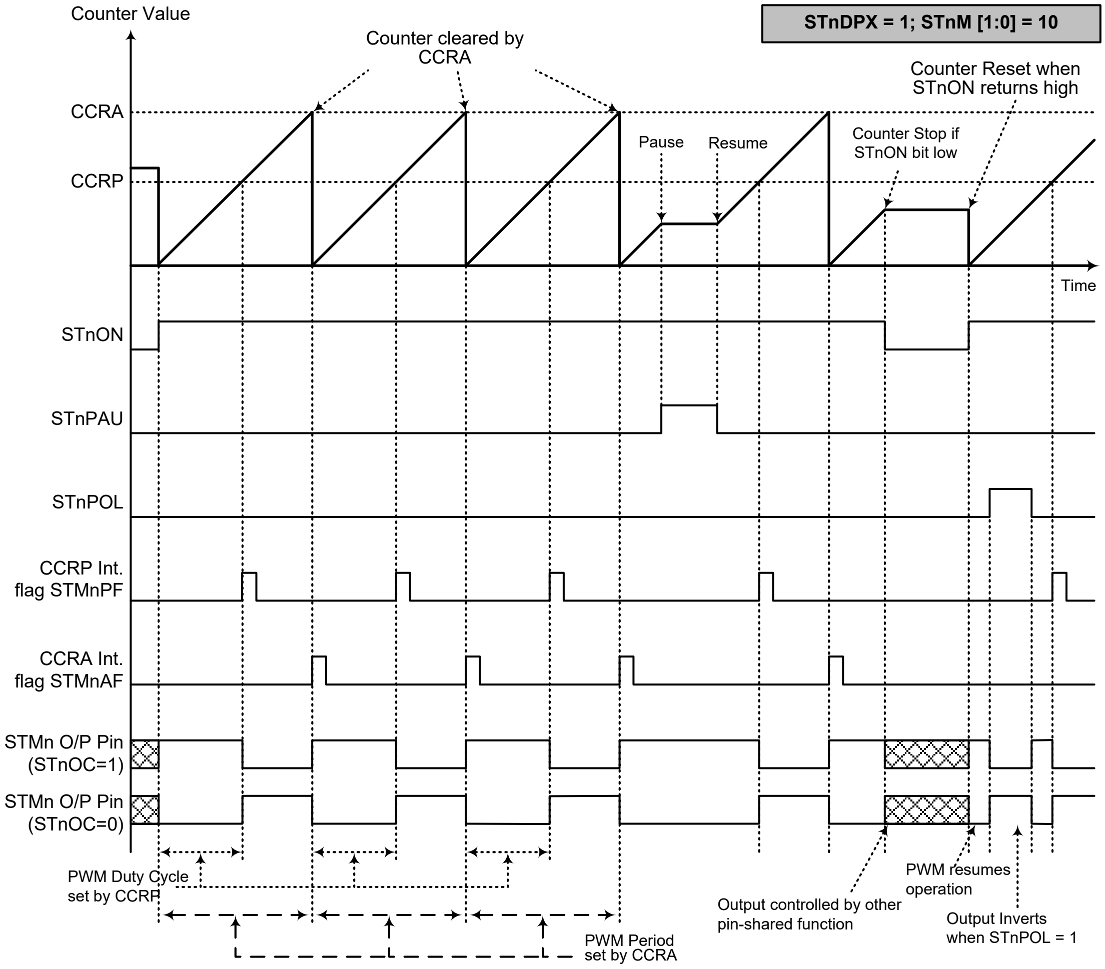
CCRP	1~255	0
Period	CCRA	
Duty	CCRP×256	65536

The PWM output period is determined by the CCRA register value together with the TM clock while the PWM duty cycle is defined by the CCRP register value except when the CCRP value is equal to 0.



PWM Output Mode – STnDPX = 0

Note: 1. Here STnDPX=0 – Counter cleared by CCRP
2. A counter clear sets the PWM Period
3. The internal PWM function continues running even when STnIO [1:0] = 00 or 01
4. The STnCCLR bit has no influence on PWM operation
5. n=0~2



PWM Output Mode – STnDPX = 1

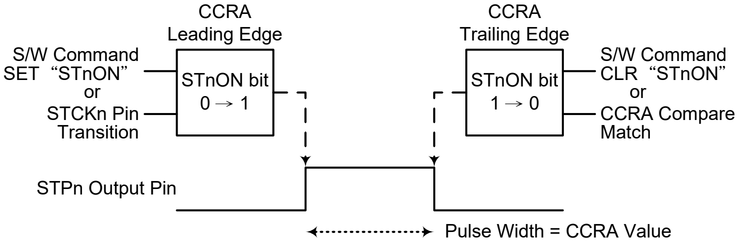
Note: 1. Here STnDPX=1 – Counter cleared by CCRA
2. A counter clear sets the PWM Period
3. The internal PWM function continues even when STnIO [1:0] = 00 or 01
4. The STnCCLR bit has no influence on PWM operation
5. n=0~2

Single Pulse Output Mode

To select this mode, bits STnM1 and STnM0 in the STMnC1 register should be set to 10 respectively and also the STnIO1 and STnIO0 bits should be set to 11 respectively. The Single Pulse Output Mode, as the name suggests, will generate a single shot pulse on the STMn output pin.

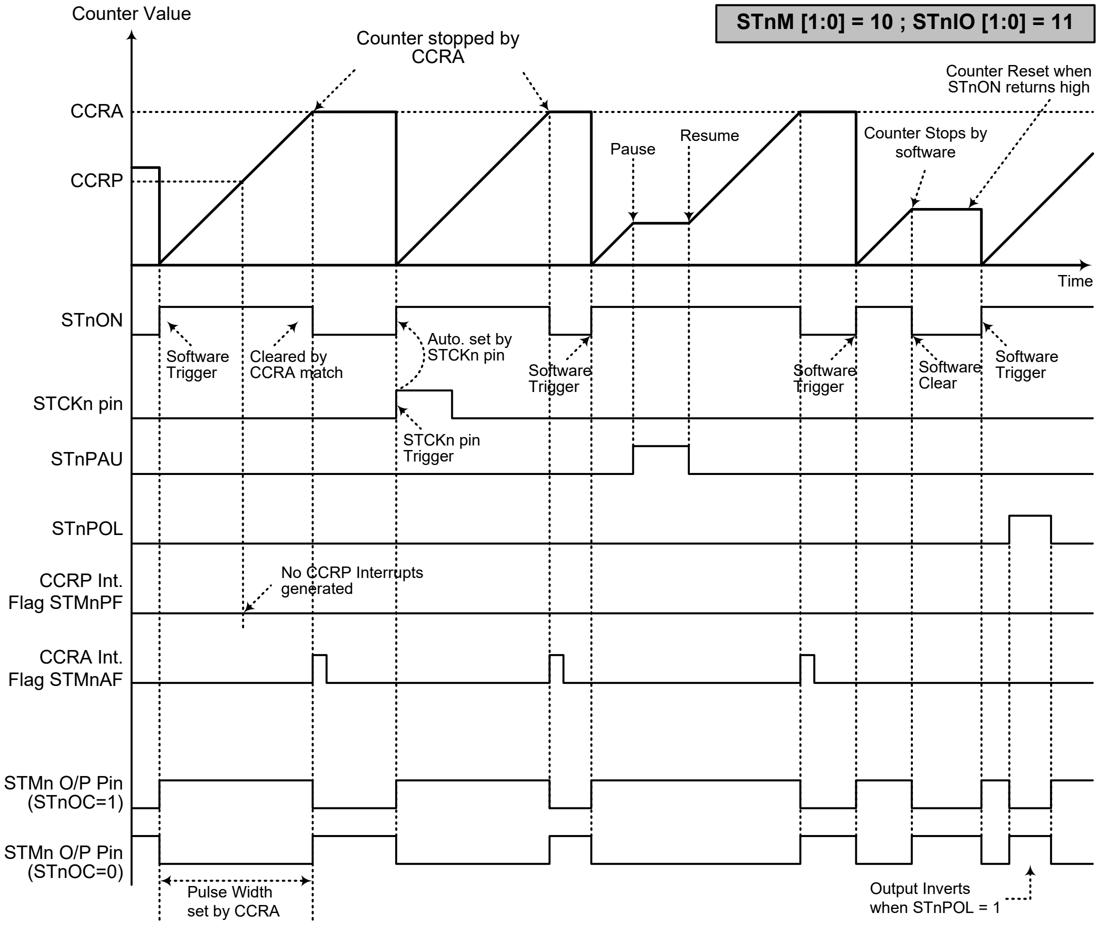
The trigger for the pulse output leading edge is a low to high transition of the STnON bit, which can be implemented using the application program. However in the Single Pulse Output Mode, the STnON bit can also be made to automatically change from low to high using the external STCKn pin, which will in turn initiate the Single Pulse output. When the STnON bit transitions to a high level, the counter will start running and the pulse leading edge will be generated. The STnON bit should remain high when the pulse is in its active state. The generated pulse trailing edge will be generated when the STnON bit is cleared to zero, which can be implemented using the application program or when a compare match occurs from Comparator A.

However a compare match from Comparator A will also automatically clear the STnON bit and thus generate the Single Pulse output trailing edge. In this way the CCRA value can be used to control the pulse width. A compare match from Comparator A will also generate a STMn interrupt. The counter can only be reset back to zero when the STnON bit changes from low to high when the counter restarts. In the Single Pulse Output Mode CCRP is not used. The STnCCLR and STnDPX bits are not used in this Mode.



Single Pulse Generation

Note: It is recommended that before the STMn operates in the single pulse output mode, the STnON bit should be cleared to zero and the STCKn pin state should be low to avoid unexpected trigger.



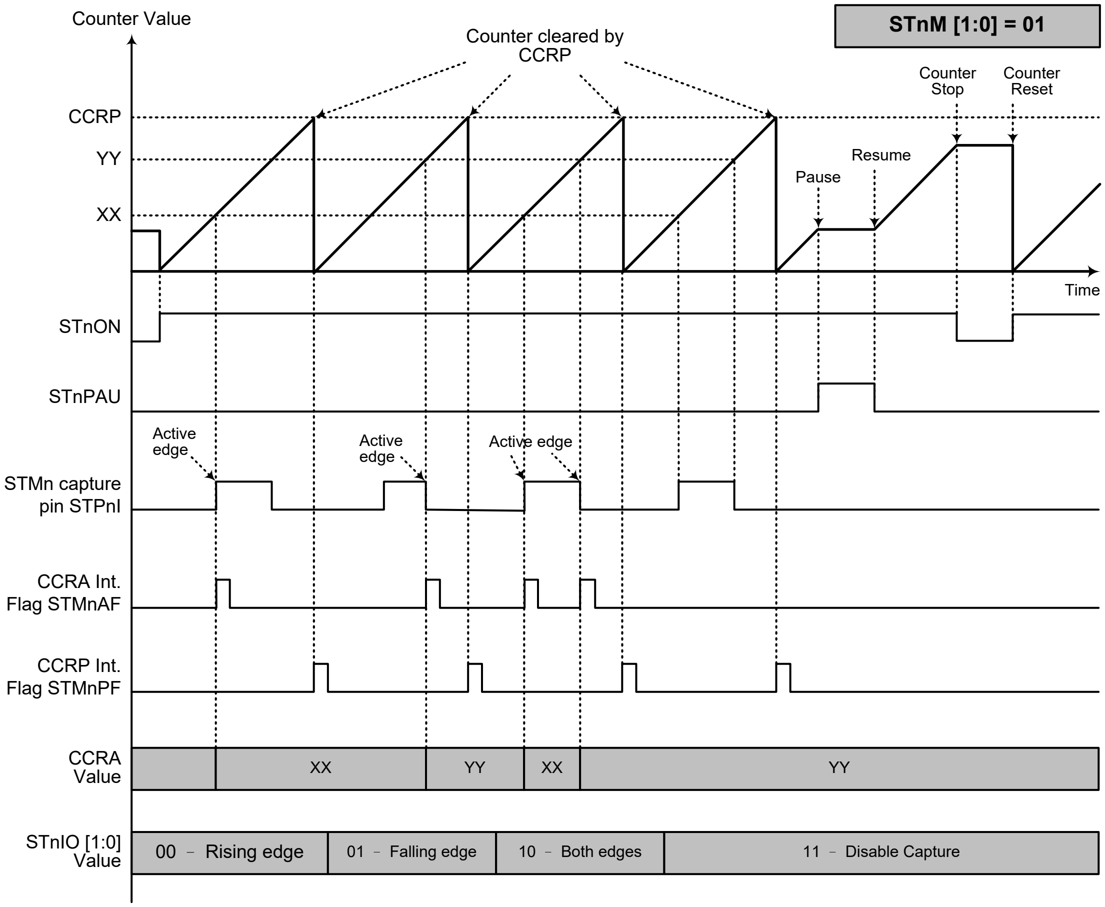
Single Pulse Output Mode

Note: 1. Counter stopped by CCRA
2. CCRP is not used
3. The pulse triggered by the STCKn pin or by setting the STnON bit high
4. A STCKn pin active edge will automatically set the STnON bit high
5. In the Single Pulse Output Mode, STnIO [1:0] must be set to “11” and can not be changed
6. n=0~2

Capture Input Mode

To select this mode bits STnM1 and STnM0 in the STMnC1 register should be set to 01 respectively. This mode enables external signals to capture and store the present value of the internal counter and can therefore be used for applications such as pulse width measurements. The external signal is supplied on the STPnI pin, whose active edge can be a rising edge, a falling edge or both rising and falling edges; the active edge transition type is selected using the STnIO1 and STnIO0 bits in the STMnC1 register. The counter is started when the STnON bit changes from low to high which is initiated using the application program.

When the required edge transition appears on the STPnI pin the present value in the counter will be latched into the CCRA registers and a STMn interrupt generated. Irrespective of what events occur on the STPnI pin the counter will continue to free run until the STnON bit changes from high to low. When a CCRP compare match occurs the counter will reset back to zero; in this way the CCRP value can be used to control the maximum counter value. When a CCRP compare match occurs from Comparator P, a STMn interrupt will also be generated. Counting the number of overflow interrupt signals from the CCRP can be a useful method in measuring long pulse widths. The STnIO1 and STnIO0 bits can select the active trigger edge on the STPnI pin to be a rising edge, falling edge or both edge types. If the STnIO1 and STnIO0 bits are both set high, then no capture operation will take place irrespective of what happens on the STPnI pin, however it must be noted that the counter will continue to run. The STnCCLR and STnDPX bits are not used in this Mode.



Capture Input Mode

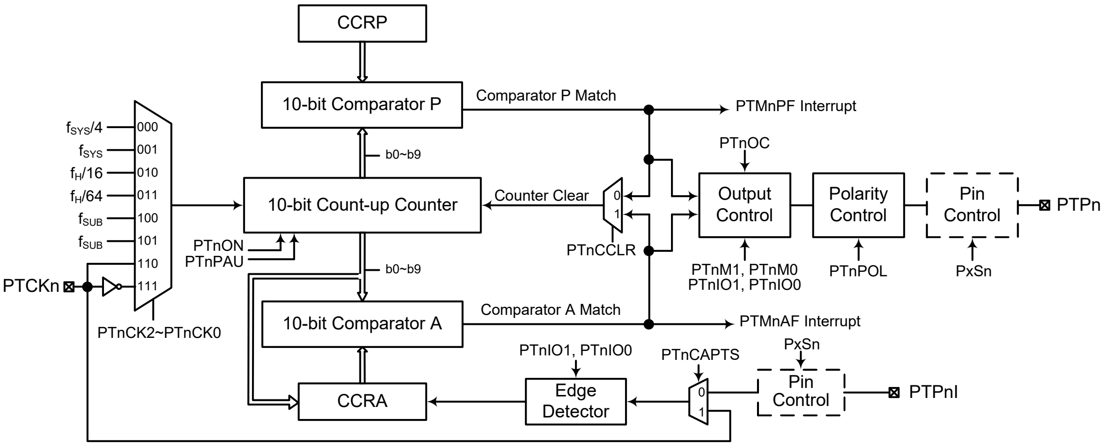
Note: 1. STnM [1:0] = 01 and active edge set by the STnIO [1:0] bits

2. A STMn Capture input pin active edge transfers the counter value to CCRA
3. STnCCLR bit not used
4. No output function – STnOC and STnPOL bits are not used
5. CCRP determines the counter value and the counter has a maximum count value when CCRP is equal to zero.
6. n=0~2

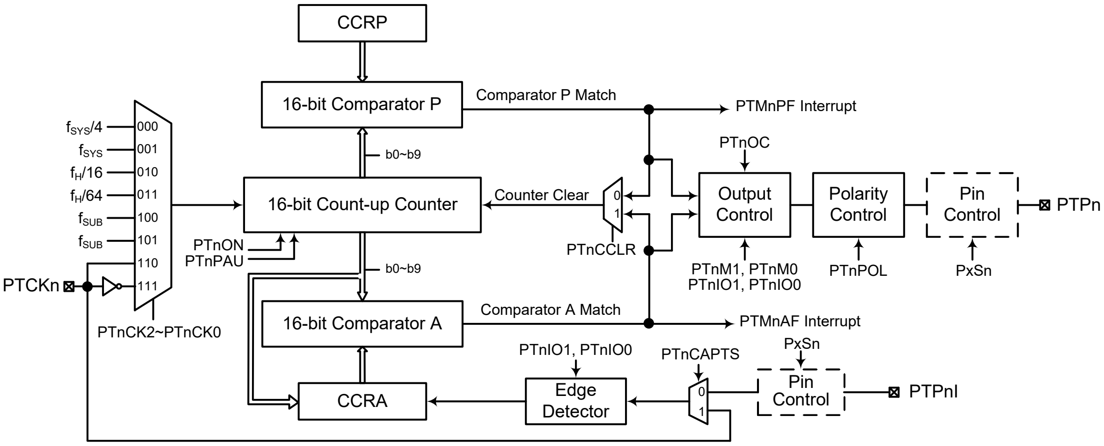
Periodic Type TM – PTM

The Periodic Type TM contains five operating modes, which are Compare Match Output, Timer/Event Counter, Capture Input, Single Pulse Output and PWM Output modes. The Periodic TM can also be controlled with two external input pins and can drive one external output pin.

PTM Core	PTM Input Pin	PTM Output Pin
10-bit PTM (PTM0, PTM1)	PTCK0, PTP0I PTCK1, PTP1I	PTP0 PTP1
16-bit PTM (PTM2, PTM3)	PTCK2, PTP2I PTCK3, PTP3I	PTP2 PTP3



10-bit Periodic Type TM Block Diagram (n = 0~1)



16-bit Periodic Type TM Block Diagram (n = 2~3)

Periodic TM Operation

The size of Periodic TM is 10-/16-bit wide and its core is a 10-/16-bit count-up counter which is driven by a user selectable internal or external clock source. There are also two internal comparators with the names, Comparator A and Comparator P. These comparators will compare the value in the counter with CCRP and CCRA registers. The CCRP and CCRA comparators are 10-/16-bit wide whose value is respectively compared with all counter bits.

The only way of changing the value of the 10-/16-bit counter using the application program is to clear the counter by changing the PTnON bit from low to high. The counter will also be cleared automatically by a counter overflow or a compare match with one of its associated comparators. When these conditions occur, a PTM interrupt signal will also usually be generated. The Periodic Type TM can operate in a number of different operational modes, can be driven by different clock sources including an input pin and can also control the output pins. All operating setup conditions are selected using relevant internal registers.

Periodic Type TM Register Description

Overall operation of the Periodic TM is controlled using a series of registers. A read only register pair exists to store the internal counter 10-/16-bit value, while two read/write register pairs exist to store the internal 10-/16-bit CCRA and CCRP value. The remaining two registers are control registers which setup the different operating and control modes.

Register Name	Bit							
	7	6	5	4	3	2	1	0
PTMnC0	PTnPAU	PTnCK2	PTnCK1	PTnCK0	PTnON	—	—	—
PTMnC1	PTnM1	PTnM0	PTnIO1	PTnIO0	PTnOC	PTnPOL	PTnCAPTS	PTnCCLR
PTMnDL	D7	D6	D5	D4	D3	D2	D1	D0
PTMnDH	—	—	—	—	—	—	D9	D8
PTMnAL	D7	D6	D5	D4	D3	D2	D1	D0
PTMnAH	—	—	—	—	—	—	D9	D8
PTMnRPL	D7	D6	D5	D4	D3	D2	D1	D0
PTMnRPH	—	—	—	—	—	—	D9	D8

10-bit Periodic TM Registers List (n = 0~1)

Register Name	Bit							
	7	6	5	4	3	2	1	0
PTMnC0	PTnPAU	PTnCK2	PTnCK1	PTnCK0	PTnON	—	—	—
PTMnC1	PTnM1	PTnM0	PTnIO1	PTnIO0	PTnOC	PTnPOL	PTnCAPTS	PTnCCLR
PTMnDL	D7	D6	D5	D4	D3	D2	D1	D0
PTMnDH	D15	D14	D13	D12	D11	D10	D9	D8
PTMnAL	D7	D6	D5	D4	D3	D2	D1	D0
PTMnAH	D15	D14	D13	D12	D11	D10	D9	D8
PTMnRPL	D7	D6	D5	D4	D3	D2	D1	D0
PTMnRPH	D15	D14	D13	D12	D11	D10	D9	D8

16-bit Periodic TM Registers List (n = 2~3)

- **PTMnDL Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 PTMn Counter Low Byte Register bit 7 ~ bit 0
PTMn 10-/16-bit Counter bit 7 ~ bit 0

- **PTMnDH Register (n=0~1)**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	D9	D8
R/W	—	—	—	—	—	—	R	R
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as “0”

Bit 1~0 PTMn Counter High Byte Register bit 1 ~ bit 0
PTMn 10-bit Counter bit 9 ~ bit 8

- **PTMnDH Register (n=2~3)**

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 PTMn Counter High Byte Register bit 7 ~ bit 0
PTMn 16-bit Counter bit 15 ~ bit 8

- **PTMnAL Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 PTMn CCRA Low Byte Register bit 7 ~ bit 0
PTMn 10-/16-bit CCRA bit 7 ~ bit 0

- **PTMnAH Register (n=0~1)**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	D9	D8
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as “0”

Bit 1~0 PTMn CCRA High Byte Register bit 1 ~ bit 0
PTMn 10-bit CCRA bit 9 ~ bit 8

- **PTMnAH Register (n=2~3)**

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 PTMn CCRA High Byte Register bit 7 ~ bit 0
PTMn 16-bit CCRA bit 15 ~ bit 8

- **PTMnRPL Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: PTMn CCRP Low Byte Register bit 7 ~ bit 0
PTMn 10-/16-bit CCRP bit 7 ~ bit 0

- **PTMnRPH Register (n = 0~1)**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	D9	D8
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as "0"

Bit 1~0 **D9~D8**: PTMn CCRP High Byte Register bit 1 ~ bit 0
PTMn 10-bit CCRP bit 9 ~ bit 8

- **PTMnRPH Register (n = 2~3)**

Bit	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D15~D8**: PTMn CCRP High Byte Register bit 7 ~ bit 0
PTMn 16-bit CCRP bit 15 ~ bit 8

- **PTMnC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	PTnPAU	PTnCK2	PTnCK1	PTnCK0	PTnON	—	—	—
R/W	R/W	R/W	R/W	R/W	R/W	—	—	—
POR	0	0	0	0	0	—	—	—

Bit 7 **PTnPAU**: PTMn Counter Pause control
0: Run
1: Pause

The counter can be paused by setting this bit high. Clearing the bit to zero restores normal counter operation. When in a Pause condition the PTMn will remain powered up and continue to consume power. The counter will retain its residual value when this bit changes from low to high and resume counting from this value when the bit changes to a low value again.

Bit 6~4 **PTnCK2~PTnCK0**: Select PTMn Counter clock
000: $f_{SYS}/4$
001: f_{SYS}
010: $f_H/16$
011: $f_H/64$
100: f_{SUB}
101: f_{SUB}
110: PTCKn rising edge clock
111: PTCKn falling edge clock

These three bits are used to select the clock source for the PTMn. The external pin clock source can be chosen to be active on the rising or falling edge. The clock source f_{SYS} is the system clock, while f_H and f_{SUB} are other internal clocks, the details of which can be found in the oscillator section.

Bit 3 **PTnON**: PTMn Counter On/Off control
0: Off
1: On

This bit controls the overall on/off function of the PTMn. Setting the bit high enables the counter to run while clearing the bit disables the PTMn. Clearing this bit to zero will stop the counter from counting and turn off the PTMn which will reduce its power consumption. When the bit changes state from low to high the internal counter value will be reset to zero, however when the bit changes from high to low, the internal counter will retain its residual value until the bit returns high again. If the PTMn is in the Compare Match Output Mode or the PWM Output Mode or Single Pulse Output Mode then the PTMn output pin will be reset to its initial condition, as specified by the PTnOC bit, when the PTnON bit changes from low to high.

Bit 2~0 Unimplemented, read as "0"

- **PTMnC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	PTnM1	PTnM0	PTnIO1	PTnIO0	PTnOC	PTnPOL	PTnCAPTS	PTnCCLR
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **PTnM1~PTnM0**: Select PTMn Operating Mode
00: Compare Match Output Mode
01: Capture Input Mode
10: PWM Output Mode or Single Pulse Output Mode
11: Timer/Counter Mode

These bits setup the required operating mode for the PTMn. To ensure reliable operation the PTMn should be switched off before any changes are made to the PTnM1 and PTnM0 bits. In the Timer/Counter Mode, the PTMn output pin state is undefined.

Bit 5~4 **PTnIO1~PTnIO0**: Select PTMn external pin PTPn or PTPnI function

Compare Match Output Mode
00: No change
01: Output low
10: Output high
11: Toggle output

PWM Output Mode/Single Pulse Output Mode
00: PWM output inactive state
01: PWM output active state
10: PWM output
11: Single Pulse Output

Capture Input Mode
00: Input capture at rising edge of PTPnI or PTCKn
01: Input capture at falling edge of PTPnI or PTCKn
10: Input capture at rising/falling edge of PTPnI or PTCKn
11: Input capture disabled

Timer/Counter Mode
Unused

These two bits are used to determine how the PTMn output pin changes state when a certain condition is reached. The function that these bits select depends upon in which mode the PTMn is running.

In the Compare Match Output Mode, the PTnIO1 and PTnIO0 bits determine how the PTMn output pin changes state when a compare match occurs from the Comparator A. The PTMn output pin can be setup to switch high, switch low or to toggle its present state when a compare match occurs from the Comparator A. When the bits are both zero, then no change will take place on the output. The initial value of the PTMn output pin should be setup using the PTnOC bit in the PTMnC1 register. Note that the output level requested by the PTnIO1 and PTnIO0 bits must be different from the initial value setup using the PTnOC bit otherwise no change will occur on the PTMn output pin when a compare match occurs. After the PTMn output pin changes state, it can be reset to its initial level by changing the level of the PTnON bit from low to high.

In the PWM Output Mode, the PTnIO1 and PTnIO0 bits determine how the TM output pin changes state when a certain compare match condition occurs. The PTMn output function is modified by changing these two bits. It is necessary to only change the values of the PTnIO1 and PTnIO0 bits only after the PTMn has been switched off. Unpredictable PWM outputs will occur if the PTnIO1 and PTnIO0 bits are changed when the PTMn is running.

Bit 3 **PTnOC**: PTMn PTPn Output control

Compare Match Output Mode
0: Initial low
1: Initial high

PWM Output Mode/Single Pulse Output Mode
0: Active low
1: Active high

This is the output control bit for the PTMn output pin. Its operation depends upon whether PTMn is being used in the Compare Match Output Mode or in the PWM Output Mode/Single Pulse Output Mode. It has no effect if the PTMn is in the Timer/Counter Mode. In the PWM Output Mode it determines if the PWM signal is active high or active low. In the Single Pulse Output Mode it determines the logic level of the PTM output pin when the PTnON bit changes from low to high.

Bit 2 **PTnPOL**: PTMn PTPn Output polarity control
0: Non-invert
1: Invert

This bit controls the polarity of the PTPn output pin. When the bit is set high the PTMn output pin will be inverted and not inverted when the bit is zero. It has no effect if the PTMn is in the Timer/Counter Mode.

Bit 1 **PTnCAPTS**: PTMn Capture Trigger Source selection
0: From PTPnI pin
1: From PTCKn pin

Bit 0 **PTnCCLR**: PTMn Counter Clear condition selection
0: Comparator P match
1: Comparator A match

This bit is used to select the method which clears the counter. Remember that the Periodic TM contains two comparators, Comparator A and Comparator P, either of which can be selected to clear the internal counter. With the PTnCCLR bit set high, the counter will be cleared when a compare match occurs from the Comparator A. When the bit is low, the counter will be cleared when a compare match occurs from the Comparator P or with a counter overflow. A counter overflow clearing method can only be implemented if the CCRP bits are all cleared to zero. The PTnCCLR bit is not used in the PWM Output, Single Pulse Output or Capture Input Mode.

Periodic Type TM Operation Modes

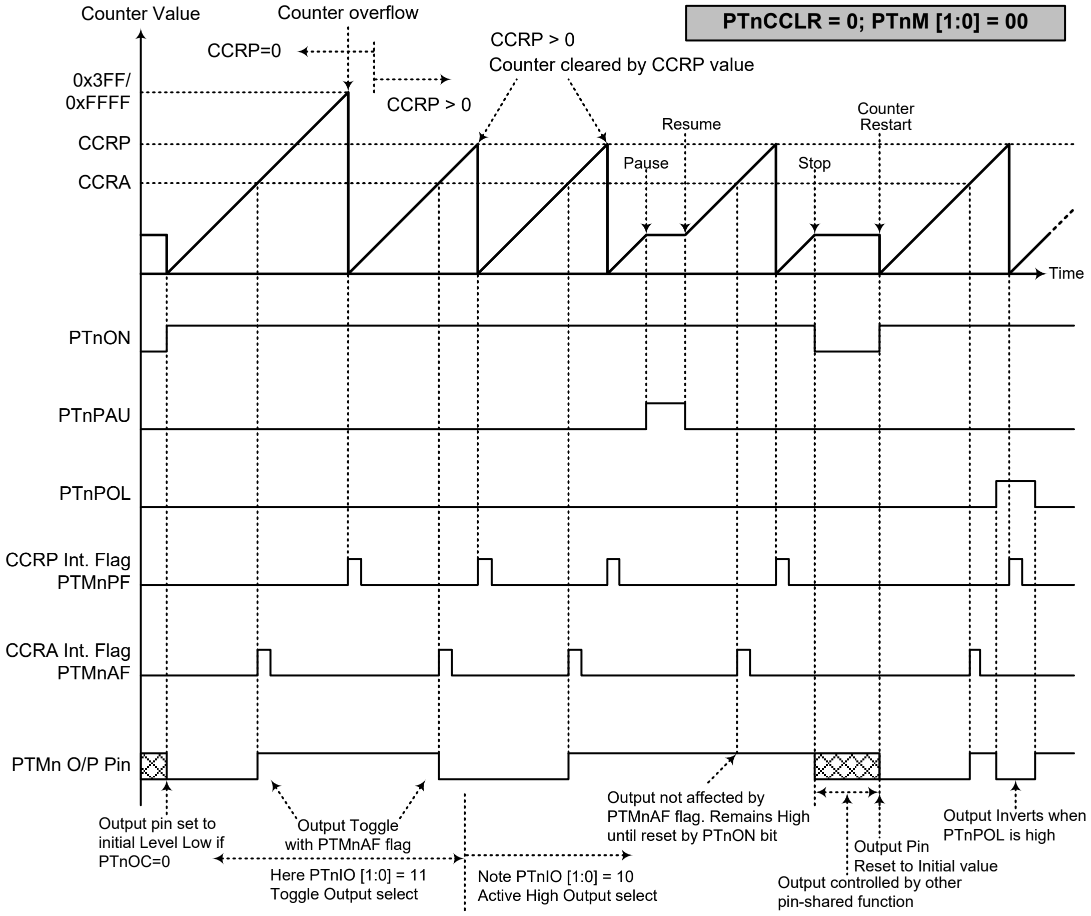
The Periodic Type TM can operate in one of five operating modes, Compare Match Output Mode, PWM Output Mode, Single Pulse Output Mode, Capture Input Mode or Timer/Counter Mode. The operating mode is selected using the PTnM1 and PTnM0 bits in the PTMnC1 register.

Compare Match Output Mode

To select this mode, bits PTnM1 and PTnM0 in the PTMnC1 register, should be set to 00 respectively. In this mode once the counter is enabled and running it can be cleared by three methods. These are a counter overflow, a compare match from Comparator A and a compare match from Comparator P. When the PTnCCLR bit is low, there are two ways in which the counter can be cleared. One is when a compare match from Comparator P, the other is when the CCRP bits are all zero which allows the counter to overflow. Here both PTMnAF and PTMnPF interrupt request flags for Comparator A and Comparator P respectively, will both be generated.

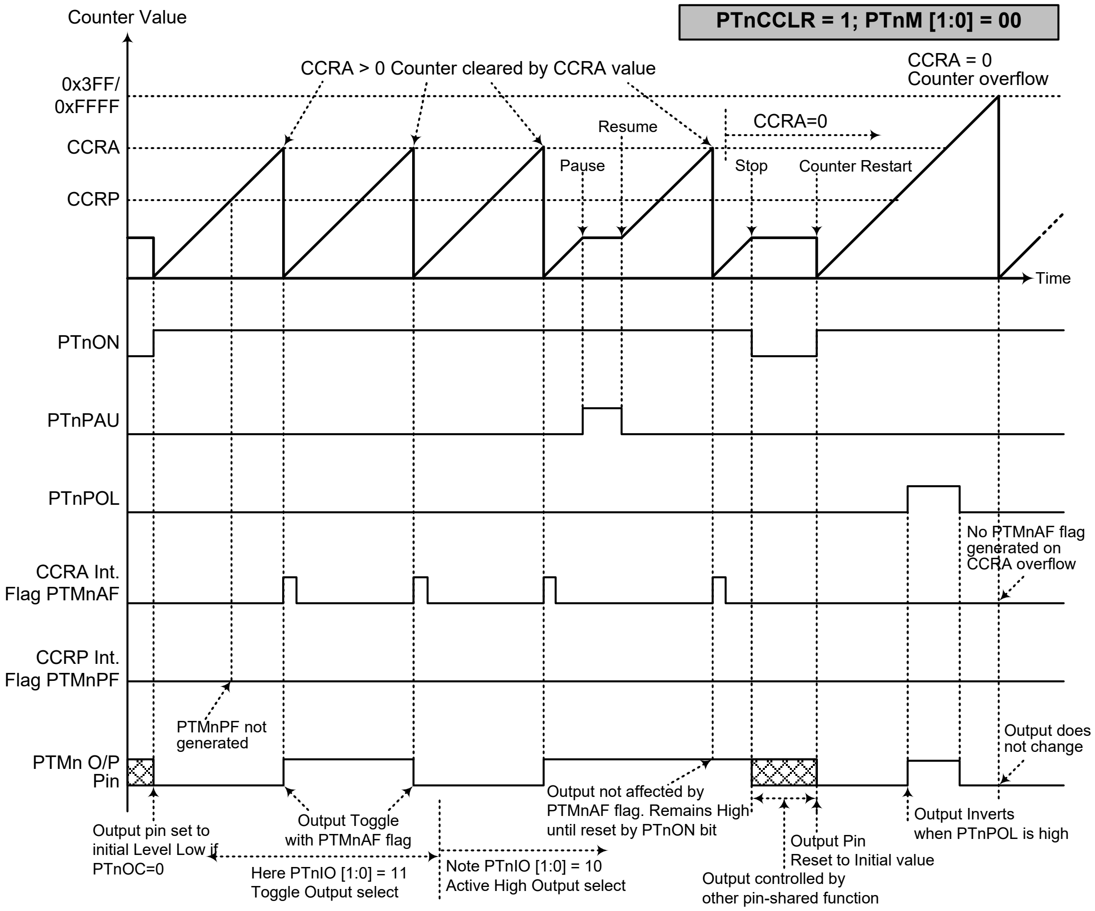
If the PTnCCLR bit in the PTMnC1 register is high then the counter will be cleared when a compare match occurs from Comparator A. However, here only the PTMnAF interrupt request flag will be generated even if the value of the CCRP bits is less than that of the CCRA registers. Therefore when PTnCCLR is high no PTMnPF interrupt request flag will be generated. In the Compare Match Output Mode, the CCRA can not be set to “0”.

As the name of the mode suggests, after a comparison is made, the PTMn output pin will change state. The PTMn output pin condition however only changes state when a PTMnAF interrupt request flag is generated after a compare match occurs from Comparator A. The PTMnPF interrupt request flag, generated from a compare match occurs from Comparator P, will have no effect on the PTMn output pin. The way in which the PTMn output pin changes state are determined by the condition of the PTnIO1 and PTnIO0 bits in the PTMnC1 register. The PTMn output pin can be selected using the PTnIO1 and PTnIO0 bits to go high, to go low or to toggle from its present condition when a compare match occurs from Comparator A. The initial condition of the PTMn output pin, which is setup after the PTnON bit changes from low to high, is setup using the PTnOC bit. Note that if the PTnIO1 and PTnIO0 bits are zero then no pin change will take place.



Compare Match Output Mode – PTnCCLR = 0

Note: 1. With PTnCCLR=0, a Comparator P match will clear the counter
2. The PTMn output pin is controlled only by the PTMnAF flag
3. The output pin is reset to its initial state by a PTnON bit rising edge
4. The 10-bit PTM maximum counter value is 0x3FF while the 16-bit PTM maximum counter value is 0xFFFF.
5. n=0~1 for 10-bit PTM while n=2~3 for 16-bit PTM



Compare Match Output Mode – PTnCCLR = 1

Note: 1. With PTnCCLR=1, a Comparator A match will clear the counter
2. The PTMn output pin is controlled only by the PTMnAF flag
3. The output pin is reset to its initial state by a PTnON bit rising edge
4. A PTMnPF flag is not generated when PTnCCLR =1
5. The 10-bit PTM maximum counter value is 0x3FF while the 16-bit PTM maximum counter value is 0xFFFF.
6. n = 0 ~1 for 10-bit PTM while n = 2 ~3 for 16-bit PTM

Timer/Counter Mode

To select this mode, bits PTnM1 and PTnM0 in the PTMnC1 register should be set to 11 respectively. The Timer/Counter Mode operates in an identical way to the Compare Match Output Mode generating the same interrupt flags. The exception is that in the Timer/Counter Mode the PTMn output pin is not used. Therefore the above description and Timing Diagrams for the Compare Match Output Mode can be used to understand its function. As the PTMn output pin is not used in this mode, the pin can be used as a normal I/O pin or other pin-shared function.

PWM Output Mode

To select this mode, bits PTnM1 and PTnM0 in the PTMnC1 register should be set to 10 respectively and also the PTnIO1 and PTnIO0 bits should be set to 10 respectively. The PWM function within the PTMn is useful for applications which require functions such as motor control, heating control, illumination control, etc. By providing a signal of fixed frequency but of varying duty cycle on the PTMn output pin, a square wave AC waveform can be generated with varying equivalent DC RMS values.

As both the period and duty cycle of the PWM waveform can be controlled, the choice of generated waveform is extremely flexible. In the PWM Output Mode, the PTnCCLR bit has no effect as the PWM period. Both of the CCRP and CCRA registers are used to generate the PWM waveform, one register is used to clear the internal counter and thus control the PWM waveform frequency, while the other one is used to control the duty cycle. The PWM waveform frequency and duty cycle can therefore be controlled by the values in the CCRA and CCRP registers.

An interrupt flag, one for each of the CCRA and CCRP, will be generated when a compare match occurs from either Comparator A or Comparator P. The PTnOC bit in the PTMnC1 register is used to select the required polarity of the PWM waveform while the two PTnIO1 and PTnIO0 bits are used to enable the PWM output or to force the PTMn output pin to a fixed high or low level. The PTnPOL bit is used to reverse the polarity of the PWM output waveform.

- **10-bit PTMn, PWM Output Mode**

CCRP	1~1023	0
Period	1~1023	1024
Duty	CCRA	

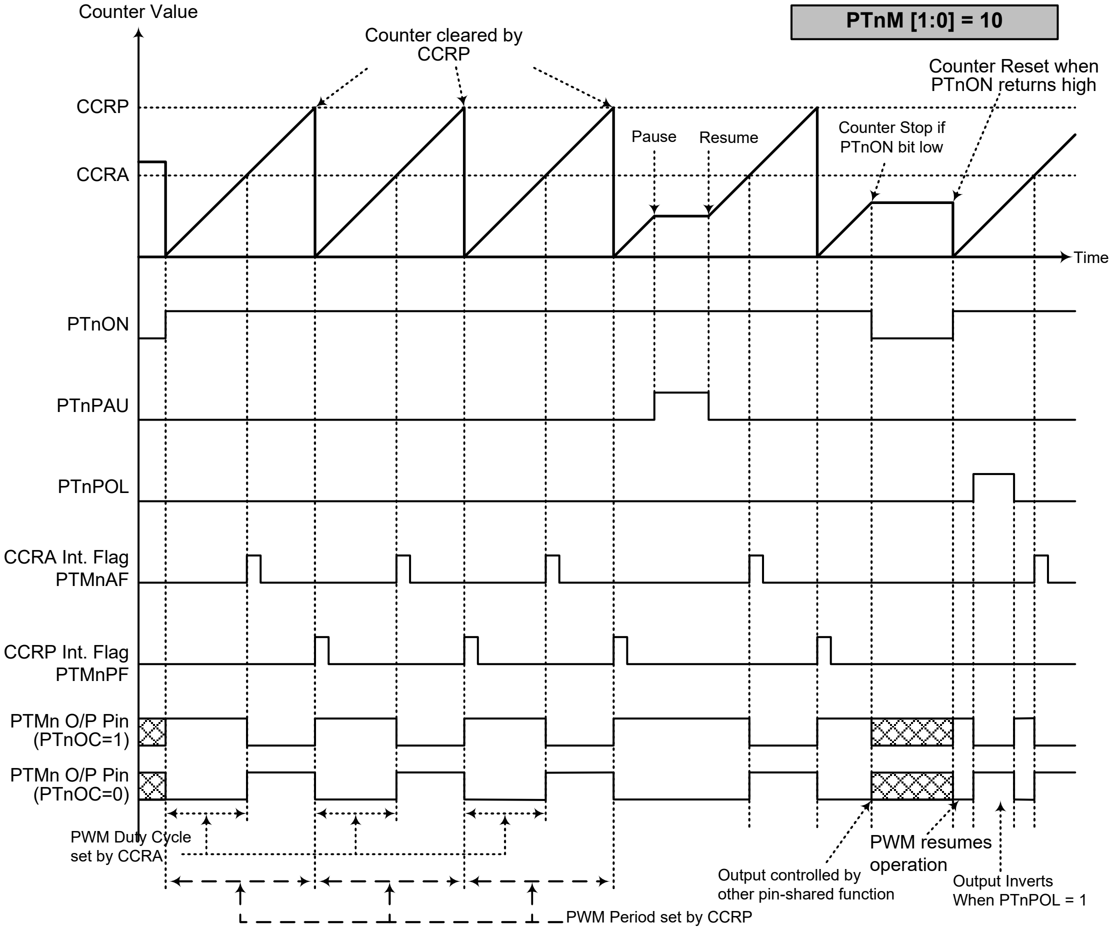
- **16-bit PTMn, PWM Output Mode**

CCRP	1~65535	0
Period	1~65535	65536
Duty	CCRA	

If f_{SYS}=16MHz, TM clock source select f_{SYS}/4, CCRP=512 and CCRA=128,

The PTMn PWM output frequency=(f_{SYS}/4)/512=f_{SYS}/2048=8kHz, duty=128/512=25%,

If the Duty value defined by the CCRA register is equal to or greater than the Period value, then the PWM output duty is 100%.



PWM Output Mode

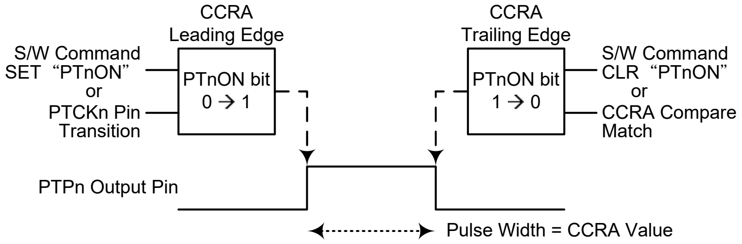
Note: 1. The counter is cleared by CCRP.
2. A counter clear sets the PWM Period
3. The internal PWM function continues running even when PTnIO [1:0] = 00 or 01
4. The PTnCCLR bit has no influence on PWM operation
5. n=0~1 for 10-bit PTM while n=2~3 for 16-bit PTM

Single Pulse Output Mode

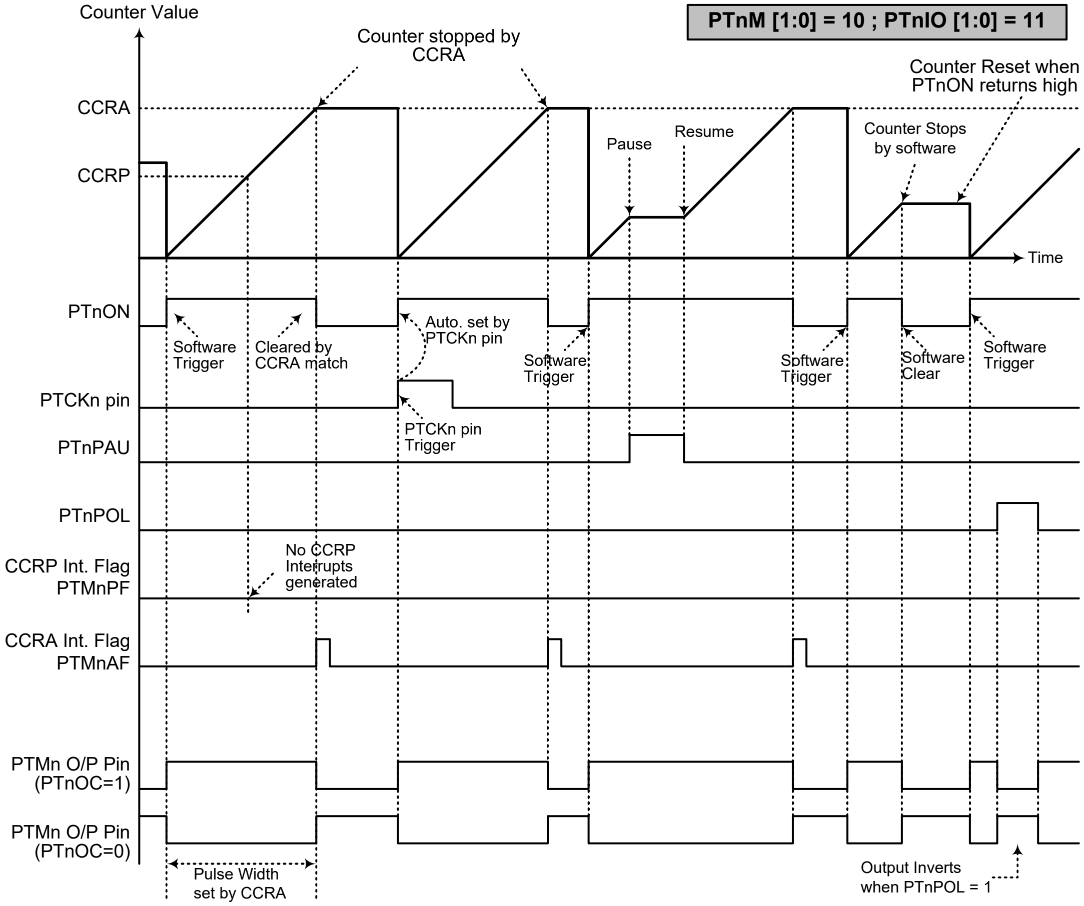
To select this mode, bits PTnM1 and PTnM0 in the PTMnC1 register should be set to 10 respectively and also the PTnIO1 and PTnIO0 bits should be set to 11 respectively. The Single Pulse Output Mode, as the name suggests, will generate a single shot pulse on the PTMn output pin.

The trigger for the pulse output leading edge is a low to high transition of the PTnON bit, which can be implemented using the application program. However in the Single Pulse Output Mode, the PTnON bit can also be made to automatically change from low to high using the external PTCKn pin, which will in turn initiate the Single Pulse output. When the PTnON bit transitions to a high level, the counter will start running and the pulse leading edge will be generated. The PTnON bit should remain high when the pulse is in its active state. The generated pulse trailing edge will be generated when the PTnON bit is cleared to zero, which can be implemented using the application program or when a compare match occurs from Comparator A.

However a compare match from Comparator A will also automatically clear the PTnON bit and thus generate the Single Pulse output trailing edge. In this way the CCRA value can be used to control the pulse width. A compare match from Comparator A will also generate a PTMn interrupt. The counter can only be reset back to zero when the PTnON bit changes from low to high when the counter restarts. In the Single Pulse Output Mode CCRP is not used. The PTnCCLR is not used in this Mode.



