Multi-channel RGB LED Flash MCU

HT45F0062

Revision: V1.30 Date: September 06, 2022

www.holtek.com

Table of Contents

Features

CPU Features

- Operating Voltage
  - f_{SYS}=8MHz: 2.2V~5.5V
- Up to 0.5μs instruction cycle with 8MHz system clock at V_{DD}=5V
- Power down and wake-up functions to reduce power consumption
- Oscillators
  - Internal High Speed RC – HIRC
  - Internal 32kHz RC – LIRC
- Multi-mode operation: NORMAL, SLOW, IDLE and SLEEP
- Fully integrated oscillators require no external components
- All instructions executed in one or two instruction cycles
- Table read instructions
- 61 powerful instructions
- 4-level subroutine nesting
- Bit manipulation instruction

Peripheral Features

- Flash Program Memory: 2K×16
- RAM Data Memory: 128×8
- Watchdog Timer function
- 14 bidirectional I/O lines
- One 10-bit CTM for time measure, compare match output and PWM output functions
- Constant current LED driver
- COM scan output driver
- Cascading transceiver interface
- I^2C Interface
- Single Time-Base function for generation of fixed time interrupt signals
- Flash program memory can be re-programmed up to 10,000 times
- Flash program memory data retention > 10 years
- Package types: 16-pin NSOP-EP, 16-pin QFN

General Description