Single Pulse Generation



Single Pulse Output Mode

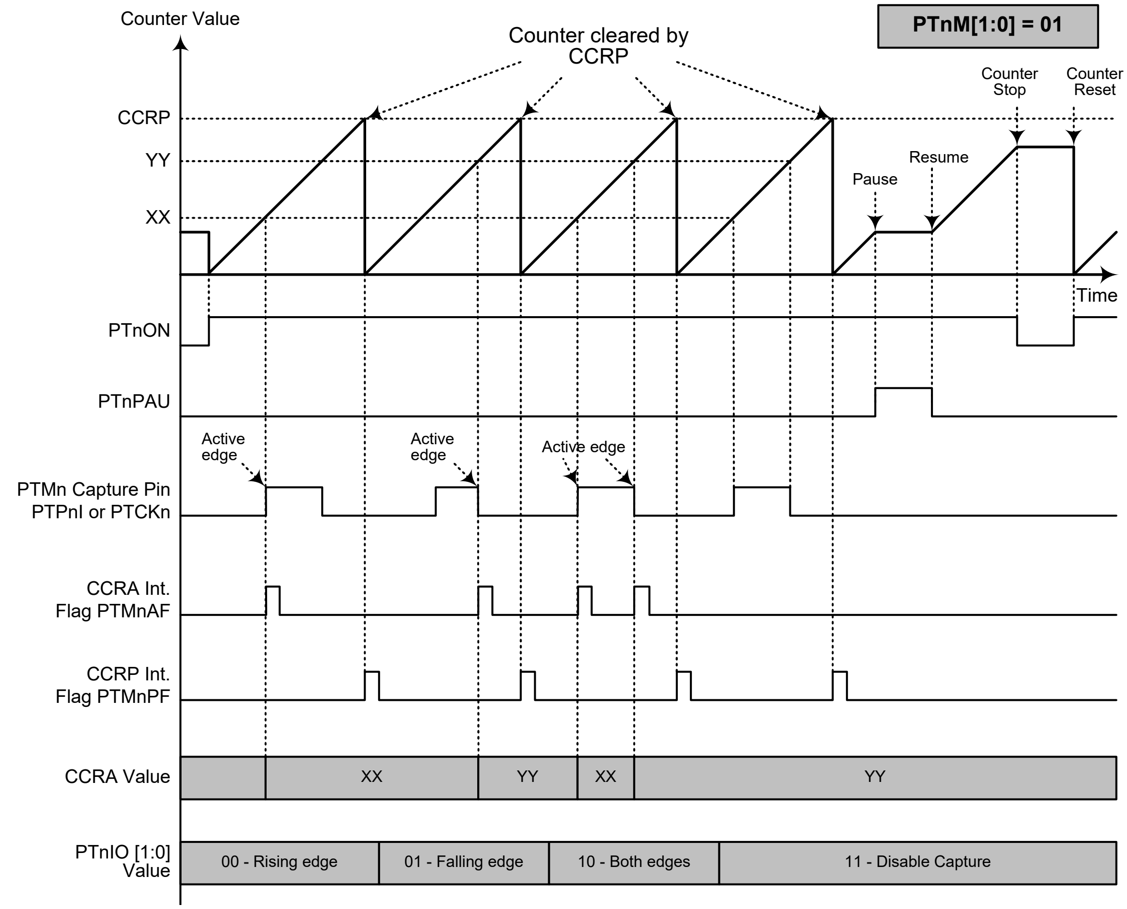
Note: 1. Counter stopped by CCRA
2. CCRP is not used
3. The pulse triggered by the PTCKn pin or by setting the PTnON bit high
4. A PTCKn pin active edge will automatically set the PTnON bit high.
5. In the Single Pulse Output Mode, PTnIO [1:0] must be set to “11” and can not be changed.
6. n = 0~1 for 10-bit PTM while n = 2~3 for 16-bit PTM

Capture Input Mode

To select this mode bits PTnM1 and PTnM0 in the PTMnC1 register should be set to 01 respectively. This mode enables external signals to capture and store the present value of the internal counter and can therefore be used for applications such as pulse width measurements. The external signal is supplied on the PTPnI or PTCKn pin, selected by the PTnCAPTS bit in the PTMnC1 register. The input pin active edge can be either a rising edge, a falling edge or both rising and falling edges; the active edge transition type is selected using the PTnIO1 and PTnIO0 bits in the PTMnC1 register. The counter is started when the PTnON bit changes from low to high which is initiated using the application program.

When the required edge transition appears on the PTPnI or PTCKn pin the present value in the counter will be latched into the CCRA registers and a PTMn interrupt generated. Irrespective of what events occur on the PTPnI or PTCKn pin the counter will continue to free run until the PTnON bit changes from high to low. When a CCRP compare match occurs the counter will reset back to zero; in this way the CCRP value can be used to control the maximum counter value. When a CCRP compare match occurs from Comparator P, a PTMn interrupt will also be generated. Counting the number of overflow interrupt signals from the CCRP can be a useful method in measuring long pulse widths. The PTnIO1 and PTnIO0 bits can select the active trigger edge on the PTPnI or PTCKn pin to be a rising edge, falling edge or both edge types. If the PTnIO1 and PTnIO0 bits are both set high, then no capture operation will take place irrespective of what happens on the PTPnI or PTCKn pin, however it must be noted that the counter will continue to run.

As the PTPnI or PTCKn pin is pin shared with other functions, care must be taken if the PTMn is in the Input Capture Mode. This is because if the pin is setup as an output, then any transitions on this pin may cause an input capture operation to be executed. The PTnCCLR, PTnOC and PTnPOL bits are not used in this Mode.



Capture Input Mode

Note: 1. PTnM [1:0] = 01 and active edge set by the PTnIO [1:0] bits

2. A PTMn Capture input pin active edge transfers the counter value to CCRA
3. PTnCCLR bit not used
4. No output function – PTnOC and PTnPOL bits are not used
5. CCRP determines the counter value and the counter has a maximum count value when CCRP is equal to zero.
6. n=0~1 for 10-bit PTM while n=2~3 for 16-bit PTM

Analog to Digital Converter

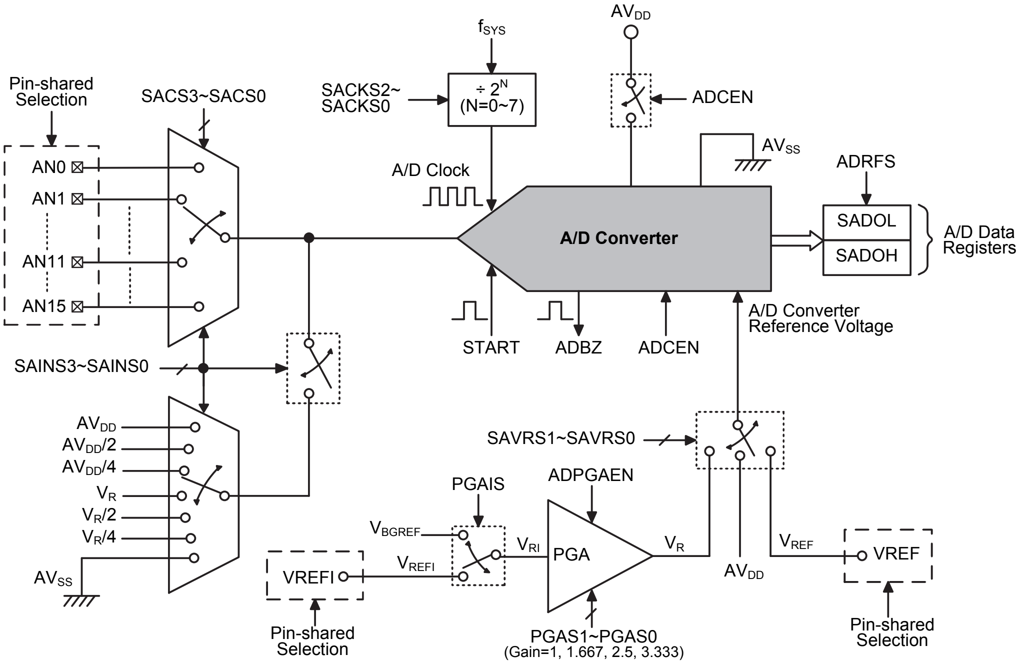
The need to interface to real world analog signals is a common requirement for many electronic systems. However, to properly process these signals by a microcontroller, they must first be converted into digital signals by A/D converters. By integrating the A/D conversion electronic circuitry into the microcontroller, the need for external components is reduced significantly with the corresponding follow-on benefits of lower costs and reduced component space requirements.

A/D Overview

The device contains a multi-channel analog to digital converter which can directly interface to external analog signals, such as that from sensors or other control signals and convert these signals directly into a 12-bit digital value. It also can convert the internal signals, such as the internal reference voltage, into a 12-bit digital value. The external or internal analog signal to be converted is determined by the SAINS and SACS bit fields. Note that when the internal analog signal is selected to be converted using the SAINS field, the external channel analog input will automatically be switched off. More detailed information about the A/D input signal selection will be described in the "A/D Converter Input Signals" section.

The accompanying block diagram shows the internal structure of the A/D converter with temperature sensor together with its associated registers and control bits.

External Input Channels	Internal Signals	A/D Signal Select
16: AN0~AN15	6: AV_{DD}, $AV_{DD}/2$, $AV_{DD}/4$, V_R, $V_R/2$, $V_R/4$,	SAINS3~SAINS0 SACS3~SACS0



A/D Converter Structure

Registers Descriptions

Overall operation of the A/D converter is controlled using six registers. A read only register pair exists to store the A/D Converter data 12-bit value. Three registers, SADC0, SADC1 and SADC2, are the control registers which setup the operating conditions and control function of the A/D converter. The VBGRC register contains the VBGREN bit to control the bandgap reference voltage.

Register Name	Bit							
	7	6	5	4	3	2	1	0
SADOL (ADRFS=0)	D3	D2	D1	D0	—	—	—	—
SADOL (ADRFS=1)	D7	D6	D5	D4	D3	D2	D1	D0
SADOH (ADRFS=0)	D11	D10	D9	D8	D7	D6	D5	D4
SADOH (ADRFS=1)	—	—	—	—	D11	D10	D9	D8
SADC0	START	ADBZ	ADCEN	ADRFS	SACS3	SACS2	SACS1	SACS0
SADC1	SAINS3	SAINS2	SAINS1	SAINS0	—	SACKS2	SACKS1	SACKS0
SADC2	ADPGAEN	—	—	PGAIS	SAVRS1	SAVRS0	PGAGS1	PGAGS0
VBGRC	—	—	—	—	—	—	—	VBGREN

A/D Converter Registers List

A/D Converter Data Registers – SADOL, SADOH

As the device contains an internal 12-bit A/D converter, it requires two data registers to store the converted value. These are a high byte register, known as SADOH, and a low byte register, known as SADOL. After the conversion process takes place, these registers can be directly read by the microcontroller to obtain the digitised conversion value. As only 12 bits of the 16-bit register space is utilised, the format in which the data is stored is controlled by the ADRFS bit in the SADC0 register as shown in the accompanying table. D0~D11 are the A/D conversion result data bits. Any unused bits will be read as zero. The A/D data registers contents will be unchanged if the A/D converter is disabled.

ADRFS	SADOH								SADOL							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	0	0	0	0
1	0	0	0	0	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0

A/D Converter Data Registers

A/D Converter Control Registers – SADC0, SADC1, SADC2

To control the function and operation of the A/D converter, three control registers known as SADC0, SADC1 and SADC2 are provided. These 8-bit registers define functions such as the selection of which analog signal is connected to the internal A/D converter, the digitised data format, the A/D clock source as well as controlling the start function and monitoring the A/D converter busy status. As the device contains only one actual analog to digital converter hardware circuit, each of the external and internal analog signals must be routed to the converter. The SAINS field in the SADC1 register and SACS field in the SADC0 register are used to determine which analog signal derived from the external or internal signals will be connected to the A/D converter. The A/D converter also contains a programmable gain amplifier, PGA, to generate the A/D converter internal reference voltage. The overall operation of the PGA is controlled using the SADC2 register.

The relevant pin-shared function selection bits determine which pins on I/O Ports are used as analog inputs for the A/D converter input and which pins are not. When the pin is selected to be an A/D input, its original function whether it is an I/O or other pin-shared function will be removed. In addition, any internal pull-high resistor connected to the pin will be automatically removed if the pin is selected to be an A/D converter input.

- **SADC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	START	ADBZ	ADCEN	ADRFS	SACS3	SACS2	SACS1	SACS0
R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **START**: Start the A/D Conversion
0→1→0: Start
This bit is used to initiate an A/D conversion process. The bit is normally low but if set high and then cleared low again, the A/D converter will initiate a conversion process.

Bit 6 **ADBZ**: A/D Converter busy flag
0: No A/D conversion is in progress
1: A/D conversion is in progress
This read only flag is used to indicate whether the A/D conversion is in progress or not. When the START bit is set from low to high and then to low again, the ADBZ flag will be set to 1 to indicate that the A/D conversion is initiated. The ADBZ flag will be cleared to 0 after the A/D conversion is complete.

Bit 5 **ADCEN**: A/D Converter function enable control
0: Disable
1: Enable
This bit controls the A/D internal function. This bit should be set to one to enable the A/D converter. If the bit is set low, then the A/D converter will be switched off reducing the device power consumption. When the A/D converter function is disabled, the contents of the A/D data register pair known as SADOH and SADOL will be unchanged.

Bit 4 **ADRFS**: A/D conversion data format select
0: A/D converter data format → SADOH = D [11:4]; SADOL = D [3:0]
1: A/D converter data format → SADOH = D [11:8]; SADOL = D [7:0]
This bit controls the format of the 12-bit converted A/D value in the two A/D data registers. Details are provided in the A/D converter data register section.

Bit 3~0 **SACS3~SACS0**: A/D converter external analog input channel select
0000: External AN0 input
0001: External AN1 input
0010: External AN2 input
0011: External AN3 input
0100: External AN4 input
0101: External AN5 input
0110: External AN6 input
0111: External AN7 input
1000: External AN8 input
1001: External AN9 input
1010: External AN10 input
1011: External AN11 input
1100: External AN12 input
1101: External AN13 input
1110: External AN14 input
1111: External AN15 input
These bits are used to select which external analog input channel is to be converted. When the external analog input channel is selected, the SAINS bit field must set to “0000”, “0100” or “11xx”. Details are summarized in the “A/D Converter Input Signal Selection” table.

- **SADC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	SAINS3	SAINS2	SAINS1	SAINS0	—	SACKS2	SACKS1	SACKS0
R/W	R/W	R/W	R/W	R/W	—	R/W	R/W	R/W
POR	0	0	0	0	—	0	0	0

Bit 7~4 **SAINS3~SAINS0**: A/D converter input signal select
0000: External source – External analog channel intput, ANn
0001: Internal source – Internal signal derived from AV_{DD}
0010: Internal source – Internal signal derived from $AV_{DD}/2$
0011: Internal source – Internal signal derived from $AV_{DD}/4$
0100: External source – External analog channel intput, ANn
0101: Internal source – Internal signal derived from PGA output V_R
0110: Internal source – Internal signal derived from PGA output $V_R/2$
0111: Internal source – Internal signal derived from PGA output $V_R/4$
10xx: Internal source – Ground
11xx: External source – External analog channel intput, ANn

When the internal analog signal is selected to be converted, the external channel signal input will automatically be switched off regardless of the SACS field value. It will prevent the external channel input from being connected together with the internal analog signal.

Bit 3 Unimplemented, read as "0"

Bit 2~0 **SACKS2~SACKS0**: A/D conversion clock source select
000: f_{SYS}
001: $f_{SYS}/2$
010: $f_{SYS}/4$
011: $f_{SYS}/8$
100: $f_{SYS}/16$
101: $f_{SYS}/32$
110: $f_{SYS}/64$
111: $f_{SYS}/128$

These bits are used to select the clock source for the A/D converter.

- **SADC2 Register**

Bit	7	6	5	4	3	2	1	0
Name	ADPGAEN	—	—	PGAIS	SAVRS1	SAVRS0	PGAGS1	PGAGS0
R/W	R/W	—	—	R/W	R/W	R/W	R/W	R/W
POR	0	—	—	0	0	0	0	0

Bit 7 **ADPGAEN**: PGA enable control
0: Disable
1: Enable

Bit 6~5 Unimplemented, read as "0"

Bit 4 **PGAIS**: PGA input voltage selection
0: From VREFI pin
1: From internal reference voltage V_{BGREF}

Bit 3~2 **SAVRS1~SAVRS0**: A/D converter reference voltage select
00: Internal A/D converter power, AV_{DD}
01: External VREF pin
1x: Internal PGA output voltage, V_R

These bits are used to select the A/D converter reference voltage source. When the internal reference voltage source is selected, the reference voltage derived from the external VREF pin will automatically be switched off.

Bit 1~0 **PGAGS1~PGAGS0**: PGA gain select
00: Gain=1
01: Gain=1.667 – V_R=2V as V_{RI}=1.2V
10: Gain=2.5 – V_R=3V as V_{RI}=1.2V
11: Gain=3.333 – V_R=4V as V_{RI}=1.2V
These bits are used to select the PGA gain. Note that here the gain is guaranteed only when the PGA input voltage is equal to 1.2V.

- **VBGRC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	—	VBGREN
R/W	—	—	—	—	—	—	—	R/W
POR	—	—	—	—	—	—	—	0

Bit 7~1 Unimplemented, read as "0"

Bit 0 **VBGREN**: Bandgap reference voltage control
0: Disable
1: Enable
This bit is used to enable the internal Bandgap reference circuit. The internal Bandgap reference circuit should first be enabled before the V_{BGREF} voltage is selected to be used. A specific start-up time is necessary for the Bandgap circuit to become stable and accurate.

A/D Converter Reference Voltage

The actual reference voltage supply to the A/D Converter can be supplied from the positive power supply, AV_{DD}, an external reference source supplied on pin VREF or an internal reference voltage V_R determined by the SAVRS1~SAVRS0 bits in the SADC2 register. The internal reference voltage is amplified through a programmable gain amplifier, PGA, which is controlled by the ADPGAEN bit in the SADC2 register. The PGA gain can be equal to 1, 1.667, 2.5 or 3.333 and selected using the PGAGS1~PGAGS0 bits in the SADC2 register. The PGA input can come from the external reference input pin, VREFI, or an internal Bandgap reference voltage, V_{BGREF}, selected by the PGAIS bit in the SADC2 register. As the VREFI and VREF pin both are pin-shared with other functions, when the VREFI or VREF pin is selected as the reference voltage pin, the VREFI or VREF pin-shared function selection bits should first be properly configured to disable other pin-shared functions. However, if the internal reference signal is selected as the reference source, the external reference input from the VREFI or VREF pin will automatically be switched off by hardware. The analog input values must not be allowed to exceed the value of the selected reference voltage, V_{REF}.

A/D Converter Input Signals

All of the external A/D analog input pins are pin-shared with the I/O pins as well as other functions. The corresponding pin-shared function selection bits in the PxS1 and PxS0 registers, determine whether the external input pins are setup as A/D converter analog channel inputs or whether they have other functions. If the corresponding pin is setup to be an A/D converter analog channel input, the original pin function will be disabled. In this way, pins can be changed under program control to change their function between A/D inputs and other functions. All pull-high resistors, which are setup through register programming, will be automatically disconnected if the pins are setup as A/D inputs. Note that it is not necessary to first setup the A/D pin as an input in the port control register to enable the A/D input as when the relevant A/D input function selection bits enable an A/D input, the status of the port control register will be overridden.

As the device contains only one actual analog to digital converter hardware circuit, each of the external and internal analog signals must be routed to the converter. The SAINS3~SAINS0 bits in the SADC1 register are used to determine that the analog signal to be converted comes from the external channel input or internal analog signal. The SACS3~SACS0 bits in the SADC0 register are used to determine which external channel input is selected to be converted. If the SAINS3~SAINS0 bits are set to "0000", the external channel input will be selected to be converted and the SACS3~SACS0 bits can determine which external channel is selected.

When the SAINS field is set to the value of "0x01", "0x10" or "0x11", the internal analog signal will be selected. If the internal analog signal is selected to be converted, the external channel signal input will automatically be switched off regardless of the SACS field value. It will prevent the external channel input from being connected together with the internal analog signal.

SAINS [3:0]	SACS [3:0]	Input Signals	Description
0000, 0100, 11xx	0000~1111	AN0~AN15	External channel analog input ANn
0001	xxxx	AV_{DD}	Internal signal derived from AV_{DD}
0010	xxxx	$AV_{DD}/2$	Internal signal derived from $AV_{DD}/2$
0011	xxxx	$AV_{DD}/4$	Internal signal derived from $AV_{DD}/4$
0101	xxxx	V_R	Internal signal derived from PGA output V_R
0110	xxxx	$V_R/2$	Internal signal derived from PGA output $V_R/2$
0111	xxxx	$V_R/4$	Internal signal derived from PGA output $V_R/4$
10xx	xxxx	GND	Connected to the ground

"x": don't care

A/D Converter Input Signal Selection

A/D Operation

The START bit in the SADC0 register is used to start the AD conversion. When the microcontroller sets this bit from low to high and then low again, an analog to digital conversion cycle will be initiated.

The ADBZ bit in the SADC0 register is used to indicate whether the analog to digital conversion process is in progress or not. This bit will be automatically set to 1 by the microcontroller after an A/D conversion is successfully initiated. When the A/D conversion is complete, the ADBZ bit will be cleared to 0. In addition, the corresponding A/D interrupt request flag will be set in the interrupt control register, and if the interrupts are enabled, an internal interrupt signal will be generated. This A/D internal interrupt signal will direct the program flow to the associated A/D internal interrupt address for processing. If the A/D internal interrupt is disabled, the microcontroller can poll the ADBZ bit in the SADC0 register to check whether it has been cleared as an alternative method of detecting the end of an A/D conversion cycle.

The clock source for the A/D converter, which originates from the system clock f_{SYS}, can be chosen to be either f_{SYS} or a subdivided version of f_{SYS}. The division ratio value is determined by the SACKS2~SACKS0 bits in the SADC1 register. Although the A/D clock source is determined by the system clock f_{SYS} and by bits SACKS2~SACKS0, there are some limitations on the maximum A/D clock source speed that can be selected. As the recommended range of permissible A/D clock period, t_{ADCK}, is from 0.5μs to 10μs, care must be taken for system clock frequencies. For example, if the system clock operates at a frequency of 8MHz, the SACKS2~SACKS0 bits should not be set to 000, 001 or 111. Doing so will give A/D clock periods that are less than the minimum or larger than the maximum A/D clock period which may result in inaccurate A/D conversion values. Refer to the following table for examples, where values marked with an asterisk * show where special care must be taken, as the values may be out of the specified A/D clock period.

f_{SYS}	A/D Clock Period (t_{ADCK})							
	SACKS [2:0]=000 (f_{SYS})	SACKS [2:0]=001 (f_{SYS}/2)	SACKS [2:0]=010 (f_{SYS}/4)	SACKS [2:0]=011 (f_{SYS}/8)	SACKS [2:0]=100 (f_{SYS}/16)	SACKS [2:0]=101 (f_{SYS}/32)	SACKS [2:0]=110 (f_{SYS}/64)	SACKS [2:0]=111 (f_{SYS}/128)
1 MHz	1μs	2μs	4μs	8μs	16μs *	32μs *	64μs *	128μs *
2 MHz	500ns	1μs	2μs	4μs	8μs	16μs *	32μs *	64μs *
4 MHz	250ns *	500ns	1μs	2μs	4μs	8μs	16μs *	32μs *
8 MHz	125ns *	250ns *	500ns	1μs	2μs	4μs	8μs	16μs *
12 MHz	83ns *	167ns *	333ns *	667ns	1.33μs	2.67μs	5.33μs	10.67μs *
16 MHz	62.5ns *	125ns *	250ns *	500ns	1μs	2μs	4μs	8μs

A/D Clock Period Examples

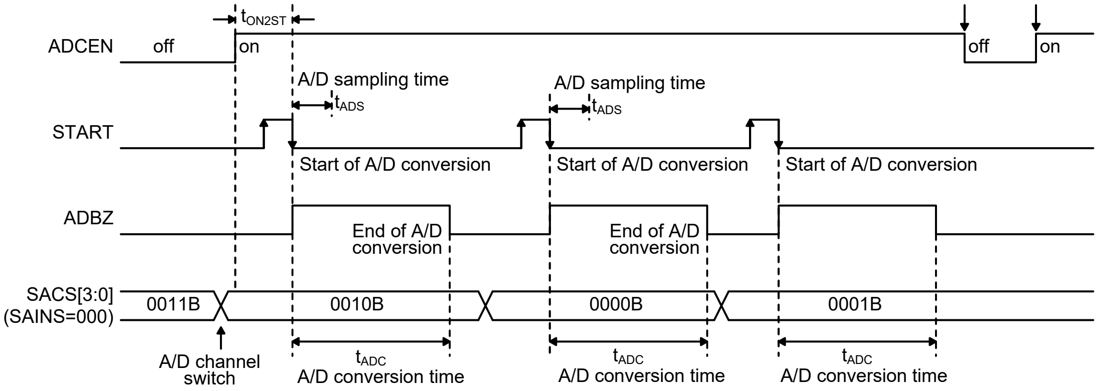
Controlling the power on/off function of the A/D converter circuitry is implemented using the ADCEN bit in the SADC0 register. This bit must be set high to power on the A/D converter. When the ADCEN bit is set high to power on the A/D converter internal circuitry a certain delay, as indicated in the timing diagram, must be allowed before an A/D conversion is initiated. Even if no pins are selected for use as A/D inputs, if the ADCEN bit is high, then some power will still be consumed. In power conscious applications it is therefore recommended that the ADCEN is set low to reduce power consumption when the A/D converter function is not being used.

Conversion Rate and Timing Diagram

A complete A/D conversion contains two parts, data sampling and data conversion. The data sampling which is defined as t_{ADS} takes 4 A/D clock cycles and the data conversion takes 12 A/D clock cycles. Therefore a total of 16 A/D clock cycles for an analog signal A/D conversion which is defined as t_{ADC} are necessary.

Maximum single A/D conversion rate = A/D clock period / 16

The accompanying diagram shows graphically the various stages involved in an external channel input signal analog to digital conversion process and its associated timing. After an A/D conversion process has been initiated by the application program, the microcontroller internal hardware will begin to carry out the conversion, during which time the program can continue with other functions. The time taken for the A/D conversion is 16 t_{ADCK} clock cycles where t_{ADCK} is equal to the A/D clock period.



A/D Conversion Timing

Summary of A/D Conversion Steps

The following summarises the individual steps that should be executed in order to implement an A/D conversion process.

- Step 1
  Select the required A/D conversion clock by properly programming the SACKS2~SACKS0 bits in the SADC1 register.
- Step 2
  Enable the A/D converter by setting the ADCEN bit in the SADC0 register to one.
- Step 3
  Select which signal is to be connected to the internal A/D converter by correctly configuring the SACS and SAINS bit fields
  Selecting the external channel input to be converted, go to Step 4.
  Selecting the internal analog signal to be converted, go to Step 5.
- Step 4
  If the SAINS field is 0000, 0100 or 11xx, the external channel input can be selected. The desired external channel input is selected by configuring the SACS field. When the A/D input signal comes from the external channel input, the corresponding pin should be configured as an A/D input function by selecting the relevant pin-shared function control bits. Then go to Step 6.
- Step 5
  If the SAINS field is set to 0x01, 0x10 or 0x11, the relevant internal analog signal will be selected. When the internal analog signal is selected to be converted, the external channel analog input will automatically be disconnected. Then go to Step 6.
- Step 6
  Select the A/D converter output data format by configuring the ADRFS bit.
- Step 7
  Select the A/D converter reference voltage source by configuring the SAVRS bit field.
  Select the PGA input signal and the desired PGA gain if the PGA output voltage, V_R, is selected as the A/D converter reference voltage.
- Step 8
  If A/D conversion interrupt is used, the interrupt control registers must be correctly configured to ensure the A/D interrupt function is active. The master interrupt control bit, EMI, and the A/D conversion interrupt control bit, ADE, must both be set high in advance.
- Step 9
  The A/D conversion procedure can now be initialized by setting the START bit from low to high and then low again.
- Step 10
  If A/D conversion is in progress, the ADBZ flag will be set high. After the A/D conversion process is complete, the ADBZ flag will go low and then the output data can be read from SADOH and SADOL registers.

Note: When checking for the end of the conversion process, if the method of polling the ADBZ bit in the SADC0 register is used, the interrupt enable step above can be omitted.

Programming Considerations

During microcontroller operations where the A/D converter is not being used, the A/D internal circuitry can be switched off to reduce power consumption, by setting bit ADCEN low in the SADC0 register. When this happens, the internal A/D converter circuits will not consume power irrespective of what analog voltage is applied to their input lines. If the A/D converter input lines are used as normal I/Os, then care must be taken as if the input voltage is not at a valid logic level, then this may lead to some increase in power consumption.

A/D Transfer Function

As the device contains a 12-bit A/D converter, its full-scale converted digitised value is equal to FFFH. Since the full-scale analog input value is equal to the actual A/D converter reference voltage, V_{REF}, this gives a single bit analog input value of reference voltage value divided by 4096.

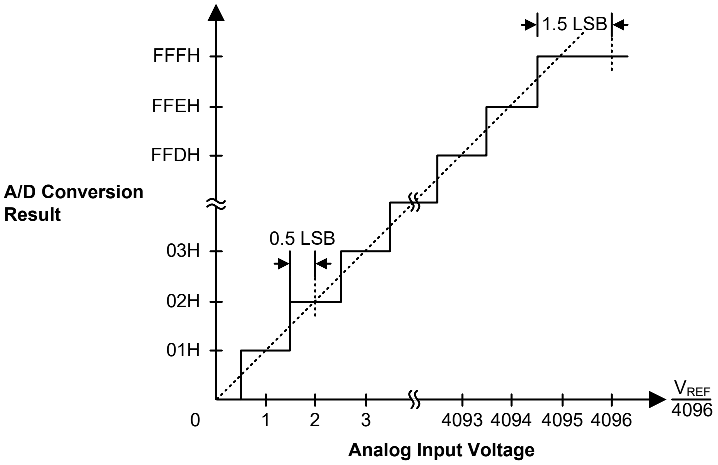
$$1\ \text{LSB} = V_{REF} \div 4096$$

The A/D Converter input voltage value can be calculated using the following equation:

$$\text{A/D input voltage} = \text{A/D output digital value} \times V_{REF} \div 4096$$

The diagram shows the ideal transfer function between the analog input value and the digitised output value for the A/D converter. Except for the digitised zero value, the subsequent digitised values will change at a point 0.5 LSB below where they would change without the offset, and the last full scale digitised value will change at a point 1.5 LSB below the V_{REF} level.

Note that here the V_{REF} voltage is the actual A/D converter reference voltage determined by the SAVRS field.



Ideal A/D Transfer Function

A/D Programming Examples

The following two programming examples illustrate how to setup and implement an A/D conversion. In the first example, the method of polling the ADBZ bit in the SADC0 register is used to detect when the conversion cycle is complete, whereas in the second example, the A/D interrupt is used to determine when the conversion is complete.

Example: Using an ADBZ polling method to detect the end of conversion

```
clr ADE               ; disable ADC interrupt
mov a,03H             ; select fSYS/8 as A/D clock and A/D input
mov SADC1,a           ; signal comes from external channel
mov a,00H             ; select AVDD as the A/D reference voltage source
mov SADC2,a
set ADCEN
mov a,03H             ; setup PCS0 to configure pin AN0
mov PCS0,a
mov a,00H             ; select AN0 as the A/D external channel input
mov SADC0,a
:
start_conversion:
clr START             ; high pulse on start bit to initiate conversion
set START             ; reset A/D
clr START             ; start A/D
:
polling_EOC:
sz  ADBZ              ; poll the SADC0 register ADBZ bit to detect end of A/D conversion
jmp polling_EOC       ; continue polling
:
mov a,SADOL           ; read low byte conversion result value
mov SADOL_buffer,a    ; save result to user defined register
mov a,SADOH           ; read high byte conversion result value
mov SADOH_buffer,a    ; save result to user defined register
:
jmp start_conversion  ; start next A/D conversion
```

Example: Using the interrupt method to detect the end of conversion

```
clr ADE               ; disable ADC interrupt
mov a,03H             ; select fSYS/8 as A/D clock and A/D input
mov SADC1,a           ; signal comes from external channel
mov a,00H             ; select AVDD as the A/D reference voltage source
mov SADC2,a
set ADCEN
mov a,03h             ; setup PCS0 to configure pin AN0
mov PCS0,a
mov a,00h
mov SADC0,a           ; select AN0 as the A/D external channel input
:
Start_conversion:
clr START             ; high pulse on START bit to initiate conversion
set START             ; reset A/D
clr START             ; start A/D
clr ADF               ; clear ADC interrupt request flag
set ADE               ; enable ADC interrupt
set EMI               ; enable global interrupt
:
:
ADC_ISR:              ; ADC interrupt service routine
mov acc_stack,a       ; save ACC to user defined memory
mov a,STATUS
mov status_stack,a    ; save STATUS to user defined memory
:
mov a,SADOL           ; read low byte conversion result value
mov SADOL_buffer,a    ; save result to user defined register
mov a,SADOH           ; read high byte conversion result value
mov SADOH_buffer,a      ; save result to user defined register
:
EXIT_INT_ISR:
mov a,status_stack
mov STATUS,a            ; restore STATUS from user defined memory
mov a,acc_stack         ; restore ACC from user defined memory
reti
```

Serial Interface Module – SIM

The device contains a Serial Interface Module, which includes both the four-line SPI interface or two-line I^{2}C interface types, to allow an easy method of communication with external peripheral hardware. Having relatively simple communication protocols, these serial interface types allow the microcontroller to interface to external SPI or I^{2}C based hardware such as sensors, Flash or EEPROM memory, etc. The SIM interface pins are pin-shared with other I/O pins and therefore the SIM interface functional pins must first be selected using the corresponding pin-shared function selection bits. As both interface types share the same pins and registers, the choice of whether the SPI or I^{2}C type is used is made using the SIM operating mode control bits, named SIM2~SIM0, in the SIMC0 register. These pull-high resistors of the SIM pin-shared I/O pins are selected using pull-high control registers when the SIM function is enabled and the corresponding pins are used as SIM input pins.

SPI Interface

The SPI interface is often used to communicate with external peripheral devices such as sensors, Flash or EEPROM memory devices, etc. Originally developed by Motorola, the four line SPI interface is a synchronous serial data interface that has a relatively simple communication protocol simplifying the programming requirements when communicating with external hardware devices.

The communication is full duplex and operates as a slave/master type, where the devices can be either master or slave. Although the SPI interface specification can control multiple slave devices from a single master, these devices provided only one $\overline{\text{SCS}}$ pin. If the master needs to control multiple slave devices from a single master, the master can use I/O pin to select the slave devices.

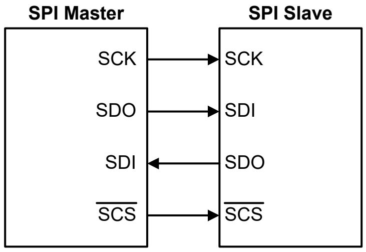
SPI Interface Operation

The SPI interface is a full duplex synchronous serial data link. It is a four line interface with pin names SDI, SDO, SCK and $\overline{\text{SCS}}$. Pins SDI and SDO are the Serial Data Input and Serial Data Output lines, SCK is the Serial Clock line and $\overline{\text{SCS}}$ is the Slave Select line. As the SPI interface pins are pin-shared with normal I/O pins and with the I^{2}C function pins, the SPI interface pins must first be selected by configuring the pin-shared function selection bits and setting the correct bits in the SIMC0 and SIMC2 registers. After the desired SPI configuration has been set it can be disabled or enabled using the SIMEN bit in the SIMC0 register. Communication between devices connected to the SPI interface is carried out in a slave/master mode with all data transfer initiations being implemented by the master. The Master also controls the clock signal. As the device only contains a single $\overline{\text{SCS}}$ pin only one slave device can be utilized. The $\overline{\text{SCS}}$ pin is controlled by software, set CSEN bit to 1 to enable $\overline{\text{SCS}}$ pin function, set CSEN bit to 0 the $\overline{\text{SCS}}$ pin will be floating state.

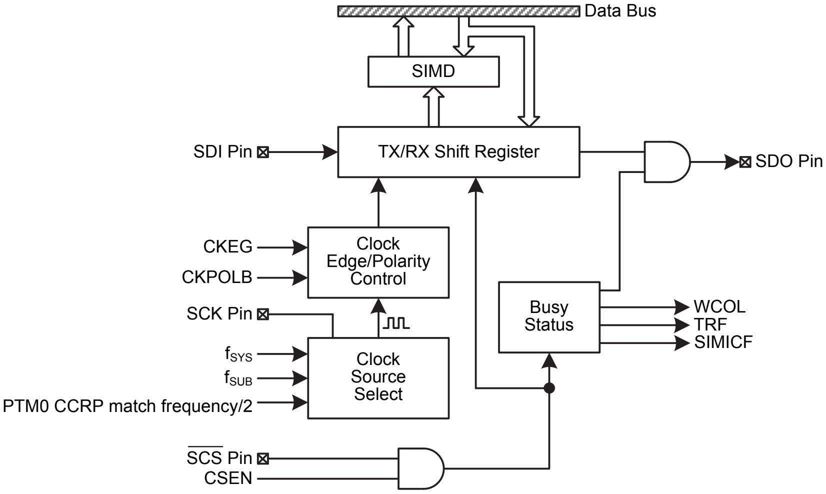
The SPI function in this device offers the following features:

- Full duplex synchronous data transfer
- Both Master and Slave modes
- LSB first or MSB first data transmission modes
- Transmission complete flag
- Rising or falling active clock edge

The status of the SPI interface pins is determined by a number of factors such as whether the device is in the master or slave mode and upon the condition of certain control bits such as CSEN and SIMEN.



SPI Master/Slave Connection



SPI Block Diagram

SPI Registers

There are three internal registers which control the overall operation of the SPI interface. These are the SIMD data register and two registers SIMC0 and SIMC2. Note that the SIMC1 register is only used by the I²C interface.

Register Name	Bit							
	7	6	5	4	3	2	1	0
SIMC0	SIM2	SIM1	SIM0	—	SIMDEB1	SIMDEB0	SIMEN	SIMICF
SIMC2	D7	D6	CKPOLB	CKEG	MLS	CSEN	WCOL	TRF
SIMD	D7	D6	D5	D4	D3	D2	D1	D0

SPI Registers List

SIMD Register

The SIMD register is used to store the data being transmitted and received. The same register is used by both the SPI and I²C functions. Before the device writes data to the SPI bus, the actual data to be transmitted must be placed in the SIMD register. After the data is received from the SPI bus, the device can read it from the SIMD register. Any transmission or reception of data from the SPI bus must be made via the SIMD register.

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	x	x	x	x	x	x	x	x

"x": unknown

There are also two control registers for the SPI interface, SIMC0 and SIMC2. Note that the SIMC2 register also has the name SIMA which is used by the I²C function. The SIMC1 register is not used by the SPI function, only by the I²C function. Register SIMC0 is used to control the enable/disable function and to set the data transmission clock frequency. Register SIMC2 is used for other control functions such as LSB/MSB selection, write collision flag, etc.

- **SIMC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	SIM2	SIM1	SIM0	—	SIMDEB1	SIMDEB0	SIMEN	SIMICF
R/W	R/W	R/W	R/W	—	R/W	R/W	R/W	R/W
POR	1	1	1	—	0	0	0	0

Bit 7~5 **SIM2~SIM0**: SIM Operating Mode Control
000: SPI master mode; SPI clock is $f_{SYS}/4$
001: SPI master mode; SPI clock is $f_{SYS}/16$
010: SPI master mode; SPI clock is $f_{SYS}/64$
011: SPI master mode; SPI clock is f_{SUB}
100: SPI master mode; SPI clock is PTM0 CCRP match frequency/2
101: SPI slave mode
110: I²C slave mode
111: Non SIM function

These bits setup the overall operating mode of the SIM function. As well as selecting if the I²C or SPI function, they are used to control the SPI Master/Slave selection and the SPI Master clock frequency. The SPI clock is a function of the system clock but can also be chosen to be sourced from PTM0. If the SPI Slave Mode is selected then the clock will be supplied by an external Master device.

Bit 4 Unimplemented, read as "0"

Bit 3~2 **SIMDEB1~SIMDEB0**: I²C Debounce Time Selection

The SIMDEB1~SIMDEB0 bits are only used in the I²C mode and the detailed definition is described in the I²C section.

Bit 1 **SIMEN**: SIM Enable Control
0: Disable
1: Enable

The bit is the overall on/off control for the SIM interface. When the SIMEN bit is cleared to zero to disable the SIM interface, the SDI, SDO, SCK and $\overline{SCS}$, or SDA and SCL lines will lose their SPI or I²C function and the SIM operating current will be reduced to a minimum value. When the bit is high the SIM interface is enabled. The SIM configuration option must have first enabled the SIM interface for this bit to be effective. If the SIM is configured to operate as an SPI interface via the SIM2~SIM0 bits, the contents of the SPI control registers will remain at the previous settings when the SIMEN bit changes from low to high and should therefore be first initialised by the application program. If the SIM is configured to operate as an I²C interface via the SIM2~SIM0 bits and the SIMEN bit changes from low to high, the contents of the I²C control bits such as HTX and TXAK will remain at the previous settings and should therefore be first initialised by the application program while the relevant I²C flags such as HCF, HAAS, HBB, SRW and RXAK will be set to their default states.

Bit 0 **SIMICF**: SIM SPI slave mode Incomplete Transfer Flag
0: SIM SPI slave mode incomplete condition not occurred
1: SIM SPI slave mode incomplete condition occurred

This bit is only available when the SIM is configured to operate in an SPI slave mode. If the SPI operates in the slave mode with the SIMEN and CSEN bits both being set to 1 but the $\overline{\text{SCS}}$ line is pulled high by the external master device before the SPI data transfer is completely finished, the SIMICF bit will be set to 1 together with the TRF bit. When this condition occurs, the corresponding interrupt will occur if the interrupt function is enabled. However, the TRF bit will not be set to 1 if the SIMICF bit is set to 1 by software application program.

- **SIMC2 Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	CKPOLB	CKEG	MLS	CSEN	WCOL	TRF
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **D7~D6**: Undefined bits
These bits can be read or written by the application program

Bit 5 **CKPOLB**: SPI clock line base condition selection
0: The SCK line will be high when the clock is inactive
1: The SCK line will be low when the clock is inactive

The CKPOLB bit determines the base condition of the clock line, if the bit is high, then the SCK line will be low when the clock is inactive. When the CKPOLB bit is low, then the SCK line will be high when the clock is inactive.

Bit 4 **CKEG**: SPI SCK clock active edge type selection
CKPOLB=0
0: SCK is high base level and data capture at SCK rising edge
1: SCK is high base level and data capture at SCK falling edge
CKPOLB=1
0: SCK is low base level and data capture at SCK falling edge
1: SCK is low base level and data capture at SCK rising edge

The CKEG and CKPOLB bits are used to setup the way that the clock signal outputs and inputs data on the SPI bus. These two bits must be configured before data transfer is executed otherwise an erroneous clock edge may be generated. The CKPOLB bit determines the base condition of the clock line, if the bit is high, then the SCK line will be low when the clock is inactive. When the CKPOLB bit is low, then the SCK line will be high when the clock is inactive. The CKEG bit determines active clock edge type which depends upon the condition of CKPOLB bit.

Bit 3 **MLS**: SPI data shift order
0: LSB first
1: MSB first

This is the data shift select bit and is used to select how the data is transferred, either MSB or LSB first. Setting the bit high will select MSB first and low for LSB first.

Bit 2 **CSEN**: SPI $\overline{\text{SCS}}$ pin control
0: Disable
1: Enable

The CSEN bit is used as an enable/disable for the $\overline{\text{SCS}}$ pin. If this bit is low, then the $\overline{\text{SCS}}$ pin will be disabled and placed into I/O pin or other pin-shared functions. If the bit is high, the $\overline{\text{SCS}}$ pin will be enabled and used as a select pin.

Bit 1 **WCOL**: SPI write collision flag
0: No collision
1: Collision

The WCOL flag is used to detect whether a data collision has occurred or not. If this bit is high, it means that data has been attempted to be written to the SIMD register during a data transfer operation. This writing operation will be ignored if data is being transferred. This bit can be cleared by the application program.

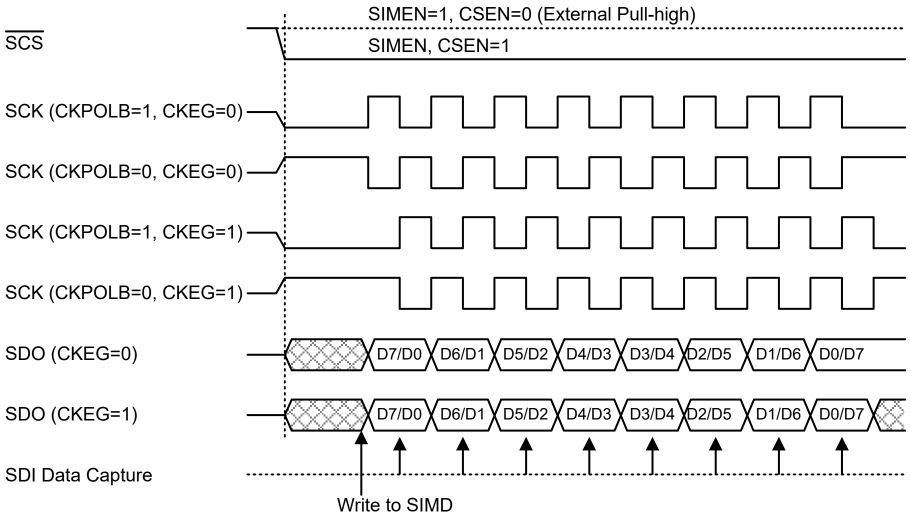
Bit 0 **TRF**: SPI Transmit/Receive complete flag
0: SPI data is being transferred
1: SPI data transfer is completed

The TRF bit is the Transmit/Receive Complete flag and is set to 1 automatically when an SPI data transfer is completed, but must cleared to 0 by the application program. It can be used to generate an interrupt.

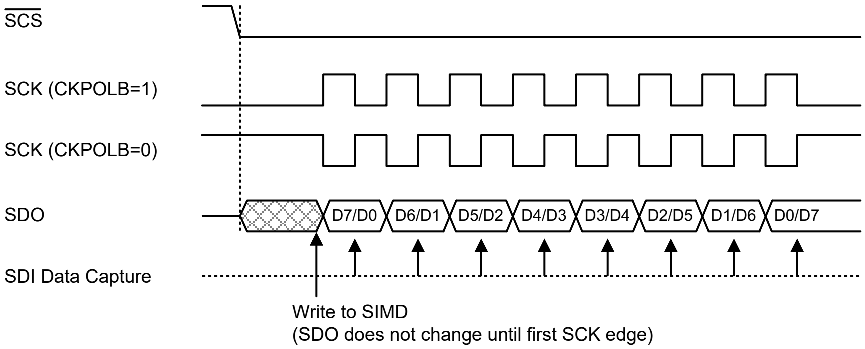
SPI Communication

After the SPI interface is enabled by setting the SIMEN bit high, then in the Master Mode, when data is written to the SIMD register, transmission/reception will begin simultaneously. When the data transfer is complete, the TRF flag will be set automatically, but must be cleared using the application program. In the Slave Mode, when the clock signal from the master has been received, any data in the SIMD register will be transmitted and any data on the SDI pin will be shifted into the SIMD register. The master should output a $\overline{SCS}$ signal to enable the slave devices before a clock signal is provided. The slave data to be transferred should be well prepared at the appropriate moment relative to the $\overline{SCS}$ signal depending upon the configurations of the CKPOLB bit and CKEG bit. The accompanying timing diagram shows the relationship between the slave data and $\overline{SCS}$ signal for various configurations of the CKPOLB and CKEG bits.

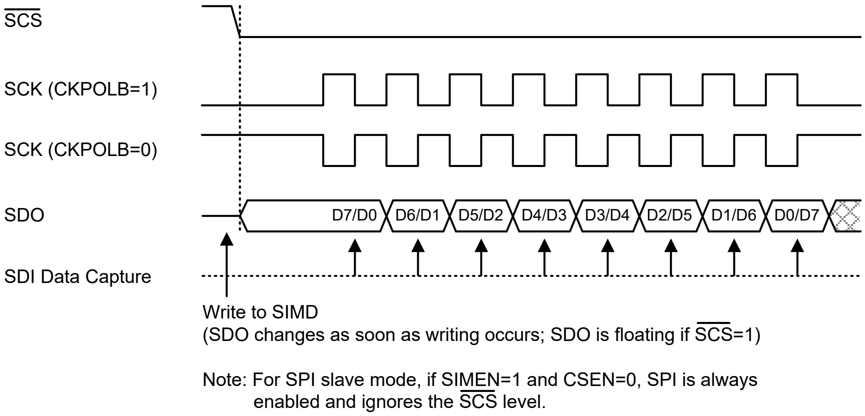
The SPI will continue to function even in certain IDLE Modes if the SPI clock source is active.



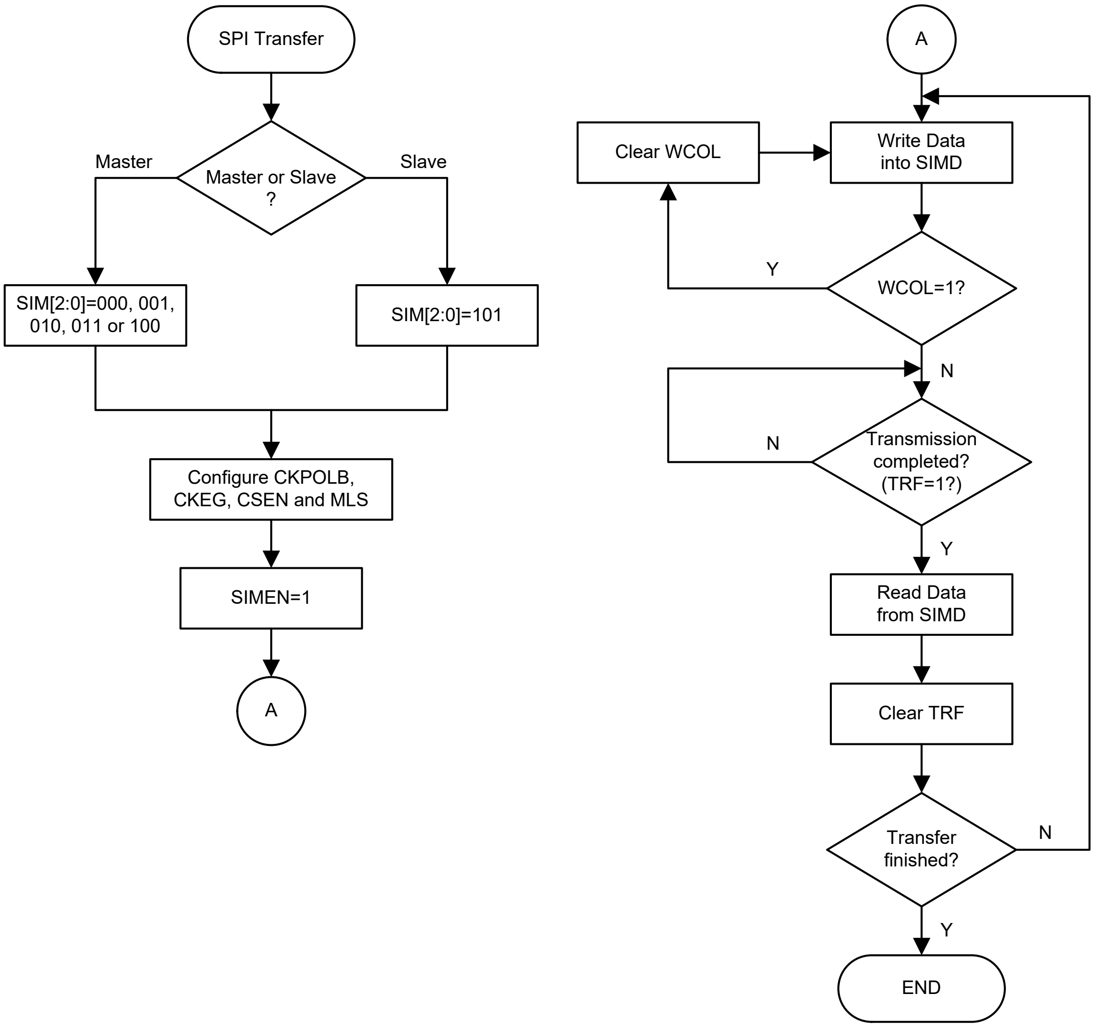
SPI Master Mode Timing



SPI Slave Mode Timing – CKEG = 0


SCS
SCK (CKPOLB=1)
SCK (CKPOLB=0)
SDO
SDI Data Capture
D7/D0
D6/D1
D5/D2
D4/D3
D3/D4
D2/D5
D1/D6
D0/D7
Write to SIMD
(SDO changes as soon as writing occurs; SDO is floating if SCS=1)
Note: For SPI slave mode, if SIMEN=1 and CSEN=0, SPI is always enabled and ignores the SCS level.

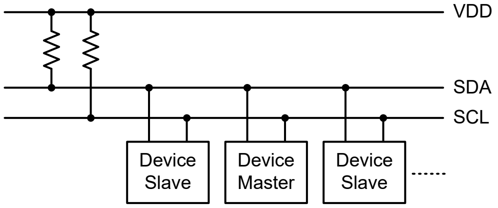
SPI Slave Mode Timing – CKEG = 1



SPI Transfer Control Flow Chart

I²C Interface

The I²C interface is used to communicate with external peripheral devices such as sensors, EEPROM memory etc. Originally developed by Philips, it is a two line low speed serial interface for synchronous serial data transfer. The advantage of only two lines for communication, relatively simple communication protocol and the ability to accommodate multiple devices on the same bus has made it an extremely popular interface type for many applications.

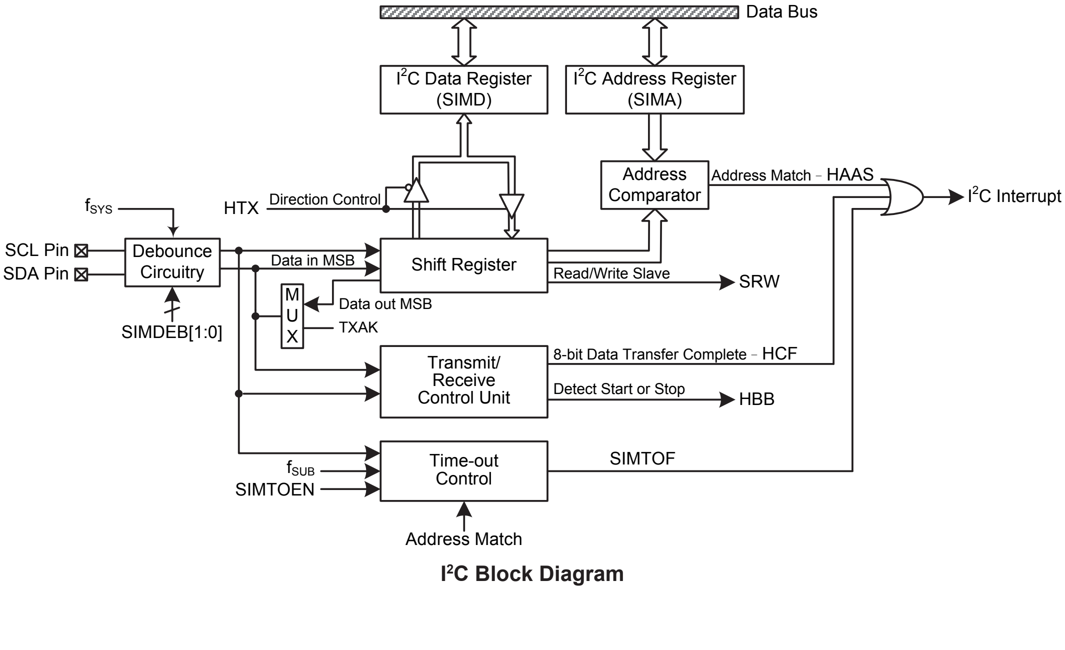


I²C Master Slave Bus Connection

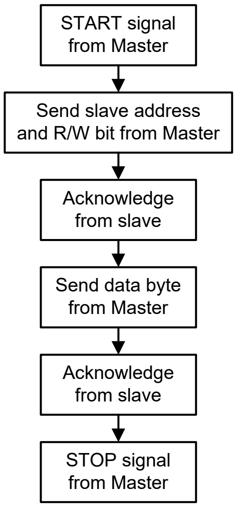
I²C interface Operation

The I²C serial interface is a two line interface, a serial data line, SDA, and serial clock line, SCL. As many devices may be connected together on the same bus, their outputs are both open drain types. For this reason it is necessary that external pull-high resistors are connected to these outputs. Note that no chip select line exists, as each device on the I²C bus is identified by a unique address which will be transmitted and received on the I²C bus.

When two devices communicate with each other on the bidirectional I²C bus, one is known as the master device and one as the slave device. Both master and slave can transmit and receive data; however, it is the master device that has overall control of the bus. For these devices, which only operate in slave mode, there are two methods of transferring data on the I²C bus, the slave transmit mode and the slave receive mode. The pull-high control function pin-shared with SCL/SDA pin is still applicable even if the I²C device is activated and the related internal pull-high register could be controlled by its corresponding pull-high control register.



I²C Block Diagram



I²C Operation

The SIMDEB1 and SIMDEB0 bits determine the debounce time of the I²C interface. This uses the system clock to in effect add a debounce time to the external clock to reduce the possibility of glitches on the clock line causing erroneous operation. The debounce time, if selected, can be chosen to be either 2 or 4 system clocks. To achieve the required I²C data transfer speed, there exists a relationship between the system clock, f_{SYS}, and the I²C debounce time. For either the I²C Standard or Fast mode operation, users must take care of the selected system clock frequency and the configured debounce time to match the criterion shown in the following table.

I²C Debounce Time Selection	I²C Standard Mode (100kHz)	I²C Fast Mode (400kHz)
No Devounce	f_{SYS} > 2MHz	f_{SYS} > 5MHz
2 system clock debounce	f_{SYS} > 4MHz	f_{SYS} > 10MHz
4 system clock debounce	f_{SYS} > 8MHz	f_{SYS} > 20MHz

I²C Minimum f_{SYS} Frequency Requirement

I²C Registers

There are three control registers associated with the I²C bus, SIMC0, SIMC1 and SIMA, and one data register, SIMD. The SIMD register, which is shown in the above SPI section, is used to store the data being transmitted and received on the I²C bus. Before the microcontroller writes data to the I²C bus, the actual data to be transmitted must be placed in the SIMD register. After the data is received from the I²C bus, the microcontroller can read it from the SIMD register. Any transmission or reception of data from the I²C bus must be made via the SIMD register.

Note that the SIMA register also has the name SIMC2 which is used by the SPI function. Bit SIMEN and bits SIM2~SIM0 in register SIMC0 are used by the I²C interface.

Register Name	Bit							
	7	6	5	4	3	2	1	0
SIMC0	SIM2	SIM1	SIM0	—	SIMDEB1	SIMDEB0	SIMEN	SIMICF
SIMC1	HCF	HAAS	HBB	HTX	TXAK	SRW	IAMWU	RXAK
SIMA	SIMA6	SIMA5	SIMA4	SIMA3	SIMA2	SIMA1	SIMA0	—
SIMD	D7	D6	D5	D4	D3	D2	D1	D0
SIMTOC	SIMTOEN	SIMTOF	SIMTOS5	SIMTOS4	SIMTOS3	SIMTOS2	SIMTOS1	SIMTOS0

I²C Registers List

- **SIMD Register**

The SIMD register is used to store the data being transmitted and received. The same register is used by both the SPI and I²C functions. Before the device writes data to the I²C bus, the actual data to be transmitted must be placed in the SIMD register. After the data is received from the I²C bus, the device can read it from the SIMD register. Any transmission or reception of data from the I²C bus must be made via the SIMD register.

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	x	x	x	x	x	x	x	x

"x": unknown

- **SIMA Register**

The SIMA register is also used by the SPI interface but has the name SIMC2. The SIMA register is the location where the 7-bit slave address of the slave device is stored. Bits 7~1 of the SIMA register define the device slave address. Bit 0 is not implemented.

When a master device, which is connected to the I²C bus, sends out an address, which matches the slave address in the SIMA register, the slave device will be selected. Note that the SIMA register is the same register address as SIMC2 which is used by the SPI interface.

Bit	7	6	5	4	3	2	1	0
Name	SIMA6	SIMA5	SIMA4	SIMA3	SIMA2	SIMA1	SIMA0	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	x	x	x	x	x	x	x	0

"x": unknown

Bit 7~1 **SIMA6~SIMA0**: I²C slave address
SIMA6~SIMA0 is the I²C slave address bit 6 ~ bit 0

Bit 0 **D0**: reserved, can be read/written by application program

There are also two control registers for the I²C interface, SIMC0 and SIMC1. The register SIMC0 is used to control the enable/disable function and to set the data transmission clock frequency. The SIMC1 register contains the relevant flags which are used to indicate the I²C communication status.

- **SIMC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	SIM2	SIM1	SIM0	—	SIMDEB1	SIMDEB0	SIMEN	SIMICF
R/W	R/W	R/W	R/W	—	R/W	R/W	R/W	R/W
POR	1	1	1	—	0	0	0	0

Bit 7~5 **SIM2~SIM0**: SIM Operating Mode Control
000: SPI master mode; SPI clock is $f_{SYS}/4$
001: SPI master mode; SPI clock is $f_{SYS}/16$
010: SPI master mode; SPI clock is $f_{SYS}/64$
011: SPI master mode; SPI clock is f_{SUB}
100: SPI master mode; SPI clock is PTM0 CCRP match frequency/2
101: SPI slave mode
110: I²C slave mode
111: Non SIM function

These bits setup the overall operating mode of the SIM function. As well as selecting if the I²C or SPI function, they are used to control the SPI Master/Slave selection and the SPI Master clock frequency. The SPI clock is a function of the system clock but can also be chosen to be sourced from PTM0. If the SPI Slave Mode is selected then the clock will be supplied by an external Master device.

Bit 4 Unimplemented, read as “0”

Bit 3~2 **SIMDEB1~SIMDEB0**: I²C Debounce Time Selection
00: No debounce
01: 2 system clock debounce
1x: 4 system clock debounce

These bits are used to select the I²C debounce time when the SIM is configured as the I²C interface function by setting the SIM2~SIM0 bits to “110”.

Bit 1 **SIMEN**: SIM Enable Control
0: Disable
1: Enable

The bit is the overall on/off control for the SIM interface. When the SIMEN bit is cleared to zero to disable the SIM interface, the SDI, SDO, SCK and $\overline{SCS}$, or SDA and SCL lines will lose their SPI or I²C function and the SIM operating current will be reduced to a minimum value. When the bit is high the SIM interface is enabled. The SIM configuration option must have first enabled the SIM interface for this bit to be effective.If the SIM is configured to operate as an SPI interface via the SIM2~SIM0 bits, the contents of the SPI control registers will remain at the previous settings when the SIMEN bit changes from low to high and should therefore be first initialised by the application program. If the SIM is configured to operate as an I²C interface via the SIM2~SIM0 bits and the SIMEN bit changes from low to high, the contents of the I²C control bits such as HTX and TXAK will remain at the previous settings and should therefore be first initialised by the application program while the relevant I²C flags such as HCF, HAAS, HBB, SRW and RXAK will be set to their default states.

Bit 0 **SIMICF**: SIM SPI Incomplete Flag

The SIMICF bit is only used in the SPI mode and the detailed definition is described in the SPI section.

- **SIMC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	HCF	HAAS	HBB	HTX	TXAK	SRW	IAMWU	RXAK
R/W	R	R	R	R/W	R/W	R/W	R/W	R
POR	1	0	0	0	0	0	0	1

Bit 7 **HCF**: I²C Bus data transfer completion flag
0: Data is being transferred
1: Completion of an 8-bit data transfer

The HCF flag is the data transfer flag. This flag will be zero when data is being transferred. Upon completion of an 8-bit data transfer the flag will go high and an interrupt will be generated.

Bit 6 **HAAS**: I²C Bus data transfer completion flag
0: Not address match
1: Address match

The HAAS flag is the address match flag. This flag is used to determine if the slave device address is the same as the master transmit address. If the addresses match then this bit will be high, if there is no match then the flag will be low.

Bit 5 **HBB**: I²C Bus busy flag
0: I²C Bus is not busy
1: I²C Bus is busy

The HBB flag is the I²C busy flag. This flag will be “1” when the I²C bus is busy which will occur when a START signal is detected. The flag will be set to “0” when the bus is free which will occur when a STOP signal is detected.

Bit 4 **HTX**: I²C slave device transmitter/receiver selection
0: Slave device is the receiver
1: Slave device is the transmitter

Bit 3 **TXAK**: I^{2}C bus transmit acknowledge flag
0: Slave send acknowledge flag
1: Slave does not send acknowledge flag

The TXAK bit is the transmit acknowledge flag. After the slave device receipt of 8-bits of data, this bit will be transmitted to the bus on the 9th clock from the slave device. The slave device must always set TXAK bit to "0" before further data is received.

Bit 2 **SRW**: I^{2}C slave read/write flag
0: Slave device should be in receive mode
1: Slave device should be in transmit mode

The SRW flag is the I^{2}C Slave Read/Write flag. This flag determines whether the master device wishes to transmit or receive data from the I^{2}C bus. When the transmitted address and slave address is match, that is when the HAAS flag is set high, the slave device will check the SRW flag to determine whether it should be in transmit mode or receive mode. If the SRW flag is high, the master is requesting to read data from the bus, so the slave device should be in transmit mode. When the SRW flag is zero, the master will write data to the bus, therefore the slave device should be in receive mode to read this data.

Bit 1 **IAMWU**: I^{2}C Address Match Wake-Up control
0: Disable
1: Enable – must be cleared by the application program after wake-up

This bit should be set to 1 to enable the I^{2}C address match wake up from the SLEEP or IDLE Mode. If the IAMWU bit has been set before entering either the SLEEP or IDLE mode to enable the I^{2}C address match wake up, then this bit must be cleared by the application program after wake-up to ensure correction device operation.

Bit 0 **RXAK**: I^{2}C bus receive acknowledge flag
0: Slave receives acknowledge flag
1: Slave does not receive acknowledge flag

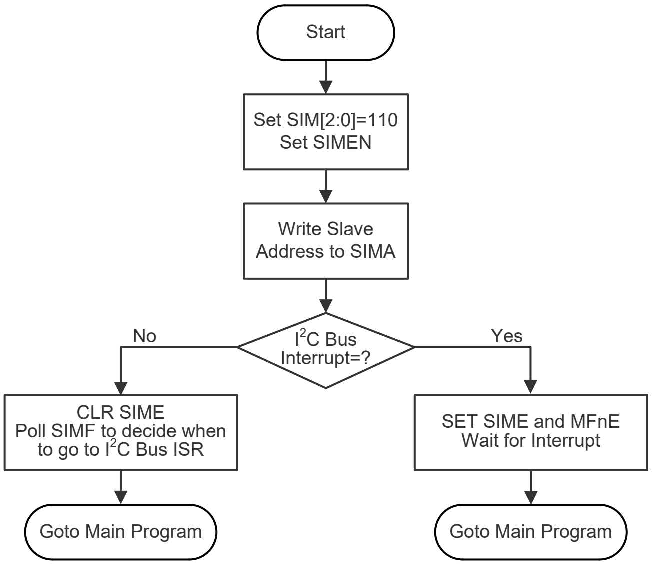
The RXAK flag is the receiver acknowledge flag. When the RXAK flag is "0", it means that a acknowledge signal has been received at the 9th clock, after 8 bits of data have been transmitted. When the slave device in the transmit mode, the slave device checks the RXAK flag to determine if the master receiver wishes to receive the next byte. The slave transmitter will therefore continue sending out data until the RXAK flag is "1". When this occurs, the slave transmitter will release the SDA line to allow the master to send a STOP signal to release the I^{2}C Bus.

I^{2}C Bus Communication

Communication on the I^{2}C bus requires four separate steps, a START signal, a slave device address transmission, a data transmission and finally a STOP signal. When a START signal is placed on the I^{2}C bus, all devices on the bus will receive this signal and be notified of the imminent arrival of data on the bus. The first seven bits of the data will be the slave address with the first bit being the MSB. If the address of the slave device matches that of the transmitted address, the HAAS bit in the SIMC1 register will be set and an I^{2}C interrupt will be generated. After entering the interrupt service routine, the slave device must first check the condition of the HAAS and SIMTOF bits to determine whether the interrupt source originates from an address match, 8-bit data transfer completion or I^{2}C bus time-out occurrence. During a data transfer, note that after the 7-bit slave address has been transmitted, the following bit, which is the 8th bit, is the read/write bit whose value will be placed in the SRW bit. This bit will be checked by the slave device to determine whether to go into transmit or receive mode. Before any transfer of data to or from the I^{2}C bus, the microcontroller must initialise the bus; the following are steps to achieve this:

- Step 1
  Set the SIM2~SIM0 bits to "110" and SIMEN bit to "1" in the SIMC0 register to enable the I^{2}C bus.

- Step 2
  Write the slave address of the device to the I²C bus address register SIMA.
- Step 3
  Set the SIME and the related Multi-Function interrupt enable bit of the interrupt control register to enable the SIM interrupt and Multi-function interrupt.



I²C Bus Initialisation Flow Chart

I²C Bus Start Signal

The START signal can only be generated by the master device connected to the I²C bus and not by the slave device. This START signal will be detected by all devices connected to the I²C bus. When detected, this indicates that the I²C bus is busy and therefore the HBB bit will be set. A START condition occurs when a high to low transition on the SDA line takes place when the SCL line remains high.

I²C Slave Address

The transmission of a START signal by the master will be detected by all devices on the I²C bus. To determine which slave device the master wishes to communicate with, the address of the slave device will be sent out immediately following the START signal. All slave devices, after receiving this 7-bit address data, will compare it with their own 7-bit slave address. If the address sent out by the master matches the internal address of the microcontroller slave device, then an internal I²C bus interrupt signal will be generated. The next bit following the address, which is the 8th bit, defines the read/write status and will be saved to the SRW bit of the SIMC1 register. The slave device will then transmit an acknowledge bit, which is a low level, as the 9th bit. The slave device will also set the status flag HAAS when the addresses match.

As an SIM I²C bus interrupt can come from three sources, when the program enters the interrupt subroutine, the HAAS and SIMTOF bits should be examined to see whether the interrupt source has come from a matching slave address, the completion of a data byte transfer or the I²C bus time-out occurrence. When a slave address is matched, the devices must be placed in either the transmit mode and then write data to the SIMD register, or in the receive mode where it must implement a dummy read from the SIMD register to release the SCL line.

I²C Bus Read/Write Signal

The SRW bit in the SIMC1 register defines whether the master device wishes to read data from the I²C bus or write data to the I²C bus. The slave device should examine this bit to determine if it is to be a transmitter or a receiver. If the SRW flag is "1" then this indicates that the master device wishes to read data from the I²C bus, therefore the slave device must be setup to send data to the I²C bus as a transmitter. If the SRW flag is "0" then this indicates that the master wishes to send data to the I²C bus, therefore the slave device must be setup to read data from the I²C bus as a receiver.

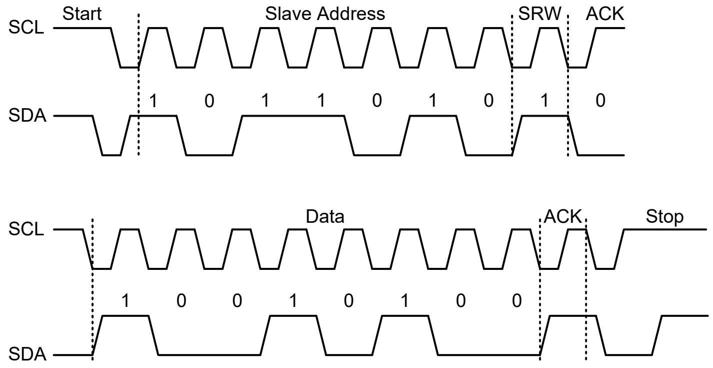
I²C Bus Slave Address Acknowledge Signal

After the master has transmitted a calling address, any slave device on the I²C bus, whose own internal address matches the calling address, must generate an acknowledge signal. The acknowledge signal will inform the master that a slave device has accepted its calling address. If no acknowledge signal is received by the master then a STOP signal must be transmitted by the master to end the communication. When the HAAS flag is high, the addresses have matched and the slave device must check the SRW flag to determine if it is to be a transmitter or a receiver. If the SRW flag is high, the slave device should be setup to be a transmitter so the HTX bit in the SIMC1 register should be set to "1". If the SRW flag is low, then the microcontroller slave device should be setup as a receiver and the HTX bit in the SIMC1 register should be set to "0".

I²C Bus Data and Acknowledge Signal

The transmitted data is 8-bits wide and is transmitted after the slave device has acknowledged receipt of its slave address. The order of serial bit transmission is the MSB first and the LSB last. After receipt of 8-bits of data, the receiver must transmit an acknowledge signal, level "0", before it can receive the next data byte. If the slave transmitter does not receive an acknowledge bit signal from the master receiver, then the slave transmitter will release the SDA line to allow the master to send a STOP signal to release the I²C Bus. The corresponding data will be stored in the SIMD register. If setup as a transmitter, the slave device must first write the data to be transmitted into the SIMD register. If setup as a receiver, the slave device must read the transmitted data from the SIMD register.

When the slave receiver receives the data byte, it must generate an acknowledge bit, known as TXAK, on the 9th clock. The slave device, which is setup as a transmitter will check the RXAK bit in the SIMC1 register to determine if it is to send another data byte, if not then it will release the SDA line and await the receipt of a STOP signal from the master.

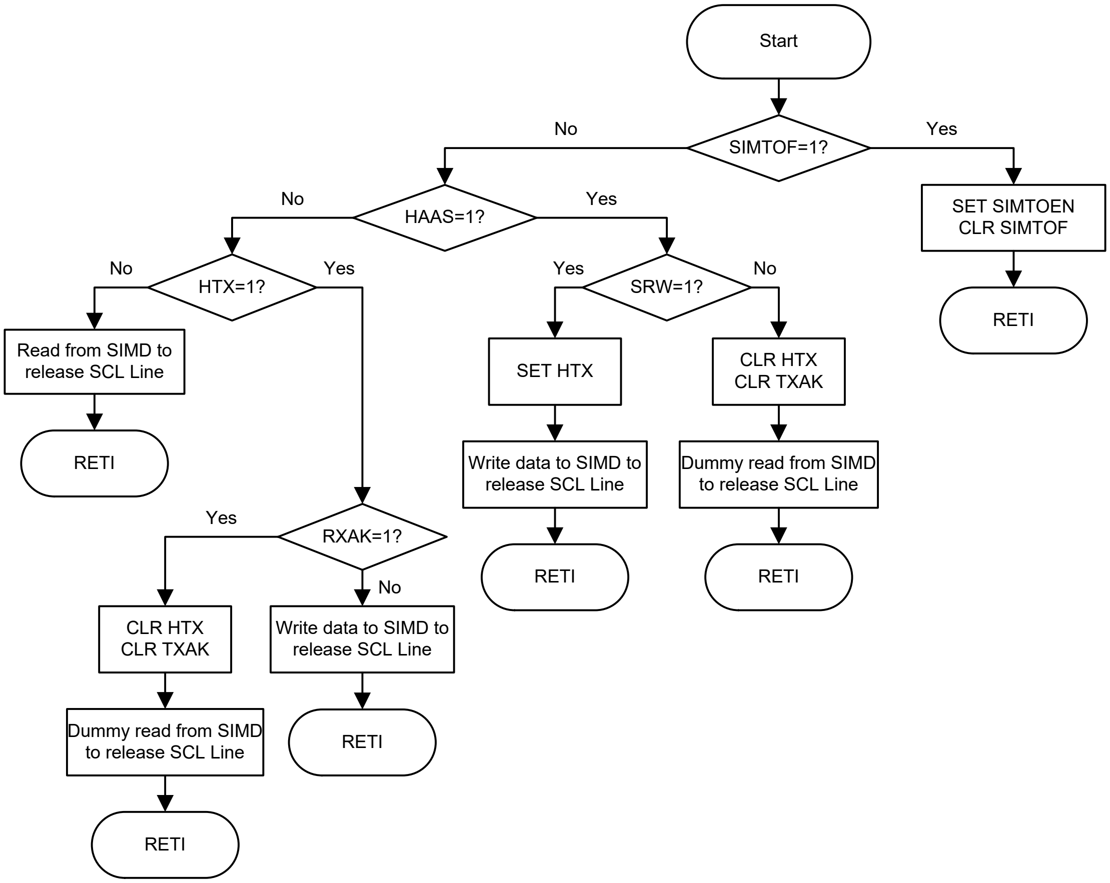


S=Start (1 bit)
SA=Slave Address (7 bits)
SR=SRW bit (1 bit)
M=Slave device send acknowledge bit (1 bit)
D=Data (8 bits)
A=ACK (RXAK bit for transmitter, TXAK bit for receiver, 1 bit)
P=Stop (1 bit)

S	SA	SR	M	D	A	D	A	⋯⋯	S	SA	SR	M	D	A	D	A	⋯⋯	P

Note: *When a slave address is matched, the devices must be placed in either the transmit mode and then write data to the SIMD register, or in the receive mode where it must implement a dummy read from the SIMD register to release the SCL line.

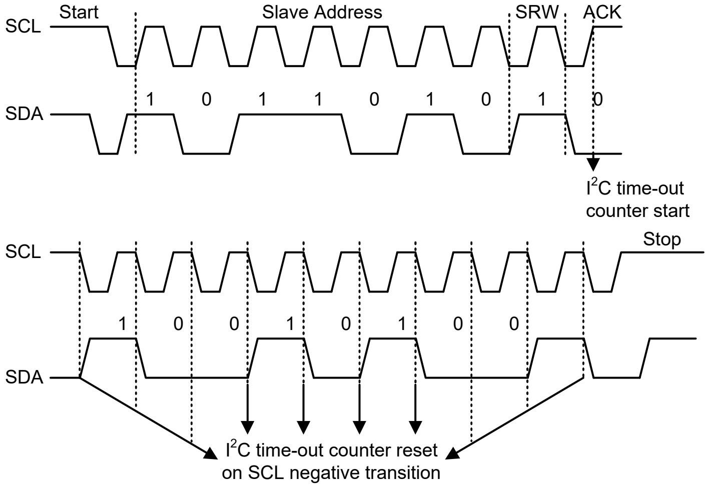
I²C Communication Timing Diagram


Start
SIMTOF=1?
No
Yes
SET SIMTOEN
CLR SIMTOF
RETI
HAAS=1?
No
Yes
HTX=1?
No
Yes
Read from SIMD to release SCL Line
RETI
SRW=1?
Yes
No
SET HTX
CLR HTX
CLR TXAK
Write data to SIMD to release SCL Line
Dummy read from SIMD to release SCL Line
RETI
RETI
RXAK=1?
Yes
No
CLR HTX
CLR TXAK
Write data to SIMD to release SCL Line
Dummy read from SIMD to release SCL Line
RETI
RETI

I²C Bus ISR Flow Chart

I²C Time-out Control

In order to reduce the I²C lockup problem due to reception of erroneous clock sources, a time-out function is provided. If the clock source connected to the I²C bus is not received for a while, then the I²C circuitry and registers will be reset after a certain time-out period. The time-out counter starts to count on an I²C bus "START" & "address match" condition, and is cleared by an SCL falling edge. Before the next SCL falling edge arrives, if the time elapsed is greater than the time-out period specified by the SIMTOC register, then a time-out condition will occur. The time-out function will stop when an I²C "STOP" condition occurs.


SCL
Start
Slave Address
SRW
ACK
SDA
1
0
1
1
0
1
0
1
0
I²C time-out counter start
SCL
Stop
1
0
0
1
0
1
0
0
SDA
I²C time-out counter reset on SCL negative transition

I²C Time-out

When an I²C time-out counter overflow occurs, the counter will stop and the SIMTOEN bit will be cleared to zero and the SIMTOF bit will be set high to indicate that a time-out condition has occurred. The time-out condition will also generate an interrupt which uses the SIM interrupt vector. When an I²C time-out occurs, the I²C internal circuitry will be reset and the registers will be reset into the following condition:

Register	After I²C Time-out
SIMD, SIMA, SIMC0	No change
SIMC1	Reset to POR condition

I²C Register after Time-out

The SIMTOF flag can be cleared by the application program. There are 64 time-out period selections which can be selected using the SIMTOS bits in the SIMTOC register. The time-out duration is calculated by the formula: ((1~64) × (32/f_{SUB})). This gives a time-out period which ranges from about 1ms to 64ms.

- **SIMTOC Register**

Bit	7	6	5	4	3	2	1	0
Name	SIMTOEN	SIMTOF	SIMTOS5	SIMTOS4	SIMTOS3	SIMTOS2	SIMTOS1	SIMTOS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **SIMTOEN**: SIM I²C Time-out control
0: Disable
1: Enable

Bit 6 **SIMTOF**: SIM I^{2}C Time-out flag
0: No time-out occurred
1: Time-out occurred

Bit 5~0 **SIMTOS5~SIMTOS0**: SIM I^{2}C Time-out period selection
I^{2}C Time-out clock source is $f_{SUB}/32$
I^{2}C Time-out period is equal to $(SIMTOS[5:0] + 1) \times \frac{32}{f_{SUB}}$

Serial Interface – SPIA

The device contains an independent SPI function. It is important not to confuse this independent SPI function with the additional one contained within the combined SIM function, which is described in another section of this datasheet. This independent SPI function will carry the name SPIA to distinguish it from the other one in the SIM.

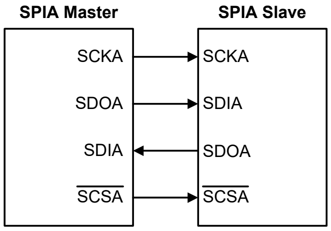
This SPIA interface is often used to communicate with external peripheral devices such as sensors, Flash or EEPROM memory devices, etc. Originally developed by Motorola, the four line SPI interface is a synchronous serial data interface that has a relatively simple communication protocol simplifying the programming requirements when communicating with external hardware devices.

The communication is full duplex and operates as a slave/master type, where the device can be either master or slave. Although the SPIA interface specification can control multiple slave devices from a single master, this device is provided only one $\overline{\text{SCSA}}$ pin. If the master needs to control multiple slave devices from a single master, the master can use I/O pins to select the slave devices.

SPIA Interface Operation

The SPIA interface is a full duplex synchronous serial data link. It is a four line interface with pin names SDIA, SDOA, SCKA and $\overline{\text{SCSA}}$. Pins SDIA and SDOA are the Serial Data Input and Serial Data Output lines, SCKA is the Serial Clock line and $\overline{\text{SCSA}}$ is the Slave Select line. As the SPIA interface pins are pin-shared with other functions, the SPIA interface pins must first be selected by configuring the corresponding selection bits in the pin-shared function selection registers. The SPIA interface function is disabled or enabled using the SPIAEN bit in the SPIAC0 register. Communication between devices connected to the SPIA interface is carried out in a slave/master mode with all data transfer initiations being implemented by the master. The master also controls the clock/signal. As the device only contains a single $\overline{\text{SCSA}}$ pin only one slave device can be utilised.

The $\overline{\text{SCSA}}$ pin is controlled by the application program, set the SACSEN bit to "1" to enable the $\overline{\text{SCSA}}$ pin function and clear the SACSEN bit to "0" to place the $\overline{\text{SCSA}}$ pin into an floating state.

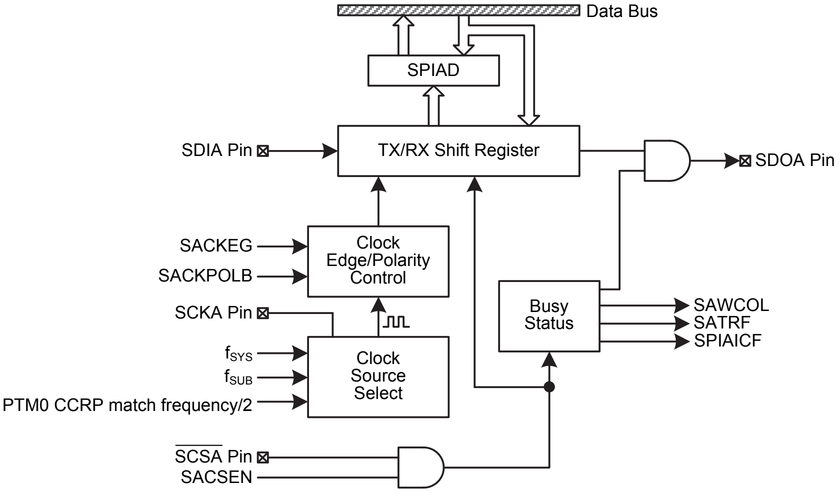


SPIA Master/Slave Connection

The SPIA Serial Interface function includes the following features:

- Full-duplex synchronous data transfer
- Both Master and Slave mode
- LSB first or MSB first data transmission modes
- Transmission complete flag
- Rising or falling active clock edge

The status of the SPIA interface pins is determined by a number of factors such as whether the device is in the master or slave mode and upon the condition of certain control bits such as SACSEN and SPIAEN.



SPIA Block Diagram

SPIA Registers

There are three internal registers which control the overall operation of the SPIA interface. These are the SPIAD data register and two registers SPIAC0 and SPIAC1.

Register Name	Bit							
	7	6	5	4	3	2	1	0
SPIAC0	SASPI2	SASPI1	SASPI0	—	—	—	SPIAEN	SPIAICF
SPIAC1	—	—	SACKPOLB	SACKEG	SAMLS	SACSEN	SAWCOL	SATRF
SPIAD	D7	D6	D5	D4	D3	D2	D1	D0

SPIA Registers List

SPIAD Register

The SPIAD register is used to store the data being transmitted and received. Before the device writes data to the SPIA bus, the actual data to be transmitted must be placed in the SPIAD register. After the data is received from the SPIA bus, the device can read it from the SPIAD register. Any transmission or reception of data from the SPIA bus must be made via the SPIAD register.

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	x	x	x	x	x	x	x	x

"x": unknown

There are also two control registers for the SPIA interface, SPIAC0 and SPIAC1. Register SPIAC0 is used to control the enable/disable function and to set the data transmission clock frequency. Register SPIAC1 is used for other control functions such as LSB/MSB selection, write collision flag, etc.

- **SPIAC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	SASPI2	SASPI1	SASPI0	—	—	—	SPIAEN	SPIAICF
R/W	R/W	R/W	R/W	—	—	—	R/W	R/W
POR	1	1	1	—	—	—	0	0

Bit 7~5 **SASPI2~SASPI0**: SPIA Master/Slave clock select
000: SPIA master mode with clock $f_{SYS}/4$
001: SPIA master mode with clock $f_{SYS}/16$
010: SPIA master mode with clock $f_{SYS}/64$
011: SPIA master mode with clock f_{SUB}
100: SPIA master mode with clock PTM0 CCRP match frequency/2
101: SPIA slave mode
11x: SPIA disable

These bits setup the SPIA Master/Slave mode and select the SPIA Master clock frequency. The SPIA clock is a function of the system clock but can also be chosen to be sourced from PTM0. If the SPIA Slave Mode is selected then the clock will be supplied by an external Master device.

Bit 4~2 Unimplemented, read as “0”

Bit 1 **SPIAEN**: SPIIA Enable Control
0: Disable
1: Enable

The bit is the overall on/off control for the SPIA interface. When the SPIAEN bit is cleared to zero to disable the SPIA interface, the SDIA, SDOA, SCKA and $\overline{\text{SCSA}}$ lines will lose the SPI function and the SPIA operating current will be reduced to a minimum value. When the bit is high the SPIA interface is enabled.

Bit 0 **SPIAICF**: SPIA Incomplete Flag
0: SPIA incomplete condition not occurred
1: SPIA incomplete condition occurred

This bit is only available when the SPIA is configured to operate in an SPIA slave mode. If the SPIA operates in the slave mode with the SPIAEN and SACSEN bits both being set to 1 but the $\overline{\text{SCSA}}$ line is pulled high by the external master device before the SPIA data transfer is completely finished, the SPIAICF bit will be set to 1 together with the SATRF bit. When this condition occurs, the corresponding interrupt will occur if the interrupt function is enabled. However, the SATRF bit will not be set to 1 if the SPIAICF bit is set to 1 by software application program.

- **SPIAC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	SACKPOLB	SACKEG	SAMLS	SACSEN	SAWCOL	SATRF
R/W	—	—	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	—	0	0	0	0	0	0

Bit 7~6 Unimplemented, read as “0”

Bit 5 **SACKPOLB**: SPIA clock line base condition selection
0: The SCKA line will be high when the clock is inactive.
1: The SCKA line will be low when the clock is inactive.

The SACKPOLB bit determines the base condition of the clock line, if the bit is high, then the SCKA line will be low when the clock is inactive. When the SACKPOLB bit is low, then the SCKA line will be high when the clock is inactive.

Bit 4 **SACKEG**: SPIA SCKA clock active edge type selection
SACKPOLB=0
0: SCKA is high base level and data capture at SCKA rising edge
1: SCKA is high base level and data capture at SCKA falling edge

SACKPOLB=1
0: SCKA is low base level and data capture at SCKA falling edge
1: SCKA is low base level and data capture at SCKA rising edge

The SACKEG and SACKPOLB bits are used to setup the way that the clock signal outputs and inputs data on the SPIA bus. These two bits must be configured before data transfer is executed otherwise an erroneous clock edge may be generated. The SACKPOLB bit determines the base condition of the clock line, if the bit is high, then the SCKA line will be low when the clock is inactive. When the SACKPOLB bit is low, then the SCKA line will be high when the clock is inactive. The SACKEG bit determines active clock edge type which depends upon the condition of SACKPOLB bit.

Bit 3 **SAMLS**: SPIA data shift order
0: LSB first
1: MSB first

This is the data shift select bit and is used to select how the data is transferred, either MSB or LSB first. Setting the bit high will select MSB first and low for LSB first.

Bit 2 **SACSEN**: SPIA $\overline{\text{SCSA}}$ pin control
0: Disable
1: Enable

The SACSEN bit is used as an enable/disable for the $\overline{\text{SCSA}}$ pin. If this bit is low, then the $\overline{\text{SCSA}}$ pin function will be disabled and can be placed into I/O pin or other pin-shared functions. If the bit is high, the $\overline{\text{SCSA}}$ pin will be enabled and used as a select pin.

Bit 1 **SAWCOL**: SPIA write collision flag
0: No collision
1: Collision

The SAWCOL flag is used to detect whether a data collision has occurred or not. If this bit is high, it means that data has been attempted to be written to the SPIAD register during a data transfer operation. This writing operation will be ignored if data is being transferred. This bit can be cleared by the application program.

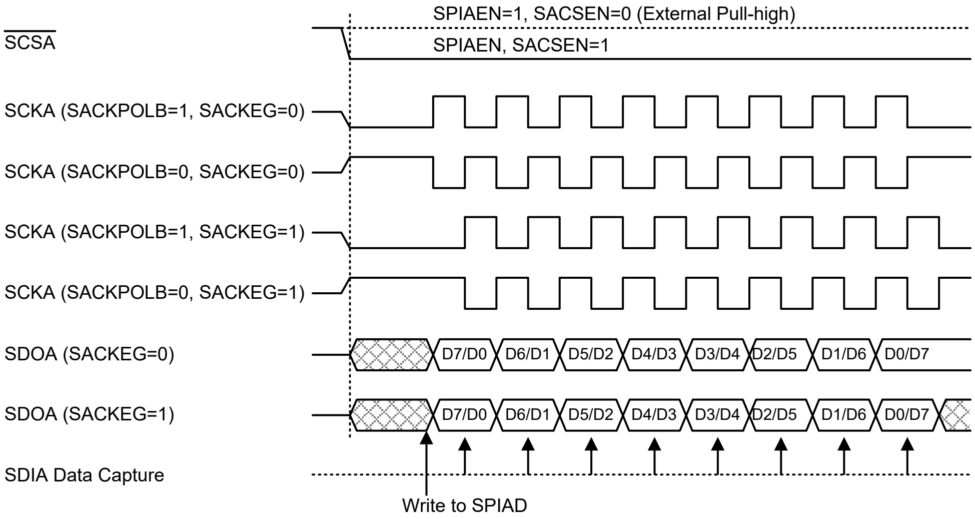
Bit 0 **SATRF**: SPIA Transmit/Receive complete flag
0: SPIA data is being transferred
1: SPIA data transfer is completed

The SATRF bit is the Transmit/Receive Complete flag and is set to 1 automatically when an SPIA data transfer is completed, but must cleared to 0 by the application program. It can be used to generate an interrupt.

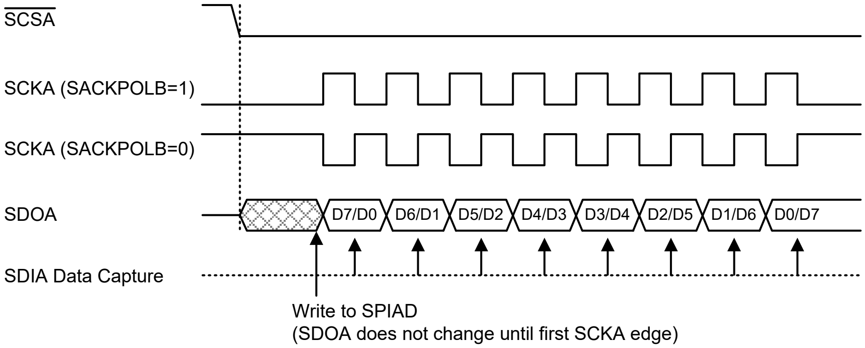
SPIA Communication

After the SPIA interface is enabled by setting the SPIAEN bit high, then in the Master Mode, when data is written to the SPIAD register, transmission/reception will begin simultaneously. When the data transfer is complete, the SATRF flag will be set automatically, but must be cleared using the application program. In the Slave Mode, when the clock signal from the master has been received, any data in the SPIAD register will be transmitted and any data on the SDIA pin will be shifted into the SPIAD registers.

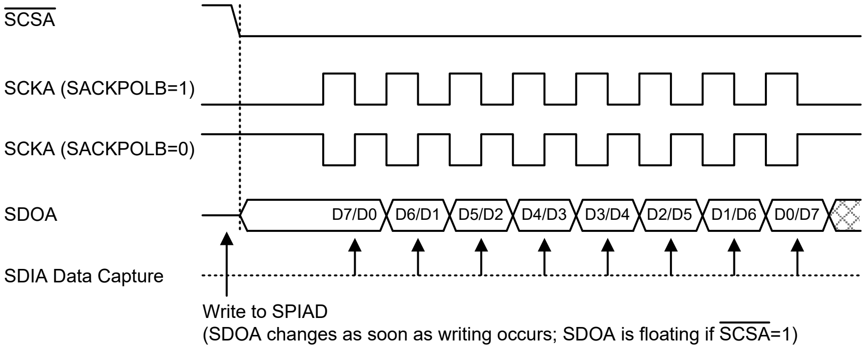
The master should output a $\overline{\text{SCSA}}$ signal to enable the slave device before a clock signal is provided. The slave data to be transferred should be well prepared at the appropriate moment relative to the $\overline{\text{SCSA}}$ signal depending upon the configurations of the SACKPOLB bit and SACKEG bit. The accompanying timing diagram shows the relationship between the slave data and $\overline{\text{SCSA}}$ signal for various configurations of the SACKPOLB and SACKEG bits. The SPIA will continue to function in certain IDLE Modes if the SPIA clock source is active.



SPIA Master Mode Timing

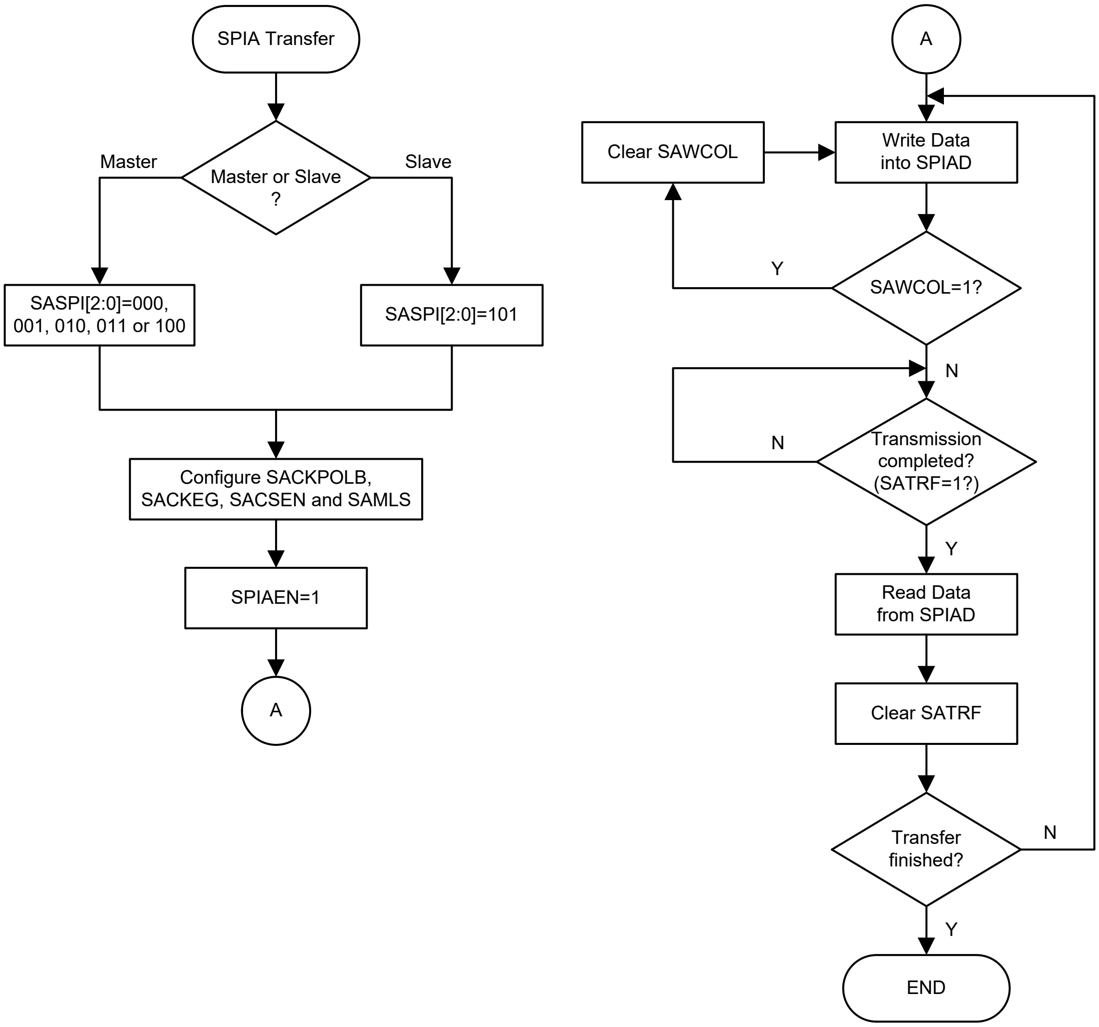


SPIA Slave Mode Timing – SACKEG=0



Note: For SPIA slave mode, if SPIAEN=1 and SACSEN=0, SPIA is always enabled and ignores the SCSA level.

SPIA Slave Mode Timing – SACKEG=1



SPIA Transfer Control Flow Chart

SPIA Bus Enable/Disable

To enable the SPIA bus, set SACSEN=1 and $\overline{\text{SCSA}}$=0, then wait for data to be written into the SPIAD (TXRX buffer) register. For the Master Mode, after data has been written to the SPIAD (TXRX buffer) register, then transmission or reception will start automatically. When all the data has been transferred the SATRF bit should be set. For the Slave Mode, when clock pulses are received on SCKA, data in the TXRX buffer will be shifted out or data on SDIA will be shifted in.

When the SPIA bus is disabled, the SCKA, SDIA, SDOA and $\overline{\text{SCSA}}$ pins can become I/O pins or other pin-shared functions using the corresponding pin-shared function selection bits.

SPIA Operation

All communication is carried out using the 4-line interface for either Master or Slave Mode.

The SACSEN bit in the SPIAC1 register controls the overall function of the SPIA interface. Setting this bit high will enable the SPIA interface by allowing the $\overline{\text{SCSA}}$ line to be active, which can then be used to control the SPIA interface. If the SACSEN bit is low, the SPIA interface will be disabled and the $\overline{\text{SCSA}}$ line will be floating and can therefore not be used for control of the SPIA interface. If the SACSEN bit and the SPIAEN bit in the SPIAC0 register are set high, this will place the SDIA line in a floating condition and the SDOA line high. If in Master Mode the SCKA line will be either high or low depending upon the clock polarity selection bit SACKPOLB in the SPIAC1 register. If in Slave Mode the SCKA line will be in a floating condition. If SPIAEN is low, then the bus will be disabled and $\overline{\text{SCSA}}$, SDIA, SDOA and SCKA pins will all become I/O pins or other pin-shared functions using the corresponding pin-shared function selection bits. In the Master Mode the Master will always generate the clock signal. The clock and data transmission will be initiated after data has been written into the SPIAD register. In the Slave Mode, the clock signal will be received from an external master device for both data transmission and reception. The following sequences show the order to be followed for data transfer in both Master and Slave Mode.

Master Mode

- Step 1
  Select the clock source and Master mode using the SASPI2~SASPI0 bits in the SPIAC0 control register.
- Step 2
  Setup the SACSEN bit and setup the SAMLS bit to choose if the data is MSB or LSB shifted first, this must be same as the Slave device.
- Step 3
  Setup the SPIAEN bit in the SPIAC0 control register to enable the SPIA interface.
- Step 4
  For write operations: write the data to the SPIAD register, which will actually place the data into the TXRX buffer. Then use the SCKA and $\overline{\text{SCSA}}$ lines to output the data. After this go to step 5.
  For read operations: the data transferred in on the SDIA line will be stored in the TXRX buffer until all the data has been received at which point it will be latched into the SPIAD register.
- Step 5
  Check the SAWCOL bit if set high then a collision error has occurred so return to step 4. If equal to zero then go to the following step.
- Step 6
  Check the SATRF bit or wait for a SPIA serial bus interrupt.
- Step 7
  Read data from the SPIAD register.
- Step 8
  Clear SATRF.
- Step 9
  Go to step 4.

Slave Mode

- Step 1
  Select the SPI Slave mode using the SASPI2~SASPI0 bits in the SPIAC0 control register
- Step 2
  Setup the SACSEN bit and setup the SAMLS bit to choose if the data is MSB or LSB shifted first, this setting must be the same with the Master device.
- Step 3
  Setup the SPIAEN bit in the SPIAC0 control register to enable the SPIA interface.
- Step 4
  For write operations: write the data to the SPIAD register, which will actually place the data into the TXRX buffer. Then wait for the master clock SCKA and $\overline{\text{SCSA}}$ signal. After this, go to step 5.
  For read operations: the data transferred in on the SDIA line will be stored in the TXRX buffer until all the data has been received at which point it will be latched into the SPIAD register.

- Step 5
  Check the SAWCOL bit if set high then a collision error has occurred so return to step 4. If equal to zero then go to the following step.
- Step 6
  Check the SATRF bit or wait for a SPIA serial bus interrupt.
- Step 7
  Read data from the SPIAD register.
- Step 8
  Clear SATRF.
- Step 9
  Go to step 4.

Error Detection

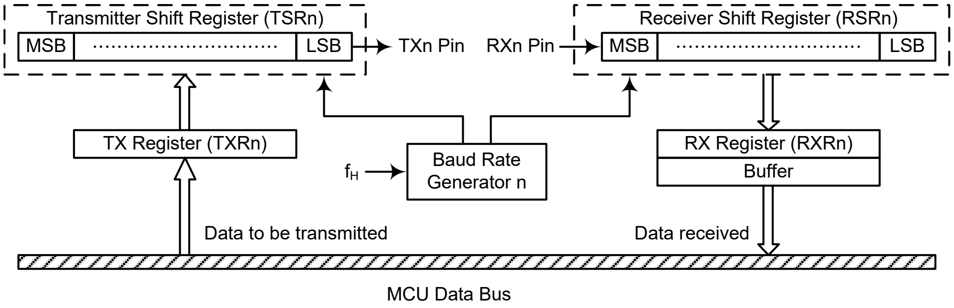
The SAWCOL bit in the SPIAC1 register is provided to indicate errors during data transfer. The bit is set by the SPIA serial Interface but must be cleared by the application program. This bit indicates a data collision has occurred which happens if a write to the SPIAD register takes place during a data transfer operation and will prevent the write operation from continuing.

UART Interface

The device contains up to three integrated full-duplex asynchronous serial communications UART interfaces that enable communication with external devices that contain a serial interface. The UART function has many features and can transmit and receive data serially by transferring a frame of data with eight or nine data bits per transmission as well as being able to detect errors when the data is overwritten or incorrectly framed. Each UART function possesses its own internal interrupt which can be used to indicate when a reception occurs or when a transmission terminates.

Each integrated UART function contains the following features:

- Full-duplex, asynchronous communication
- 8 or 9 bits character length
- Even, odd or no parity options
- One or two stop bits
- Baud rate generator with 8-bit prescaler
- Parity, framing, noise and overrun error detection
- Support for interrupt on address detect (last character bit=1)
- Separately enabled transmitter and receiver
- 2-byte Deep FIFO Receive Data Buffer
- RX pin wake-up function
- Transmit and receive interrupts
- Interrupts can be triggered by the following conditions:
  - Transmitter Empty
  - Transmitter Idle
  - Receiver Full
  - Receiver Overrun
  - Address Mode Detect



UART Data Transfer Block Diagram (n=0~2)

UART External Pins

To communicate with an external serial interface, the internal UARTn has two external pins known as TXn and RXn. The TXn and RXn pins are the UARTn transmitter and receiver pins respectively. The TXn and RXn pin function should first be selected by the corresponding pin-shared function selection register before the UARTn function is used. Along with the UARTENn bit, the TXENn and RXENn bits, if set, will automatically setup the TXn and RXn pins to their respective TXn output and RXn input conditions and disable any pull-high resistor option which may exist on the TXn and RXn pins. When the TXn or RXn pin function is disabled by clearing the UARTENn, TXENn or RXENn bit, the TXn or RXn pin will be set to a floating state. At this time whether the internal pull-high resistor is connected to the TXn or RXn pin or not is determined by the corresponding I/O pull-high function control bit.

UART Data Transfer Scheme

The above diagram shows the overall data transfer structure arrangement for the UARTn interface. The actual data to be transmitted from the MCU is first transferred to the TXRn register by the application program. The data will then be transferred to the Transmit Shift Register from where it will be shifted out, LSB first, onto the TXn pin at a rate controlled by the Baud Rate Generator n. Only the TXRn register is mapped onto the MCU Data Memory, the Transmit Shift Register is not mapped and is therefore inaccessible to the application program.

Data to be received by the UARTn is accepted on the external RXn pin, from where it is shifted in, LSB first, to the Receiver Shift Register at a rate controlled by the Baud Rate Generator. When the shift register is full, the data will then be transferred from the shift register to the internal RXRn register, where it is buffered and can be manipulated by the application program. Only the TXRn register is mapped onto the MCU Data Memory, the Receiver Shift Register is not mapped and is therefore inaccessible to the application program.

It should be noted that the actual register for data transmission and reception, although referred to in the text, and in UARTn block diagram, as separate TXRn and RXRn registers, only exists as a single shared register in the Data Memory. This shared register known as the TXR_RXRn register is used for both data transmission and data reception.

UART Status and Control Registers

There are five control registers associated with the UARTn function. The UnSR, UnCR1 and UnCR2 registers control the overall function of the UARTn, while the BRGn register controls the Baud rate. The actual data to be transmitted and received on the serial interface is managed through the TXR_RXRn data registers.

- **TXR_RXRn Register**

The TXR_RXRn register is the data register which is used to store the data to be transmitted on the TXn pin or being received from the RXn pin.

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	x	x	x	x	x	x	x	x

"x": unknown

Bit 7~0 **D7~D0**: UARTn Transmit/Receive Data bits

- **UnSR Register**

The UnSR register is the status register for the UARTn, which can be read by the program to determine the present status of the UARTn. All flags within the UnSR register are read only and further explanations are given below.

Bit	7	6	5	4	3	2	1	0
Name	PERRn	NFn	FERRn	OERRn	RIDLEn	RXIFn	TIDLEn	TXIFn
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	1	0	1	1

Bit 7 **PERRn**: Parity error flag
0: No parity error is detected
1: Parity error is detected

The PERRn flag is the parity error flag. When this read only flag is "0", it indicates a parity error has not been detected. When the flag is "1", it indicates that the parity of the received word is incorrect. This error flag is applicable only if Parity mode (odd or even) is selected. The flag can also be cleared by a software sequence which involves a read to the status register UnSR followed by an access to the TXR_RXRn data register.

Bit 6 **NFn**: Noise flag
0: No noise is detected
1: Noise is detected

The NFn flag is the noise flag. When this read only flag is "0", it indicates no noise condition. When the flag is "1", it indicates that the UARTn has detected noise on the receiver input. The NFn flag is set during the same cycle as the RXIFn flag but will not be set in the case of as overrun. The NFn flag can be cleared by a software sequence which will involve a read to the status register UnSR followed by an access to the TXR_RXRn data register.

Bit 5 **FERRn**: Framing error flag
0: No framing error is detected
1: Framing error is detected

The FERRn flag is the framing error flag. When this read only flag is "0", it indicates that there is no framing error. When the flag is "1", it indicates that a framing error has been detected for the current character. The flag can also be cleared by a software sequence which will involve a read to the status register UnSR followed by an access to the TXR_RXRn data register.

Bit 4 **OERRn**: Overrun error flag
0: No overrun error is detected
1: Overrun error is detected

The OERRn flag is the overrun error flag which indicates when the receiver buffer has overflowed. When this read only flag is "0", it indicates that there is no overrun error. When the flag is "1", it indicates that an overrun error occurs which will inhibit further transfers to the TXR_RXRn receive data register. The flag is cleared by a software sequence, which is a read to the status register UnSR followed by an access to the TXR_RXRn data register.

Bit 3 **RIDLEn**: Receiver status
0: Data reception is in progress (data being received)
1: No data reception is in progress (receiver is idle)

The RIDLEn flag is the receiver status flag. When this read only flag is "0", it indicates that the receiver is between the initial detection of the start bit and the completion of the stop bit. When the flag is "1", it indicates that the receiver is idle. Between the completion of the stop bit and the detection of the next start bit, the RIDLEn bit is "1" indicating that the UARTn receiver is idle and the RXn pin stays in logic high condition.

Bit 2 **RXIFn**: Receive TXR_RXRn data register status
0: TXR_RXRn data register is empty
1: TXR_RXRn data register has available data

The RXIFn flag is the receive data register status flag. When this read only flag is "0", it indicates that the TXR_RXRn read data register is empty. When the flag is "1", it indicates that the TXR_RXRn read data register contains new data. When the contents of the shift register are transferred to the TXR_RXRn register, an interrupt is generated if RIEn=1 in the UnCR2 register. If one or more errors are detected in the received word, the appropriate receive-related flags NFn, FERRn, and/or PERRn are set within the same clock cycle. The RXIFn flag is will eventually be cleared when the UnSR register is read with RXIFn set, followed by a read from the TXR_RXRn register, and if the TXR_RXRn register has no more new data available.

Bit 1 **TIDLEn**: Transmission status
0: Data transmission is in progress (data being transmitted)
1: No data transmission is in progress (transmitter is idle)

The TIDLEn flag is known as the transmission complete flag. When this read only flag is "0", it indicates that a transmission is in progress. This flag will be set to "1" when the TXIFn flag is "1" and when there is no transmit data or break character being transmitted. When TIDLEn is equal to 1, the TXn pin becomes idle with the pin state in logic high condition. The TIDLEn flag is cleared by reading the UnSR register with TIDLEn set and then writing to the TXR_RXRn register. The flag is not generated when a data character or a break is queued and ready to be sent.

Bit 0 **TXIFn**: Transmit TXR data register status
0: Character is not transferred to the transmit shift register
1: Character has transferred to the transmit shift register (TXR_RXRn data register is empty)

The TXIFn flag is the transmit data register empty flag. When this read only flag is "0", it indicates that the character is not transferred to the transmitter shift register. When the flag is "1", it indicates that the transmitter shift register has received a character from the TXR_RXRn data register. The TXIFn flag is cleared by reading the UARTn status register (UnSR) with TXIFn set and then writing to the TXR_RXRn data register. Note that when the TXENn bit is set, the TXIFn flag bit will also be set since the transmit data register is not yet full.

- **UnCR1 Register**

The UnCR1 register together with the UnCR2 register are the UARTn control registers that are used to set the various options for the UARTn function such as overall on/off control, parity control, data transfer bit length, etc. Further explanation on each of the bits is given below.

Bit	7	6	5	4	3	2	1	0
Name	UARTENn	BNOn	PRENn	PRTn	STOPSn	TXBRKn	RX8n	TX8n
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R	W
POR	0	0	0	0	0	0	x	0

"x": unknown

Bit 7 **UARTENn**: UARTn function enable control
0: Disable UARTn; TXn and RXn pins are in a floating state.
1: Enable UARTn; TXn and RXn pins function as UARTn pins

The UARTENn bit is the UARTn enable bit. When this bit is equal to "0", the UARTn will be disabled and the RXn pin as well as the TXn pin will be set in a floating state. When the bit is equal to "1", the UARTn will be enabled and the TXn and RXn pins will function as defined by the TXENn and RXENn enable control bits. When the UARTn is disabled, it will empty the buffer so any character remaining in the buffer will be discarded. In addition, the value of the baud rate counter will be reset. If the UARTn is disabled, all error and status flags will be reset. Also the TXENn, RXENn, TXBRKn, RXIFn, OERRn, FERRn, PERRn and NFn bits will be cleared, while the TIDLEn, TXIFn and RIDLEn bits will be set. Other control bits in UnCR1, UnCR2 and BRGn registers will remain unaffected. If the UARTn is active and the UARTENn bit is cleared, all pending transmissions and receptions will be terminated and the module will be reset as defined above. When the UARTn is re-enabled, it will restart in the same configuration.

Bit 6 **BNOn**: Number of data transfer bits selection
0: 8-bit data transfer
1: 9-bit data transfer

This bit is used to select the data length format, which can have a choice of either 8-bit or 9-bit format. When this bit is equal to "1", a 9-bit data length format will be selected. If the bit is equal to "0", then an 8-bit data length format will be selected. If 9-bit data length format is selected, then bits RX8n and TX8n will be used to store the 9th bit of the received and transmitted data respectively.

Bit 5 **PRENn**: Parity function enable control
0: Parity function is disabled
1: Parity function is enabled

This bit is the parity function enable bit. When this bit is equal to 1, the parity function will be enabled. If the bit is equal to 0, then the parity function will be disabled.

Bit 4 **PRTn**: Parity type selection bit
0: Even parity for parity generator
1: Odd parity for parity generator

This bit is the parity type selection bit. When this bit is equal to 1, odd parity type will be selected. If the bit is equal to 0, then even parity type will be selected.

Bit 3 **STOPSn**: Number of stop bits selection
0: One stop bit format is used
1: Two stop bits format is used

This bit determines if one or two stop bits are to be used. When this bit is equal to "1", two stop bits format are used. If the bit is equal to "0", then only one stop bit format is used.

Bit 2 **TXBRKn**: Transmit break character
0: No break character is transmitted
1: Break characters transmit

The TXBRKn bit is the Transmit Break Character bit. When this bit is equal to “0”, there are no break characters and the TXn pin operates normally. When the bit is equal to “1”, there are transmit break characters and the transmitter will send logic zeros. When this bit is equal to “1”, after the buffered data has been transmitted, the transmitter output is held low for a minimum of a 13-bit length and until the TXBRKn bit is reset.

Bit 1 **RX8n**: Receive data bit 8 for 9-bit data transfer format (read only)

This bit is only used if 9-bit data transfers are used, in which case this bit location will store the 9th bit of the received data known as RX8n. The BNOn bit is used to determine whether data transfers are in 8-bit or 9-bit format.

Bit 0 **TX8n**: Transmit data bit 8 for 9-bit data transfer format (write only)

This bit is only used if 9-bit data transfers are used, in which case this bit location will store the 9th bit of the transmitted data known as TX8n. The BNOn bit is used to determine whether data transfers are in 8-bit or 9-bit format.

- **UnCR2 Register**

The UnCR2 register is the second of the UARTn control registers and serves several purposes. One of its main functions is to control the basic enable/disable operation if the UARTn Transmitter and Receiver as well as enabling the various UARTn interrupt sources. The register also serves to control the baud rate speed, receiver wake-up function enable and the address detect function enable. Further explanation on each of the bits is given below.

Bit	7	6	5	4	3	2	1	0
Name	TXENn	RXENn	BRGHn	ADDENn	WAKEn	RIEn	TIIEn	TEIEn
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **TXENn**: UARTn Transmitter enable control
0: UARTn Transmitter is disabled
1: UARTn Transmitter is enabled

The TXENn bit is the Transmitter Enable Bit. When this bit is equal to “0”, the transmitter will be disabled with any pending data transmissions being aborted. In addition the buffers will be reset. If the TXENn bit is equal to “1” and the UARTENn bit is also equal to 1, the transmitter will be enabled and the TXn pin will be controlled by the UARTn. Clearing the TXENn bit during a transmission will cause the data transmission to be aborted and will reset the transmitter.

Bit 6 **RXENn**: UARTn Receiver enable control
0: UARTn Receiver is disabled
1: UARTn Receiver is enabled

The RXENn bit is the Receiver Enable Bit. When this bit is equal to “0”, the receiver will be disabled with any pending data receptions being aborted. In addition the receiver buffers will be reset. If the RXENn bit is equal to “1” and the UARTENn bit is also equal to 1, the receiver will be enabled and the RXn pin will be controlled by the UARTn. Clearing the RXENn bit during a reception will cause the data reception to be aborted and will reset the receiver.

Bit 5 **BRGHn**: Baud Rate speed selection
0: Low speed baud rate
1: High speed baud rate

The bit named BRGHn selects the high or low speed mode of the Baud Rate Generator n. This bit, together with the value placed in the baud rate register, BRGn, controls the baud rate of the UARTn. If the bit is equal to 0, the low speed mode is selected.

Bit 4 **ADDENn**: Address detect function enable control
0: Address detection function is disabled
1: Address detection function is enabled

The bit named ADDENn is the address detection function enable control bit. When this bit is equal to 1, the address detection function is enabled. When it occurs, if the 8th bit, which corresponds to RX7n if BNO=0, or the 9th bit, which corresponds to RX8n if BNOn=1, has a value of "1", then the received word will be identified as an address, rather than data. If the corresponding interrupt is enabled, an interrupt request will be generated each time the received word has the address bit set, which is the 8th or 9th bit depending on the value of the BNOn bit. If the address bit known as the 8th or 9th bit of the received word is "0" with the address detection function being enabled, an interrupt will not be generated and the received data will be discarded.

Bit 3 **WAKEn**: RXn pin falling edge wake-up function enable control
0: RXn pin wake-up UARTn function is disabled
1: RXn pin wake-up UARTn function is enabled

The bit is used to control the wake-up UARTn function when a falling edge on the RXn pin occurs. Note that this bit is only available when the UARTn clock, f_H, is switched off. There will be no RXn pin wake-up UARTn function if the UARTn clock, f_H, exists. If the WAKEn bit is equal to 1 and the UARTn clock, f_H, is switched off, a UARTn wake-up request will be initiated when a falling edge on the RXn pin occurs. When this request happens and the corresponding interrupt is enabled, an RXn pin wake-up UARTn interrupt will be generated to inform the MCU to wake up the UARTn function by switching on the UARTn clock, f_H, via the application programs. Otherwise, the UARTn function can not resume even if there is a falling edge on the RXn pin when the WAKEn bit is cleared to 0.

Bit 2 **RIEn**: Receiver interrupt enable control
0: Receiver related interrupt is disabled
1: Receiver related interrupt is enabled

The bit enables or disables the receiver interrupt. If this bit is equal to 1 and when the receiver overrun flag OERRn or received data available flag RXIFn is set, the UARTn interrupt request flag will be set. If this bit is equal to 0, the UARTn interrupt request flag will not be influenced by the condition of the OERRn or RXIFn flags.

Bit 1 **TIIEn**: Transmitter Idle interrupt enable control
0: Transmitter idle interrupt is disabled
1: Transmitter idle interrupt is enabled

The bit enables or disables the transmitter idle interrupt. If this bit is equal to 1 and when the transmitter idle flag TIDLEn is set, due to a transmitter idle condition, the UARTn interrupt request flag will be set. If this bit is equal to 0, the UARTn interrupt request flag will not be influenced by the condition of the TIDLEn flag.

Bit 0 **TEIEn**: Transmitter Empty interrupt enable control
0: Transmitter empty interrupt is disabled
1: Transmitter empty interrupt is enabled

The bit enables or disables the transmitter empty interrupt. If this bit is equal to 1 and when the transmitter empty flag TXIFn is set, due to a transmitter empty condition, the UARTn interrupt request flag will be set. If this bit is equal to 0, the UARTn interrupt request flag will not be influenced by the condition of the TXIFn flag.

Baud Rate Generator

To setup the speed of the serial data communication, the UARTn function contains its own dedicated baud rate generator. The baud rate is controlled by its own internal free running 8-bit timer, the period of which is determined by two factors. The first of these is the value placed in the BRGn register and the second is the value of the BRGHn bit within the UnCR2 control register. The BRGHn bit decides, if the baud rate generator is to be used in a high speed mode or low speed mode, which in turn determines the formula that is used to calculate the baud rate. The value in the BRGn register, N, which is used in the following baud rate calculation formula determines the division factor. Note that N is the decimal value placed in the BRGn register and has a range of between 0 and 255.

UnCR2 BRGHn Bit	0	1
Baud Rate (BR)	$\frac{f_H}{[64(N+1)]}$	$\frac{f_H}{[16(N+1)]}$

By programming the BRGHn bit which allows selection of the related formula and programming the required value in the BRGn register, the required baud rate can be setup. Note that because the actual baud rate is determined using a discrete value, N, placed in the BRGn register, there will be an error associated between the actual and requested value. The following example shows how the BRGn register value N and the error value can be calculated.

- **BRGn Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	x	x	x	x	x	x	x	x

"x": unknown

Bit 7~0 **D7~D0**: Baud Rate values

By programming the BRGHn bit in the UnCR2 register which allows selection of the related formula described above and programming the required value in the BRGn register, the required baud rate can be setup.

Calculating the Baud Rate and Error Values

For a clock frequency of 4MHz, and with BRGHn set to 0 determine the BRGn register value N, the actual baud rate and the error value for a desired baud rate of 4800.

From the above table the desired baud rate BR = $\frac{f_H}{[64(N+1)]}$

Re-arranging this equation gives N = $\frac{f_H}{(BR\times64)} - 1$

Giving a value for N = $\frac{4000000}{(4800\times64)} - 1 = 12.0208$

To obtain the closest value, a decimal value of 12 should be placed into the BRGn register. This gives an actual or calculated baud rate value of BR = $\frac{4000000}{[64(12+1)]} = 4808$

Therefore the error is equal to $\frac{4808 - 4800}{4800} = 0.16\%$

UART Setup and Control

For data transfer, the UARTn function utilizes a non-return-to-zero, more commonly known as NRZ, format. This is composed of one start bit, eight or nine data bits and one or two stop bits. Parity is supported by the UARTn hardware and can be setup to be even, odd or no parity. For the most common data format, 8 data bits along with no parity and one stop bit, denoted as 8, N, 1, is used as the default setting, which is the setting at power-on. The number of data bits and stop bits, along with the parity, are setup by programming the corresponding BNOn, PRTn, PRENn and STOPSn bits in the UnCR1 register. The baud rate used to transmit and receive data is setup using the internal 8-bit baud rate generator, while the data is transmitted and received LSB first. Although the transmitter and receiver of the UARTn are functionally independent, they both use the same data format and baud rate. In all cases stop bits will be used for data transmission.

Enabling/Disabling the UART Interface

The basic on/off function of the internal UARTn function is controlled using the UARTENn bit in the UnCR1 register. If the UARTENn, TXENn and RXENn bits are set, then these two UARTn pins will act as normal TXn output pin and RXn input pin respectively. If no data is being transmitted on the TXn pin, then it will default to a logic high value.

Clearing the UARTENn bit will disable the TXn and RXn pins and then these two pins can be used as an I/O or other pin-shared functional pins by properly configurations. When the UARTn function is disabled, the buffer will be reset to an empty condition, at the same time discarding any remaining residual data. Disabling the UARTn will also reset the enable control, the error and status flags with bits TXENn, RXENn, TXBRKn, RXIFn, OERRn, FERRn, PERRn and NFn being cleared while bits TIDLEn, TXIFn and RIDLEn will be set. The remaining control bits in the UnCR1, UnCR2 and BRGn registers will remain unaffected. If the UARTENn bit in the UnCR1 register is cleared while the UARTn is active, then all pending transmissions and receptions will be immediately suspended and the UARTn will be reset to a condition as defined above. If the UARTn is then subsequently re-enabled, it will restart again in the same configuration.

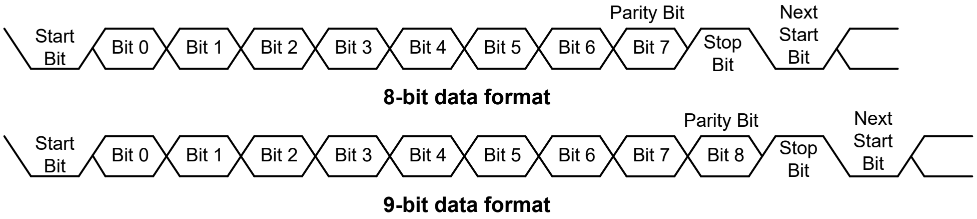
Data, Parity and Stop Bit Selection

The format of the data to be transferred is composed of various factors such as data bit length, parity on/off, parity type, address bits and the number of stop bits. These factors are determined by the setup of various bits within the UnCR1 register. The BNOn bit controls the number of data bits which can be set to either 8 or 9. The PRTn bit controls the choice if odd or even parity. The PRENn bit controls the parity on/off function. The STOPSn bit decides whether one or two stop bits are to be used. The following table shows various formats for data transmission. The address bit, which is the MSB of the data byte, identifies the frame as an address character or data if the address detect function is enabled. The number of stop bits, which can be either one or two, is independent of the data length.

Start Bit	Data Bits	Address Bits	Parity Bit	Stop Bit
Example of 8-bit Data Formats				
1	8	0	0	1
1	7	0	1	1
1	7	1	0	1
Example of 9-bit Data Formats				
1	9	0	0	1
1	8	0	1	1
1	8	1	0	1

Transmitter Receiver Data Format

The following diagram shows the transmit and receive waveforms for both 8-bit and 9-bit data formats.



UART Transmitter

Data word lengths of either 8 or 9 bits can be selected by programming the BNOn bit in the UnCR1 register. When BNOn bit is set, the word length will be set to 9 bits. In this case the 9th bit, which is the MSB, needs to be stored in the TX8n bit in the UnCR1 register. At the transmitter core lies the Transmitter Shift Register, more commonly known as the TSRn, whose data is obtained from the transmit data register, which is known as the TXR_RXRn register. The data to be transmitted is loaded into this TXR_RXRn register by the application program. The TSRn register is not written to with new data until the stop bit from the previous transmission has been sent out. As soon as this stop bit has been transmitted, the TSRn can then be loaded with new data from the TXR_RXRn register, if it is available. It should be noted that the TSRn register, unlike many other registers, is not directly mapped into the Data Memory area and as such is not available to the application program for direct read/write operations. An actual transmission of data will normally be enabled when the TXENn bit is set, but the data will not be transmitted until the TXR_RXRn register has been loaded with data and the baud rate generator has defined a shift clock source. However, the transmission can also be initiated by first loading data into the TXR_RXRn register, after which the TXENn bit can be set. When a transmission of data begins, the TSRn is normally empty, in which case a transfer to the TXR_RXRn register will result in an immediate transfer to the TSRn. If during a transmission the TXENn bit is cleared, the transmission will immediately cease and the transmitter will be reset. The TXn output pin can then return to the I/O or other pin-shared function by properly configurations.

Transmitting Data

When the UARTn is transmitting data, the data is shifted on the TXn pin from the shift register, with the least significant bit LSB first. In the transmit mode, the TXR_RXRn register forms a buffer between the internal bus and the transmitter shift register. It should be noted that if 9-bit data format has been selected, then the MSB will be taken from the TX8n bit in the UnCR1 register. The steps to initiate a data transfer can be summarized as follows:

- Make the correct selection of the BNOn, PRTn, PRENn and STOPSn bits to define the required word length, parity type and number of stop bits.
- Setup the BRGn register to select the desired baud rate.
- Set the TXENn bit to ensure that the UARTn transmitter is enabled and the TXn pin is used as a UARTn transmitter pin.
- Access the UnSR register and write the data that is to be transmitted into the TXR_RXRn register. Note that this step will clear the TXIFn bit.

This sequence of events can now be repeated to send additional data. It should be noted that when TXIFn=0, data will be inhibited from being written to the TXR_RXRn register. Clearing the TXIFn flag is always achieved using the following software sequence:

1. A UnSR register access
2. A TXR_RXRn register write execution

The read-only TXIFn flag is set by the UARTn hardware and if set indicates that the TXR_RXRn register is empty and that other data can now be written into the TXR_RXRn register without overwriting the previous data. If the TEIEn bit is set, then the TXIFn flag will generate an interrupt. During a data transmission, a write instruction to the TXR_RXRn register will place the data into the TXR_RXRn register, which will be copied to the shift register at the end of the present transmission. When there is no data transmission in progress, a write instruction to the TXR_RXRn register will place the data directly into the shift register, resulting in the commencement of data transmission, and the TXIFn bit being immediately set. When a frame transmission is complete, which happens after stop bits are sent or after the break frame, the TIDLEn bit will be set. To clear the TIDLEn bit the following software sequence is used:

1. A UnSR register access
2. A TXR_RXRn register write execution

Note that both the TXIFn and TIDLEn bits are cleared by the same software sequence.

Transmitting Break

If the TXBRKn bit is set, then the break characters will be sent on the next transmission. Break character transmission consists of a start bit, followed by 13xN "0" bits, where N=1, 2, etc. If a break character is to be transmitted, then the TXBRKn bit must be first set by the application program and then cleared to generate the stop bits. Transmitting a break character will not generate a transmit interrupt. Note that a break condition length is at least 13 bits long. If the TXBRKn bit is continually kept at a logic high level, then the transmitter circuitry will transmit continuous break characters. After the application program has cleared the TXBRKn bit, the transmitter will finish transmitting the last break character and subsequently send out one or two stop bits. The automatic logic high at the end of the last break character will ensure that the start bit of the next frame is recognized.

UART Receiver

The UARTn is capable of receiving word lengths of either 8 or 9 bits can be selected by programming the BNOn bit in the UnCR1 register. When BNOn bit is set, the word length will be set to 9 bits. In this case the 9th bit, which is the MSB, will be stored in the RX8n bit in the UnCR1 register. At the receiver core lies the Receiver Shift Register more commonly known as the RSRn. The data which is received on the RXn external input pin is sent to the data recovery block. The data recovery block operating speed is 16 times that of the baud rate, while the main receive serial shifter operates at the baud rate. After the RXn pin is sampled for the stop bit, the received data in RSRn is transferred to the receive data register, if the register is empty. The data which is received on the external RXn input pin is sampled three times by a majority detect circuit to determine the logic level that has been placed onto the RXn pin. It should be noted that the RSRn register, unlike many other registers, is not directly mapped into the Data Memory area and as such is not available to the application program for direct read/write operations.

Receiving Data

When the UARTn receiver is receiving data, the data is serially shifted in on the external RXn input pin to the shift register, with the least significant bit LSB first. The TXR_RXRn register is a two byte deep FIFO data buffer, where two bytes can be held in the FIFO while the 3rd byte can continue to be received. Note that the application program must ensure that the data is read from TXR_RXRn before the 3rd byte has been completely shifted in, otherwise the 3rd byte will be discarded and an overrun error OERRn will be subsequently indicated. The steps to initiate a data transfer can be summarized as follows:

- Make the correct selection of the BNOn, PRTn, PRENn and STOPSn bits to define the required word length, parity type and number of stop bits.

- Setup the BRGn register to select the desired baud rate.
- Set the RXENn bit to ensure that the UARTn receiver is enabled and the RXn pin is used as a UARTn receiver pin.

At this point the receiver will be enabled which will begin to look for a start bit.

When a character is received, the following sequence of events will occur:

- The RXIFn bit in the UnSR register will be set when the TXR_RXRn register has data available. There will be at most one more character that can be read.
- When the contents of the shift register have been transferred to the TXR_RXRn register and if the RIEn bit is set, then an interrupt will be generated.
- If during reception, a frame error, noise error, parity error or an overrun error has been detected, then the error flags can be set.

The RXIFn bit can be cleared using the following software sequence:

1. A UnSR register access
2. A TXR_RXRn register read execution

Receiving Break

Any break character received by the UARTn will be managed as a framing error. The receiver will count and expect a certain number of bit times as specified by the values programmed into the BNOn and STOPSn bits. If the break is much longer than 13 bit times, the reception will be considered as complete after the number of bit times specified by BNOn and STOPSn. The RXIFn bit is set, FERRn is set, zeros are loaded into the receive data register, interrupts are generated if appropriate and the RIDLEn bit is set. A break is regarded as a character that contains only zeros with the FERRn flag being set. If a long break signal has been detected, the receiver will regard it as a data frame including a start bit, data bits and the invalid stop bit and the FERRn flag will be set. The receiver must wait for a valid stop bit before looking for the next start bit. The receiver will not make the assumption that the break condition on the line is the next start bit. The break character will be loaded into the buffer and no further data will be received until stop bits are received. It should be noted that the RIDLEn read only flag will go high when the stop bits have not yet been received. The reception of a break character on the UARTn registers will result in the following:

- The framing error flag, FERRn, will be set.
- The receive data register, TXR_RXRn, will be cleared.
- The OERRn, NFn, PERRn, RIDLEn or RXIFn flags will possibly be set.

Idle Status

When the receiver is reading data, which means it will be in between the detection of a start bit and the reading of a stop bit, the receiver status flag in the UnSR register, otherwise known as the RIDLEn flag, will have a zero value. In between the reception of a stop bit and the detection of the next start bit, the RIDLEn flag will have a high value, which indicates the receiver is in an idle condition.

Receiver Interrupt

The read only receive interrupt flag, RXIFn, in the UnSR register is set by an edge generated by the receiver. An interrupt is generated if RIEn=1, when a word is transferred from the Receive Shift Register, RSRn, to the Receive Data Register, TXR_RXRn. An overrun error can also generate an interrupt if RIEn=1.

Managing Receiver Errors

Several types of reception errors can occur within the UARTn module, the following section describes the various types and how they are managed by the UARTn.

Overrun Error – OERRn

The TXR_RXRn register is composed of a two byte deep FIFO data buffer, where two bytes can be held in the FIFO register, while a 3th byte can continue to be received. Before the 3th byte has been entirely shifted in, the data should be read from the TXR_RXRn register. If this is not done, the overrun error flag OERRn will be consequently indicated.

In the event of an overrun error occurring, the following will happen:

- The OERRn flag in the UnSR register will be set.
- The TXR_RXRn contents will not be lost.
- The shift register will be overwritten.
- An interrupt will be generated if the RIEn bit is set.

The OERRn flag can be cleared by an access to the UnSR register followed by a read to the TXR_RXRn register.

Noise Error – NFn

Over-sampling is used for data recovery to identify valid incoming data and noise. If noise is detected within a frame, the following will occur:

- The read only noise flag, NFn, in the UnSR register will be set on the rising edge of the RXIFn bit.
- Data will be transferred from the shift register to the TXR_RXRn register.
- No interrupt will be generated. However this bit rises at the same time as the RXIFn bit which itself generates an interrupt.

Note that the NFn flag is reset by an UnSR register read operation followed by a TXR_RXRn register read operation.

Framing Error – FERRn

The read only framing error flag, FERRn, in the UnSR register, is set if a zero is detected instead of stop bits. If two stop bits are selected, both stop bits must be high. Otherwise the FERRn flag will be set. The FERRn flag and the received data will be recorded in the UnSR and TXR_RXRn registers respectively and the FERRn flag will be cleared in any reset.

Parity Error – PERRn

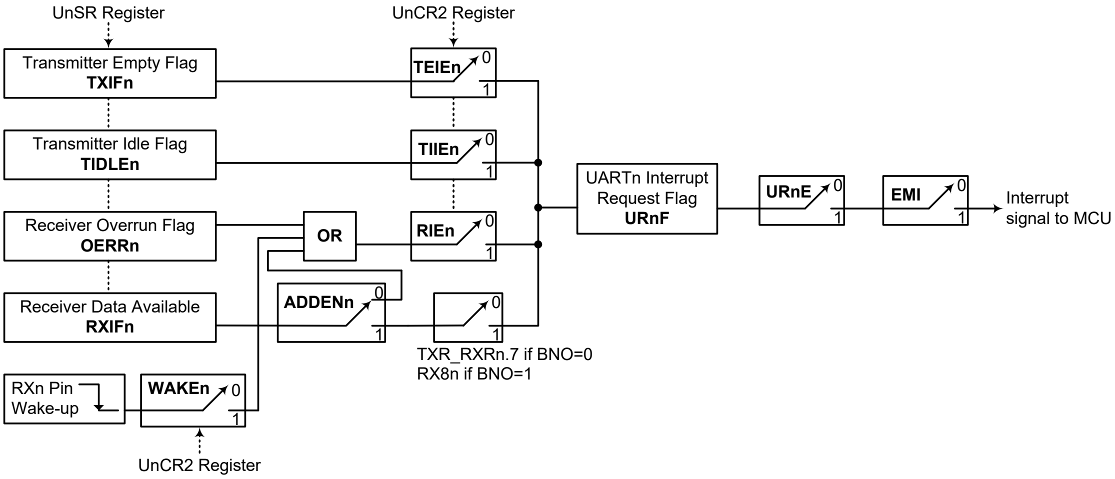
The read only parity error flag, PERRn, in the UnSR register, is set if the parity of the received word is incorrect. This error flag is only applicable if the parity function is enabled, PRENn=1, and if the parity type, odd or even, is selected. The read only PERRn flag and the received data will be recorded in the UnSR and TXR_RXRn registers respectively and the flag will be cleared on any reset. It should be noted that the FERRn and PERRn flags in the UnSR register should first be read by the application programs before reading the data word.

UART Interrupt Structure

Several individual UARTn conditions can generate a UARTn interrupt. When these conditions exist, a low pulse will be generated to get the attention of the microcontroller. These conditions are a transmitter data register empty, transmitter idle, receiver data available, receiver overrun, address detect and an RXn pin wake-up. When any of these conditions are created, if its corresponding interrupt control is enabled and the stack is not full, the program will jump to its corresponding interrupt vector where it can be serviced before returning to the main program. Four of these conditions have the corresponding UnSR register flags which will generate a UARTn interrupt if its associated interrupt enable control bit in the UnCR2 register is set. The two transmitter interrupt conditions have their own corresponding enable control bits, while the two receiver interrupt conditions have a shared enable control bit. These enable bits can be used to mask out individual UARTn interrupt sources.

The address detect condition, which is also a UARTn interrupt source, does not have an associated flag, but will generate a UARTn interrupt when an address detect condition occurs if its function is enabled by setting the ADDENn bit in the UnCR2 register. An RXn pin wake-up, which is also a UARTn interrupt source, does not have an associated flag, but will generate a UARTn interrupt if the UARTn clock source, f_H, is switched off and the WAKEn and RIEn bits in the UnCR2 register are set when a falling edge on the RXn pin occurs. Note that in the event of an RXn wake-up interrupt occurring, there will be a certain period of delay, commonly known as the System Start-up Time, for the oscillator to restart and stabilize before the system resumes normal operation.

Note that the UnSR register flags are read only and cannot be cleared or set by the application program, neither will they be cleared when the program jumps to the corresponding interrupt servicing routine, as is the case for some of the other interrupts. The flags will be cleared automatically when certain actions are taken by the UARTn, the details of which are given in the UARTn register section. The overall UARTn interrupt can be disabled or enabled by the related interrupt enable control bits in the interrupt control registers of the microcontroller to decide whether the interrupt requested by the UARTn module is masked out or allowed.



UARTn Interrupt Structure

Address Detect Mode

Setting the Address Detect function enable control bit, ADDENn, in the UnCR2 register, enables this special function. If this bit is set to 1, then an additional qualifier will be placed on the generation of a Receiver Data Available interrupt, which is requested by the RXIFn flag. If the ADDENn bit is equal to 1, then when the data is available, an interrupt will only be generated, if the highest received bit has a high value. Note that the related interrupt enable control bit and the EMI bit of the microcontroller must also be enabled for correct interrupt generation. The highest address bit is the 9th bit if the bit BNOn=1 or the 8th bit if the bit BNOn=0. If the highest bit is high, then the received word will be defined as an address rather than data. A Data Available interrupt will be generated every time the last bit of the received word is set. If the ADDENn bit is equal to 0, then a Receive Data Available interrupt will be generated each time the RXIFn flag is set, irrespective of the data last bit status. The address detection and parity functions are mutually exclusive functions. Therefore, if the address detect function is enabled, then to ensure correct operation, the parity function should be disabled by resetting the parity function enable bit PRENn to zero.

ADDENn	Bit 9 if BNOn=1 Bit 8 if BNOn=0	UARTn Interrupt Generated
0	0	√
	1	√
1	0	X
	1	√

ADDENn Bit Function

UART Power Down and Wake-up

When the UARTn clock, f_H, is switched off, the UARTn will cease to function. If the MCU switches off the UARTn clock f_H and enters the power down mode while a transmission is still in progress, then the transmission will be paused until the UARTn clock source derived from the microcontroller is activated. In a similar way, if the MCU switches off the UART clock f_H and enters the power down mode by executing the “HALT” instruction while receiving data, then the reception of data will likewise be paused. When the MCU enters the power down mode, note that the UnSR, UnCR1, UnCR2, transmit and receive registers, as well as the BRGn register will not be affected. It is recommended to make sure first that the UARTn data transmission or reception has been finished before the microcontroller enters the power down mode.

The UARTn function contains a receiver RXn pin wake-up function, which is enabled or disabled by the WAKEn bit in the UnCR2 register. If this bit, along with the UARTn enable bit, UARTENn, the receiver enable bit, RXENn and the receiver interrupt bit, RIEn, are all set before the MCU enters the power down mode with the UARTn clock f_H being switched off, then a falling edge on the RXn pin will initiate a RXn pin wake-up UARTn interrupt. Note that as it takes certain system clock cycles after a wake-up, before normal microcontroller operation resumes, any data received during this time on the RXn pin will be ignored.

For a UARTn wake-up interrupt to occur, in addition to the bits for the wake-up being set, the global interrupt enable bit, EMI, and the UARTn interrupt enable bit, URnE, must be set. If the EMI and URnE bits are not set then only a wake up event will occur and no interrupt will be generated. Note also that as it takes certain system clock cycles after a wake-up before normal microcontroller resumes, the UARTn interrupt will not be generated until after this time has elapsed.

CAN Bus

The device contains a fully integrated CAN (Controller Area Network) Bus interface. It was originally developed as a vehicle electrical bus standard to implement multiplexed wiring to allow multiple automotive device hardware and microcontrollers to communicate with each other without requiring a host computer. Although primarily used in automotive applications the CAN Bus is also seeing use on other application areas.

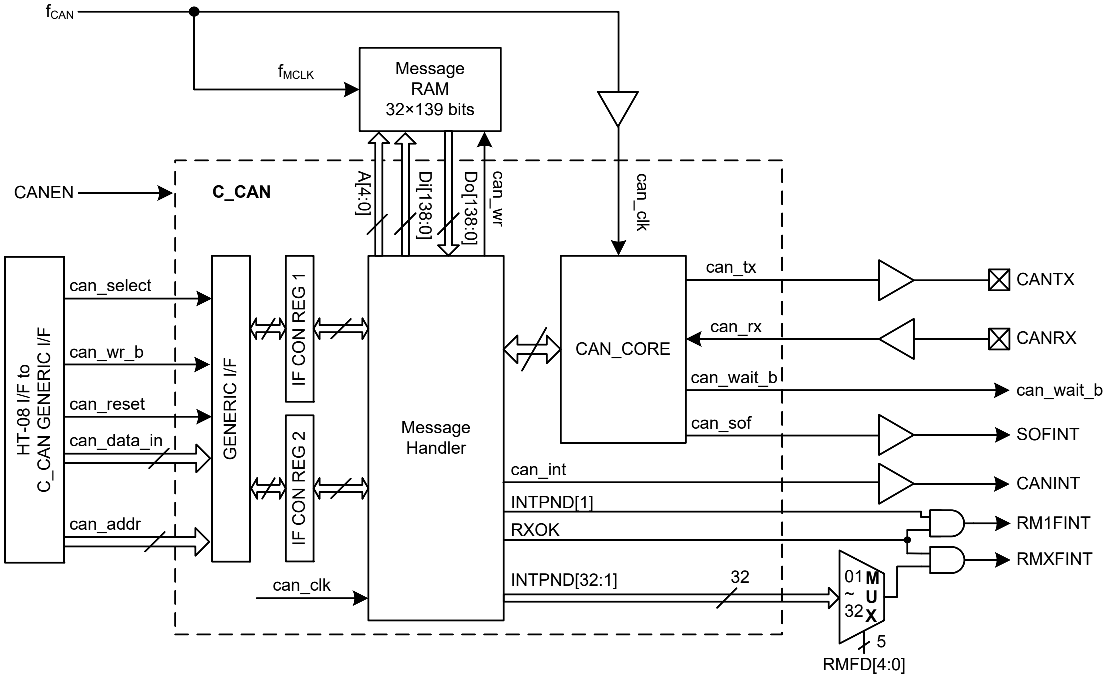
The Holtek CAN Module licensed from Bosch, abbreviation C_CAN. The CAN module includes an area of Message RAM, for which there are two interface control register sets to access the Message Object Data contained within the Message RAM.

The CAN Bus contains the following features:

- Conforms to ISO11898-1, 2003
- 32 Message Objects
- Each Message Object has its own identifier mask
- Programmable FIFO mode – concatenation of Message Objects
- Maskable interrupt
- Programmable loop-back mode for self-test operation

CAN Bus Overview

The CAN module consists of the CAN Core, Message RAM, Message Handler and Interface Control Register sets. An additional HT-8 Interface to the C_CAN generic interface is provided by the CAN module, via which the external 8-bit MCUs can access the C_CAN registers. The CAN Bus clock f_{CAN} comes from the high frequency f_H clock.



CAN Module Block Diagram

1. C_CAN – CAN Core

Refer to the following Operating Mode Description and Application section for further CAN module operation details.

- CAN_Core
  The CAN_Core performs communication according to the CAN protocol version 2.0 A, B and ISO 11898-1.
- Registers
  All registers are used to control and configure the module.
- Message Handler
  The internal State Machine controls the data transfer between the RX/TX Shift Register of the CAN_Core and the Message RAM as well as the generation of interrupts as programmed in the Control and Configuration Registers. All functions concerning the handling of messages are implemented in the Message Handler. Those functions are the acceptance filtering, the transfer of messages between the CAN Core and the Message RAM and the handling of transmission requests as well as the generation of the module interrupt.
- Interface Control Registers 1 and 2
  The function of the two interface control register sets is identical (except in Basic mode).
  The interface control register sets are used for data transfer between the external bus and the Message RAM.
- Message RAM Interface
  Message RAM size: 139-bit × 32

2. Message RAM IP

- Stores 32 Message Objects and Identifier Masks.
- Each Message Object together with Identifier Mask has a length of 139-bit

3. HT-8 Interface to C_CAN Generic Interface

- The external 8-bit microcontroller can access the C_CAN registers through this transform interface.
- The MCU defines the desired register address in the application program. It is only allowed to access one register each time. To implement consecutive access operations, the address should be incremented by one using the application program.

CAN Bus Pins

This device provides two CAN Bus TX and RX pins, namely CANTX and CANRX. The CANTX and CANRX pins are the CAN Bus digital transmit and receive pins respectively. These pin functions should first be selected by the correspongding pin-shared function selection register before the CAN Bus is used.

CAN Bus Registers

The CAN Bus is controlled using a series of registers as shown in the following table.

Register Name	Bit							
	7	6	5	4	3	2	1	0
CTRLRL	TEST	CCE	DAR	—	EIE	SIE	CANIE	INIT
STATRL	BOFF	EWARN	EPASS	RXOK	TXOK	LEC2	LEC1	LEC0
ERRCNTL	TEC7	TEC6	TEC5	TEC4	TEC3	TEC2	TEC1	TEC0
ERRCNTH	RP	REC6	REC5	REC4	REC3	REC2	REC1	REC0
BTRL	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
BTRH	—	TSG2D2	TSG2D1	TSG2D0	TSG1D3	TSG1D2	TSG1D1	TSG1D0
INTRL	INTID7	INTID6	INTID5	INTID4	INTID3	INTID2	INTID1	INTID0
INTRH	INTID15	INTID14	INTID13	INTID12	INTID11	INTID10	INTID9	INTID8
TESTRL	RX	TX1	TX0	LBACK	SILENT	BASIC	—	—
BRPERL	—	—	—	—	BRPE3	BRPE2	BRPE1	BRPE0
IF1CREQL	—	—	MSG1N5	MSG1N4	MSG1N3	MSG1N2	MSG1N1	MSG1N0
IF1CREQH	BUSY1	—	—	—	—	—	—	—
IF1CMSKL	TD1DIR	MASK1	ARB1	CTRL1	CINTPND1	TQ1DTA	DATA1A	DATA1B
IF1MSK1L	MSK107	MSK106	MSK105	MSK104	MSK103	MSK102	MSK101	MSK100
IF1MSK1H	MSK115	MSK114	MSK113	MSK112	MSK111	MSK110	MSK109	MSK108
IF1MSK2L	MSK123	MSK122	MSK121	MSK120	MSK119	MSK118	MSK117	MSK116
IF1MSK2H	MXTD1	MDIR1	—	MSK128	MSK127	MSK126	MSK125	MSK124
IF1ARB1L	ID107	ID106	ID105	ID104	ID103	ID102	ID101	ID100
IF1ARB1H	ID115	ID114	ID113	ID112	ID111	ID110	ID109	ID108
IF1ARB2L	ID123	ID122	ID121	ID120	ID119	ID118	ID117	ID116
IF1ARB2H	MSG1VA	XTD1	DIR1	ID128	ID127	ID126	ID125	ID124
IF1MCTRL	EOB1	—	—	—	DLC13	DLC12	DLC11	DLC10
IF1MCTRH	N1DTA	MSG1LST	INT1PND	UMASK1	TX1IEN	RX1IEN	RMT1EN	T1REQ
IF1DTA1L	D7	D6	D5	D4	D3	D2	D1	D0
IF1DTA1H	D7	D6	D5	D4	D3	D2	D1	D0
IF1DTA2L	D7	D6	D5	D4	D3	D2	D1	D0
IF1DTA2H	D7	D6	D5	D4	D3	D2	D1	D0
IF1DTB1L	D7	D6	D5	D4	D3	D2	D1	D0
IF1DTB1H	D7	D6	D5	D4	D3	D2	D1	D0
IF1DTB2L	D7	D6	D5	D4	D3	D2	D1	D0
IF1DTB2H	D7	D6	D5	D4	D3	D2	D1	D0
CRLL	DAY7	DAY6	DAY5	DAY4	DAY3	DAY2	DAY1	DAY0
CRLH	MON7	MON6	MON5	MON4	MON3	MON2	MON1	MON0
CRHL	SUBSTEP3	SUBSTEP2	SUBSTEP1	SUBSTEP0	YEAR3	YEAR2	YEAR1	YEAR0
CRHH	REL3	REL2	REL1	REL0	STEP3	STEP2	STEP1	STEP0
IF2CREQL	—	—	MSG2N5	MSG2N4	MSG2N3	MSG2N2	MSG2N1	MSG2N0
IF2CREQH	BUSY2	—	—	—	—	—	—	—
IF2CMSKL	TD2DIR	MASK2	ARB2	CTRL2	CINTPND2	TQ2DTA	DATA2A	DATA2B
IF2MSK1L	MSK207	MSK206	MSK205	MSK204	MSK203	MSK202	MSK201	MSK200
IF2MSK1H	MSK215	MSK214	MSK213	MSK212	MSK211	MSK210	MSK209	MSK208
IF2MSK2L	MSK223	MSK222	MSK221	MSK220	MSK219	MSK218	MSK217	MSK216
IF2MSK2H	MXTD2	MDIR2	—	MSK228	MSK227	MSK226	MSK225	MSK224
IF2ARB1L	ID207	ID206	ID205	ID204	ID203	ID202	ID201	ID200
IF2ARB1H	ID215	ID214	ID213	ID212	ID211	ID210	ID209	ID208

Register Name	Bit							
	7	6	5	4	3	2	1	0
IF2ARB2L	ID223	ID222	ID221	ID220	ID219	ID218	ID217	ID216
IF2ARB2H	MSG2VA	XTD2	DIR2	ID228	ID227	ID226	ID225	ID224
IF2MCTRL	EOB2	—	—	—	DLC23	DLC22	DLC21	DLC20
IF2MCTRH	N2DTA	MSG2LST	INT2PND	UMASK2	TX2IEN	RX2IEN	RMT2EN	T2REQ
IF2DTA1L	D7	D6	D5	D4	D3	D2	D1	D0
IF2DTA1H	D7	D6	D5	D4	D3	D2	D1	D0
IF2DTA2L	D7	D6	D5	D4	D3	D2	D1	D0
IF2DTA2H	D7	D6	D5	D4	D3	D2	D1	D0
IF2DTB1L	D7	D6	D5	D4	D3	D2	D1	D0
IF2DTB1H	D7	D6	D5	D4	D3	D2	D1	D0
IF2DTB2L	D7	D6	D5	D4	D3	D2	D1	D0
IF2DTB2H	D7	D6	D5	D4	D3	D2	D1	D0
TREQR1L	TREQ8	TREQ7	TREQ6	TREQ5	TREQ4	TREQ3	TREQ2	TREQ1
TREQR1H	TREQ16	TREQ15	TREQ14	TREQ13	TREQ12	TREQ11	TREQ10	TREQ9
TREQR2L	TREQ24	TREQ23	TREQ22	TREQ21	TREQ20	TREQ19	TREQ18	TREQ17
TREQR2H	TREQ32	TREQ31	TREQ30	TREQ29	TREQ28	TREQ27	TREQ26	TREQ25
NEWDT1L	NDTA8	NDTA7	NDTA6	NDTA5	NDTA4	NDTA3	NDTA2	NDTA1
NEWDT1H	NDTA16	NDTA15	NDTA14	NDTA13	NDTA12	NDTA11	NDTA10	NDTA9
NEWDT2L	NDTA24	NDTA23	NDTA22	NDTA21	NDTA20	NDTA19	NDTA18	NDTA17
NEWDT2H	NDTA32	NDTA31	NDTA30	NDTA29	NDTA28	NDTA27	NDTA26	NDTA25
INTPND1L	INTPND8	INTPND7	INTPND6	INTPND5	INTPND4	INTPND3	INTPND2	INTPND1
INTPND1H	INTPND16	INTPND15	INTPND14	INTPND13	INTPND12	INTPND11	INTPND10	INTPND9
INTPND2L	INTPND24	INTPND23	INTPND22	INTPND21	INTPND20	INTPND19	INTPND18	INTPND17
INTPND2H	INTPND32	INTPND31	INTPND30	INTPND29	INTPND28	INTPND27	INTPND26	INTPND25
MSGVAL1L	MSGVA8	MSGVA7	MSGVA6	MSGVA5	MSGVA4	MSGVA3	MSGVA2	MSGVA1
MSGVAL1H	MSGVA16	MSGVA15	MSGVA14	MSGVA13	MSGVA12	MSGVA11	MSGVA10	MSGVA9
MSGVAL2L	MSGVA24	MSGVA23	MSGVA22	MSGVA21	MSGVA20	MSGVA19	MSGVA18	MSGVA17
MSGVAL2H	MSGVA32	MSGVA31	MSGVA30	MSGVA29	MSGVA28	MSGVA27	MSGVA26	MSGVA25
CANCFG	D7	CANEN	—	RMFD4	RMFD3	RMFD2	RMFD1	RMFD0

"—": unimplemented, read as "0"

Programmer's Model

Register	Description	Note
CTRLRL	CAN Control Register	—
STATRL	Status Register	—
ERRCNTH/ERRCNTL	Error Counter	Read only
BTRH/BTRL	Bit Timing Register	Write enabled by CCE
INTRH/INTRL	Interrupt Register	Read only
TESTRL	Test Register	Write enabled by TEST
BRPERL	BRP Extension Register	Write enabled by CCE
IF1CREQH/IF1CREQL	IF1 Command Request	—
IF1CMSKL	IF1 Command Mask	—
IF1MSK1H/IF1MSK1L	IF1 Mask 1	—
IF1MSK2H/IF1MSK2L	IF1 Mask 2	—
IF1ARB1H/IF1ARB1L	IF1 Arbitration 1	—
IF1ARB2H/IF1ARB2L	IF1 Arbitration 2	—
IF1MCTRH/IF1MCTRL	IF1 Message Control	—

Register	Description	Note
IF1DTA1H/IF1DTA1L	IF1 Data A 1	—
IF1DTA2H/IF1DTA2L	IF1 Data A 2	—
IF1DTB1H/IF1DTB1L	IF1 Data B 1	—
IF1DTB2H/IF1DTB2L	IF1 Data B 2	—
IF2CREQH/IF2CREQL	IF2 Command Request	—
IF2CMSKL	IF2 Command Mask	—
IF2MSK1H/IF2MSK1L	IF2 Mask 1	—
IF2MSK2H/IF2MSK2L	IF2 Mask 2	—
IF2ARB1H/IF2ARB1L	IF2 Arbitration 1	—
IF2ARB2H/IF2ARB2L	IF2 Arbitration 2	—
IF2MCTRH/IF2MCTRL	IF2 Message Control	—
IF2DTA1H/IF2DTA1L	IF2 Data A 1	—
IF2DTA2H/IF2DTA2L	IF2 Data A 2	—
IF2DTB1H/IF2DTB1L	IF2 Data B 1	—
IF2DTB2H/IF2DTB2L	IF2 Data B 2	—
CRLH/CRLL	Core Release Low	Read only
CRHH/CRHL	Core Release High	Read only
TREQR1H/TREQR1L	Transmission Request 1	Read only
TREQR2H/TREQR2L	Transmission Request 2	Read only
NEWDT1H/NEWDT1L	New Data 1	Read only
NEWDT2H/NEWDT2L	New Data 2	Read only
INTPND1H/INTPND1L	Interrupt Pending 1	Read only
INTPND2H/INTPND2L	Interrupt Pending 2	Read only
MSGVAL1H/MSGVAL1L	Message Valid 1	Read only
MSGVAL2H/MSGVAL2L	Message Valid 2	Read only
CANCFG	CAN Module Configuration	—

CAN Module Configuration Register

- **CANCFG Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	CANEN	—	RMFD4	RMFD3	RMFD2	RMFD1	RMFD0
R/W	R/W	R/W	—	R/W	R/W	R/W	R/W	R/W
POR	1	0	—	0	0	0	0	0

Bit 7 **D7**: Reserved, must be fixed at "1"

Bit 6 **CANEN**: CAN Module Core Enable Control
0: Disable
1: Enable
When this bit is cleared to zero, CAN core remains in reset state and cannot be written in.

Bit 5 Unimplemented, read as "0"

Bit 4~0 **RMFD4~RMFD0**: Set receive FIFO threshold
0x00: Message Object Number 32
0x01~0x1F: Message Object Number 1 ~ Message Object Number 31
These bits are used to select one of the Message Object Number 1 ~ Message Object Number 32. When the selected Message Object receives a Message successfully, an interrrupt signal RMXFINT will be generated.

CAN Protocol Related Registers

These registers are related to the CAN protocol controller in the CAN Core. They control the operating modes and the configuration of the CAN bit timing and provide status information.

CAN Module Control Register

The contents of the control register are used to change the behavior of the CAN module operation.

- **CTRLRL Register**

Bit	7	6	5	4	3	2	1	0
Name	TEST	CCE	DAR	—	EIE	SIE	CANIE	INIT
R/W	R/W	R/W	R/W	—	R/W	R/W	R/W	R/W
POR	0	0	0	—	0	0	0	1

Bit 7 **TEST**: Test Mode Enable Control
0: Normal Operation
1: Test Mode

Bit 6 **CCE**: Configuration Change Enable
0: The CPU has no write access to protected register bits
1: While INIT = ‘1’, the CPU has write access to protected register bits

Bit 5 **DAR**: Disable Automatic Retransmission
0: Enable Automatic Retransmission of disturbed messages
1: Disable Automatic Retransmission of disturbed messages

Bit 4 Unimplemented, read as “0”

Bit 3 **EIE**: Error Interrupt Enable
0: Disabled – No Error Status Interrupt will be generated
1: Enabled – A change of bits BOFF or EWARN in the Status Register will cause the Interrupt Register to be set to Status Interrupt (INTID15~INTID0=0x8000)

Bit 2 **SIE**: Status Change Interrupt Enable
0: Disabled – No Status Change Interrupt will be generated
1: Enabled – The Interrupt Register will be set to Status Interrupt (INTID15~INTID0=0x8000) when the CAN module sets LEC[2:0] to a value other than ‘7’.

Bit 1 **CANIE**: Module Interrupt Enable
0: Disabled – Module Interrupt can_int is always inactive
1: Enabled – When the Interrupt Register is non-zero, the interrupt line can_int is set to active. can_int remains active until all interrupts are processed (Interrupt Register returns to zero)

Bit 0 **INIT**: Initialisation
0: Normal Operation
1: Initialisation is started

The busoff recovery sequence (see CAN Specification Rev.2.0) cannot be shortened by setting or resetting INIT. If the device goes busoff, it will set INIT of its own accord, stopping all bus activities.

Once INIT has been cleared by the CPU, the device will then wait for 129 occurrences of Bus Idle (129×11 consecutive recessive bits) before resuming normal operations. At the end of the busoff recovery sequence, the Error Management Counters will be reset.

Test Register

Write access to the Test Register TESTRL is enabled by setting the bit, TEST, in the CAN Molule Control Register. Different test functions may be combined, but TX[1:0] ≠ "00" disturbs message transfer. The TESTRL register should be cleared to zero before exiting the Test Mode.

- **TESTRL Register**

Bit	7	6	5	4	3	2	1	0
Name	RX	TX1	TX0	LBACK	SILENT	BASIC	—	—
R/W	R	R/W	R/W	R/W	R/W	R/W	—	—
POR	x	0	0	0	0	0	—	—

Bit 7 **RX**: Monitors the actual value of the CANRX Pin
0: The CAN bus is dominant (can_rx = '0')
1: The CAN bus is recessive (can_rx = '1')
Note: The POR value of 'x' signifies the actual POR value of the CANRX pin.

Bit 6~5 **TX1~TX0**: Control of CANTX pin
00: Reset value, CANTX is controlled by the CAN Core
01: Sample Point can be monitored at CANTX pin
10: CANTX pin drives a dominant ('0') value
11: CANTX pin drives a recessive ('1') value

Bit 4 **LBACK**: Loop Back Mode Control
0: Loop Back Mode is disabled
1: Loop Back Mode is enabled

Bit 3 **SILENT**: Silent Mode Control
0: Normal operation
1: The module is in Silent Mode

Bit 2 **BASIC**: Basic Mode Control
0: Basic Mode disabled
1: Basic Mode, IF1 Registers used as TX Buffer, IF2 Registers used as RX Buffer

Bit 1~0 Unimplemented, read as "0"

Status Register

The content of this register reflects the CAN module status. Some bits can only be read while some are read/write bits.

- **STATRL Register**

Bit	7	6	5	4	3	2	1	0
Name	BOFF	EWARN	EPASS	RXOK	TXOK	LEC2	LEC1	LEC0
R/W	R	R	R	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **BOFF**: Busoff Status
0: The CAN module is not in the busoff state
1: The CAN module is in the busoff state

Bit 6 **EWARN**: Error Warning Status
0: Both error counters are below the error warning limit of 96
1: At least one of the error counters in the EML has reached the error warning limit of 96

Bit 5 **EPASS**: Error Passive
0: The CAN Core is error active
1: The CAN Core is error passive as defined in the CAN Specification

Bit 4 **RXOK**: Received a Message Successfully

0: Since this bit was last reset by the CPU, no message has been successfully received

1: Since this bit was last reset by the CPU, a message has been successfully received independent of the result of acceptance filtering

This RXOK bit is never reset by the CAN Core.

Bit 3 **TXOK**: Transmitted a Message Successfully

0: Since this bit was reset by the CPU, no message has been successfully transmitted

1: Since this bit was last reset by the CPU, a message has been successfully transmitted error free and acknowledged by at least one other node

This TXOK bit is never reset by the CAN Core.

Bit 2~0 **LEC2~LEC0**: Last Error Code (Type of the last protocol event to occur on the CAN bus)

000: No Error

Message successfully transmitted or received.

001: Stuff Error

More than 5 equal bits in a sequence have occurred in a part of a received message where this is not allowed.

010: Form Error

Fixed format part of a received frame has the wrong format.

011: Ack Error

CAN Core transmitted message was not acknowledged by another node.

100: Bit1 Error

During the transmission of a message, with the exception of the arbitration field, the device wished to send a recessive level (bit of logical value ‘1’), but the monitored bus value was dominant.

101: Bit0 Error

During the transmission of a message or an acknowledge bit, active error flag, or overload flag, the device wished to send a dominant level (data or identifier bit of logical value ‘0’), but the monitored Bus value was recessive. During busoff recovery this status is set each time a sequence of 11 recessive bits has been monitored. This enables the CPU to monitor the busoff recovery sequence, indicating that the bus is not stuck at dominant or is continuously disturbed. During the waiting time after an INIT reset, each time a sequence of 11 recessive bits has been monitored, a Bit0 Error code is written to the Status Register. This enables the CPU to readily check up whether the CAN bus is stuck at dominant or is continuously disturbed and to monitor the busoff recovery sequence.

110: CRC Error

The CRC check sum was incorrect in the message received, the CRC received for an incoming message wished to send a recessive level (bit of logical value ‘1’), but the monitored bus value was dominant.

111: No Change

When the LEC bit field shows the value ‘7’, no CAN bus event was detected since the CPU wrote this value to the LEC bit field.

The LEC field holds a code which indicates the type of the last error to occur on the CAN bus. This field will be cleared to ‘0’ when a message has been transferred (reception or transmission) without error. A code ‘7’ may be written by the CPU to check for updates.

Note: A Status Interrupt is generated by bits BOFF and EWARN (Error Interrupt) or by RXOK, TXOK and LEC (Status Change Interrupt) assumed that the corresponding enable bits in the CAN Control Register are set. A change of bit EPASS or a write to RXOK, TXOK or LEC will never generate a Status Interrupt.

Reading the Status Register will clear the Status Interrupt value (INTID15~INTID0=0x8000) in the Interrupt Register, if it is pending.

CAN Error Counter Registers

- **ERRCNTL Register**

Bit	7	6	5	4	3	2	1	0
Name	TEC7	TEC6	TEC5	TEC4	TEC3	TEC2	TEC1	TEC0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **TEC7~TEC0**: Transmit Error Counter
Actual state of the Transmit Error Counter

- **ERRCNTH Register**

Bit	7	6	5	4	3	2	1	0
Name	RP	REC6	REC5	REC4	REC3	REC2	REC1	REC0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7 **RP**: Receive Error Passive
0: The Receive Error Counter is below the error passive level
1: The Receive Error Counter has reached the error passive level as defined in the CAN Specification

Bit 6~0 **REC6~REC0**: Receive Error Counter
Actual state of the Receive Error Counter

Bit Timing Registers

The bit time is divided into four segments which are the Synchronisation Segment, the Propagation Time Segment, the Phase Buffer Segment 1 and the Phase Buffer Segment 2. Each segment consists of a specific, programmable number of time quanta. The length of the time quantum (t_q), which is the basic time unit of the bit time, is defined by the CAN module input clock f_{CAN} and the Baud Rate Prescaler (BRP) and the BRP Extension Register.

- The time quantum is defined as:
  $t_q = (BRPE[3:0] \times 0x40 + BRP[5:0] + 1)/f_{CAN}$.

Where f_{CAN} = CAN module clock frequency

The contents of the BTRL register define the values of the Baud Rate Prescaler and the (Re) Synchronisation Jump Width (SJW). The BTRH register bits define the length of the time segment before and after the sample point. The registers are only writable if bits CCE and INIT in the CAN Control Register are set.

- **BTRL Register**

Bit	7	6	5	4	3	2	1	0
Name	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	1

Bit 7~6 **SJW1~SJW0**: (Re)Synchronisation Jump Width
0x00~0x03: Valid programmed values are 0~3.

The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

Bit 5~0 **BRP5~BRP0**: Baud Rate Prescaler (base value)
0x01~0x3F: The value by which the CAN module system clock frequency is divided to generate the bit time quanta. The bit time is built up from a multiple of this time quanta. Valid values for BRP[5:0] are 0~63.

The actual interpretation by the hardware of this value is such that one more than the programmed value is used.

- **BTRH Register**

Bit	7	6	5	4	3	2	1	0
Name	—	TSG2D2	TSG2D1	TSG2D0	TSG1D3	TSG1D2	TSG1D1	TSG1D0
R/W	—	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	0	1	0	0	0	1	1

Bit 7 Unimplemented, read as "0"

Bit 6~4 **TSG2D2~TSG2D0**: The time segment after the sample point
0x00~0x07: Valid values for TSG2D[2:0] are 0~7. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

Bit 3~0 **TSG1D3~TSG1D0**: The time segment before the sample point
0x01~0x0F: Valid values for TSG1D[3:0] are 1~15. The actual interpretation by the hardware of this value is such that one more than the value programmed here is used.

Note: If f_{CAN}=8MHz, the reset value of BTRL=0x01 and BTRH=0x23 configures the CAN module for a bit rate of 500kBit/s.

- **BRPERL register**

This BRPERL register configures the BRP extension for Classic CAN operation. The register is writable by setting CCE bit.

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	BRPE3	BRPE2	BRPE1	BRPE0
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	0	0	0

Bit 7~4 Unimplemented, read as "0"

Bit 3~0 **BRPE3~BRPE0**: Baud Rate Prescaler Extension
0x00~0x0F: By programming BRPE[3:0] the Baud Rate Prescaler can be extended to values up to 1023.
The actual interpretation by the hardware is that one more than the value programmed by BRPE[3:0] (MSBs) and BRP[5:0] (LSBs) is used.

Message Interface Registers

There are two sets of Interface Registers which are used to control the CPU access to the Message RAM. The Interface Registers avoid conflicts between CPU access to the Message RAM and CAN message reception and transmission by buffering the data to be transferred. A complete Message Object or parts of the Message Object may be transferred between the Message RAM and the IFn Message Buffer registers in one single transfer.

The function of the two interface register sets is identical, except for the Basic test mode. They can be used in the way that one set of registers is used for data transfer to the Message RAM while the other set of registers is used for the data transfer from the Message RAM, allowing both processes to be interrupted by each other.

Each set of Interface Registers consists of Message Buffer Registers controlled by their own Command Registers. The Command Mask Register specifies the direction of the data transfer and which parts of a Message Object will be transferred. The Command Request Register is used to select a Message Object in the Message RAM as target or source for the transfer and to start the action specified in the Command Mask Register.

IF1 Register Set	Description	IF2 Register Set	Description
IF1CREQH/IF1CREQL	IF1 Command Request	IF2CREQH/IF2CREQL	IF2 Command Request
IF1CMSKL	IF1 Command Mask	IF2CMSKL	IF2 Command Mask
IF1MSK1H/IF1MSK1L	IF1 Mask 1	IF2MSK1H/IF2MSK1L	IF2 Mask 1
IF1MSK2H/IF1MSK2L	IF1 Mask 2	IF2MSK2H/IF2MSK2L	IF2 Mask 2
IF1ARB1H/IF1ARB1L	IF1 Arbitration 1	IF2ARB1H/IF2ARB1L	IF2 Arbitration 1
IF1ARB2H/IF1ARB2L	IF1 Arbitration 2	IF2ARB2H/F2ARB2L	IF2 Arbitration 2
IF1MCTRH/IF1MCTRL	IF1 Message Control	IF2MCTRH/IF2MCTRL	IF2 Message Control
IF1DTA1H/IF1DTA1L	IF1 Data A1	IF2DTA1H/IF2DTA1L	IF2 Data A1
IF1DTA2H/IF1DTA2L	IF1 Data A2	IF2DTA2H/IF2DTA2L	IF2 Data A2
IF1DTB1H/IF1DTB1L	IF1 Data B1	IF2DTB1H/IF2DTB1L	IF2 Data B1
IF1DTB2H/IF1DTB2L	IF1 Data B2	IF2DTB2H/IF2DTB2L	IF2 Data B2

IF1 and IF2 Message Interface Register Sets

IFn Command Request Registers

A message transfer is started as soon as the CPU has written the message number to the Command Request Register. With this write operation the BUSYn bit is automatically set to ‘1’ and output can_wait_b is activated to notify the CPU that a transfer is in progress. After a wait time of 3 to 6 can_clk periods, the transfer between the Interface Register and the Message RAM will have completed. The BUSYn bit is set back to zero and can_wait_b is deactivated (see Module Integration Guide).

- **IFnCREQL Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	MSGnN5	MSGnN4	MSGnN3	MSGnN2	MSGnN1	MSGnN0
R/W	—	—	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	—	0	0	0	0	0	1

Bit 7~6 Unimplemented, read as “0”

Bit 5~0 **MSGnN5~MSGnN0**: Message Number
0x00: Not a valid Message Number – interpreted as 0x20
0x01~0x20: Valid Message Number – the Message Object in the Message RAM is selected for data transfer.
0x21~0x3F: Not a valid Message Number – interpreted as 0x01-0x1F
Note: When a Message Number that is not valid is written into the Command Request Register, the Message Number will be transformed into a valid value and that Message Object will be transferred.

- **IFnCREQH Register**

Bit	7	6	5	4	3	2	1	0
Name	BUSYn	—	—	—	—	—	—	—
R/W	R/W	—	—	—	—	—	—	—
POR	0	—	—	—	—	—	—	—

Bit 7 **BUSYn**: Busy Flag
0: Reset to zero when a read/write action has finished
1: Set to one when writing to the IFn Command Request Register

Bit 6~0 Unimplemented, read as “0”

IFn Command Mask Registers

The control bits of the IFn Command Mask Register specify the transfer direction and select which of the IFn Message Buffer Registers is to be used as a source or target of the data transfer.

- **IFnCMSKL Register**

Bit	7	6	5	4	3	2	1	0
Name	TDnDIR	MASKn	ARBn	CTRLn	CINTPNDn	TQnDTA	DATAnA	DATAnB
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **TDnDIR**: Write / Read Selection
0: Read – Transfer data from the Message Object addressed by the Command Request Register into the selected Message Buffer Registers.
1: Write – Transfer data from the selected Message Buffer Registers to the Message Object addressed by the Command Request Register.

The other bits of IFn Command MASKn Register have different functions depending on the transfer direction:

- **Direction = Write (TDnDIR=1)**

Bit 6 **MASKn**: Access MASK Bits
0: MASK bits unchanged.
1: Transfer Identifier MASKn + MDIRn + MXTDn to Message Object.

Bit 5 **ARBn**: Access Arbitration Bits
0: Arbitration bits unchanged.
1: Transfer Identifier + DIRn + XTDn + MSGnVA to Message Object.

Bit 4 **CTRLn**: Access Control Bits
0: Control Bits unchanged.
1: Transfer Control Bits to Message Object.

Bit 3 **CINTPNDn**: Clear Interrupt Pending Bit
0: INTPND bit remains unchanged.
1: Clear INTPND bit in the Message Object.
Note: When writing to a Message Object, this bit is ignored.

Bit 2 **TQnDTA**: Access Transmission Request Bit
0: TREQ bit unchanged
1: Set TREQ bit
If a transmission is requested by programming bit TQnDTA in the IFn Command Mask Register, bit TnREQ in the IFn Message Control Register will be ignored.

Bit 1 **DATAnA**: Access Data Bytes 0-3
0: Data Bytes 0-3 unchanged.
1: Transfer Data Bytes 0-3 to Message Object.

Bit 0 **DATAnB**: Access Data Bytes 4-7
0: Data Bytes 4-7 unchanged
1: Transfer Data Bytes 4-7 to Message Object.

- **Direction = Read (TDnDIR=0)**

Bit 6 **MASKn**: Access MASK Bits
0: MASK bits unchanged.
1: Transfer Identifier MASKn + MDIRn + MXTDn to IFn Message Buffer Register.

Bit 5 **ARBn**: Access Arbitration Bits
0: Arbitration bits unchanged.
1: transfer Identifier + DIRn + XTDn + MSGnVA to IFn Message Buffer Register.

Bit 4 **CTRLn**: Access Control Bits
0: Control Bits unchanged.
1: Transfer Control Bits to IFn Message Buffer Register.

Bit 3 **CINTPNDn**: Clear Interrupt Pending Bit
0: INTPND bit remains unchanged.
1: Clear INTPND bit in the Message Object.

Bit 2 **TQnDTA**: Access New Data Bit
0: NDTA bit remains unchanged.
1: Clear NDTA bit in the Message Object.

Note: A read access to a Message Object can be combined with the reset of the control bits INTnPND and NnDTA.The values of these bits transferred to the IFn Message Control Register always reflect the status before resetting these bits.

Bit 1 **DATAnA**: Access Data Bytes 0-3
0: Data Bytes 0-3 unchanged.
1: Transfer Data Bytes 0-3 to IFn Message Buffer Register.

Bit 0 **DATAnB**: Access Data Bytes 4-7
0: Data Bytes 4-7 unchanged.
1: Transfer Data Bytes 4-7 to IFn Message Buffer Register.

IFn Message Buffer Registers

The bits of the Message Buffer registers mirror the Message Objects in the Message RAM.

- **IFnMSK1L Register**

Bit	7	6	5	4	3	2	1	0
Name	MSKn07	MSKn06	MSKn05	MSKn04	MSKn03	MSKn02	MSKn01	MSKn00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	1	1	1	1	1	1	1

Bit 7~0 **MSKn07~MSKn00**: Identifier MASK
0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering.
1: The corresponding identifier bit is used for acceptance filtering.

- **IFnMSK1H Register**

Bit	7	6	5	4	3	2	1	0
Name	MSKn15	MSKn14	MSKn13	MSKn12	MSKn11	MSKn10	MSKn09	MSKn08
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	1	1	1	1	1	1	1

Bit 7~0 **MSKn15~MSKn08**: Identifier MASK
0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering.
1: The corresponding identifier bit is used for acceptance filtering.

- **IFnMSK2L Register**

Bit	7	6	5	4	3	2	1	0
Name	MSKn23	MSKn22	MSKn21	MSKn20	MSKn19	MSKn18	MSKn17	MSKn16
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	1	1	1	1	1	1	1

Bit 7~0 **MSKn23~MSKn16**: Identifier MASK
0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering
1: The corresponding identifier bit is used for acceptance filtering

- **IFnMSK2H Register**

Bit	7	6	5	4	3	2	1	0
Name	MXTDn	MDIRn	—	MSKn28	MSKn27	MSKn26	MSKn25	MSKn24
R/W	R/W	R/W	—	R/W	R/W	R/W	R/W	R/W
POR	1	1	—	1	1	1	1	1

Bit 7 **MXTDn**: MASK Extended Identifier
0: The extended identifier bit (IDE) has no effect on the acceptance filtering
1: The extended identifier bit (IDE) is used for acceptance filtering

Bit 6 **MDIRn**: MASK Message Direction
0: The message direction bit (DIR) has no effect on the acceptance filtering
1: The message direction bit (DIR) is used for acceptance filtering

Bit 5 Unimplemented, read as “1”

Bit 4~0 **MSKn28~MSKn24**: Identifier MASK
0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering
1: The corresponding identifier bit is used for acceptance filtering

- **IFnARB1L Register**

Bit	7	6	5	4	3	2	1	0
Name	IDn07	IDn06	IDn05	IDn04	IDn03	IDn02	IDn01	IDn00
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **IDn07~IDn00**: Message Identifier 7~0

- **IFnARB1H Register**

Bit	7	6	5	4	3	2	1	0
Name	IDn15	IDn14	IDn13	IDn12	IDn11	IDn10	IDn09	IDn08
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **IDn15~IDn8**: Message Identifier 15~8

- **IFnARB2L Register**

Bit	7	6	5	4	3	2	1	0
Name	IDn23	IDn22	IDn21	IDn20	IDn19	IDn18	IDn17	IDn16
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **IDn23~IDn16**: Message Identifier 23~16

- **IFnARB2H Register**

Bit	7	6	5	4	3	2	1	0
Name	MSGnVA	XTDn	DIRn	IDn28	IDn27	IDn26	IDn25	IDn24
R/W	RW	RW	RW	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **MSGnVA**: Message Valid Bits – all Message Objects
0: This Message Object is ignored by the Message Handler
1: This Message Object is configured and should be considered by the Message Handler

Bit 6 **XTDn**: Extended Identifier
0: The 11-bit (“standard”) Identifier will be used for this Message Object.
1: The 29-bit (“extended”) Identifier will be used for this Message Object.

Bit 5 **DIRn**: Message Direction
0: Direction=receive. On TREQ, a Remote Frame with the identifier of this Message Object is transmitted. On reception of a Data Frame with matching identifier, that message is stored in this Message Object.
1: Direction=transmit. On TREQ, the respective Message Object is transmitted as a Data Frame. On reception of a Remote Frame with matching identifier, the TREQ bit of this Message Object is set if RMTnEN = 1.

Bit 4~0 **IDn28~IDn24**: Message Identifier 28~24

IFn Message Control Registers

- **IFnMCTRL Register**

Bit	7	6	5	4	3	2	1	0
Name	EOBn	—	—	—	DLCn3	DLCn2	DLCn1	DLCn0
R/W	R/W	—	—	—	R/W	R/W	R/W	R/W
POR	0	—	—	—	0	0	0	0

Bit 7 **EOBn**: End of Buffer
0: Message Object belongs to a FIFO Buffer and is not the last Message Object of that FIFO Buffer.
1: Single Message Object or last Message Object of a FIFO Buffer.

This bit is used to concatenate two or more Message Objects (up to 32) to build a FIFO Buffer. For single Message Objects not belonging to a FIFO Buffer, this bit must always be set to one.

Bit 6~4 Unimplemented, read as “0”

Bit 3~0 **DLCn3~DLCn0**: Data Length Code
0~8: CAN: Frame has 0~8 data bytes
9~15: CAN: Frame has 8 data bytes

The Data Length Code of a Message Object must be defined the same as in all the corresponding objects with the same identifier at other nodes.

When the Message Handler stores a data frame, it will write the DLCn to the value given by the received message.

- **IFnMCTRH Register**

Bit	7	6	5	4	3	2	1	0
Name	NnDTA	MSGnLST	INTnPND	UMASKn	TXnIEN	RXnIEN	RMTnEN	TnREQ
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **NnDTA**: New Data Bits
0: No new data has been written into the data portion of this Message Object by the Message Handler since the last time this flag was cleared by the CPU.
1: The Message Handler or the CPU has written new data into the data portion of this Message Object

Bit 6 **MSGnLST**: Message Lost (only valid for Message Objects with direction = receive)
0: No message lost since the last time this bit was reset by the CPU
1: The Message Handler stored a new message into this object when NDTA was still set, the CPU has lost a message

Bit 5 **INTnPND**: Interrupt Pending Bits
0: This message object is not the source of an interrupt
1: This message object is the source of an interrupt. The Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with a higher priority

Bit 4 **UMASKn**: Use Acceptance Mask
0: MASK ignored
1: Use MASK (MSKn[28:00], MXTDn, and MDIRn) for acceptance filtering

Bit 3 **TXnIEN**: Transmit Interrupt Enable
0: INTPND will be left unchanged after the successful transmission of a frame
1: INTPND will be set after a successful transmission of a frame

Bit 2 **RXnIEN**: Receive Interrupt Enable
0: INTPND will be left unchanged after a successful reception of a frame
1: INTPND will be set after a successful reception of a frame

Bit 1 **RMTnEN**: Remote Enable
0: At the reception of a Remote Frame, TREQ is left unchanged
1: At the reception of a Remote Frame, TREQ is set

Bit 0 **TnREQ**: Transmission Request Bits
0: This Message Object is not waiting for transmission
1: The transmission of this Message Object is requested and is not yet completed

IFn Data A and Data B Registers

In a CAN Data Frame Data0 is the first and Data7 is the last byte to be transmitted or received. In a CAN serial bit stream, the MSB of each byte will be transmitted first.

DATA0: 1st data byte of a CAN Data Frame

DATA1: 2nd data byte of a CAN Data Frame

DATA2: 3rd data byte of a CAN Data Frame

DATA3: 4th data byte of a CAN Data Frame

DATA4: 5th data byte of a CAN Data Frame

DATA5: 6th data byte of a CAN Data Frame

DATA6: 7th data byte of a CAN Data Frame

DATA7: 8th data byte of a CAN Data Frame

The data bytes of CAN messages are stored in the IFn Message Buffer Registers in the following order:

- **IFnDTA1L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA0, 1st data byte of a CAN Data Frame

- **IFnDTA1H Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA1, 2nd data byte of a CAN Data Frame

- **IFnDTA2L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA2, 3rd data byte of a CAN Data Frame

- **IFnDTA2H Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA3, 4th data byte of a CAN Data Frame

- **IFnDTB1L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA4, 5th data byte of a CAN Data Frame

- **IFnDTB1H Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA5, 6th data byte of a CAN Data Frame

- **IFnDTB2L Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA6, 7th data byte of a CAN Data Frame

- **IFnDTB2H Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: DATA7, 8th data byte of a CAN Data Frame

Note: Byte DATA0 is the first data byte shifted into the shift register of the CAN Core during a reception and byte DATA7 is the last. When the Message Handler stores a Data Frame, it will write all the eight data bytes into a Message Object. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by non specified values.

Message Handler Registers

All Message Handler registers are read-only. Their contents, which include the TREQ, NDTA, INTPND, and MSGVA bits of each Message Object and the Interrupt Identifier are status information provided by the Message Handler FSM (Finite State Machine).

Interrupt Registers

The interrupt registers allow the identification of an interrupt source. When an interrupt occurs, a CAN interrupt will be indicated to inform the CPU. The interrupt register appears to the CPU as a read only memory.

- **INTRL Register**

Bit	7	6	5	4	3	2	1	0
Name	INTID7	INTID6	INTID5	INTID4	INTID3	INTID2	INTID1	INTID0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTID7~INTID0**: Interrupt Identifier

- **INTRH Register**

Bit	7	6	5	4	3	2	1	0
Name	INTID15	INTID14	INTID13	INTID12	INTID11	INTID10	INTID9	INTID8
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTID15~INTID8**: Interrupt Identifier
The interrupt identifier INTID[15:0] indicates the source of the interrupt.

INTID[15:0] Value	Indicated Interrupt
0000H	No interrupt is pending
0001H~0020H	Number of Message Object which caused the interrupt
0021H~7FFFH	Unused
8000H	Status Interrupt
8001H~FFFFH	Unused

If several interrupts are pending, the CAN Interrupt Register will point to the pending interrupt with the highest priority disregarding their chronological order. An interrupt remains pending until the CPU has cleared it. If INTID15~INTID0 is different from 0x0000 and CANIE is set, the interrupt line to the CPU, can_int, is active until INTID15~INTID0 is back to value 0x0000. This occurs when the reason for the interrupt is reset or until CANIE is reset.

The Status Interrupt has the highest priority. Among the message interrupts, the Message Object's interrupt priority decreases with increasing message number.

A message interrupt is cleared by clearing the Message Object's INTPND bit. The Status Interrupt is cleared by reading the Status Register.

The Status Interrupt value 0x8000 (INTID15~INTID0) indicates that an interrupt is pending because the CAN Core has updated (not necessarily changed) the Status Register (Error Interrupt or Status Interrupt). This interrupt has the highest priority. The CPU can update (reset) the Status Register bits RXOK, TXOK and LEC by writing to the Status Register. A write access by the CPU to the Status Registers can never generate or reset an interrupt.

All other values indicate that the source of the interrupt is one of the Message Objects. INTID points to the pending message interrupt with the highest interrupt priority.

The CPU controls whether a change of the Status Registers may cause the Interrupt Register to be set to INTID Status interrupt (bits EIE and SIE in the CAN Control Register) and whether the interrupt line becomes active when the Interrupt Register is not equal to zero (bit CANIE in the CAN Control Register). The Interrupt Register will be updated even when CANIE is not set.

The CPU has two possibilities to follow the source of a message interrupt. First it can follow the INTID in the Interrupt Register and second it can poll the Interrupt Pending Register.

An interrupt service routine reading the message that is the source of the interrupt may read the message and reset the Message Object's INTPND at the same time (bit CINTPNDn in the IFn Command Mask Register). When INTPND is cleared, the Interrupt Register will point to the next Message Object with a pending interrupt.

Transmission Request Registers

These registers hold the TREQ bits of the 32 Message Objects. By reading out the TREQ bits, the CPU can check for which Message Object a Transmission Request is pending. The TREQ bit of a specific Message Object can be set or reset by the CPU via the IFn Message Interface Registers or by the Message Handler after reception of a Remote Frame or after a successful transmission.

- **TREQR1L Register**

Bit	7	6	5	4	3	2	1	0
Name	TREQ8	TREQ7	TREQ6	TREQ5	TREQ4	TREQ3	TREQ2	TREQ1
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **TREQ8~TREQ1**: Transmission Request Bits of Message Object 8 ~ Message Object 1
0: This Message Object is not waiting for transmission.
1: The transmission of this Message Object is requested and is not yet completed.

- **TREQR1H Register**

Bit	7	6	5	4	3	2	1	0
Name	TREQ16	TREQ15	TREQ14	TREQ13	TREQ12	TREQ11	TREQ10	TREQ9
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **TREQ16~TREQ9**: Transmission Request Bits of Message Object 16 ~ Message Object 9
0: This Message Object is not waiting for transmission
1: The transmission of this Message Object is requested and is not yet completed

- **TREQR2L Register**

Bit	7	6	5	4	3	2	1	0
Name	TREQ24	TREQ23	TREQ22	TREQ21	TREQ20	TREQ19	TREQ18	TREQ17
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **TREQ24~TREQ17**: Transmission Request Bits of Message Object 24 ~ Message Object 17
0: This Message Object is not waiting for transmission
1: The transmission of this Message Object is requested and is not yet completed

- **TREQR2H Register**

Bit	7	6	5	4	3	2	1	0
Name	TREQ32	TREQ31	TREQ30	TREQ29	TREQ28	TREQ27	TREQ26	TREQ25
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **TREQ32~TREQ25**: Transmission Request Bits of Message Object 32 ~ Message Object 25
0: This Message Object is not waiting for transmission
1: The transmission of this Message Object is requested and is not yet done

New Data Registers

These registers hold the NDTA bits of the 32 Message Objects. By reading out the NDTA bits, the CPU can check for which Message Object the data portion was updated. The NDTA bit of a specific Message Object can be set/reset by the CPU via the IFn Message Interface Registers or by the Message Handler after reception of a Data Frame or after a successful transmission.

- **NEWDT1L Register**

Bit	7	6	5	4	3	2	1	0
Name	NDTA8	NDTA7	NDTA6	NDTA5	NDTA4	NDTA3	NDTA2	NDTA1
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **NDTA8~NDTA1**: New Data Bits of Message Object 8 ~ Message Object 1
0: No new data has been written into the data portion of this Message Object by the Message Handler since the last time this flag was cleared by the CPU.
1: The Message Handler or the CPU has written new data into the data portion of this Message Object.

- **NEWDT1H Register**

Bit	7	6	5	4	3	2	1	0
Name	NDTA16	NDTA15	NDTA14	NDTA13	NDTA12	NDTA11	NDTA10	NDTA9
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **NDTA16~NDTA9**: New Data Bits of Message Object 16 ~ Message Object 9
0: No new data has been written into the data portion of this Message Object by the Message Handler since the last time this flag was cleared by the CPU.
1: The Message Handler or the CPU has written new data into the data portion of this Message Object.

- **NEWDT2L Register**

Bit	7	6	5	4	3	2	1	0
Name	NDTA24	NDTA23	NDTA22	NDTA21	NDTA20	NDTA19	NDTA18	NDTA17
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **NDTA24~NDTA17**: New Data Bits of Message Object 24 ~ Message Object 17
0: No new data has been written into the data portion of this Message Object by the Message Handler since the last time this flag was cleared by the CPU.
1: The Message Handler or the CPU has written new data into the data portion of this Message Object.

- **NEWDT2H Register**

Bit	7	6	5	4	3	2	1	0
Name	NDTA32	NDTA31	NDTA30	NDTA29	NDTA28	NDTA27	NDTA26	NDTA25
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **NDTA32~NDTA25**: New Data Bits of Message Object 32 ~ Message Object 25
0: No new data has been written into the data portion of this Message Object by the Message Handler since the last time this flag was cleared by the CPU.
1: The Message Handler or the CPU has written new data into the data portion of this Message Object.

Interrupt Pending Registers

These registers hold the INTPND bits of the 32 Message Objects. By reading out the INTPND bits, the CPU can check for which Message Object an interrupt is pending. The INTPND bit of a specific Message Object can be set/reset by the CPU via the IFn Message Interface Registers or by the Message Handler after reception or after a successful transmission of a frame. This will also affect the value of INTID in the Interrupt Register.

- **INTPND1L Register**

Bit	7	6	5	4	3	2	1	0
Name	INTPND8	INTPND7	INTPND6	INTPND5	INTPND4	INTPND3	INTPND2	INTPND1
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTPND8~INTPND1**: Interrupt Pending Bits of Message Object 8 ~ Message Object 1
0: This message object is not the source of an interrupt.
1: This message object is the source of an interrupt.

If the message object whose interrupt pending bit is set high is the source of an interrupt, the Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with a higher priority.

- **INTPND1H Register**

Bit	7	6	5	4	3	2	1	0
Name	INTPND16	INTPND15	INTPND14	INTPND13	INTPND12	INTPND11	INTPND10	INTPND9
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTPND16~INTPND9**: Interrupt Pending Bits of Message Object 16 ~ Message Object 9
0: This message object is not the source of an interrupt
1: This message object is the source of an interrupt

If the message object whose interrupt pending bit is set high is the source of an interrupt, the Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with a higher priority.

- **INTPND2L Register**

Bit	7	6	5	4	3	2	1	0
Name	INTPND24	INTPND23	INTPND22	INTPND21	INTPND20	INTPND19	INTPND18	INTPND17
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTPND24~INTPND17**: Interrupt Pending Bits of Message Object 24 ~ Message Object 17
0: This message object is not the source of an interrupt
1: This message object is the source of an interrupt

If the message object whose interrupt pending bit is set high is the source of an interrupt, the Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with a higher priority.

- **INTPND2H Register**

Bit	7	6	5	4	3	2	1	0
Name	INTPND32	INTPND31	INTPND30	INTPND29	INTPND28	INTPND27	INTPND26	INTPND25
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **INTPND32~INTPND25**: Interrupt Pending Bits of Message Object 32 ~ Message Object 25
0: This message object is not the source of an interrupt
1: This message object is the source of an interrupt

If the message object whose interrupt pending bit is set high is the source of an interrupt, the Interrupt Identifier in the Interrupt Register will point to this message object if there is no other interrupt source with higher priority.

Message Valid Registers

These registers hold the MSGVA bits of the 32 Message Objects. By reading out the MSGVA bits, the CPU can check which Message Object is valid. The MSGVA bit of a specific Message Object can be set/reset by the CPU via the IFn Message Interface Registers.

- **MSGVAL1L Register**

Bit	7	6	5	4	3	2	1	0
Name	MSGVA8	MSGVA7	MSGVA6	MSGVA5	MSGVA4	MSGVA3	MSGVA2	MSGVA1
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **MSGVA8~MSGVA1**: Message Valid Bits of Message Object 8 ~ Message Object 1
0: This Message Object is ignored by the Message Handler
1: This Message Object is configured and should be considered by the Message Handler

- **MSGVAL1H Register**

Bit	7	6	5	4	3	2	1	0
Name	MSGVA16	MSGVA15	MSGVA14	MSGVA13	MSGVA12	MSGVA11	MSGVA10	MSGVA9
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **MSGVA16~MSGVA9**: Message Valid Bits of Message Object 16 ~ Message Object 9
0: This Message Object is ignored by the Message Handler
1: This Message Object is configured and should be considered by the Message Handler

- **MSGVAL2L Register**

Bit	7	6	5	4	3	2	1	0
Name	MSGVA24	MSGVA23	MSGVA22	MSGVA21	MSGVA20	MSGVA19	MSGVA18	MSGVA17
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **MSGVA24~MSGVA17**: Message Valid Bits of Message Object 24 ~ Message Object 17.
0: This Message Object is ignored by the Message Handler
1: This Message Object is configured and should be considered by the Message Handler

- **MSGVAL2H Register**

Bit	7	6	5	4	3	2	1	0
Name	MSGVA32	MSGVA31	MSGVA30	MSGVA29	MSGVA28	MSGVA27	MSGVA26	MSGVA25
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7~0 **MSGVA32~MSGVA25**: Message Valid Bits of Message Object 32 ~ Message Object 25
0: This Message Object is ignored by the Message Handler
1: This Message Object is configured and should be considered by the Message Handler

Note: The CPU must reset the MSGVA bit of all unused Messages Objects during initialisation before it resets bit INIT in the CAN Control Register. This bit must also be reset before the identifier IDn[28:00], the control bits XTDn, DIRn, or the Data Length Code DLCn[3:0] are modified, or if the Messages Object is no longer required.

Core Release Registers

The design step for the CAN Bus implementation can be identified by reading the Core Release Registers Low/High.

Release	Step	SubStep	Year	Month	Day	Name
2	1	0	5	02	27	Revision 2.1.0, Date 2015/02/27

Example for Coding of Revisions

- **CRLL Register**

Bit	7	6	5	4	3	2	1	0
Name	DAY7	DAY6	DAY5	DAY4	DAY3	DAY2	DAY1	DAY0
R/W	R	R	R	R	R	R	R	R
POR	0	0	1	0	0	1	1	1

Bit 7~0 **DAY7~DAY0**: Time Stamp Day
Two digits, BCD-coded.

- **CRLH Register**

Bit	7	6	5	4	3	2	1	0
Name	MON7	MON6	MON5	MON4	MON3	MON2	MON1	MON0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	1	0

Bit 7~0 **MON7~MON0**: Time Stamp Month
Two digits, BCD-coded.

- **CRHL Register**

Bit	7	6	5	4	3	2	1	0
Name	SUBSTEP3	SUBSTEP2	SUBSTEP1	SUBSTEP0	YEAR3	YEAR2	YEAR1	YEAR0
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	1	0	1

Bit 7~4 **SUBSTEP3~SUBSTEP0**: Core Release Sub-step
One digit, BCD-coded.

Bit 3~0 **YEAR3~YEAR0**: Time Stamp Year (2010 + digit)
One digit, BCD-coded.

- **CRHH Register**

Bit	7	6	5	4	3	2	1	0
Name	REL3	REL2	REL1	REL0	STEP3	STEP2	STEP1	STEP0
R/W	R	R	R	R	R	R	R	R
POR	0	0	1	0	0	0	0	1

Bit 7~4 **REL3~REL0**: Core Release
One digit, BCD-coded.

Bit 3~0 **STEP3~STEP0**: Step of Core Release
One digit, BCD-coded.

Message RAM and FIFO Buffer Configuration

For communication on a CAN network, individual Message Objects are configured. The Message Objects and Identifier Masks for acceptance filtering of received messages are stored in the Message RAM. The CAN module includes a Message Memory capacity of 139-bit × 32 for storing 32 Message Objects and Identifier Masks. A Message Objects and Identifier Masks is 139 bits which is shown in the following table.

Structure of a Message Object in the Message RAM							
MSKn28~00	MXTDn	MDIRn	UMASKn	TXnIEN	RXnIEN	RMTnEN	EOBn
IDn28~00	XTDn	DIRn	MSGnLST	—	—	—	DLCn[3:0]
DATA0	DATA1	DATA2	DATA3	DATA4	DATA5	DATA6	DATA7

MSKn28~00 Identifier Mask
0: The corresponding bit in the identifier of the message object cannot inhibit the match in the acceptance filtering.
1: The corresponding identifier bit is used for acceptance filtering.

IDn28~00 Message Identifier
IDn28 – IDn00: 29-bit Identifier ("Extended Frame")
IDn28 – IDn18: 11-bit Identifier ("Standard Frame")

MXTDn Mask Extended Identifier
0: The extended identifier bit (IDE) has no effect on the acceptance filtering
1: The extended identifier bit (IDE) is used for acceptance filtering

Note: When 11-bit ("standard") Identifiers are used for a Message Object, the identifiers of received Data Frames are written into bits IDn28 to IDn18. For acceptance filtering, only these bits together with MASK bits MSKn28 to MSKn18 are considered.

XTDn Extended Identifier
0: The 11-bit ("standard") Identifier will be used for this Message Object
1: The 29-bit ("extended") Identifier will be used for this Message Object

MDIRn Mask Message Direction
0: The message direction bit (DIR) has no effect on the acceptance filtering
1: The message direction bit (DIR) is used for acceptance filtering

Note: The Arbitration Registers IDn28-00, XTDn, and DIRn are used to define the identifier and type of outgoing messages and are used, together with the mask registers MSKn28-00, MXTDn, and MDIRn, for acceptance filtering of incoming messages. A received message is stored into the valid Message Object with matching identifier and Direction=receive (Data Frame) or Direction=transmit (Remote Frame). Extended frames can be stored only in Message Objects with XTDn=one, standard frames in Message Objects with XTDn=zero. If a received message (Data Frame or Remote Frame) matches with more than one valid Message Object, it is stored into that with the lowest message number. For details see chapter Acceptance Filtering of Received Messages.

DIRn Message Direction
0: Direction=receive: On TREQ, a Remote Frame with the identifier of this Message Object is transmitted. On reception of a Data Frame with matching identifier, that message is stored in this Message Object.
1: Direction=transmit: On TREQ, the respective Message Object is transmitted as a Data Frame. On reception of a Remote Frame with matching identifier, the TREQ bit of this Message Object is set (if RMTnEN=one).

UMASKn Use Acceptance Mask
0: MASK ignored
1: Use MASK (MSKn28~00, MXTDn and MDIRn) for acceptance filtering

If the UMASKn bit is set to one, the Message Object's mask bits have to be programmed during initialisation of the Message Object before MSGnVA is set to one.

MSGnLST Message Lost (only valid for Message Objects with direction=receive)
0: No message lost since last time this bit was reset by the CPU
1: The Message Handler stored a new message into this object when NDTA was still set – the CPU has lost a message

TXnIEN Transmit Interrupt Enable
0: INTPND will be left unchanged after the successful transmission of a frame
1: INTPND will be set after a successful transmission of a frame

RXnIEN Receive Interrupt Enable
0: INTPND will be left unchanged after a successful reception of a frame
1: INTPND will be set after a successful reception of a frame

RMTnEN Remote Enable
0: At the reception of a Remote Frame, TREQ is left unchanged
1: At the reception of a Remote Frame, TREQ is set

EOBn End of Buffer
0: Message Object belongs to a FIFO Buffer and is not the last Message Object of that FIFO Buffer
1: Single Message Object or last Message Object of a FIFO Buffer

This bit is used to concatenate two ore more Message Objects (up to 32) to build a FIFO Buffer. For single Message Objects, not belonging to a FIFO Buffer, this bit must always be set to one.

DLCn3~0 Data Length Code
0~8: CAN: Frame has 0-8 data bytes
9~15: CAN: Frame has 8 data bytes
Note: The Data Length Code of a Message Object must be defined the same as in all the corresponding objects with the same identifier at other nodes. When the Message Handler stores a data frame, it will write the DLC to the value given by the received message.

DATA0: 1st data byte of a CAN Data Frame

DATA1: 2nd data byte of a CAN Data Frame

DATA2: 3rd data byte of a CAN Data Frame

DATA3: 4th data byte of a CAN Data Frame

DATA4: 5th data byte of a CAN Data Frame

DATA5: 6th data byte of a CAN Data Frame

DATA6: 7th data byte of a CAN Data Frame

DATA7: 8th data byte of a CAN Data Frame

Note: Byte DATA0 is the first data byte shifted into the shift register of the CAN Core during a reception and byte DATA7 is the last. When the Message Handler stores a Data Frame, it will write all the eight data bytes into a Message Object. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by non specified values.

The 32 Message Objects can be configurated to several sets of FIFO buffer. A FIFO buffer can have a single Message Object or several concatenated Message Objects. The FIFO threshold of the Message Object number is determined by the RMFD[4:0] bits in the CAN Module Configuration Register, CANCFG. When the Message Object of the selected number is received successfully, an interrupt active signal RMXFINT will be generated.

CAN Module Operating Modes

The Operating modes can be controlled by registers. For detailed information regarding the operating modes refer to the following contents and the related registers.

Software Initialisation

The software initialisation is started by setting the bit INIT in the CAN Control Register CTRLRL, either by software or with a hardware reset, or by going Bus_Off.

While INIT is set, all message transfered from and to the CAN bus is stopped and the status of the CAN bus output pin CANTX is recessive, that is HIGH. The EML (Error Management Logic) counters remain unchanged. Setting INIT does not change any configuration register.

To initialize the CAN Controller, the CPU has to set up the Bit Timing Register and each Message Object. If a Message Object is not required, it is sufficient to set its MSGnVA bit to not valid. Otherwise, the whole Message Object has to be initialized.

Access to the Bit Timing Register and to the BRP Extension Register for bit timing configuration is enabled when both of the INIT and CCE bits in the CAN Control Register are set.

Resetting INIT, by the CPU only, will end the software initialisation. Afterwards the Bit Stream Processor(BSP) synchronizes itself to the data transfer on the CAN bus by waiting for the occurrence of a sequence of 11 consecutive recessive bits (=Bus Idle) before it can take part in bus activities and start the message transfer.

The initialisation of the Message Objects is independent of INIT and can be done on the fly, but the Message Objects should all be configured to particular identifiers or set to not valid before the BSP starts the message transfer. To change the configuration of a Message Object during normal operation, the CPU has to start by setting MSGnVA to not valid. When the configuration is completed, MSGnVA is set to valid again.

CAN Message Transfer

Once the CAN module is initialized and INIT is reset to zero, the CAN Core synchronizes itself to the CAN bus and starts the message transfer.

Received messages are stored into their appropriate Message Objects if they pass the Message Handler's acceptance filtering. The whole message including all arbitration bits, DLC and eight data bytes is stored into the Message Object. If the Identifier Mask is used, the arbitration bits which are masked to "don't care" may be overwritten in the Message Object.

The CPU may read or write each message any time via the Interface Registers, the Message Handler guarantees data consistency in case of concurrent accesses.

Messages to be transmitted are updated by the CPU. If a permanent Message Object (arbitration and control bits setup during configuration) exists for the message, only the data bytes are updated and then the TQnDTA bit is set to start the transmission. If several transmit messages are assigned to the same Message Object (when the number of Message Objects is not sufficient), the whole Message Object has to be configured before the transmission of this message is requested.

The transmission of any number of Message Objects may be requested at the same time, in which case they will be transmitted subsequently according to their internal priority. Messages may be updated or set to not valid any time, even when their requested transmission is still pending. The old data will be discarded when a message is updated before its pending transmission has started.

Depending on the configuration of the Message Object, the transmission of a message may be requested autonomously by the reception of a remote frame with a matching identifier.

Note: Remote frames are always transmitted in Classical CAN format.

Disabled Automatic Retransmission

According to the CAN Specification (see ISO11898-1, 6.3.3 Recovery Management), the CAN module provides a means for automatic retransmission of frames that have lost arbitration or that have been disturbed by errors during transmission. The frame transmission service will not be confirmed to the user before the transmission is successfully completed. By default, this means that automatic retransmission is enabled.

The Disabled Automatic Retransmission mode is enabled by programming bit DAR in the CAN Control Register to '1'. In this operation mode the programmer has to consider the different behaviour of bits TREQ and NDTA in the Control Registers of the Message Buffers:

- When a transmission starts, bit TREQ of the respective Message Buffer is reset, while bit NnDTA remains set.
- When the transmission completes successfully bit NDTA is reset.
- When a transmission has failed (lost arbitration or error) bit NDTA remains set.

To restart the transmission the CPU has to set TREQ back to '1'.

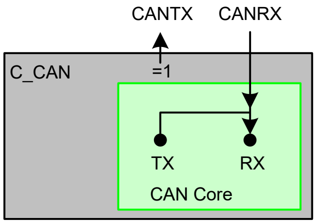
Test Mode

The Test Mode is entered by setting bit TEST in the CAN Control Register. In the Test Mode the bits TX1, TX0, LBACK, SILENT and BASIC in the Test Register are writable. Bit RX monitors the state of pin CANRX and therefore is only readable. All Test Register functions are disabled when bit TEST is reset to zero. The Test Mode functions as described in the following subsections are intended for device tests outside normal operation. These functions should be used carefully. Switching between Test Mode functions and normal operation while communication is running (INIT='0') should be avoided.

Silent Mode

In ISO 11898-1, the Silent Mode is called the Bus Monitoring Mode. The CAN Core can be set to be in the Silent Mode by programming the Test Register bit SILENT to '1'.

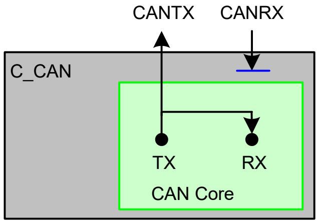
In the Silent Mode, the CAN module is able to receive valid data frames and valid remote frames, but it sends only recessive bits on the CAN bus and it cannot start a transmission. If the CAN Core is required to send a dominant bit (ACK bit, overload flag, active error flag), the bit is rerouted internally so that the CAN Core monitors this dominant bit, although the CAN bus may remain in recessive state. The Silent Mode can be used to analyse the traffic on a CAN bus without affecting it by the transmission of dominant bits (Acknowledge Bits, Error Frames).



CAN Module Core in Silent Mode

Loop Back Mode

The CAN Core can be set in Loop Back Mode by programming the Test Register bit LBACK to ‘1’. In the Loop Back Mode, the CAN Core treats its own transmitted messages as received messages and stores them (if they pass acceptance filtering) into a Receive Buffer.

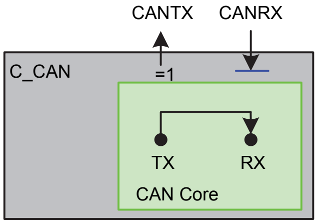


CAN Core in Loop Back Mode

This mode is provided for self-test functions. To be independent from external stimulation, the CAN Core ignores acknowledge errors (recessive bit sampled in the acknowledge slot of a data/remote frame) in the Loop Back Mode. In this mode the CAN Core performs an internal feedback from its TX output to its RX input. The actual value of the CANRX input pin is disregarded by the CAN Core. The transmitted messages can be monitored on the CANTX pin.

Loop Back combined with Silent Mode

It is also possible to combine Loop Back Mode and Silent Mode by programming both bits LBACK and SILENT to ‘1’ at the same time. This mode can be used for a “Hot Selftest”, meaning the CAN module can be tested without affecting a running CAN system connected to the pins CANTX and CANRX. In this mode the CANRX pin is disconnected from the CAN Core and the CANTX pin is held recessive.



CAN Core in Loop Back combined with Silent Mode

Basic Mode

The CAN Core can be set to be in the Basic Mode by programming the Test Register bit BASIC to ‘1’. In this mode the CAN module runs without the Message RAM.

The IF1 Registers are used as a Transmit Buffer. The transmission of the contents of the IF1 Registers is requested by writing the BUSYn bit of the IF1 Command Request Register to ‘1’. The IF1 Registers are locked while the BUSYn bit is set. The BUSYn bit indicates that the transmission is pending. As soon the CAN bus is idle, the IF1 Registers are loaded into the shift register of the CAN Core and the transmission is started. When the transmission has completed, the BUSYn bit is reset and the locked IF1 Registers are released. A pending transmission can be aborted at any time by resetting the BUSYn bit in the IF1 Command Request Register while the IF1 Registers are locked. If the CPU has reset the BUSYn bit, a possible retransmission in case of lost arbitration or in case of an error is disabled.

The IF2 Registers are used as a Receive Buffer. After the reception of a message the contents of the shift register is stored into the IF2 Registers, without any acceptance filtering. Additionally, the actual contents of the shift register can be monitored during the message transfer. Each time a read Message Object is initiated by writing the BUSYn bit of the IF2 Command Request Register to ‘1’, the contents of the shift register is stored into the IF2 Registers.

In the Basic Mode the evaluation of all Message Object related control and status bits and of the control bits of the IFn Command Mask Registers is turned off. The message number of the Command request registers is not evaluated. The NnDTA and MSGnLST bits of the IF2 Message Control Register retain their function, DLCn3~DLCn0 will show the received DLC (Data Length Code). The other control bits will be read as ‘0’.

In the Basic Mode the ready output can_wait_b is not active.

Software control of Pin CANTX

Four output functions are available for the CAN transmit pin CANTX. Additionally to its default function – the serial data output – it can drive the CAN Sample Point signal to monitor the CAN_Core’s bit timing and it can drive constant dominant or recessive values. The last two functions, combined with the readable CAN receive pin CANRX, can be used to check the CAN bus physical layer.

The output mode of pin CANTX is selected by programming the Test Register bits TX1 and TX0. The three test functions for pin CANTX interfere with all CAN protocol functions. CANTX must be left in its default function when a CAN message transfer or any of the test modes Loop Back Mode, Silent Mode, or Basic Mode are selected.

CAN Application

Management of Message Objects

The configuration of the Message Objects in the Message RAM will, with the exception of the bits MSGVA, NDTA, INTPND and TREQ, not be affected by resetting the CAN module. All the Message Objects must be initialized by the CPU or they must be set as not valid (MSGVA=‘0’). The bit timing must be configured before the CPU clears the INIT bit in the CAN Control Register.

The configuration of a Message Object is implemented by programming Mask, Arbitration, Control and Data field of one of the two interface register sets to the desired values. By writing to the corresponding IFn Command Request Register, the IFn Message Buffer Registers are loaded into the addressed Message Object in the Message RAM.

When the INIT bit in the CAN Control Register is cleared, the CAN Protocol Controller state machine of the CAN_Core and the Message Handler State Machine control the CAN module’s internal data flow. Received messages that pass the acceptance filtering are stored into the Message RAM, messages with pending transmission request are loaded into the CAN_Core’s Shift Register and are transmitted via the CAN bus.

The CPU reads received messages and updates messages to be transmitted via the IFn Interface Registers. Depending on the configuration, the CPU is interrupted on certain CAN message and CAN error events.

Message Handler State Machine

The Message Handler controls the data transfer between the RX/TX Shift Register of the CAN Core, the Message RAM and the IFn Registers.

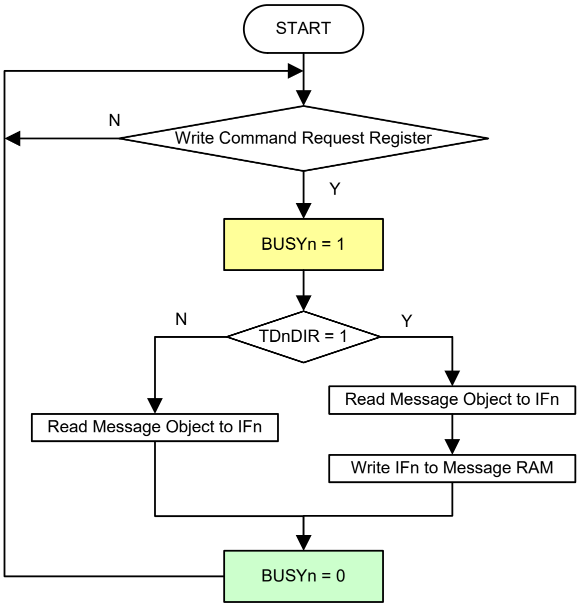
The Message Handler FSM (Finite State Machine) controls the following functions:

- Data Transfer from the IFn Registers to the Message RAM
- Data Transfer from the Message RAM to the IFn Registers
- Data Transfer from the Shift Register to the Message RAM
- Data Transfer from the Message RAM to Shift Register
- Data Transfer from the Shift Register to the Acceptance Filtering unit
- Scanning of Message RAM for a matching Message Object
- TREQ flag handling
- Interrupt handling

Data Transfer from/to Message RAM

When the CPU initiates a data transfer between the IFn Registers and the Message RAM, the Message Handler sets the BUSYn bit in the respective IFn Command Request Register to ‘1’. After the transfer has completed, the BUSYn bit is reset back to ‘0’.

The respective IFn Command Mask Register specifies whether a complete Message Object or only parts of it will be transferred. Due to the structure of the Message RAM it is not possible to write single bits/bytes of one Message Object, it is always necessary to write a complete Message Object to the Message RAM. Therefore the data transfer from the IFn Message Buffer Registers to the Message RAM (TDnDIR=‘1’) requires a read-modify-write cycle. First those parts of the Message Object that are not to be changed are read from the Message RAM to the selected IFn Message Buffer Registers and then the complete contents of the selected IFn Message Buffer Registers are written to the Message Object.


START
N
Write Command Request Register
Y
BUSYn = 1
N
TDnDIR = 1
Y
Read Message Object to IFn
Read Message Object to IFn
Write IFn to Message RAM
BUSYn = 0

Data Transfer between IFn Registers and Message RAM

After a partial write of a Message Object (TDnDIR="1"), the IFn Message Buffer Registers that are not selected by the respective IFn Command Mask Register will be set to the actual contents of the selected Message Object.

After a partial read of a Message Object (TDnDIR="0"), the IFn Message Buffer Registers that are not selected by the respective IFn Command Mask Register will be left unchanged.

Transmission of Messages

If the shift register of the CAN Core cell is ready for loading and if there is no data transfer between the IFn Registers and Message RAM, the MSGVA bits in the Message Valid Register and the TREQ bits in the Transmission Request Register are evaluated. The valid Message Object with the highest priority pending transmission request is loaded into the shift register by the Message Handler and the transmission is started. The Message Object's NDTA bit will be reset.

After a successful transmission and if no new data was written to the Message Object (NDTA ='0') since the start of the transmission, the TREQ bit will be reset. If TXnIEN is set, INTPND will be set after a successful transmission. If the CAN module has lost the arbitration or if an error occurred during the transmission, the message will be retransmitted as soon as the CAN bus is free again. If meanwhile the transmission of a message with higher priority has been requested, the messages will be transmitted in the order of their priority.

Acceptance Filtering of Received Messages

When the arbitration and control field (Identifier+DLC) of an incoming message is completely shifted into the RX/TX Shift Register of the CAN Core, the Message Handler FSM (Finite State Machine) starts the scanning of the Message RAM for a matching valid Message Object.

To scan the Message RAM for a matching Message Object, the Acceptance Filtering unit is loaded with the arbitration bits from the CAN Core shift register. Then the arbitration and mask fields (including MSGVA, UMASKn, NDTA, and EOBn) of Message Object 1 are loaded into the Acceptance Filtering unit and are compared with the arbitration field from the shift register. This is repeated with each following Message Object until a matching Message Object is found or until the end of the Message RAM is reached.

If a match occurs, the scanning is stopped and the Message Handler FSM (Finite State Machine) proceeds depending on the type of frame (Data Frame or Remote Frame) received.

Reception of Data Frame

The Message Handler FSM (Finite State Machine) stores the message from the CAN Core shift register into the respective Message Object in the Message RAM. Not only the data bytes, but also all arbitration bits and the Data Length Code are stored into the corresponding Message Object. This is implemented to keep the data bytes connected with the identifier even if arbitration mask registers are used.

The NDTA bit is set to indicate that new data (not yet seen by the CPU) has been received. The CPU should reset NDTA when it reads the Message Object. If at the time of the reception, the NDTA bit was already set, MSGLST is set to indicate that the previous data (supposedly not seen by the CPU) is lost. If the RXnIE bit is set, the INTPND bit is set, causing the Interrupt Register to point to this Message Object.

The TREQ bit of this Message Object is reset to prevent the transmission of a Remote Frame, while the requested Data Frame has just been received.

Reception of Remote Frame

When a Remote Frame is received, three different configurations of the matching Message Object have to be considered:

1) DIRn='1' (direction=transmit), RMTnEN='1', UMASKn='1' or '0'

At the reception of a matching Remote Frame, the TREQ bit of this Message Object is set high.

The rest of the Message Object remains unchanged.

2) DIRn='1' (direction=transmit), RMTnEN='0', UMASKn='0'

At the reception of a matching Remote Frame, the TREQ bit of this Message Object remains unchanged; the Remote Frame is ignored.

3) DIRn='1' (direction=transmit), RMTnEN='0', UMASKn='1'

At the reception of a matching Remote Frame, the TREQ bit of this Message Object is reset. The arbitration and control field (Identifier + IDE + RTR + DLC) from the shift register is stored into the Message Object in the Message RAM and the NDTA bit of this Message Object is set high.

Note: Remote frames are always transmitted in Classical CAN format.

Receive / Transmit Priority

The receive/transmit priority for the Message Objects is attached to the message number. Message Object 1 has the highest priority, while Message Object 32 has the lowest priority. If more than one transmission request is pending, they are serviced according to the priority of the corresponding Message Object.

Configuration of a Transmit Object

MSGnVA	ARBn	DATA	MASK	EOBn	DIRn	NnDTA
1	appl.	appl.	appl.	1	1	0
MSGnLST	**RXnIEN**	**TXnIEN**	**INTnPND**	**RMTnEN**	**TnREQ**	
0	0	appl.	0	appl.	0	

Initialisation of a Transmit Object

Note: "appl." means by application.

The Arbitration Registers (IDn28~00 and XTDn bit) are given by the application. They define the identifier and type of the outgoing message. If an 11-bit Identifier ("Standard Frame") is used, it is programmed to IDn28~IDn18, IDn17~IDn00 can then be disregarded.

If the TXnIEN bit is set, the INTPND bit will be set after a successful transmission of the Message Object.

If the RMTnEN bit is set, a matching received Remote Frame will cause the TREQ bit to be set; the Remote Frame will autonomously be answered by a Data Frame.

The Data Registers (DLCn3-0, DATA0-7) are given by the application, TnREQ and RMTnEN may not be set before the data is valid.

The Mask Registers (MSKn28-00, UMASKn, MXTDn and MDIRn bits) may be used (UMASKn='1') to allow groups of Remote Frames with similar identifiers to set the TnREQ bit. The DIRn bit should not be masked.

Updating a Transmit Object

The CPU may update the data bytes of a Transmit Object any time via the IFn Interface registers, neither MSGVA nor TREQ have to be reset before the update.

Even if only a part of the data bytes are to be updated, all four bytes of the corresponding IFn DATAnA Register or IFn DATAnB Register have to be valid before the content of that register is transferred to the Message Object. Either the CPU has to write all four bytes into the IFn Data Register or the Message Object is transferred to the IFn Data Register before the CPU writes the new data bytes.

When only the (eight) data bytes are updated, first 0x87 is written to the IFn Command Mask Register and then the number of the Message Object is written to the IFn Command Request Register, concurrently updating the data bytes and setting TQnDTA.

To prevent the reset of TREQ at the end of a transmission that may already be in progress while the data is updated, NDTA has to be set together with TREQ.

When NDTA is set together with TREQ, NDTA will be reset as soon as the new transmission has started.

Configuration of a Receive Object

MSGnVA	ARBn	DATA	MASK	EOBn	DIRn	NnDTA
1	appl.	appl.	appl.	1	0	0
MSGnLST	**RXnIEN**	**TXnIEN**	**INTnPND**	**RMTnEN**	**TnREQ**	
0	appl.	0	0	0	0	

Initialisation of a Receive Object

Note: “appl.” means by application.

The Arbitration Registers (IDn[28:00] and XTDn bit) are given by the application. They define the identifier and type of accepted received messages. If an 11-bit Identifier (“Standard Frame”) is used, it is programmed to IDn28~IDn18 and IDn17~IDn00 can then be disregarded. When a Data Frame with an 11-bit Identifier is received, IDn17~IDn00 will be set to ‘0’.

If the RXnIEN bit is set, the INTPND bit will be set when a received Data Frame is accepted and stored in the Message Object.

The Data Length Code (DLCn[3:0]) is given by the application. When the Message Handler stores a Data Frame in the Message Object, it will store the received Data Length Code and eight data bytes. If the Data Length Code is less than 8, the remaining bytes of the Message Object will be overwritten by non specified values.

The Mask Registers (MSKn[28:00], UMASKn, MXTDn, and MDIRn bits) may be used (UMASKn=‘1’) to allow groups of Data Frames with similar identifiers to be accepted . The DIRn bit should not be masked in typical applications.

Handling of Received Messages

The CPU may read a received message any time via the IFn Interface registers. The data consistency is guaranteed by the Message Handler state machine.

Typically the CPU will write first 0x7F to the IFn Command Mask Register and then the number of the Message Object to the IFn Command Request Register. That combination will transfer the whole received message from the Message RAM into the IFn Message Buffer Register. Additionally, the bits NDTA and INTPND are cleared in the Message RAM (not in the Message Buffer).

If the Message Object uses masks for acceptance filtering, the arbitration bits show which of the matching messages has been received.

The actual value of NDTA shows whether a new message has been received since last time this Message Object was read. The actual value of MSGLST shows whether more than one message has been received since last time this Message Object was read. MSGLST will not be automatically reset.

By means of a Remote Frame, the CPU may request another CAN node to provide new data for a receive object. Setting the TREQ bit of a receive object will cause the transmission of a Remote Frame with the receive object’s identifier. This Remote Frame triggers the other CAN node to start the transmission of the matching Data Frame. If the matching Data Frame is received before the Remote Frame could be transmitted, the TREQ bit is automatically reset.

Configuration of a FIFO Buffer

With the exception of the EOBn bit, the configuration of Receive Objects belonging to a FIFO Buffer is the same as the configuration of a (single) Receive Object.

To concatenate two or more Message Objects into a FIFO Buffer, the identifiers and masks (if used) of these Message Objects have to be programmed to matching values. Due to the implicit priority of the Message Objects, the Message Object with the lowest number will be the first Message Object of the FIFO Buffer. The EOBn bit of all Message Objects of a FIFO Buffer except the last have to be programmed to zero. The EOBn bits of the last Message Object of a FIFO Buffer is set to one, configuring it as the End of the Block.

Reception of Messages with FIFO Buffers

Received messages with identifiers matching to a FIFO Buffer are stored into a Message Object of this FIFO Buffer starting with the Message Object with the lowest message number.

When a message is stored into a Message Object of a FIFO Buffer the NDTA bit of this Message Object is set. By setting NDTA while EOBn is zero the Message Object is locked for further write accesses by the Message Handler until the CPU has written the NDTA bit back to zero.

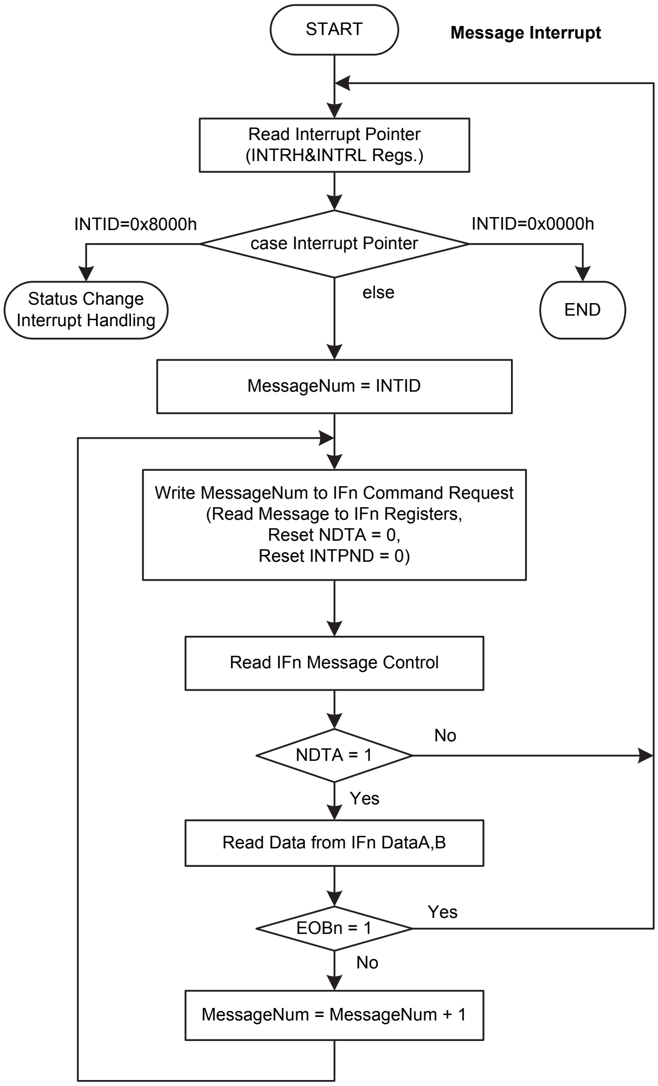
Messages are stored into a FIFO Buffer until the last Message Object of this FIFO Buffer is reached. If none of the preceding Message Objects is released by writing NDTA to zero, all further messages for this FIFO Buffer will be written into the last Message Object of the FIFO Buffer and therefore overwrite previous messages.

Reading from a FIFO Buffer

When the CPU transfers the contents of Message Object to the IFn Message Buffer Registers by writing its number to the IFn Command Request Register, the corresponding IFn Command Mask Register should be programmed the way that bits NDTA and INTPND are reset to zero (TQnDTA=‘1’ and CINTPNDn=‘1’). The values of these bits in the IFn Message Control Registers always reflect the status before resetting the bits.

To assure the correct function of a FIFO Buffer, the CPU should read out the Message Objects starting at the FIFO Object with the lowest message number.

The following figure shows how a set of Message Objects which are concatenated to a FIFO Buffer can be handled by the CPU.


START
Message Interrupt
Read Interrupt Pointer (INTRH&INTRL Regs.)
case Interrupt Pointer
INTID=0x8000h
INTID=0x0000h
Status Change Interrupt Handling
END
else
MessageNum = INTID
Write MessageNum to IFn Command Request (Read Message to IFn Registers, Reset NDTA = 0, Reset INTPND = 0)
Read IFn Message Control
NDTA = 1
No
Yes
Read Data from IFn DataA,B
EOBn = 1
Yes
No
MessageNum = MessageNum + 1

CPU Handling of a FIFO Buffer

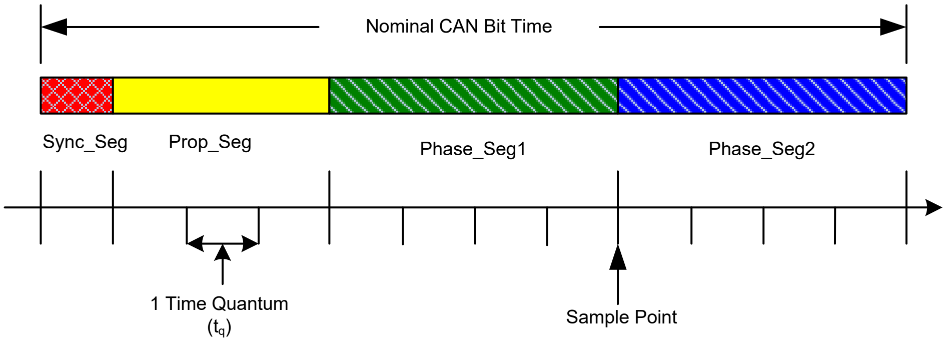
Bit Time and Bit Rate

CAN supports bit rates in the range of 1kBit/s to 1000kBit/s. Each member of the CAN network has its own clock generator, usually a quartz oscillator. The timing parameter of the bit time, i.e. the reciprocal of the bit rate, can be configured individually for each CAN node, creating a common bit rate even though the CAN nodes' oscillator periods (fosc) may be different.

The frequencies of these oscillators are not absolutely stable, as small variations are caused by changes in temperature or voltage and by deteriorating components. As long as the variations remain within a specific oscillator tolerance range (dF), the CAN nodes are able to compensate for the different bit rates by resynchronising to the bit stream.

According to the CAN specification, the bit time is divided into four segments which are the Synchronisation Segment, the Propagation Time Segment, the Phase Buffer Segment 1 and the Phase Buffer Segment 2. Each segment consists of a specific, programmable number of time quanta. The length of the time quantum (t_q), which is the basic time unit of the bit time, is defined by the CAN controller's system clock f_{CAN} and the Baud Rate Prescaler(BRP): t_q=BRP/f_{CAN}. The CAN module system clock f_{CAN} is the frequency of its can_clk input.

The Synchronisation Segment Sync_Seg is that part of the bit time where edges of the CAN bus level are expected to occur. The distance between an edge that occurs outside of Sync_Seg and the Sync_Seg is called the phase error of that edge. The Propagation Time Segment Prop_Seg is intended to compensate for the physical delay times within the CAN network. The Phase Buffer Segments Phase_Seg1 and Phase_Seg2 surround the Sample Point. The (Re-) Synchronisation Jump Width (SJW) defines how far a resynchronisation may move the Sample Point inside the limits defined by the Phase Buffer Segments to compensate for edge phase errors.



Bit Timing

Parameter	Range	Remark
BRP	[1 .. 32]	Defines the length of the time quantum t_q
Sync_Seg	1 t_q	Fixed length, synchronisation of bus input to CAN system clock
Prop_Seg	[1 .. 8] t_q	Compensates for the physical delay times
Phase_Seg1	[1 .. 8] t_q	May be lengthened temporarily by synchronisation
Phase_Seg2	[1 .. 8] t_q	May be shortened temporarily by synchronisation
SJW	[1 .. 4] t_q	May not be longer than either Phase Buffer Segment
This table describes the minimum programmable ranges required by the CAN protocol		

mtq (minimum time quantum) = CAN system clock period = $1/f_{CAN}$

t_q (time quantum) = (BRPE[3:0] × 0x40 + BRP[5:0] + 1) × mtq

SYNC_SEG = 1 t_q

SEG1 = PROP_SEG + PHASE_SEG1

Bit Time = $t_{SYNC\_SEG} + t_{SEG1} + t_{PHASE\_SEG2}$

For example:

f_{CAN} = 8MHz , Bit Rate = 500Kbps , PROP_SEG = 0 , Sample point = 50% , SYNC_SEG = 1 $t_{q,}$ then to calculate the SEG1 & PHASE_SEG2 values.

Sol: mtq = $1/f_{CAN}$ = 1/8MHz = 0.125μs

set Baud Rate Prescaler (BRP) = 1 → BRP[5:0] = (1-1) = 0 #

t_q = (BRPE[3:0] × 0x40 + BRP[5:0] + 1) × mtq = 1 × mtq = 0.125μs

Bit Time = 1/Bit Rate = 1/500Kbps = 0.002ms = 2μs

Nominal Bit Time = Bit Rate / t_q = 2μs / (0.125μs) = 16 t_q

(1) PHASE_SEG2 = Nominal Bit Time – (Nominal Bit Time • Sample point)

= $16t_q - (16t_q \bullet 50\%) = 16t_q - 8t_q = 8t_q$

TSG2D[2:0] = (8 – 1) = 7 #

(2) SEG1 = (Nominal Bit Time – SYNC_SEG – PHASE_SEG2) = $(16t_q - 1t_q - 8t_q) = 7t_q$

TSG1D[3:0] = (7 – 1) = 6 #

CAN Bus Interrupt Structure

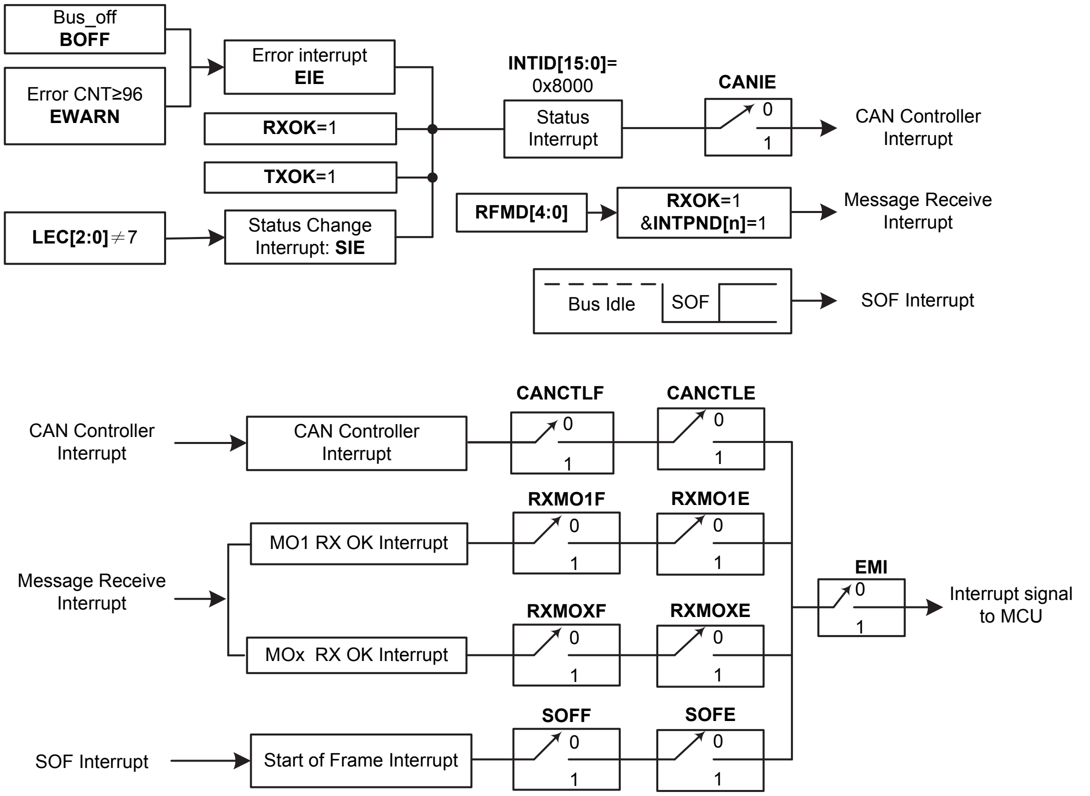
The CAN Bus interrupt contains four interrupt sources, namely CAN controller interrupt, Message Object 1 Successful Message Reception Interrupt, Message Object x Successful Message Reception Interrupt and SOF interrupt.

Several individual conditions can generate a CAN controller interrupt. When these conditions exist, an interrupt will be generated to get the attention of the microcontroller. These conditions are a CAN module busoff state, an error limit warning, a successfull message reception, a successfull message transmission or a CAN bus status change. When any of these conditions is created, if the CAN Module interrupt enable bit CANIE is set, the corresponding interrupt control is enabled and the stack is not full, the program will jump to its corresponding interrupt vector where it can be serviced before returning to the main program. Three of these conditions will generate a CAN Controller interrupt if its associated interrupt enable control bit in the CAN control register is set and bit CANIE is set. While the transmittion interrupt and the receive interrupt conditions will also generate a CAN controller interrupt if the CANIE bit is set high. These enable bits can be used to mask out individual CAN controller interrupt sources.

The CAN module includes 32 Message Objects, which can be scheduled for multiple sets of FIFO, where a single set of FIFO depth can be a single Message Object or multiple Message Objects in series. When the Message Object 1 successfully receives a Message, bit RXOK is set and INTPND1 will be high, an interrupt signal will be sent to inform the MCU. The FIFO threshold can be set using the RMFD[4:0] to select the Message Object number. When the selected Message Object successfully receives a Message, bit RXOK is set, an interrupt signal will not be sent to inform the MCU until the INTPND[n] bit is high.

When a Message is start to be transferred (reception or transmission), a start of frame signal will be detected, an SOF interrupt signal will then be sent to inform MCU. This SOF signal is used as hardware synchronizing signal.

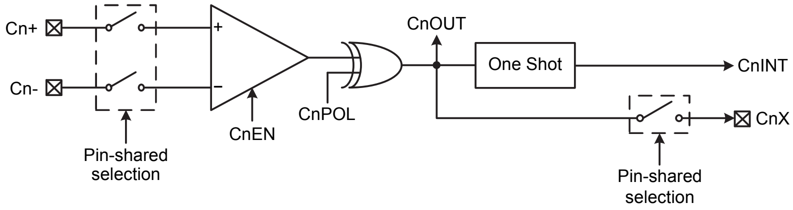
The overall CAN Bus interrupts can be disabled or enabled by the related interrupt enable control bits in the interrupt control registers of the microcontroller to decide whether the interrupt requested by the CAN module is masked out or allowed.



CAN Bus Interrupt Structure

Comparators

Two independent analog conparators are contained in the device. The comparator functions offer flexibility via their register controlled features such as power-down, polarity select, response time, etc. In sharing their pins with normal I/O pins the comparators do not waste precious I/O pins if the comparator functions are otherwise unused.



Comparators (n=0~1)

Comparator Operation

The device contains two comparator functions which are used to compare two analog voltages and provide an output based on their difference. Full control over the two internal comparators is provided via the control register, CP0C and CP1C, one assigned to each comparator. The comparator output is recorded via a bit in the control register, but can also be transferred out onto a shared I/O pin. Additional comparator functions include output polarity, response time and power down control.

Any pull-high resistors connected to the shared comparator input pins will be automatically disconnected when the comparator is enabled. As the comparator inputs approach their switching level, some spurious output signals may be generated on the comparator output due to the slow rising or falling nature of the input signals. This can be minimised by the hysteresis function which will apply a small amount of positive feedback to the comparator. When the comparator operates in the normal mode, the hysteresis function will automatically be enabled. However, the hysteresis function will be disabled when the comparator operates in the input offset calibration mode.

Ideally the comparator should switch at the point where the positive and negative inputs signals are at the same voltage level. However, unavoidable input offsets introduce some uncertainties here. The offset calibration function, if executed, will minimise the switching offset value. The comparator also provides the output response time select function using the CNVTn1~CNVTn0 bits in the CPnC register.

Comparator Registers

There are four registers for overall comparator operation, two registers, CPnC and CPnVOS, for each comparator. As corresponding bits in these registers have identical functions, the following register table applies to the registers.

Register Name	Bit							
	7	6	5	4	3	2	1	0
CPnC	—	CnEN	CnPOL	CnOUT	CNVTn1	CNVTn0	—	—
CPnVOS	—	CnOFM	CnRSP	CnOF4	CnOF3	CnOF2	CnOF1	CnOF0

Comparator Registers List (n = 0~1)

- **CPnC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	CnEN	CnPOL	CnOUT	CNVTn1	CNVTn0	—	—
R/W	—	R/W	R/W	R	R/W	R/W	—	—
POR	—	0	0	0	0	0	—	—

Bit 7 Unimplemented, read as "0"

Bit 6 **CnEN**: Comparator enable control
0: Disable
1: Enable

This bit is used to enable the comparator function. If this bit is cleared to zero, the comparator will be switched off and no power consumed even if analog voltages are applied to its inputs. When the comparator function is disabled, the comparator output will be set to zero.

Bit 5 **CnPOL**: Comparator output polarity selection
0: Non-invert
1: Invert

If this bit is cleared to zero, the CnOUT bit will reflect the non-inverted output condition of the comparator. If this bit is set high, the CnOUT bit will be inverted.

Bit 4 **CnOUT**: Comparator output bit

CnPOL=0
0: Cn+ < Cn–
1: Cn+ > Cn–

CnPOL=1
0: Cn+ > Cn–
1: Cn+ < Cn–

This bit is used to store the comparator output bit. The polarity of this bit is determined by the voltages on the comparator inputs and by the condition of the CnPOL bit.

Bit 3~2 **CNVTn1~CNVTn0**: Comparator response time selection
00: Response time 0 (max.)
01: Response time 1
10: Response time 2
11: Response time 3 (min.)

These bits are used to select the comparator response time. The detailed response time specifications are listed in the Comparator Characteristics.

Bit 1~0 Unimplemented, read as "0"

- **CPnVOS Register**

Bit	7	6	5	4	3	2	1	0
Name	—	CnOFM	CnRSP	CnOF4	CnOF3	CnOF2	CnOF1	CnOF0
R/W	—	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	0	0	1	0	0	0	0

Bit 7 Unimplemented, read as "0"

Bit 6 **CnOFM**: Comparator normal operation or input offset calibration mode selection
0: Normal operation mode
1: Input offset calibration mode

This bit is used to enable the comparator input offset calibration function. Refer to the "Input Offset Calibration" section for the detailed input offset calibration procedures.

Bit 5 **CnRSP**: Comparator input offset calibration reference input selection
0: Cn– is selected as reference input
1: Cn+ is selected as reference input

Bit 4~0 **CnOF4~CnOF0**: Comparator input offset calibration value

This 5-bit field is used to perform the comparator input offset calibration operation and the value after the input offset calibration can be restored into this bit field. Refer to the "Input Offset Calibration" section for more detailed information.

Input Offset Calibration

To operate in the input offset calibration mode, the comparator input pins to be used should first be selected by properly configuring the corresponding pin-shared function selection bits followed by setting the CnOFM bit high. The procedure is described in the following.

Step 1. Set CnOFM = 1 to enable the comparator input offset calibration mode.

Step 2. Set CnOF [4:0] = 00000 and read the CnOUT bit.

Step 3. Increase the CnOF [4:0] value by 1 and then read the CnOUT bit.

If the CnOUT bit state does not changed, then repeat Step 3 until the CnOUT bit state changes.

If the CnOUT bit state changes, record the CnOF field value as VCnOS1 and then go to Step 4.

Step 4. Set CnOF [4:0] = 11111 and read the CnOUT bit.

Step 5. Decrease the CnOF [4:0] value by 1 and then read the CnOUT bit.

If the CnOUT bit state does not changed, then repeat Step 5 until the CnOUT bit state changes.

If the CnOUT bit state changes, record the CnOF field value as VCnOS2 and then go to Step 6.

Step 6. Restore the comparator input offset calibration value VCnOS into the CnOF [4:0] bit field. The offset calibration procedure is now finished.

Where $VCnOS = \frac{VCnOS1 + VCnOS2}{2}$

Comparator Interrupt

Each comparator possesses its own interrupt function. When the comparator output bit changes state, its relevant interrupt flag will be set, and if the corresponding interrupt enable bit is set, then a jump to its relevant interrupt vector will be executed. Note that it is the changing state of the CnOUT bit and not the output pin which generates an interrupt. If the microcontroller is in the SLEEP or IDLE Mode and the Comparator is enabled, then if the external input lines cause the Comparator output to change state, the resulting generated interrupt flag will also generate a wake-up. If it is required to disable a wake-up from occurring, then the interrupt flag should be first set high before entering the SLEEP or IDLE Mode.

Programming Considerations

If the comparator is enabled, it will remain active when the microcontroller enters the SLEEP or IDLE Mode, however as it will consume a certain amount of power, the user may wish to consider disabling it before the SLEEP or IDLE Mode is entered. As comparator pins are shared with normal I/O pins the I/O data bits for these pins will be read as zero regardless of the port control register bit value due to normal I/O path being switched off if the comparator function is enabled.

Software Controlled LCD Driver

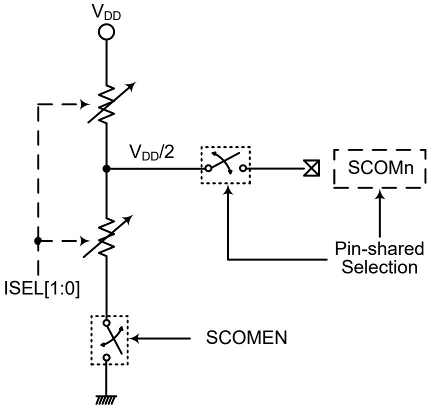
The device has the capability of driving external LCD panels. The common pins, SCOM0~SCOM3, for LCD driving are pin-shared with certain pins on the I/O ports. The LCD signals (COM and SEG) are generated using the application program.

LCD Operation

An external LCD panel can be driven using the device by configuring the I/O pins as common pins and segment pins. The LCD driver function is controlled using the LCD control register which in addition to controlling the overall on/off function also controls the R-type bias current on the

SCOMn pins. This enables the LCD COM driver to generate the necessary voltage levels, V_{SS}, $V_{DD}/2$ and V_{DD}, for LCD 1/2 bias operation.

The SCOMEN bit in the SCOMC register is the overall master control for the LCD driver. The SCOMn pin is selected to be used for LCD driving by the corresponding pin-shared function selection bits. Note that the corresponding Port Control register does not need to first setup the pins as outputs to enable the LCD driver operation.



Software Controlled LCD Driver Structure

LCD Bias Current Control

The LCD COM driver enables a range of selections to be provided to suit the requirement of the LCD panel which is being used. The bias current choice is implemented using the ISEL1 and ISEL0 bits in the SCOMC register. All COM pins are pin-shared with I/O pins and selected as SCOM pins using the corresponding pin-shared function selection bits.

- **SCOMC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	ISEL1	ISEL0	SCOMEN	—	—	—	—
R/W	—	R/W	R/W	R/W	—	—	—	—
POR	—	0	0	0	—	—	—	—

Bit 7 Unimplemented, read as “0”

Bit 6~5 **ISEL1~ISEL0**: SCOM typical bias current selection (@V_{DD}=5V)
00: 25μA
01: 50μA
10: 100μA
11: 200μA

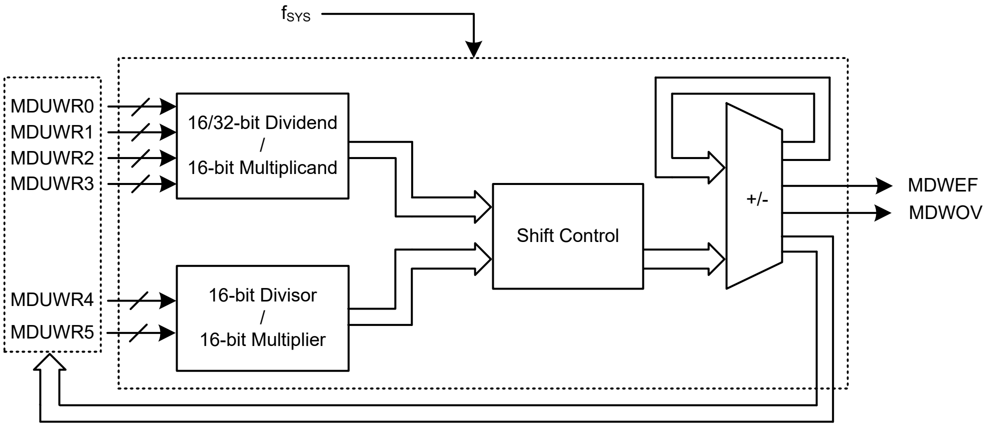
Bit 4 **SCOMEN**: Software controlled LCD Driver enable control
0: Disable
1: Enable

The SCOMn lines can be enabled using the corresponding pin-shared selection bits if the SCOMEN bit is set to 1. When the SCOMEN bit is cleared to 0, then the SCOMn outputs will be fixed at a V_{DD} level. Note that the corresponding pin-shared selection bits should first be properly configured before the SCOMn function is enabled.

Bit 3~0 Unimplemented, read as “0”

16-bit Multiplication Division Unit – MDU

The device has a 16-bit Multiplication Division Unit, MDU, which integrates a 16-bit unsigned multiplier and a 32-bit/16-bit divider. The MDU, in replacing the software multiplication and division operations, can therefore save large amounts of computing time as well as the Program and Data Memory space. It also reduces the overall microcontroller loading and results in the overall system performance improvements.


f_{SYS}
MDUWR0
MDUWR1
MDUWR2
MDUWR3
MDUWR4
MDUWR5
16/32-bit Dividend / 16-bit Multiplicand
16-bit Divisor / 16-bit Multiplier
Shift Control
+/-
MDWEF
MDWOV

16-Bit MDU Block Diagram

MDU Registers

The multiplication and division operations are implemented in a specific way, a specific write access sequence of a series of MDU data registers. The status register, MDUWCTRL, provides the indications for the MDU operation. The data register each is used to store the data regarded as the different operand corresponding to different MDU operations.

Register Name	Bit							
	7	6	5	4	3	2	1	0
MDUWR0	D7	D6	D5	D4	D3	D2	D1	D0
MDUWR1	D7	D6	D5	D4	D3	D2	D1	D0
MDUWR2	D7	D6	D5	D4	D3	D2	D1	D0
MDUWR3	D7	D6	D5	D4	D3	D2	D1	D0
MDUWR4	D7	D6	D5	D4	D3	D2	D1	D0
MDUWR5	D7	D6	D5	D4	D3	D2	D1	D0
MDUWCTRL	MDWEF	MDWOV	—	—	—	—	—	—

MDU Registers List

- **MDUWRn Register (n=0~5)**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	x	x	x	x	x	x	x	x

"x": unknown

Bit 7~0 **D7~D0**: 16-bit MDU data register n

- **MDUWCTRL Register**

Bit	7	6	5	4	3	2	1	0
Name	MDWEF	MDWOV	—	—	—	—	—	—
R/W	R	R	—	—	—	—	—	—
POR	0	0	—	—	—	—	—	—

Bit 7 **MDWEF**: 16-bit MDU error flag
0: Normal
1: Abnormal
This bit will be set to 1 if the data register MDUWRn is written or read as the MDU operation is executing. This bit should be cleared to 0 by reading the MDUWCTRL register if it is equal to 1 and the MDU operation is completed.

Bit 6 **MDWOV**: 16-bit MDU overflow flag
0: No overflow occurs
1: Multiplication product > FFFFH or Divisor = 0
When an operation is completed, this bit will be updated by hardware to a new value corresponding to the current operation situation.

Bit 5~0 Unimplemented, read as “0”

MDU Operation

For this MDU the multiplication or division operation is carried out in a specific way and is determined by the write access sequence of the six MDU data registers, MDUWR0~MDUWR5. The low byte data, regardless of the dividend, multiplicand, divisor or multiplier, must first be written into the corresponding MDU data register followed by the high byte data. All MDU operations will be executed after the MDUWR5 register is write-accessed together with the correct specific write access sequence of the MDUWRn. Note that it is not necessary to consecutively write data into the MDU data registers but must be in a correct write access sequence. Therefore, a non-write MDUWRn instruction or an interrupt, etc., can be inserted into the correct write access sequence without destroying the write operation. The relationship between the write access sequence and the MDU operation is shown in the following.

- 32-bit/16-bit division operation: Write data sequentially into the six MDU data registers from MDUWR0 to MDUWR5.
- 16-bit/16-bit division operation: Write data sequentially into the specific four MDU data registers in a sequence of MDUWR0, MDUWR1, MDUWR4 and MDUWR5 with no write access to MDUWR2 and MDUWR3.
- 16-bit × 16-bit multiplication operation: Write data sequentially into the specific four MDU data register in a sequence of MDUWR0, MDUWR4, MDUWR1 and MDUWR5 with no write access to MDUWR2 and MDUWR3.

After the specific write access sequence is determined, the MDU will start to perform the corresponding operation. The calculation time necessary for these MDU operations are different. During the calculation time any read/write access to the six MDU data registers is forbidden. After the completion of each operation, it is necessary to check the operation status in the MDUWCTRL register to make sure that whether the operation is correct or not. Then the operation result can be read out from the corresponding MDU data registers in a specific read access sequence if the operation is correctly finished. The necessary calculation time for different MDU operations is listed in the following.

- 32-bit/16-bit division operation: 17 × t_{SYS}
- 16-bit/16-bit division operation: 9 × t_{SYS}
- 16-bit × 16-bit multiplication operation: 11 × t_{SYS}

The operation results will be stored in the corresponding MDU data registers and should be read out from the MDU data registers in a specific read access sequence after the operation is completed. Noe that it is not necessary to consecutively read data out from the MDU data registers but must be in a correct read access sequence. Therefore, a non-read MDUWRn instruction or an interrupt, etc., can be inserted into the correct read access sequence without destroying the read operation. The relationship between the operation result read access sequence and the MDU operation is shown in the following.

- 32-bit/16-bit division operation: Read the quotient from MDUWR0 to MDUWR3 and remainder from MDUWR4 and MDUWR5 sequentially.
- 16-bit/6-bit division operation: Read the quotient from MDUWR0 and MDUWR1 and remainder from MDUWR4 and MDUWR5 sequentially.
- 16-bit × 16-bit multiplication operation: Read the product sequentially from MDUWR0 to MDUWR3.

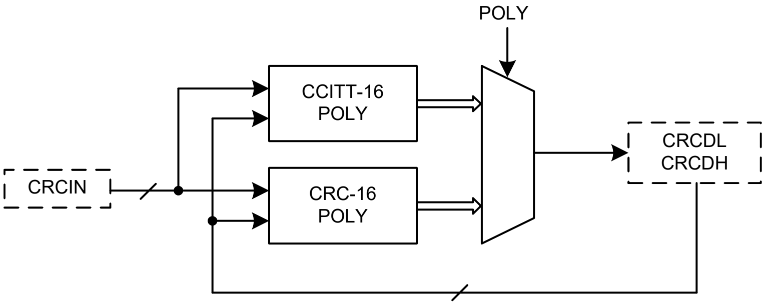
The overall important points for the MDU read/write access sequence and calculation time are summarized in the following table. Note that the device should not enter the IDLE or SLEEP mode until the MDU operation is totally completed, otherwise the MDU operation will fail.

Operations Items	32-bit / 16-bit Division	16-bit / 16-bit Division	16-bit × 16-bit Multiplication
Write Sequence First write ↓ ↓ ↓ ↓ Last write	Dividend Byte 0 written to MDUWR0 Dividend Byte 1 written to MDUWR1 Dividend Byte 2 written to MDUWR2 Dividend Byte 3 written to MDUWR3 Divisor Byte 0 written to MDUWR4 Divisor Byte 1 written to MDUWR5	Dividend Byte 0 written to MDUWR0 Dividend Byte 1 written to MDUWR1 Divisor Byte 0 written to MDUWR4 Divisor Byte 1 written to MDUWR5	Multiplicand Byte 0 written to MDUWR0 Multiplier Byte 0 written to MDUWR4 Multiplicand Byte 1 written to MDUWR1 Multiplier Byte 1 written to MDUWR5
Calculation Time	17 × t_{SYS}	9 × t_{SYS}	11 × t_{SYS}
Read Sequence First read ↓ ↓ ↓ ↓ Last read	Quotient Byte 0 read from MDUWR0 Quotient Byte 1 read from MDUWR1 Quotient Byte 2 read from MDUWR2 Quotient Byte 3 read from MDUWR3 Remainder Byte 0 read from MDUWR4 Remainder Byte 1 read from MDUWR5	Quotient Byte 0 read from MDUWR0 Quotient Byte 1 read from MDUWR1 Remainder Byte 0 read from MDUWR4 Remainder Byte 1 read from MDUWR5	Product Byte 0 read from MDUWR0 Product Byte 1 read from MDUWR1 Product Byte 2 read from MDUWR2 Product Byte 3 read from MDUWR3

MDU Operations Summary

Cyclic Redundancy Check – CRC

The Cyclic Redundancy Check, CRC, calculation unit is an error detection technique test algorithm and uses to verify data transmission or storage data correctness. A CRC calculation takes a data stream or a block of data as input and generates a 16-bit output remainder. Ordinarily, a data stream is suffixed by a CRC code which is used as a checksum when the data stream is sent or stored. Therefore, the received or restored data stream is calculated by the same generator polynomial as described in the following section.



CRC Block Diagram

CRC Registers

The CRC generator contains an 8-bit CRC data input register, CRCIN, and a CRC checksum register pair, CRCDH and CRCDL. The CRCIN register is used to input new data and the CRCDH and CRCDL registers are used to hold the previous CRC calculation result. A CRC control register, CRCCR, is used to select which CRC generating polynomial is used.

Register Name	Bit							
	7	6	5	4	3	2	1	0
CRCIN	D7	D6	D5	D4	D3	D2	D1	D0
CRCDL	D7	D6	D5	D4	D3	D2	D1	D0
CRCDH	D7	D6	D5	D4	D3	D2	D1	D0
CRCCR	—	—	—	—	—	—	—	POLY

CRC Registers List

- **CRCIN Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: CRC input data register

- **CRCDL Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: 16-bit CRC checksum low byte data register

- **CRCDH Register**

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **D7~D0**: 16-bit CRC checksum high byte data register

- **CRCCR Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	—	POLY
R/W	—	—	—	—	—	—	—	R/W
POR	—	—	—	—	—	—	—	0

Bit 7~1 Unimplemented, read as “0”

Bit 0 **POLY**: 16-bit CRC generating polynomial selection
0: CRC-CCITT: $X^{16} + X^{12} + X^5 + 1$
1: CRC-16: $X^{16} + X^{15} + X^2 + 1$

CRC Operation

The CRC generator provides the 16-bit CRC result calculation based on the CRC16 and CCITT CRC16 polynomials. In this CRC generator, there are only these two polynomials available for the numeric values calculation. It can not support the 16-bit CRC calculations based on any other polynomials.

The following two expressions can be used for the CRC generating polynomial which is determined using the POLY bit in the CRC control register, CRCCR. The CRC calculation result is called as the CRC checksum, CRCSUM, and stored in the CRC checksum register pair, CRCDH and CRCDL.

- CRC-CCITT: $X^{16} + X^{12} + X^5 + 1$
- CRC-16: $X^{16} + X^{15} + X^2 + 1$

CRC Computation

Each write operation to the CRCIN register creates a combination of the previous CRC value stored in the CRCDH and CRCDL registers and the new data input. The CRC unit calculates the CRC data register value byte by byte. It will take one MCU instruction cycle to calculate the CRC checksum.

CRC Calculation Procedures

1. Clear the checksum register pair, CRCDH and CRCDL.
2. Execute an “Exclusive OR” operation with the 8-bit input data byte and the 16-bit CRCSUM high byte. The result is called the temporary CRCSUM.
3. Shift the temporary CRCSUM value left by one bit and move a “0” into the LSB.
4. Check the shifted temporary CRCSUM value after procedure 3.

   If the MSB is 0, then this shifted temporary CRCSUM will be considered as a new temporary CRCSUM.

   Otherwise, execute an “Exclusive OR” operation with the shifted temporary CRCSUM in procedure 3 and a data “8005H”. Then the operation result will be regarded as the new temporary CRCSUM.

   Note that the data to be perform an “Exclusive OR” operation is “8005H” for the CRC-16 polynomial while for the CRC-CCITT polynomial the data is “1021H”.

5. Repeat the procedure 3 ~ procedure 4 until all bits of the input data byte are completely calculated.

6. Repeat the procedure 2 ~ procedure 5 until all of the input data bytes are completely calculated. Then, the latest calculated result is the final CRC checksum, CRCSUM.

CRC Calculation Examples

1. Write 1 byte input data into the CRCIN register and the corresponding CRC checksum are individually calculated as the following table shown.

CRC Data Input CRC Polynomial	00H	01H	02H	03H	04H	05H	06H	07H
CRC-CCITT ($X^{16}+X^{12}+X^5+1$)	0000H	1021H	2042H	3063H	4084H	50A5H	60C6H	70E7H
CRC-16 ($X^{16}+X^{15}+X^2+1$)	0000H	8005H	800FH	000AH	801BH	001EH	0014H	8011H

Note: The initial value of the CRC checksum register pair, CRCDH and CRCDL, is zero before each CRC input data is written into the CRCIN register.

2. Write 4 bytes input data into the CRCIN register sequentially and the CRC checksum are sequentially listed in the following table.

CRC Data Input CRC Polynomial	CRCIN = 78h→56h→34h→12h
CRC-CCITT ($X^{16}+X^{12}+X^5+1$)	(CRCDH, CRCDL) = FF9FH→BBC3H→A367H→D0FAH
CRC-16 ($X^{16}+X^{15}+X^2+1$)	(CRCDH, CRCDL) = 0110h→91F1h→F2DEh→5C43h

Note: The initial value of the CRC checksum register pair, CRCDH and CRCDL, is zero before the sequential CRC data input operation.

Program Memory CRC Checksum Calculation Example

1. Clear the checksum register pair, CRCDH and CRCDL.

2. Select the CRC-CCITT or CRC-16 polynomial as the generating polynomial using the POLY bit in the CRCCR register.

3. Execute the table read instruction to read the program memory data value.

4. Write the table data low byte into the CRCIN register and execute the CRC calculation with the current CRCSUM value. Then a new CRCSUM result will be obtained and stored in the CRC checksum register pair, CRCDH and CRCDL.

5. Write the table data high byte into the CRCIN register and execute the CRC calculation with the current CRCSUM value. Then a new CRCSUM result will be obtained and stored in the CRC checksum register pair, CRCDH and CRCDL.

6. Repeat the procedure 3 ~ procedure 5 to read the next program memory data value and execute the CRC calculation until all program memory data are read followed by the sequential CRC calculation. Then the value in the CRC checksum register pair is the final CRC calculation result.

Interrupts

Interrupts are an important part of any microcontroller system. When an external event or an internal function such as a Timer Module or an A/D converter requires microcontroller attention, their corresponding interrupt will enforce a temporary suspension of the main program allowing the microcontroller to direct attention to their respective needs. The device contains several external interrupt and internal interrupt functions. The external interrupts are generated by the action of the external INT0~INT3 pins, while the internal interrupts are generated by various internal functions such as the Timer Module (TM), Time Bases, Low Voltage Detector (LVD), EEPROM, UART, SIM, Comparator, CAN Bus and the A/D converter.

Interrupt Registers

Overall interrupt control, which basically means the setting of request flags when certain microcontroller conditions occur and the setting of interrupt enable bits by the application program, is controlled by a series of registers, located in the Special Purpose Data Memory. The registers fall into three categories. The first is the INTC0~INTC3 registers which setup the primary interrupts, the second is the MFI0~MFI7 registers which setup the Multi-function interrupts. Finally there is an INTEG register to setup the external interrupts trigger edge type.

Each register contains a number of enable bits to enable or disable individual registers as well as interrupt flags to indicate the presence of an interrupt request. The naming convention of these follows a specific pattern. First is listed an abbreviated interrupt type, then the (optional) number of that interrupt followed by either an “E” for enable/disable bit or “F” for request flag.

Function	Enable Bit	Request Flag	Notes
Global	EMI	—	—
INTn Pin	INTnE	INTnF	n=0~3
A/D Converter	ADE	ADF	—
Multi-function	MFnE	MFnF	n=0~7
Time Base	TBnE	TBnF	n=0~1
LVD	LVE	LVF	—
EEPROM	DEE	DEF	—
SIM	SIME	SIMF	—
SPIA	SPIAE	SPIAF	—
UART	URnE	URnF	n=0~2
Comparator	CPnE	CPnF	n=0~1
CAN Bus	RXMO1E	RXMO1F	—
	RXMOXE	RXMOXF	—
	CANCTLE	CANCTLF	—
	SOFE	SOFF	—
TM	STMnPE	STMnPF	n=0~1
	STMnAE	STMnAF	
	PTMnPE	PTMnPF	n=0~3
	PTMnAE	PTMnAF	

Interrupt Register Bit Naming Conventions

Register Name	Bit							
	7	6	5	4	3	2	1	0
INTEG	INT3S1	INT3S0	INT2S1	INT2S0	INT1S1	INT1S0	INT0S1	INT0S0
INTC0	—	CP0F	INT1F	INT0F	CP0E	INT1E	INT0E	EMI
INTC1	ADF	MF1F	MF0F	CP1F	ADE	MF1E	MF0E	CP1E
INTC2	MF3F	MF7F	MF6F	MF2F	MF3E	MF7E	MF6E	MF2E
INTC3	MF5F	MF4F	INT3F	INT2F	MF5E	MF4E	INT3E	INT2E
MFI0	STM0AF	STM0PF	PTM0AF	PTM0PF	STM0AE	STM0PE	PTM0AE	PTM0PE
MFI1	STM1AF	STM1PF	PTM1AF	PTM1PF	STM1AE	STM1PE	PTM1AE	PTM1PE
MFI2	—	—	PTM2AF	PTM2PF	—	—	PTM2AE	PTM2PE
MFI3	SIMF	SPIAF	DEF	LVF	SIME	SPIAE	DEE	LVE
MFI4	STM2AF	STM2PF	PTM3AF	PTM3PF	STM2AE	STM2PE	PTM3AE	PTM3PE
MFI5	—	UR2F	UR1F	UR0F	—	UR2E	UR1E	UR0E
MFI6	—	—	TB1F	TB0F	—	—	TB1E	TB0E
MFI7	SOFF	RXMOXF	RXMO1F	CANCTLF	SOFE	RXMOXE	RXMO1E	CANCTLE

Interrupt Register List

- **INTEG Register**

Bit	7	6	5	4	3	2	1	0
Name	INT3S1	INT3S0	INT2S1	INT2S0	INT1S1	INT1S0	INT0S1	INT0S0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **INT3S1~INT3S0**: Interrupt Edge Control for INT3 Pin
00: Disable
01: Rising edge
10: Falling edge
11: Rising and falling edges

Bit 5~4 **INT5S1~INT4S0**: Interrupt Edge Control for INT2 Pin
00: Disable
01: Rising edge
10: Falling edge
11: Rising and falling edges

Bit 3~2 **INT1S1~INT1S0**: Interrupt Edge Control for INT1 Pin
00: Disable
01: Rising edge
10: Falling edge
11: Rising and falling edges

Bit 1~0 **INT0S1~INT0S0**: Interrupt Edge Control for INT0 Pin
00: Disable
01: Rising edge
10: Falling edge
11: Rising and falling edges

- **INTC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	CP0F	INT1F	INT0F	CP0E	INT1E	INT0E	EMI
R/W	—	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	0	0	0	0	0	0	0

Bit 7 Unimplemented, read as “0”

Bit 6 **CP0F**: Comparator 0 Interrupt Request Flag
0: No request
1: Interrupt request

Bit 5 **INT1F**: External Interrupt 1 Request Flag
0: No request
1: Interrupt request

Bit 4 **INT0F**: External Interrupt 0 Request Flag
0: No request
1: Interrupt request

Bit 3 **CP0E**: Comparator 0 Interrupt Control
0: Disable
1: Enable

Bit 2 **INT1E**: External Interrupt 1 Control
0: Disable
1: Enable

Bit 1 **INT0E**: External Interrupt 0 Control
0: Disable
1: Enable

Bit 0 **EMI**: Global Interrupt Control
0: Disable
1: Enable

- **INTC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	ADF	MF1F	MF0F	CP1F	ADE	MF1E	MF0E	CP1E
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **ADF**: A/D Converter Interrupt Request Flag
0: No request
1: Interrupt request

Bit 6 **MF1F**: Multi-function Interrupt 1 Request Flag
0: No request
1: Interrupt request

Bit 5 **MF0F**: Multi-function Interrupt 0 Request Flag
0: No request
1: Interrupt request

Bit 4 **CP1F**: Comparator 1 Interrupt Request Flag
0: No request
1: Interrupt request

Bit 3 **ADF**: A/D Converter Interrupt Control
0: Disable
1: Enable

Bit 2 **MF1E**: Multi-function Interrupt 1 Control
0: Disable
1: Enable

Bit 1 **MF0E**: Multi-function Interrupt 0 Control
0: Disable
1: Enable

Bit 0 **CP1E**: Comparator 1 Interrupt Control
0: Disable
1: Enable

• INTC2 Register

Bit	7	6	5	4	3	2	1	0
Name	MF3F	MF7F	MF6F	MF2F	MF3E	MF7E	MF6E	MF2E
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **MF3F**: Multi-function Interrupt 3 Request Flag
0: No request
1: Interrupt request

Bit 6 **MF7F**: Multi-function Interrupt 7 Request Flag
0: No request
1: Interrupt request

Bit 5 **MF6F**: Multi-function Interrupt 6 Request Flag
0: No request
1: Interrupt request

Bit 4 **MF2F**: Multi-function Interrupt 2 Request Flag
0: No request
1: Interrupt request

Bit 3 **MF3E**: Multi-function Interrupt 3 Control
0: Disable
1: Enable

Bit 2 **MF7E**: Multi-function Interrupt 7 Control
0: Disable
1: Enable

Bit 1 **MF6E**: Multi-function Interrupt 6 Control
0: Disable
1: Enable

Bit 0 **MF2E**: Multi-function Interrupt 2 Control
0: Disable
1: Enable

• INTC3 Register

Bit	7	6	5	4	3	2	1	0
Name	MF5F	MF4F	INT3F	INT2F	MF5E	MF4E	INT3E	INT2E
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **MF5F**: Multi-function Interrupt 5 Request Flag
0: No request
1: Interrupt request

Bit 6 **MF4F**: Multi-function Interrupt 4 Request Flag
0: No request
1: Interrupt request

Bit 5 **INT3F**: External interrupt 3 Request Flag
0: No request
1: Interrupt request

Bit 4 **INT2F**: External interrupt 2 Request Flag
0: No request
1: Interrupt request

Bit 3 **MF5E**: Multi-function Interrupt 5 Control
0: Disable
1: Enable

Bit 2 **MF4E**: Multi-function Interrupt 4 Control
0: Disable
1: Enable

Bit 1 **INT3E**: External interrupt 3 Control
0: Disable
1: Enable

Bit 0 **INT2E**: External interrupt 2 Control
0: Disable
1: Enable

- **MFI0 Register**

Bit	7	6	5	4	3	2	1	0
Name	STM0AF	STM0PF	PTM0AF	PTM0PF	STM0AE	STM0PE	PTM0AE	PTM0PE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **STM0AF**: STM0 Comparator A match interrupt request flag
0: No request
1: Interrupt request

Bit 6 **STM0PF**: STM0 Comparator P match interrupt request flag
0: No request
1: Interrupt request

Bit 5 **PTM0AF**: PTM0 Comparator A match interrupt request flag
0: No request
1: Interrupt request

Bit 4 **PTM0PF**: PTM0 Comparator P match interrupt request flag
0: No request
1: Interrupt request

Bit 3 **STM0AE**: STM0 Comparator A match interrupt control
0: Disable
1: Enable

Bit 2 **STM0PE**: STM0 Comparator P match interrupt control
0: Disable
1: Enable

Bit 1 **PTM0AE**: PTM0 Comparator A match interrupt control
0: Disable
1: Enable

Bit 0 **PTM0PE**: PTM0 Comparator P match interrupt control
0: Disable
1: Enable

- **MFI1 Register**

Bit	7	6	5	4	3	2	1	0
Name	STM1AF	STM1PF	PTM1AF	PTM1PF	STM1AE	STM1PE	PTM1AE	PTM1PE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **STM1AF**: STM1 Comparator A match interrupt request flag
0: No request
1: Interrupt request

Bit 6 **STM1PF**: STM1 Comparator P match interrupt request flag
0: No request
1: Interrupt request

Bit 5 **PTM1AF**: PTM1 Comparator A match interrupt request flag
0: No request
1: Interrupt request

Bit 4 **PTM1PF**: PTM1 Comparator P match interrupt request flag
0: No request
1: Interrupt request

Bit 3 **STM1AE**: STM1 Comparator A match interrupt control
0: Disable
1: Enable

Bit 2 **STM1PE**: STM1 Comparator P match interrupt control
0: Disable
1: Enable

Bit 1 **PTM1AE**: PTM1 Comparator A match interrupt control
0: Disable
1: Enable

Bit 0 **PTM1PE**: PTM1 Comparator P match interrupt control
0: Disable
1: Enable

- **MFI2 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	PTM2AF	PTM2PF	—	—	PTM2AE	PTM2PE
R/W	—	—	R/W	R/W	—	—	R/W	R/W
POR	—	—	0	0	—	—	0	0

Bit 7~6 Unimplemented, read as “0”

Bit 5 **PTM2AF**: PTM2 Comparator A match interrupt request flag
0: No request
1: Interrupt request

Bit 4 **PTM2PF**: PTM2 Comparator P match interrupt request flag
0: No request
1: Interrupt request

Bit 3~2 Unimplemented, read as “0”

Bit 1 **PTM2AE**: PTM2 Comparator A match interrupt control
0: Disable
1: Enable

Bit 0 **PTM2PE**: PTM2 Comparator P match interrupt control
0: Disable
1: Enable

- **MFI3 Register**

Bit	7	6	5	4	3	2	1	0
Name	SIMF	SPIAF	DEF	LVF	SIME	SPIAE	DEE	LVE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **SIMF**: SIM interrupt request flag
0: No request
1: Interrupt request

Bit 6 **SPIAF**: SPIA interrupt request flag
0: No request
1: Interrupt request

Bit 5 **DEF**: Data EEPROM interrupt request flag
0: No request
1: Interrupt request

Bit 4 **LVF**: LVD interrupt request flag
0: No request
1: Interrupt request

Bit 3 **SIME**: SIM interrupt control
0: Disable
1: Enable

Bit 2 **SPIAE**: SPIA interrupt control
0: Disable
1: Enable

Bit 1 **DEE**: Data EEPROM interrupt control
0: Disable
1: Enable

Bit 0 **LVE**: LVD interrupt control
0: Disable
1: Enable

- **MFI4 Register**

Bit	7	6	5	4	3	2	1	0
Name	STM2AF	STM2PF	PTM3AF	PTM3PF	STM2AE	STM2PE	PTM3AE	PTM3PE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **STM2AF**: STM2 Comparator A match interrupt request flag
0: No request
1: Interrupt request

Bit 6 **STM2PF**: STM2 Comparator P match interrupt request flag
0: No request
1: Interrupt request

Bit 5 **PTM3AF**: PTM3 Comparator A match interrupt request flag
0: No request
1: Interrupt request

Bit 4 **PTM3PF**: PTM3 Comparator P match interrupt request flag
0: No request
1: Interrupt request

Bit 3 **STM2AE**: STM2 Comparator A match interrupt control
0: Disable
1: Enable

Bit 2 **STM2PE**: STM2 Comparator P match interrupt control
0: Disable
1: Enable

Bit 1 **PTM3AE**: PTM3 Comparator A match interrupt control
0: Disable
1: Enable

Bit 0 **PTM3PE**: PTM3 Comparator P match interrupt control
0: Disable
1: Enable

- **MFI5 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	UR2F	UR1F	UR0F	—	UR2E	UR1E	UR0E
R/W	—	R/W	R/W	R/W	—	R/W	R/W	R/W
POR	—	0	0	0	—	0	0	0

Bit 7 Unimplemented, read as “0”

Bit 6 **UR2F**: UART2 interrupt request flag
0: No request
1: Interrupt request

Bit 5 **UR1F**: UART1 interrupt request flag
0: No request
1: Interrupt request

Bit 4 **UR0F**: UART0 interrupt request flag
0: No request
1: Interrupt request

Bit 3 Unimplemented, read as “0”

Bit 2 **UR2E**: UART2 interrupt control
0: No request
1: Interrupt request

Bit 1 **UR1E**: UART1 interrupt control
0: Disable
1: Enable

Bit 0 **UR0E**: UART0 interrupt control
0: Disable
1: Enable

- **MFI6 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	TB1F	TB0F	—	—	TB1E	TB0E
R/W	—	—	R/W	R/W	—	—	R/W	R/W
POR	—	—	0	0	—	—	0	0

Bit 7~6 Unimplemented, read as “0”

Bit 5 **TB1F**: Time Base 1 interrupt request flag
0: No request
1: Interrupt request

Bit 4 **TB0F**: Time Base 0 interrupt request flag
0: No request
1: Interrupt request

Bit 3~2 Unimplemented, read as “0”

Bit 1 **TB1E**: Time Base 1 interrupt control
0: Disable
1: Enable

Bit 0 **TB0E**: Time Base 0 interrupt control
0: Disable
1: Enable

- **MFI7 Register**

Bit	7	6	5	4	3	2	1	0
Name	SOFF	RXMOXF	RXMO1F	CANCTLF	SOFE	RXMOXE	RXMO1E	CANCTLE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **SOFF**: Start of frame interrupt request flag
0: No request
1: Interrupt request

Bit 6 **RXMOXF**: MOx successful message reception interrupt request flag
0: No request
1: Interrupt request

Bit 5 **RXMO1F**: MO1 successful message reception interrupt request flag
0: No request
1: Interrupt request

Bit 4 **CANCTLF**: CAN Controller interrupt request flag
0: No request
1: Interrupt request

Bit 3 **SOFE**: Start of frame interrupt control
0: Disable
1: Enable

Bit 2 **RXMOXE**: MOx successful message reception interrupt control
0: Disable
1: Enable

Bit 1 **RXMO1E**: MO1 successful message reception interrupt control
0: Disable
1: Enable

Bit 0 **CANCTLE**: CAN Controller interrupt control
0: Disable
1: Enable

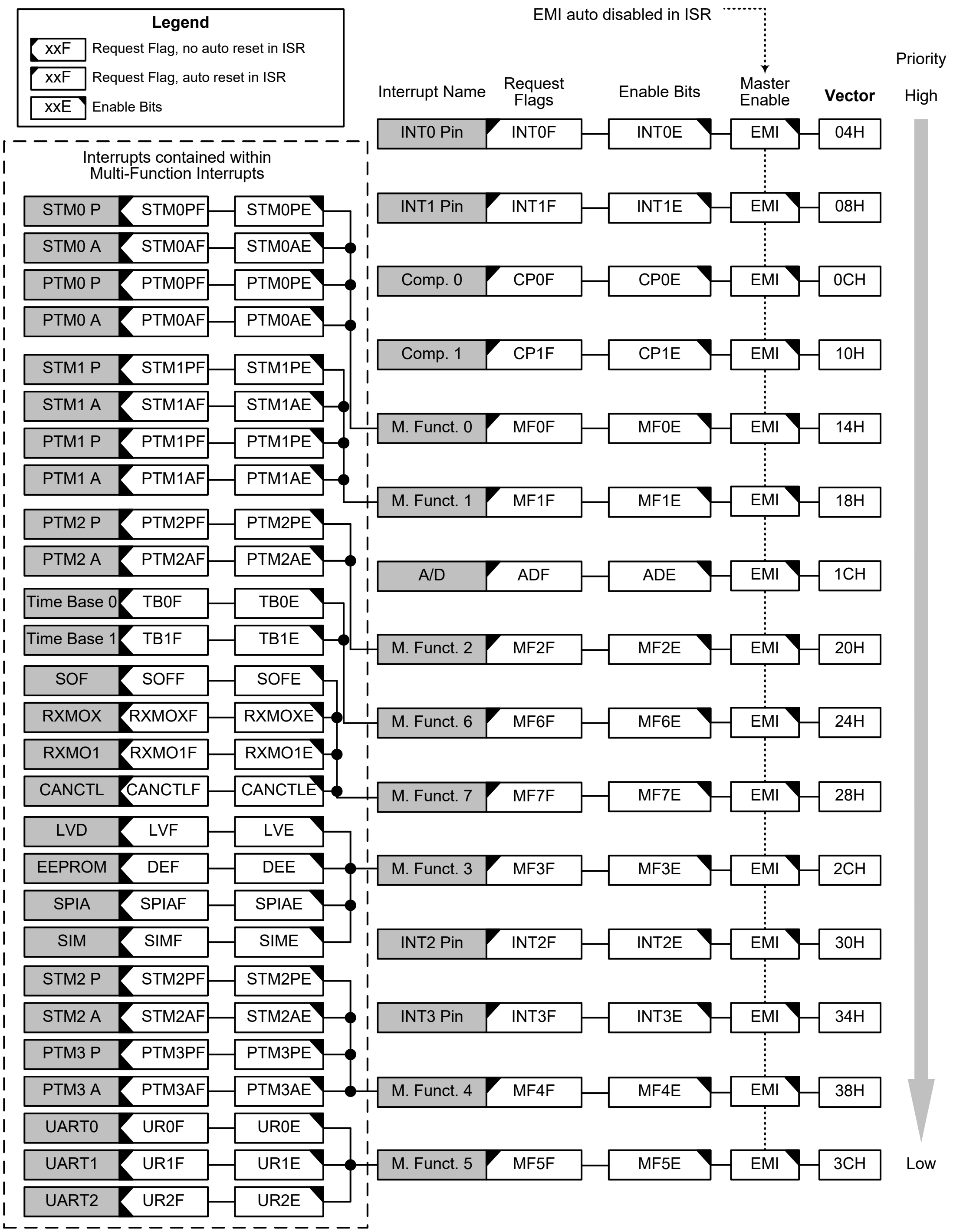
Interrupt Operation

When the conditions for an interrupt event occur, such as a TM Comparator P, Comparator A match or A/D conversion completion etc., the relevant interrupt request flag will be set. Whether the request flag actually generates a program jump to the relevant interrupt vector is determined by the condition of the interrupt enable bit. If the enable bit is set high then the program will jump to its relevant vector; if the enable bit is zero then although the interrupt request flag is set an actual interrupt will not be generated and the program will not jump to the relevant interrupt vector. The global interrupt enable bit, if cleared to zero, will disable all interrupts.

When an interrupt is generated, the Program Counter, which stores the address of the next instruction to be executed, will be transferred onto the stack. The Program Counter will then be loaded with a new address which will be the value of the corresponding interrupt vector. The microcontroller will then fetch its next instruction from this interrupt vector. The instruction at this vector will usually be a “JMP” which will jump to another section of program which is known as the interrupt service routine. Here is located the code to control the appropriate interrupt. The interrupt service routine must be terminated with a “RETI”, which retrieves the original Program Counter address from the stack and allows the microcontroller to continue with normal execution at the point where the interrupt occurred.

The various interrupt enable bits, together with their associated request flags, are shown in the accompanying diagrams with their order of priority. Some interrupt sources have their own individual vector while others share the same multi-function interrupt vector. Once an interrupt subroutine is serviced, all the other interrupts will be blocked, as the global interrupt enable bit, EMI bit will be cleared automatically. This will prevent any further interrupt nesting from occurring. However, if other interrupt requests occur during this interval, although the interrupt will not be immediately serviced, the request flag will still be recorded.

If an interrupt requires immediate servicing while the program is already in another interrupt service routine, the EMI bit should be set after entering the routine, to allow interrupt nesting. If the stack is full, the interrupt request will not be acknowledged, even if the related interrupt is enabled, until the Stack Pointer is decremented. If immediate service is desired, the stack must be prevented from becoming full. In case of simultaneous requests, the accompanying diagram shows the priority that is applied. All of the interrupt request flags when set will wake-up the device if it is in SLEEP or IDLE Mode, however to prevent a wake-up from occurring the corresponding flag should be set before the device is in SLEEP or IDLE Mode.



Interrupt Structure

External Interrupts

The external interrupts are controlled by signal transitions on the pins INT0~INT3. An external interrupt request will take place when the external interrupt request flags, INT0F~INT3F, are set, which will occur when a transition, whose type is chosen by the edge select bits, appears on the external interrupt pins. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and respective external interrupt enable bit, INT0E~INT3E, must first be set. Additionally the correct interrupt edge type must be selected using the INTEG register to enable the external interrupt function and to choose the trigger edge type. As the external interrupt pins are pin-shared with I/O pins, they can only be configured as external interrupt pins if their external interrupt enable bit in the corresponding interrupt register has been set and the external interrupt pin is selected by the corresponding pin-shared function selection bits. The pin must also be setup as an input by setting the corresponding bit in the port control register.

When the interrupt is enabled, the stack is not full and the correct transition type appears on the external interrupt pin, a subroutine call to the external interrupt vector, will take place. When the interrupt is serviced, the external interrupt request flags, INT0F~INT3F, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts. Note that any pull-high resistor selections on the external interrupt pins will remain valid even if the pin is used as an external interrupt input. The INTEG register is used to select the type of active edge that will trigger the external interrupt. A choice of either rising or falling or both edge types can be chosen to trigger an external interrupt. Note that the INTEG register can also be used to disable the external interrupt function.

Comparator Interrupts

The comparator interrupts are controlled by the two internal comparators. A comparator interrupt request will take place when the comparator interrupt request flags, CP0F or CP1F, are set, a situation that will occur when the comparator output bit changes state. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and comparator interrupt enable bit, CP0E or CP1E, must first be set. When the interrupt is enabled, the stack is not full and the comparator inputs generate a comparator output bit transition, a subroutine call to the comparator interrupt cector, will take place. When the interrupt is serviced, the comparator interrupt request flag will be automatically reset and the EMI bit will also be automatically cleared to disable other interrupts.

Multi-function Interrupts

Within the device there are seven Multi-function interrupts. Unlike the other independent interrupts, these interrupts have no independent source, but rather are formed from other existing interrupt sources, namely the TM interrupts, LVD interrupt, EEPROM write operation interrupt, CAN Bus, SIM interface, SPIA interface and UART interface interrupts.

A Multi-function interrupt request will take place when any of the Multi-function interrupt request flags MFnF are set. The Multi-function interrupt flags will be set when any of their included functions generate an interrupt request flag. When the Multi-function interrupt is enabled and the stack is not full, and either one of the interrupts contained within each of Multi-function interrupt occurs, a subroutine call to one of the Multi-function interrupt vectors will take place. When the interrupt is serviced, the related Multi-Function request flag will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

However, it must be noted that, although the Multi-function Interrupt request flags will be automatically reset when the interrupt is serviced, the request flags from the original source of the Multi-function interrupts will not be automatically reset and must be manually reset by the application program.

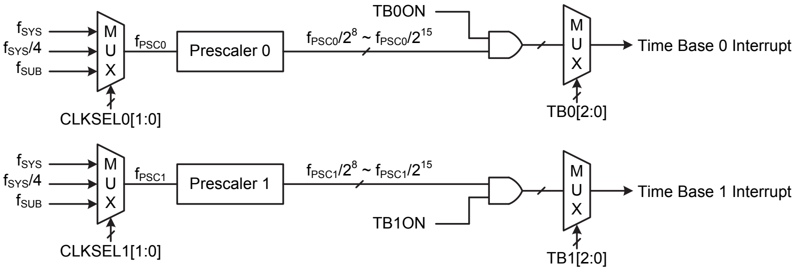
A/D Converter Interrupt

The A/D Converter Interrupt is controlled by the termination of an A/D conversion process. An A/D Converter Interrupt request will take place when the A/D Converter Interrupt request flag, ADF, is set, which occurs when the A/D conversion process finishes. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and A/D Interrupt enable bit, ADE, must first be set. When the interrupt is enabled, the stack is not full and the A/D conversion process has ended, a subroutine call to the A/D Converter Interrupt vector, will take place. When the interrupt is serviced, the A/D Converter Interrupt flag, ADF, will be automatically cleared. The EMI bit will also be automatically cleared to disable other interrupts.

Time Base Interrupts

The Time Base Interrupt is contained within the Multi-function Interrupt. The function of the Time Base Interrupts is to provide regular time signal in the form of an internal interrupt. They are controlled by the overflow signals from their respective timer functions. When these happens their respective interrupt request flags, TB0F or TB1F will be set. To allow the program to branch to their respective interrupt vector addresses, the global interrupt enable bit, EMI, Time Base enable bits, TB0E or TB1E, and associated Multi-function interrupt enable bit, must first be set. When the interrupt is enabled, the stack is not full and the Time Base overflows, a subroutine call to the revelant Multi-function Interrupt vector locations will take place. When the interrupt is serviced, the EMI bit will be automatically cleared to disable other interrupts, however only the Multi-function interrupt request flag will be also automatically cleared. As the respective interrupt request flag, TB0F or TB1F, will not be automatically cleared, it has to be cleared by the application program.

The purpose of the Time Base Interrupt is to provide an interrupt signal at fixed time periods. Its clock source, f_{PSC0} or f_{PSC1}, originates from the internal clock source f_{SYS}, $f_{SYS}/4$ or f_{SUB} and then passes through a divider, the division ratio of which is selected by programming the appropriate bits in the TB0C and TB1C registers to obtain longer interrupt periods whose value ranges. The clock source which in turn controls the Time Base interrupt period is selected using the CLKSEL0[1:0] and CLKSEL1[1:0] bits in the PSC0R and PSC1R register respectively.



Time Base Interrupt

- **PSC0R Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	CLKSEL01	CLKSEL00
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as "0"

Bit 1~0 **CLKSEL01~CLKSEL00**: Prescaler 0 clock source selection
00: f_{SYS}
01: $f_{SYS}/4$
1x: f_{SUB}

- **PSC1R Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	CLKSEL11	CLKSEL10
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as “0”

Bit 1~0 **CLKSEL11~CLKSEL10**: Prescaler 1 clock source selection
00: f_{SYS}
01: $f_{SYS}/4$
1x: f_{SUB}

- **TB0C Register**

Bit	7	6	5	4	3	2	1	0
Name	TB0ON	—	—	—	—	TB02	TB01	TB00
R/W	R/W	—	—	—	—	R/W	R/W	R/W
POR	0	—	—	—	—	0	0	0

Bit 7 **TB0ON**: Time Base 0 Control
0: Disable
1: Enable

Bit 6~3 Unimplemented, read as “0”

Bit 2~0 **TB02~TB00**: Select Time Base 0 Time-out Period
000: $2^8/f_{PSC0}$
001: $2^9/f_{PSC0}$
010: $2^{10}/f_{PSC0}$
011: $2^{11}/f_{PSC0}$
100: $2^{12}/f_{PSC0}$
101: $2^{13}/f_{PSC0}$
110: $2^{14}/f_{PSC0}$
111: $2^{15}/f_{PSC0}$

- **TB1C Register**

Bit	7	6	5	4	3	2	1	0
Name	TB1ON	—	—	—	—	TB12	TB11	TB10
R/W	R/W	—	—	—	—	R/W	R/W	R/W
POR	0	—	—	—	—	0	0	0

Bit 7 **TB1ON**: Time Base 1 Control
0: Disable
1: Enable

Bit 6~3 Unimplemented, read as “0”

Bit 2~0 **TB12~TB10**: Select Time Base 1 Time-out Period
000: $2^8/f_{PSC1}$
001: $2^9/f_{PSC1}$
010: $2^{10}/f_{PSC1}$
011: $2^{11}/f_{PSC1}$
100: $2^{12}/f_{PSC1}$
101: $2^{13}/f_{PSC1}$
110: $2^{14}/f_{PSC1}$
111: $2^{15}/f_{PSC1}$

Timer Module Interrupts

The Standard and Periodic TMs each have two interrupts, one comes from the comparator A match situation and the other comes from the comparator P match situation. All of the TM interrupts are contained within the Multi-function Interrupts. For all of the TM types there are two interrupt request flags and two enable control bits. A TM interrupt request will take place when any of the TM request flags are set, a situation which occurs when a TM comparator P or A match situation happens.

To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, respective TM Interrupt enable bit, and relevant Multi-function Interrupt enable bit, MFnE, must first be set. When the interrupt is enabled, the stack is not full and a TM comparator match situation occurs, a subroutine call to the relevant Multi-function Interrupt vector locations, will take place. When the TM interrupt is serviced, the EMI bit will be automatically cleared to disable other interrupts. However, only the related MFnF flag will be automatically cleared. As the TM interrupt request flags will not be automatically cleared, they have to be cleared by the application program.

LVD Interrupt

The Low Voltage Detector Interrupt is contained within the Multi-function Interrupt. An LVD Interrupt request will take place when the LVD Interrupt request flag, LVF, is set, which occurs when the Low Voltage Detector function detects a low power supply voltage. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, Low Voltage Interrupt enable bit, LVE, and associated Multi-function interrupt enable bit, must first be set. When the interrupt is enabled, the stack is not full and a low voltage condition occurs, a subroutine call to the Multi-function Interrupt vector, will take place. When the Low Voltage Interrupt is serviced, the EMI bit will be automatically cleared to disable other interrupts, however only the Multi-function interrupt request flag will be also automatically cleared. As the LVF flag will not be automatically cleared, it has to be cleared by the application program.

EEPROM Interrupt

The EEPROM Interrupt is contained within the Multi-function Interrupt. An EEPROM Interrupt request will take place when the EEPROM Interrupt request flag, DEF, is set, which occurs when an EEPROM Write cycle ends. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and EEPROM Interrupt enable bit, DEE, and associated Multi-function interrupt enable bit, must first be set. When the interrupt is enabled, the stack is not full and an EEPROM Write cycle ends, a subroutine call to the respective EEPROM Interrupt vector will take place. When the EEPROM Interrupt is serviced, the EMI bit will be automatically cleared to disable other interrupts, however only the Multi-function interrupt request flag will be also automatically cleared. As the DEF flag will not be automatically cleared, it has to be cleared by the application program.

Serial Interface Module Interrupt

The Serial Interface Module Interrupt, also known as the SIM interrupt, is contained within the Multi-function Interrupt. A SIM Interrupt request will take place when the SIM Interrupt request flag, SIMF, is set, which occurs when a byte of data has been received or transmitted by the SIM interface, an I^2C slave address match or I^2C bus time-out occurrence. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, the Serial Interface Interrupt enable bit, SIME, and Multi-function interrupt enable bit must first be set. When the interrupt is enabled, the stack is not full and any of the above described situations occurs, a subroutine call to the respective Multi-function Interrupt vector, will take place. When the Serial

Interface Interrupt is serviced, the EMI bit will be automatically cleared to disable other interrupts, however only the Multi-function interrupt request flag will be also automatically cleared. As the SIMF flag will not be automatically cleared, it has to be cleared by the application program.

SPIA Interface Interrupt

The SPIA Interface Module Interrupt is contained within the Multi-function Interrupt. A SPIA Interrupt request will take place when the SPIA Interrupt request flag, SPIAF, is set, which occurs when a byte of data has been received or transmitted by the SPIA interface. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, the Serial Interface Interrupt enable bit, SPIAE, and Multi-function interrupt enable bit must first be set. When the interrupt is enabled, the stack is not full and a byte of data has been transmitted or received by the SPIA interface, a subroutine call to the respective Multi-function Interrupt vector, will take place. When the SPIA Interface Interrupt is serviced, the EMI bit will be automatically cleared to disable other interrupts. However, only the Multi-function interrupt request flag will be also automatically cleared. As the SPIAF flag will not be automatically cleared, it has to be cleared by the application program.

UART Interrupts

The UARTn Transfer Interrupt is contained within the Multi-function interrupt and controlled by several UARTn transfer conditions. When one of these conditions occurs, an interrupt pulse will be generated to get the attention of the microcontroller. These conditions are a transmitter data register empty, transmitter idle, receiver data available, receiver overrun, address detect and an RXn pin wake-up. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, the UART Interrupt enable bit, URnE, and Multi-function interrupt enable bit must first be set. When the interrupt is enabled, the stack is not full and any of the conditions described above occurs, a subroutine call to the relevant Multi-function Interrupt vector, will take place. When the UARTn interrupt is serviced, the EMI bit will be automatically cleared to disable other interrupts. However, only the Multi-function interrupt request flag will be automatically cleared. As the UARTn Interrupt flag, URnF, will not be automatically cleared, it has to be cleared by the application program.

CAN Interrupts

The CAN module has four interrupt sources to be used to indicate different conditions, which are all contained within the Multi-function Interrupt. More details are described in the following subsections.

SOF Interrupt

An SOF Interrupt request will take place when the correspongding interrupt request flag, SOFF, is set, which occurs when a start of frame signal is detected. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and SOF Interrupt enable bit, SOFE, and associated Multi-function interrupt enable bit, must first be set. When the interrupt is enabled, the stack is not full and a start of frame signal is detected, a subroutine call to the respective interrupt vector will take place. When the SOF Interrupt is serviced, the EMI bit will be automatically cleared to disable other interrupts, however only the Multi-function interrupt request flag will be also automatically cleared. As the SOFF flag will not be automatically cleared, it has to be cleared by the application program.

CAN Controller Interrupt

The CAN Controller Interrupt is generated by several individual conditions. When these conditions exist, an interrupt will be generated to get the attention of the microcontroller. These conditions are a CAN module busoff state, an error limit warning, a successfull message reception, a successfull message transmission or a CAN bus status change. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and CAN Controller Interrupt enable bit, CANCTLE, and associated Multi-function interrupt enable bit, must first be set. When the interrupt is enabled, the stack is not full and any of the conditions described above occurs, a subroutine call to the respective interrupt vector will take place. When the CAN Controller Interrupt is serviced, the EMI bit will be automatically cleared to disable other interrupts, however only the Multi-function interrupt request flag will be also automatically cleared. As the CANCTLF flag will not be automatically cleared, it has to be cleared by the application program.

Message Object 1 Successful Message Reception Interrupt

A Message Object 1 Successful Message Reception Interrupt request will take place when the Interrupt request flag, RXMO1F, is set, which occurs when a Message is received into Message Object 1 successfully. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and corresponding Interrupt enable bit, RXMO1E, and associated Multi-function interrupt enable bit, must first be set. When the interrupt is enabled, the stack is not full and a Message is received into Message Object 1 successfully, a subroutine call to the respective Interrupt vector will take place. When the Interrupt is serviced, the EMI bit will be automatically cleared to disable other interrupts, however only the Multi-function interrupt request flag will be also automatically cleared. As the RXMO1F flag will not be automatically cleared, it has to be cleared by the application program.

Message Object x Successful Message Reception Interrupt

A Message Object x Successful Message Reception Interrupt request will take place when the Interrupt request flag, RXMOXF, is set, which occurs when a Message is received into Message Object x successfully. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and corresponding Interrupt enable bit, RXMOXE, and associated Multi-function interrupt enable bit, must first be set. When the interrupt is enabled, the stack is not full and a Message is received into Message Object x successfully, a subroutine call to the respective Interrupt vector will take place. When the Interrupt is serviced, the EMI bit will be automatically cleared to disable other interrupts, however only the Multi-function interrupt request flag will be also automatically cleared. As the RXMOXF flag will not be automatically cleared, it has to be cleared by the application program.

Interrupt Wake-up Function

Each of the interrupt functions has the capability of waking up the microcontroller when in the SLEEP or IDLE Mode. A wake-up is generated when an interrupt request flag changes from low to high and is independent of whether the interrupt is enabled or not. Therefore, even though the device is in the SLEEP or IDLE Mode and its system oscillator stopped, situations such as external edge transitions on the external interrupt pins or a low power supply voltage may cause their respective interrupt flag to be set high and consequently generate an interrupt. Care must therefore be taken if spurious wake-up situations are to be avoided. If an interrupt wake-up function is to be disabled then the corresponding interrupt request flag should be set high before the device enters the SLEEP or IDLE Mode. The interrupt enable bits have no effect on the interrupt wake-up function.

Programming Considerations

By disabling the relevant interrupt enable bits, a requested interrupt can be prevented from being serviced, however, once an interrupt request flag is set, it will remain in this condition in the interrupt register until the corresponding interrupt is serviced or until the request flag is cleared by the application program.

Where a certain interrupt is contained within a Multi-function interrupt, then when the interrupt service routine is executed, as only the Multi-function interrupt request flags, MFnF, will be automatically cleared, the individual request flag for the function needs to be cleared by the application program.

It is recommended that programs do not use the “CALL” instruction within the interrupt service subroutine. Interrupts often occur in an unpredictable manner or need to be serviced immediately. If only one stack is left and the interrupt is not well controlled, the original control sequence will be damaged once a CALL subroutine is executed in the interrupt subroutine.

Every interrupt has the capability of waking up the microcontroller when it is in SLEEP or IDLE Mode, the wake up being generated when the interrupt request flag changes from low to high. If it is required to prevent a certain interrupt from waking up the microcontroller then its respective request flag should be first set high before enter SLEEP or IDLE Mode.

As only the Program Counter is pushed onto the stack, then when the interrupt is serviced, if the contents of the accumulator, status register or other registers are altered by the interrupt service program, their contents should be saved to the memory at the beginning of the interrupt service routine. To return from an interrupt subroutine, either a RET or RETI instruction may be executed. The RETI instruction in addition to executing a return to the main program also automatically sets the EMI bit high to allow further interrupts. The RET instruction however only executes a return to the main program leaving the EMI bit in its present zero state and therefore disabling the execution of further interrupts.

Low Voltage Detector – LVD

The device has a Low Voltage Detector function, also known as LVD. This enables the device to monitor the power supply voltage, V_{DD}, and provide a warning signal should it fall below a certain level. This function may be especially useful in battery applications where the supply voltage will gradually reduce as the battery ages, as it allows an early warning battery low signal to be generated. The Low Voltage Detector also has the capability of generating an interrupt signal.

LVD Register

The Low Voltage Detector function is controlled using a single register with the name LVDC. Three bits in this register, VLVD2~VLVD0, are used to select one of eight fixed voltages below which a low voltage condition will be determined. A low voltage condition is indicated when the LVDO bit is set. If the LVDO bit is low, this indicates that the V_{DD} voltage is above the preset low voltage value. The LVDEN bit is used to control the overall on/off function of the low voltage detector. Setting the bit high will enable the low voltage detector. Clearing the bit to zero will switch off the internal low voltage detector circuits. As the low voltage detector will consume a certain amount of power, it may be desirable to switch off the circuit when not in use, an important consideration in power sensitive battery powered applications.

- **LVDC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	LVDO	LVDEN	VBGEN	VLVD2	VLVD1	VLVD0
R/W	—	—	R	R/W	R/W	R/W	R/W	R/W
POR	—	—	0	0	0	0	0	0

Bit 7~6 Unimplemented, read as “0”

Bit 5 **LVDO**: LVD Output Flag
0: No Low Voltage Detect
1: Low Voltage Detect

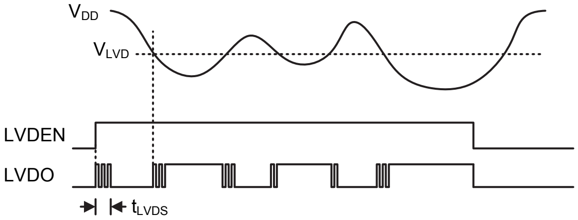
Bit 4 **LVDEN**: Low Voltage Detector Control
0: Disable
1: Enable

Bit 3 **VBGEN**: Bandgap Buffer Control
0: Disable
1: Enable
Note that the Bandgap circuit is enabled when the LVD or LVR function is enabled or when the VBGEN bit is set to 1.

Bit 2~0 **VLVD2~VLVD0**: Select LVD Voltage
000: 2.0V
001: 2.2V
010: 2.4V
011: 2.7V
100: 3.0V
101: 3.3V
110: 3.6V
111: 4.0V

LVD Operation

The Low Voltage Detector function operates by comparing the power supply voltage, V_{DD}, with a pre-specified voltage level stored in the LVDC register. This has a range of between 2.0V and 4.0V. When the power supply voltage, V_{DD}, falls below this pre-determined value, the LVDO bit will be set high indicating a low power supply voltage condition. When the device is in the SLEEP mode, the low voltage detector will be disabled even if the LVDEN bit is high. After enabling the Low Voltage Detector, a time delay t_{LVDS} should be allowed for the circuitry to stabilise before reading the LVDO bit. Note also that as the V_{DD} voltage may rise and fall rather slowly, at the voltage nears that of V_{LVD}, there may be multiple bit LVDO transitions.

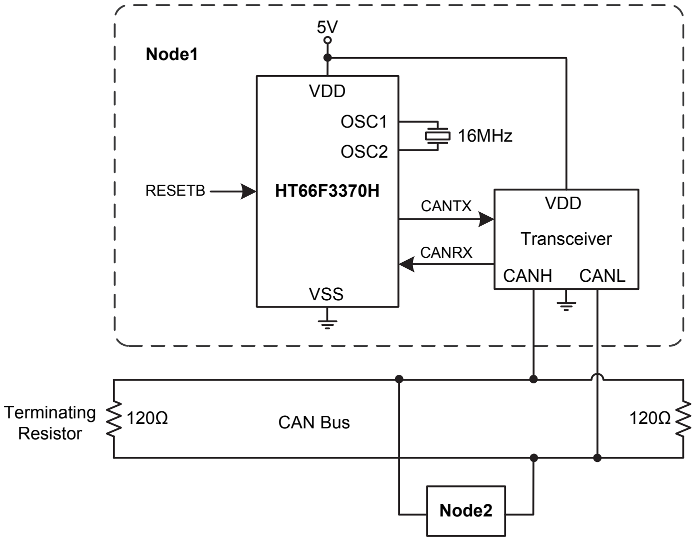


LVD Operation

The Low Voltage Detector also has its own interrupt which is contained within one of the Multi-function interrupts, providing an alternative means of low voltage detection, in addition to polling the LVDO bit. The interrupt will only be generated after a delay of t_{LVD} after the LVDO bit has been set high by a low voltage condition. In this case, the LVF interrupt request flag will be set, causing an interrupt to be generated if V_{DD} falls below the preset LVD voltage. This will cause the device to wake-up from the IDLE Mode, however if the Low Voltage Detector wake up function is not required then the LVF flag should be first set high before the device enters the IDLE Mode.

Application Circuits

For CAN Application Reference



Instruction Set

Introduction

Central to the successful operation of any microcontroller is its instruction set, which is a set of program instruction codes that directs the microcontroller to perform certain operations. In the case of Holtek microcontroller, a comprehensive and flexible set of over 60 instructions is provided to enable programmers to implement their application with the minimum of programming overheads.

For easier understanding of the various instruction codes, they have been subdivided into several functional groupings.

Instruction Timing

Most instructions are implemented within one instruction cycle. The exceptions to this are branch, call, or table read instructions where two instruction cycles are required. One instruction cycle is equal to 4 system clock cycles, therefore in the case of an 8MHz system oscillator, most instructions would be implemented within 0.5μs and branch or call instructions would be implemented within 1μs. Although instructions which require one more cycle to implement are generally limited to the JMP, CALL, RET, RETI and table read instructions, it is important to realize that any other instructions which involve manipulation of the Program Counter Low register or PCL will also take one more cycle to implement. As instructions which change the contents of the PCL will imply a direct jump to that new address, one more cycle will be required. Examples of such instructions would be “CLR PCL” or “MOV PCL, A”. For the case of skip instructions, it must be noted that if the result of the comparison involves a skip operation then this will also take one more cycle, if no skip is involved then only one cycle is required.

Moving and Transferring Data

The transfer of data within the microcontroller program is one of the most frequently used operations. Making use of several kinds of MOV instructions, data can be transferred from registers to the Accumulator and vice-versa as well as being able to move specific immediate data directly into the Accumulator. One of the most important data transfer applications is to receive data from the input ports and transfer data to the output ports.

Arithmetic Operations

The ability to perform certain arithmetic operations and data manipulation is a necessary feature of most microcontroller applications. Within the Holtek microcontroller instruction set are a range of add and subtract instruction mnemonics to enable the necessary arithmetic to be carried out. Care must be taken to ensure correct handling of carry and borrow data when results exceed 255 for addition and less than 0 for subtraction. The increment and decrement instructions such as INC, INCA, DEC and DECA provide a simple means of increasing or decreasing by a value of one of the values in the destination specified.

Logical and Rotate Operation

The standard logical operations such as AND, OR, XOR and CPL all have their own instruction within the Holtek microcontroller instruction set. As with the case of most instructions involving data manipulation, data must pass through the Accumulator which may involve additional programming steps. In all logical data operations, the zero flag may be set if the result of the operation is zero. Another form of logical data manipulation comes from the rotate instructions such as RR, RL, RRC and RLC which provide a simple means of rotating one bit right or left. Different rotate instructions exist depending on program requirements. Rotate instructions are useful for serial port programming applications where data can be rotated from an internal register into the Carry bit from where it can be examined and the necessary serial bit set high or low. Another application which rotate data operations are used is to implement multiplication and division calculations.

Branches and Control Transfer

Program branching takes the form of either jumps to specified locations using the JMP instruction or to a subroutine using the CALL instruction. They differ in the sense that in the case of a subroutine call, the program must return to the instruction immediately when the subroutine has been carried out. This is done by placing a return instruction "RET" in the subroutine which will cause the program to jump back to the address right after the CALL instruction. In the case of a JMP instruction, the program simply jumps to the desired location. There is no requirement to jump back to the original jumping off point as in the case of the CALL instruction. One special and extremely useful set of branch instructions are the conditional branches. Here a decision is first made regarding the condition of a certain data memory or individual bits. Depending upon the conditions, the program will continue with the next instruction or skip over it and jump to the following instruction. These instructions are the key to decision making and branching within the program perhaps determined by the condition of certain input switches or by the condition of internal data bits.

Bit Operations

The ability to provide single bit operations on Data Memory is an extremely flexible feature of all Holtek microcontrollers. This feature is especially useful for output port bit programming where individual bits or port pins can be directly set high or low using either the "SET [m].i" or "CLR [m].i" instructions respectively. The feature removes the need for programmers to first read the 8-bit output port, manipulate the input data to ensure that other bits are not changed and then output the port with the correct new data. This read-modify-write process is taken care of automatically when these bit operation instructions are used.

Table Read Operations

Data storage is normally implemented by using registers. However, when working with large amounts of fixed data, the volume involved often makes it inconvenient to store the fixed data in the Data Memory. To overcome this problem, Holtek microcontrollers allow an area of Program Memory to be setup as a table where data can be directly stored. A set of easy to use instructions provides the means by which this fixed data can be referenced and retrieved from the Program Memory.

Other Operations

In addition to the above functional instructions, a range of other instructions also exist such as the "HALT" instruction for Power-down operations and instructions to control the operation of the Watchdog Timer for reliable program operations under extreme electric or electromagnetic environments. For their relevant operations, refer to the functional related sections.

Instruction Set Summary

The instructions related to the data memory access in the following table can be used when the desired data memory is located in Data Memory sector 0.

Table Conventions

x: Bits immediate data
m: Data Memory address
A: Accumulator
i: 0~7 number of bits
addr: Program memory address

Mnemonic	Description	Cycles	Flag Affected
Arithmetic			
ADD A,[m]	Add Data Memory to ACC	1	Z, C, AC, OV, SC
ADDM A,[m]	Add ACC to Data Memory	1^{Note}	Z, C, AC, OV, SC
ADD A,x	Add immediate data to ACC	1	Z, C, AC, OV, SC
ADC A,[m]	Add Data Memory to ACC with Carry	1	Z, C, AC, OV, SC
ADCM A,[m]	Add ACC to Data memory with Carry	1^{Note}	Z, C, AC, OV, SC
SUB A,x	Subtract immediate data from the ACC	1	Z, C, AC, OV, SC, CZ
SUB A,[m]	Subtract Data Memory from ACC	1	Z, C, AC, OV, SC, CZ
SUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory	1^{Note}	Z, C, AC, OV, SC, CZ
SBC A,x	Subtract immediate data from ACC with Carry	1	Z, C, AC, OV, SC, CZ
SBC A,[m]	Subtract Data Memory from ACC with Carry	1	Z, C, AC, OV, SC, CZ
SBCM A,[m]	Subtract Data Memory from ACC with Carry, result in Data Memory	1^{Note}	Z, C, AC, OV, SC, CZ
DAA [m]	Decimal adjust ACC for Addition with result in Data Memory	1^{Note}	C
Logic Operation			
AND A,[m]	Logical AND Data Memory to ACC	1	Z
OR A,[m]	Logical OR Data Memory to ACC	1	Z
XOR A,[m]	Logical XOR Data Memory to ACC	1	Z
ANDM A,[m]	Logical AND ACC to Data Memory	1^{Note}	Z
ORM A,[m]	Logical OR ACC to Data Memory	1^{Note}	Z
XORM A,[m]	Logical XOR ACC to Data Memory	1^{Note}	Z
AND A,x	Logical AND immediate Data to ACC	1	Z
OR A,x	Logical OR immediate Data to ACC	1	Z
XOR A,x	Logical XOR immediate Data to ACC	1	Z
CPL [m]	Complement Data Memory	1^{Note}	Z
CPLA [m]	Complement Data Memory with result in ACC	1	Z
Increment & Decrement			
INCA [m]	Increment Data Memory with result in ACC	1	Z
INC [m]	Increment Data Memory	1^{Note}	Z
DECA [m]	Decrement Data Memory with result in ACC	1	Z
DEC [m]	Decrement Data Memory	1^{Note}	Z
Rotate			
RRA [m]	Rotate Data Memory right with result in ACC	1	None
RR [m]	Rotate Data Memory right	1^{Note}	None
RRCA [m]	Rotate Data Memory right through Carry with result in ACC	1	C
RRC [m]	Rotate Data Memory right through Carry	1^{Note}	C
RLA [m]	Rotate Data Memory left with result in ACC	1	None
RL [m]	Rotate Data Memory left	1^{Note}	None
RLCA [m]	Rotate Data Memory left through Carry with result in ACC	1	C
RLC [m]	Rotate Data Memory left through Carry	1^{Note}	C

Mnemonic	Description	Cycles	Flag Affected
Data Move			
MOV A,[m]	Move Data Memory to ACC	1	None
MOV [m],A	Move ACC to Data Memory	1Note	None
MOV A,x	Move immediate data to ACC	1	None
Bit Operation			
CLR [m].i	Clear bit of Data Memory	1Note	None
SET [m].i	Set bit of Data Memory	1Note	None
Branch Operation			
JMP addr	Jump unconditionally	2	None
SZ [m]	Skip if Data Memory is zero	1Note	None
SZA [m]	Skip if Data Memory is zero with data movement to ACC	1Note	None
SZ [m].i	Skip if bit i of Data Memory is zero	1Note	None
SNZ [m]	Skip if Data Memory is not zero	1Note	None
SNZ [m].i	Skip if bit i of Data Memory is not zero	1Note	None
SIZ [m]	Skip if increment Data Memory is zero	1Note	None
SDZ [m]	Skip if decrement Data Memory is zero	1Note	None
SIZA [m]	Skip if increment Data Memory is zero with result in ACC	1Note	None
SDZA [m]	Skip if decrement Data Memory is zero with result in ACC	1Note	None
CALL addr	Subroutine call	2	None
RET	Return from subroutine	2	None
RET A,x	Return from subroutine and load immediate data to ACC	2	None
RETI	Return from interrupt	2	None
Table Read Operation			
TABRD [m]	Read table (specific page) to TBLH and Data Memory	2Note	None
TABRDL [m]	Read table (last page) to TBLH and Data Memory	2Note	None
ITABRD [m]	Increment table pointer TBLP first and Read table to TBLH and Data Memory	2Note	None
ITABRDL [m]	Increment table pointer TBLP first and Read table (last page) to TBLH and Data Memory	2Note	None
Miscellaneous			
NOP	No operation	1	None
CLR [m]	Clear Data Memory	1Note	None
SET [m]	Set Data Memory	1Note	None
CLR WDT	Clear Watchdog Timer	1	TO, PDF
SWAP [m]	Swap nibbles of Data Memory	1Note	None
SWAPA [m]	Swap nibbles of Data Memory with result in ACC	1	None
HALT	Enter power down mode	1	TO, PDF

Note: 1. For skip instructions, if the result of the comparison involves a skip then up to three cycles are required, if no skip takes place only one cycle is required.

2. Any instruction which changes the contents of the PCL will also require 2 cycles for execution.

3. For the "CLR WDT" instruction the TO and PDF flags may be affected by the execution status. The TO and PDF flags are cleared after the "CLR WDT" instructions is executed. Otherwise the TO and PDF flags remain unchanged.

Extended Instruction Set

The extended instructions are used to support the full range address access for the data memory. When the accessed data memory is located in any data memory sector except sector 0, the extended instruction can be used to directly access the data memory instead of using the indirect addressing access. This can not only reduce the use of Flash memory space but also improve the CPU execution efficiency.

Mnemonic	Description	Cycles	Flag Affected
Arithmetic			
LADD A,[m]	Add Data Memory to ACC	2	Z, C, AC, OV, SC
LADDM A,[m]	Add ACC to Data Memory	2^{Note}	Z, C, AC, OV, SC
LADC A,[m]	Add Data Memory to ACC with Carry	2	Z, C, AC, OV, SC
LADCM A,[m]	Add ACC to Data memory with Carry	2^{Note}	Z, C, AC, OV, SC
LSUB A,[m]	Subtract Data Memory from ACC	2	Z, C, AC, OV, SC, CZ
LSUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory	2^{Note}	Z, C, AC, OV, SC, CZ
LSBC A,[m]	Subtract Data Memory from ACC with Carry	2	Z, C, AC, OV, SC, CZ
LSBCM A,[m]	Subtract Data Memory from ACC with Carry, result in Data Memory	2^{Note}	Z, C, AC, OV, SC, CZ
LDAA [m]	Decimal adjust ACC for Addition with result in Data Memory	2^{Note}	C
Logic Operation			
LAND A,[m]	Logical AND Data Memory to ACC	2	Z
LOR A,[m]	Logical OR Data Memory to ACC	2	Z
LXOR A,[m]	Logical XOR Data Memory to ACC	2	Z
LANDM A,[m]	Logical AND ACC to Data Memory	2^{Note}	Z
LORM A,[m]	Logical OR ACC to Data Memory	2^{Note}	Z
LXORM A,[m]	Logical XOR ACC to Data Memory	2^{Note}	Z
LCPL [m]	Complement Data Memory	2^{Note}	Z
LCPLA [m]	Complement Data Memory with result in ACC	2	Z
Increment & Decrement			
LINCA [m]	Increment Data Memory with result in ACC	2	Z
LINC [m]	Increment Data Memory	2^{Note}	Z
LDECA [m]	Decrement Data Memory with result in ACC	2	Z
LDEC [m]	Decrement Data Memory	2^{Note}	Z
Rotate			
LRRA [m]	Rotate Data Memory right with result in ACC	2	None
LRR [m]	Rotate Data Memory right	2^{Note}	None
LRRCA [m]	Rotate Data Memory right through Carry with result in ACC	2	C
LRRC [m]	Rotate Data Memory right through Carry	2^{Note}	C
LRLA [m]	Rotate Data Memory left with result in ACC	2	None
LRL [m]	Rotate Data Memory left	2^{Note}	None
LRLCA [m]	Rotate Data Memory left through Carry with result in ACC	2	C
LRLC [m]	Rotate Data Memory left through Carry	2^{Note}	C
Data Move			
LMOV A,[m]	Move Data Memory to ACC	2	None
LMOV [m],A	Move ACC to Data Memory	2^{Note}	None
Bit Operation			
LCLR [m].i	Clear bit of Data Memory	2^{Note}	None
LSET [m].i	Set bit of Data Memory	2^{Note}	None

Mnemonic	Description	Cycles	Flag Affected
Branch			
LSZ [m]	Skip if Data Memory is zero	2[Note]	None
LSZA [m]	Skip if Data Memory is zero with data movement to ACC	2[Note]	None
LSNZ [m]	Skip if Data Memory is not zero	2[Note]	None
LSZ [m].i	Skip if bit i of Data Memory is zero	2[Note]	None
LSNZ [m].i	Skip if bit i of Data Memory is not zero	2[Note]	None
LSIZ [m]	Skip if increment Data Memory is zero	2[Note]	None
LSDZ [m]	Skip if decrement Data Memory is zero	2[Note]	None
LSIZA [m]	Skip if increment Data Memory is zero with result in ACC	2[Note]	None
LSDZA [m]	Skip if decrement Data Memory is zero with result in ACC	2[Note]	None
Table Read			
LTABRD [m]	Read table to TBLH and Data Memory	3[Note]	None
LTABRDL [m]	Read table (last page) to TBLH and Data Memory	3[Note]	None
LITABRD [m]	Increment table pointer TBLP first and Read table to TBLH and Data Memory	3[Note]	None
LITABRDL [m]	Increment table pointer TBLP first and Read table (last page) to TBLH and Data Memory	3[Note]	None
Miscellaneous			
LCLR [m]	Clear Data Memory	2[Note]	None
LSET [m]	Set Data Memory	2[Note]	None
LSWAP [m]	Swap nibbles of Data Memory	2[Note]	None
LSWAPA [m]	Swap nibbles of Data Memory with result in ACC	2	None

Note: 1. For these extended skip instructions, if the result of the comparison involves a skip then up to four cycles are required, if no skip takes place two cycles is required.

2. Any extended instruction which changes the contents of the PCL register will also require three cycles for execution.

Instruction Definition

ADC A,[m] Add Data Memory to ACC with Carry

Description The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the Accumulator.

Operation ACC ← ACC + [m] + C

Affected flag(s) OV, Z, AC, C, SC

ADCM A,[m] Add ACC to Data Memory with Carry

Description The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the specified Data Memory.

Operation [m] ← ACC + [m] + C

Affected flag(s) OV, Z, AC, C, SC

ADD A,[m] Add Data Memory to ACC

Description The contents of the specified Data Memory and the Accumulator are added. The result is stored in the Accumulator.

Operation ACC ← ACC + [m]

Affected flag(s) OV, Z, AC, C, SC

ADD A,x Add immediate data to ACC

Description The contents of the Accumulator and the specified immediate data are added. The result is stored in the Accumulator.

Operation ACC ← ACC + x

Affected flag(s) OV, Z, AC, C, SC

ADDM A,[m] Add ACC to Data Memory

Description The contents of the specified Data Memory and the Accumulator are added. The result is stored in the specified Data Memory.

Operation [m] ← ACC + [m]

Affected flag(s) OV, Z, AC, C, SC

AND A,[m] Logical AND Data Memory to ACC

Description Data in the Accumulator and the specified Data Memory perform a bitwise logical AND operation. The result is stored in the Accumulator.

Operation ACC ← ACC ″AND″ [m]

Affected flag(s) Z

AND A,x Logical AND immediate data to ACC

Description Data in the Accumulator and the specified immediate data perform a bit wise logical AND operation. The result is stored in the Accumulator.

Operation ACC ← ACC ″AND″ x

Affected flag(s) Z

ANDM A,[m] Logical AND ACC to Data Memory

Description Data in the specified Data Memory and the Accumulator perform a bitwise logical AND operation. The result is stored in the Data Memory.

Operation [m] ← ACC ″AND″ [m]

Affected flag(s) Z

CALL addr Subroutine call

Description Unconditionally calls a subroutine at the specified address. The Program Counter then increments by 1 to obtain the address of the next instruction which is then pushed onto the stack. The specified address is then loaded and the program continues execution from this new address. As this instruction requires an additional operation, it is a two cycle instruction.

Operation Stack ← Program Counter + 1
Program Counter ← addr

Affected flag(s) None

CLR [m] Clear Data Memory

Description Each bit of the specified Data Memory is cleared to 0.

Operation [m] ← 00H

Affected flag(s) None

CLR [m].i Clear bit of Data Memory

Description Bit i of the specified Data Memory is cleared to 0.

Operation [m].i ← 0

Affected flag(s) None

CLR WDT Clear Watchdog Timer

Description The TO, PDF flags and the WDT are all cleared.

Operation WDT cleared
TO ← 0
PDF ← 0

Affected flag(s) TO, PDF

CPL [m] Complement Data Memory

Description Each bit of the specified Data Memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice versa.

Operation $[m] \leftarrow \overline{[m]}$

Affected flag(s) Z

CPLA [m] Complement Data Memory with result in ACC

Description Each bit of the specified Data Memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice versa. The complemented result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation $ACC \leftarrow \overline{[m]}$

Affected flag(s) Z

DAA [m] Decimal-Adjust ACC for addition with result in Data Memory

Description Convert the contents of the Accumulator value to a BCD (Binary Coded Decimal) value resulting from the previous addition of two BCD variables. If the low nibble is greater than 9 or if AC flag is set, then a value of 6 will be added to the low nibble. Otherwise the low nibble remains unchanged. If the high nibble is greater than 9 or if the C flag is set, then a value of 6 will be added to the high nibble. Essentially, the decimal conversion is performed by adding 00H, 06H, 60H or 66H depending on the Accumulator and flag conditions. Only the C flag may be affected by this instruction which indicates that if the original BCD sum is greater than 100, it allows multiple precision decimal addition.

Operation [m] ← ACC + 00H or
[m] ← ACC + 06H or
[m] ← ACC + 60H or
[m] ← ACC + 66H

Affected flag(s) C

DEC [m] Decrement Data Memory
Description Data in the specified Data Memory is decremented by 1.
Operation [m] ← [m] − 1
Affected flag(s) Z

DECA [m] Decrement Data Memory with result in ACC
Description Data in the specified Data Memory is decremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.
Operation ACC ← [m] − 1
Affected flag(s) Z

HALT Enter power down mode
Description This instruction stops the program execution and turns off the system clock. The contents of the Data Memory and registers are retained. The WDT and prescaler are cleared. The power down flag PDF is set and the WDT time-out flag TO is cleared.
Operation TO ← 0
PDF ← 1
Affected flag(s) TO, PDF

INC [m] Increment Data Memory
Description Data in the specified Data Memory is incremented by 1.
Operation [m] ← [m] + 1
Affected flag(s) Z

INCA [m] Increment Data Memory with result in ACC
Description Data in the specified Data Memory is incremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.
Operation ACC ← [m] + 1
Affected flag(s) Z

JMP addr Jump unconditionally
Description The contents of the Program Counter are replaced with the specified address. Program execution then continues from this new address. As this requires the insertion of a dummy instruction while the new address is loaded, it is a two cycle instruction.
Operation Program Counter ← addr
Affected flag(s) None

MOV A,[m] Move Data Memory to ACC
Description The contents of the specified Data Memory are copied to the Accumulator.
Operation ACC ← [m]
Affected flag(s) None

MOV A,x Move immediate data to ACC
Description The immediate data specified is loaded into the Accumulator.
Operation ACC ← x
Affected flag(s) None

MOV [m],A Move ACC to Data Memory
Description The contents of the Accumulator are copied to the specified Data Memory.
Operation [m] ← ACC
Affected flag(s) None

NOP No operation

Description No operation is performed. Execution continues with the next instruction.

Operation No operation

Affected flag(s) None

OR A,[m] Logical OR Data Memory to ACC

Description Data in the Accumulator and the specified Data Memory perform a bitwise logical OR operation. The result is stored in the Accumulator.

Operation ACC ← ACC ″OR″ [m]

Affected flag(s) Z

OR A,x Logical OR immediate data to ACC

Description Data in the Accumulator and the specified immediate data perform a bitwise logical OR operation. The result is stored in the Accumulator.

Operation ACC ← ACC ″OR″ x

Affected flag(s) Z

ORM A,[m] Logical OR ACC to Data Memory

Description Data in the specified Data Memory and the Accumulator perform a bitwise logical OR operation. The result is stored in the Data Memory.

Operation [m] ← ACC ″OR″ [m]

Affected flag(s) Z

RET Return from subroutine

Description The Program Counter is restored from the stack. Program execution continues at the restored address.

Operation Program Counter ← Stack

Affected flag(s) None

RET A,x Return from subroutine and load immediate data to ACC

Description The Program Counter is restored from the stack and the Accumulator loaded with the specified immediate data. Program execution continues at the restored address.

Operation Program Counter ← Stack
ACC ← x

Affected flag(s) None

RETI Return from interrupt

Description The Program Counter is restored from the stack and the interrupts are re-enabled by setting the EMI bit. EMI is the master interrupt global enable bit. If an interrupt was pending when the RETI instruction is executed, the pending Interrupt routine will be processed before returning to the main program.

Operation Program Counter ← Stack
EMI ← 1

Affected flag(s) None

RL [m] Rotate Data Memory left

Description The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0.

Operation [m].(i+1) ← [m].i; (i=0~6)
[m].0 ← [m].7

Affected flag(s) None

RLA [m] Rotate Data Memory left with result in ACC

Description: The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation:
ACC.(i+1) ← [m].i; (i=0~6)
ACC.0 ← [m].7

Affected flag(s): None

RLC [m] Rotate Data Memory left through Carry

Description: The contents of the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into bit 0.

Operation:
[m].(i+1) ← [m].i; (i=0~6)
[m].0 ← C
C ← [m].7

Affected flag(s): C

RLCA [m] Rotate Data Memory left through Carry with result in ACC

Description: Data in the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into the bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation:
ACC.(i+1) ← [m].i; (i=0~6)
ACC.0 ← C
C ← [m].7

Affected flag(s): C

RR [m] Rotate Data Memory right

Description: The contents of the specified Data Memory are rotated right by 1 bit with bit 0 rotated into bit 7.

Operation:
[m].i ← [m].(i+1); (i=0~6)
[m].7 ← [m].0

Affected flag(s): None

RRA [m] Rotate Data Memory right with result in ACC

Description: Data in the specified Data Memory is rotated right by 1 bit with bit 0 rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation:
ACC.i ← [m].(i+1); (i=0~6)
ACC.7 ← [m].0

Affected flag(s): None

RRC [m] Rotate Data Memory right through Carry

Description: The contents of the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7.

Operation:
[m].i ← [m].(i+1); (i=0~6)
[m].7 ← C
C ← [m].0

Affected flag(s): C

RRCA [m] Rotate Data Memory right through Carry with result in ACC

Description Data in the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation ACC.i ← [m].(i+1); (i=0~6)
ACC.7 ← C
C ← [m].0

Affected flag(s) C

SBC A,[m] Subtract Data Memory from ACC with Carry

Description The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation ACC ← ACC − [m] − $\overline{C}$

Affected flag(s) OV, Z, AC, C, SC, CZ

SBC A, x Subtract immediate data from ACC with Carry

Description The immediate data and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation ACC ← ACC – [m] – $\overline{C}$

Affected flag(s) OV, Z, AC, C, SC, CZ

SBCM A,[m] Subtract Data Memory from ACC with Carry and result in Data Memory

Description The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation [m] ← ACC − [m] − $\overline{C}$

Affected flag(s) OV, Z, AC, C, SC, CZ

SDZ [m] Skip if decrement Data Memory is 0

Description The contents of the specified Data Memory are first decremented by 1. If the result is 0 the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation [m] ← [m] − 1
Skip if [m]=0

Affected flag(s) None

SDZA [m] Skip if decrement Data Memory is zero with result in ACC

Description The contents of the specified Data Memory are first decremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction.

Operation ACC ← [m] − 1
Skip if ACC=0

Affected flag(s) None

SET [m] Set Data Memory

Description Each bit of the specified Data Memory is set to 1.

Operation [m] ← FFH

Affected flag(s) None

SET [m].i Set bit of Data Memory

Description Bit i of the specified Data Memory is set to 1.

Operation [m].i ← 1

Affected flag(s) None

SIZ [m] Skip if increment Data Memory is 0

Description The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation [m] ← [m] + 1
Skip if [m]=0

Affected flag(s) None

SIZA [m] Skip if increment Data Memory is zero with result in ACC

Description The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation ACC ← [m] + 1
Skip if ACC=0

Affected flag(s) None

SNZ [m].i Skip if Data Memory is not 0

Description If the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction.

Operation Skip if [m].i ≠ 0

Affected flag(s) None

SNZ [m] Skip if Data Memory is not 0

Description If the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction.

Operation Skip if [m]≠ 0

Affected flag(s) None

SUB A,[m] Subtract Data Memory from ACC

Description The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation ACC ← ACC − [m]

Affected flag(s) OV, Z, AC, C, SC, CZ

SUBM A,[m] Subtract Data Memory from ACC with result in Data Memory

Description The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation [m] ← ACC − [m]

Affected flag(s) OV, Z, AC, C, SC, CZ

SUB A,x Subtract immediate data from ACC

Description The immediate data specified by the code is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation ACC ← ACC − x

Affected flag(s) OV, Z, AC, C, SC, CZ

SWAP [m] Swap nibbles of Data Memory

Description The low-order and high-order nibbles of the specified Data Memory are interchanged.

Operation [m].3~[m].0 ↔ [m].7~[m].4

Affected flag(s) None

SWAPA [m] Swap nibbles of Data Memory with result in ACC

Description The low-order and high-order nibbles of the specified Data Memory are interchanged. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.

Operation ACC.3~ACC.0 ← [m].7~[m].4
ACC.7~ACC.4 ← [m].3~[m].0

Affected flag(s) None

SZ [m] Skip if Data Memory is 0

Description If the contents of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation Skip if [m]=0

Affected flag(s) None

SZA [m] Skip if Data Memory is 0 with data movement to ACC

Description The contents of the specified Data Memory are copied to the Accumulator. If the value is zero, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation ACC ← [m]
Skip if [m]=0

Affected flag(s) None

SZ [m].i Skip if bit i of Data Memory is 0

Description If bit i of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction.

Operation Skip if [m].i=0

Affected flag(s) None

TABRD [m] Read table (specific page) to TBLH and Data Memory

Description The low byte of the program code (specific page) addressed by the table pointer pair (TBLP and TBHP) is moved to the specified Data Memory and the high byte moved to TBLH.

Operation [m] ← program code (low byte)
TBLH ← program code (high byte)

Affected flag(s) None

TABRDL [m] Read table (last page) to TBLH and Data Memory

Description The low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.

Operation [m] ← program code (low byte)
TBLH ← program code (high byte)

Affected flag(s) None

ITABRD [m] Increment table pointer low byte first and read table to TBLH and Data Memory

Description Increment table pointer low byte, TBLP, first and then the program code addressed by the table pointer (TBHP and TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.

Operation [m] ← program code (low byte)
TBLH ← program code (high byte)

Affected flag(s) None

ITABRDL [m] Increment table pointer low byte first and read table (last page) to TBLH and Data Memory

Description Increment table pointer low byte, TBLP, first and then the low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.

Operation [m] ← program code (low byte)
TBLH ← program code (high byte)

Affected flag(s) None

XOR A,[m] Logical XOR Data Memory to ACC

Description Data in the Accumulator and the specified Data Memory perform a bitwise logical XOR operation. The result is stored in the Accumulator.

Operation ACC ← ACC ″XOR″ [m]

Affected flag(s) Z

XORM A,[m] Logical XOR ACC to Data Memory

Description Data in the specified Data Memory and the Accumulator perform a bitwise logical XOR operation. The result is stored in the Data Memory.

Operation [m] ← ACC ″XOR″ [m]

Affected flag(s) Z

XOR A,x Logical XOR immediate data to ACC

Description Data in the Accumulator and the specified immediate data perform a bitwise logical XOR operation. The result is stored in the Accumulator.

Operation ACC ← ACC ″XOR″ x

Affected flag(s) Z

Extended Instruction Definition

The extended instructions are used to directly access the data stored in any data memory sections.

LADC A,[m] Add Data Memory to ACC with Carry
Description The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the Accumulator.
Operation ACC ← ACC + [m] + C
Affected flag(s) OV, Z, AC, C, SC

LADCM A,[m] Add ACC to Data Memory with Carry
Description The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the specified Data Memory.
Operation [m] ← ACC + [m] + C
Affected flag(s) OV, Z, AC, C, SC

LADD A,[m] Add Data Memory to ACC
Description The contents of the specified Data Memory and the Accumulator are added. The result is stored in the Accumulator.
Operation ACC ← ACC + [m]
Affected flag(s) OV, Z, AC, C, SC

LADDM A,[m] Add ACC to Data Memory
Description The contents of the specified Data Memory and the Accumulator are added. The result is stored in the specified Data Memory.
Operation [m] ← ACC + [m]
Affected flag(s) OV, Z, AC, C, SC

LAND A,[m] Logical AND Data Memory to ACC
Description Data in the Accumulator and the specified Data Memory perform a bitwise logical AND operation. The result is stored in the Accumulator.
Operation ACC ← ACC ″AND″ [m]
Affected flag(s) Z

LANDM A,[m] Logical AND ACC to Data Memory
Description Data in the specified Data Memory and the Accumulator perform a bitwise logical AND operation. The result is stored in the Data Memory.
Operation [m] ← ACC ″AND″ [m]
Affected flag(s) Z

LCLR [m] Clear Data Memory
Description Each bit of the specified Data Memory is cleared to 0.
Operation [m] ← 00H
Affected flag(s) None

LCLR [m].i Clear bit of Data Memory
Description Bit i of the specified Data Memory is cleared to 0.
Operation [m].i ← 0
Affected flag(s) None

LCPL [m] Complement Data Memory

Description: Each bit of the specified Data Memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice versa.

Operation: $[m] \leftarrow \overline{[m]}$

Affected flag(s): Z

LCPLA [m] Complement Data Memory with result in ACC

Description: Each bit of the specified Data Memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice versa. The complemented result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation: $ACC \leftarrow \overline{[m]}$

Affected flag(s): Z

LDAA [m] Decimal-Adjust ACC for addition with result in Data Memory

Description: Convert the contents of the Accumulator value to a BCD (Binary Coded Decimal) value resulting from the previous addition of two BCD variables. If the low nibble is greater than 9 or if AC flag is set, then a value of 6 will be added to the low nibble. Otherwise the low nibble remains unchanged. If the high nibble is greater than 9 or if the C flag is set, then a value of 6 will be added to the high nibble. Essentially, the decimal conversion is performed by adding 00H, 06H, 60H or 66H depending on the Accumulator and flag conditions. Only the C flag may be affected by this instruction which indicates that if the original BCD sum is greater than 100, it allows multiple precision decimal addition.

Operation: $[m] \leftarrow ACC + 00H$ or
$[m] \leftarrow ACC + 06H$ or
$[m] \leftarrow ACC + 60H$ or
$[m] \leftarrow ACC + 66H$

Affected flag(s): C

LDEC [m] Decrement Data Memory

Description: Data in the specified Data Memory is decremented by 1.

Operation: $[m] \leftarrow [m] - 1$

Affected flag(s): Z

LDECA [m] Decrement Data Memory with result in ACC

Description: Data in the specified Data Memory is decremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.

Operation: $ACC \leftarrow [m] - 1$

Affected flag(s): Z

LINC [m] Increment Data Memory

Description: Data in the specified Data Memory is incremented by 1.

Operation: $[m] \leftarrow [m] + 1$

Affected flag(s): Z

LINCA [m] Increment Data Memory with result in ACC

Description: Data in the specified Data Memory is incremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.

Operation: $ACC \leftarrow [m] + 1$

Affected flag(s): Z

LMOV A,[m] Move Data Memory to ACC
Description The contents of the specified Data Memory are copied to the Accumulator.
Operation ACC ← [m]
Affected flag(s) None

LMOV [m],A Move ACC to Data Memory
Description The contents of the Accumulator are copied to the specified Data Memory.
Operation [m] ← ACC
Affected flag(s) None

LOR A,[m] Logical OR Data Memory to ACC
Description Data in the Accumulator and the specified Data Memory perform a bitwise logical OR operation. The result is stored in the Accumulator.
Operation ACC ← ACC ″OR″ [m]
Affected flag(s) Z

LORM A,[m] Logical OR ACC to Data Memory
Description Data in the specified Data Memory and the Accumulator perform a bitwise logical OR operation. The result is stored in the Data Memory.
Operation [m] ← ACC ″OR″ [m]
Affected flag(s) Z

LRL [m] Rotate Data Memory left
Description The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0.
Operation [m].(i+1) ← [m].i; (i=0~6)
[m].0 ← [m].7
Affected flag(s) None

LRLA [m] Rotate Data Memory left with result in ACC
Description The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.
Operation ACC.(i+1) ← [m].i; (i=0~6)
ACC.0 ← [m].7
Affected flag(s) None

LRLC [m] Rotate Data Memory left through Carry
Description The contents of the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into bit 0.
Operation [m].(i+1) ← [m].i; (i=0~6)
[m].0 ← C
C ← [m].7
Affected flag(s) C

LRLCA [m] Rotate Data Memory left through Carry with result in ACC
Description Data in the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into the bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.
Operation ACC.(i+1) ← [m].i; (i=0~6)
ACC.0 ← C
C ← [m].7
Affected flag(s) C

LRR [m] Rotate Data Memory right

Description The contents of the specified Data Memory are rotated right by 1 bit with bit 0 rotated into bit 7.

Operation [m].i ← [m].(i+1); (i=0~6)
[m].7 ← [m].0

Affected flag(s) None

LRRA [m] Rotate Data Memory right with result in ACC

Description Data in the specified Data Memory is rotated right by 1 bit with bit 0 rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation ACC.i ← [m].(i+1); (i=0~6)
ACC.7 ← [m].0

Affected flag(s) None

LRRC [m] Rotate Data Memory right through Carry

Description The contents of the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7.

Operation [m].i ← [m].(i+1); (i=0~6)
[m].7 ← C
C ← [m].0

Affected flag(s) C

LRRCA [m] Rotate Data Memory right through Carry with result in ACC

Description Data in the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation ACC.i ← [m].(i+1); (i=0~6)
ACC.7 ← C
C ← [m].0

Affected flag(s) C

LSBC A,[m] Subtract Data Memory from ACC with Carry

Description The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation ACC ← ACC − [m] − $\overline{C}$

Affected flag(s) OV, Z, AC, C, SC, CZ

LSBCM A,[m] Subtract Data Memory from ACC with Carry and result in Data Memory

Description The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation [m] ← ACC − [m] − $\overline{C}$

Affected flag(s) OV, Z, AC, C, SC, CZ

LSDZ [m] Skip if decrement Data Memory is 0

Description The contents of the specified Data Memory are first decremented by 1. If the result is 0 the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation [m] ← [m] − 1
Skip if [m]=0

Affected flag(s) None

LSDZA [m] Skip if decrement Data Memory is zero with result in ACC

Description The contents of the specified Data Memory are first decremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction.

Operation ACC ← [m] − 1
Skip if ACC=0

Affected flag(s) None

LSET [m] Set Data Memory

Description Each bit of the specified Data Memory is set to 1.

Operation [m] ← FFH

Affected flag(s) None

LSET [m].i Set bit of Data Memory

Description Bit i of the specified Data Memory is set to 1.

Operation [m].i ← 1

Affected flag(s) None

LSIZ [m] Skip if increment Data Memory is 0

Description The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation [m] ← [m] + 1
Skip if [m]=0

Affected flag(s) None

LSIZA [m] Skip if increment Data Memory is zero with result in ACC

Description The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation ACC ← [m] + 1
Skip if ACC=0

Affected flag(s) None

LSNZ [m].i Skip if Data Memory is not 0

Description If the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction.

Operation Skip if [m].i ≠ 0

Affected flag(s) None

LSNZ [m] Skip if Data Memory is not 0

Description If the content of the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction.

Operation Skip if [m] ≠ 0

Affected flag(s) None

LSUB A,[m] Subtract Data Memory from ACC

Description The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation ACC ← ACC − [m]

Affected flag(s) OV, Z, AC, C, SC, CZ

LSUBM A,[m] Subtract Data Memory from ACC with result in Data Memory

Description The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation [m] ← ACC − [m]

Affected flag(s) OV, Z, AC, C, SC, CZ

LSWAP [m] Swap nibbles of Data Memory

Description The low-order and high-order nibbles of the specified Data Memory are interchanged.

Operation [m].3~[m].0 ↔ [m].7~[m].4

Affected flag(s) None

LSWAPA [m] Swap nibbles of Data Memory with result in ACC

Description The low-order and high-order nibbles of the specified Data Memory are interchanged. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.

Operation ACC.3~ACC.0 ← [m].7~[m].4
ACC.7~ACC.4 ← [m].3~[m].0

Affected flag(s) None

LSZ [m] Skip if Data Memory is 0

Description If the contents of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation Skip if [m]=0

Affected flag(s) None

LSZA [m] Skip if Data Memory is 0 with data movement to ACC

Description The contents of the specified Data Memory are copied to the Accumulator. If the value is zero, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation ACC ← [m]
Skip if [m]=0

Affected flag(s) None

LSZ [m].i Skip if bit i of Data Memory is 0

Description If bit i of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction.

Operation Skip if [m].i=0

Affected flag(s) None

LTABRD [m] Read table (current page) to TBLH and Data Memory

Description The low byte of the program code (current page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.

Operation [m] ← program code (low byte)
TBLH ← program code (high byte)

Affected flag(s) None

LTABRDL [m] Read table (last page) to TBLH and Data Memory

Description The low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.

Operation [m] ← program code (low byte)
TBLH ← program code (high byte)

Affected flag(s) None

LITABRD [m] Increment table pointer low byte first and read table to TBLH and Data Memory

Description Increment table pointer low byte, TBLP, first and then the program code addressed by the table pointer (TBHP and TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.

Operation [m] ← program code (low byte)
TBLH ← program code (high byte)

Affected flag(s) None

LITABRDL [m] Increment table pointer low byte first and read table (last page) to TBLH and Data Memory

Description Increment table pointer low byte, TBLP, first and then the low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.

Operation [m] ← program code (low byte)
TBLH ← program code (high byte)

Affected flag(s) None

LXOR A,[m] Logical XOR Data Memory to ACC

Description Data in the Accumulator and the specified Data Memory perform a bitwise logical XOR operation. The result is stored in the Accumulator.

Operation ACC ← ACC ″XOR″ [m]

Affected flag(s) Z

LXORM A,[m] Logical XOR ACC to Data Memory

Description Data in the specified Data Memory and the Accumulator perform a bitwise logical XOR operation. The result is stored in the Data Memory.

Operation [m] ← ACC ″XOR″ [m]

Affected flag(s) Z

Package Information

Note that the package information provided here is for consultation purposes only. As this information may be updated at regular intervals users are reminded to consult the Holtek website for the latest version of the Package/Carton Information.

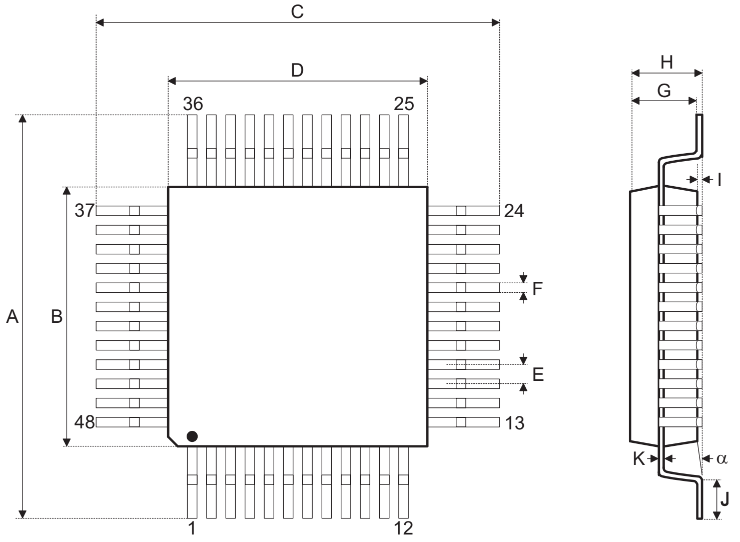
Additional supplementary information with regard to packaging is listed below. Click on the relevant section to be transferred to the relevant website page.

- Package Information (include Outline Dimensions, Product Tape and Reel Specifications)

- The Operation Instruction of Packing Materials

- Carton information

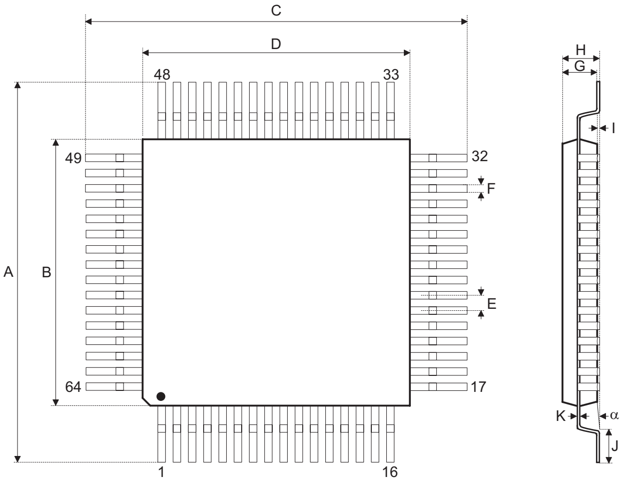
48-pin LQFP (7mm×7mm) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	—	0.354 BSC	—
B	—	0.276 BSC	—
C	—	0.354 BSC	—
D	—	0.276 BSC	—
E	—	0.020 BSC	—
F	0.007	0.009	0.011
G	0.053	0.055	0.057
H	—	—	0.063
I	0.002	—	0.006
J	0.018	0.024	0.030
K	0.004	—	0.008
α	0°	—	7°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	9.00 BSC	—
B	—	7.00 BSC	—
C	—	9.00 BSC	—
D	—	7.00 BSC	—
E	—	0.50 BSC	—
F	0.17	0.22	0.27
G	1.35	1.40	1.45
H	—	—	1.60
I	0.05	—	0.15
J	0.45	0.60	0.75
K	0.09	—	0.20
α	0°	—	7°

64-pin LQFP (7mm×7mm) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	—	0.354 BSC	—
B	—	0.276 BSC	—
C	—	0.354 BSC	—
D	—	0.276 BSC	—
E	—	0.016 BSC	—
F	0.005	0.007	0.009
G	0.053	0.055	0.057
H	—	—	0.063
I	0.002	—	0.006
J	0.018	0.024	0.030
K	0.004	—	0.008
α	0°	—	7°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	9.00 BSC	—
B	—	7.00 BSC	—
C	—	9.00 BSC	—
D	—	7.00 BSC	—
E	—	0.40 BSC	—
F	0.13	0.18	0.23
G	1.35	1.40	1.45
H	—	—	1.60
I	0.05	—	0.15
J	0.45	0.60	0.75
K	0.09	—	0.20
α	0°	—	7°

The information appearing in this Data Sheet is believed to be accurate at the time of publication. However, Holtek assumes no responsibility arising from the use of the specifications described. The applications mentioned herein are used solely for the purpose of illustration and Holtek makes no warranty or representation that such applications will be suitable without further modification, nor recommends the use of its products for application that may present a risk to human life due to malfunction or otherwise. Holtek's products are not authorized for use as critical components in life support devices or systems. Holtek reserves the right to alter its products without prior notification. For the most up-to-date information, please visit our web site at http://www.holtek.com.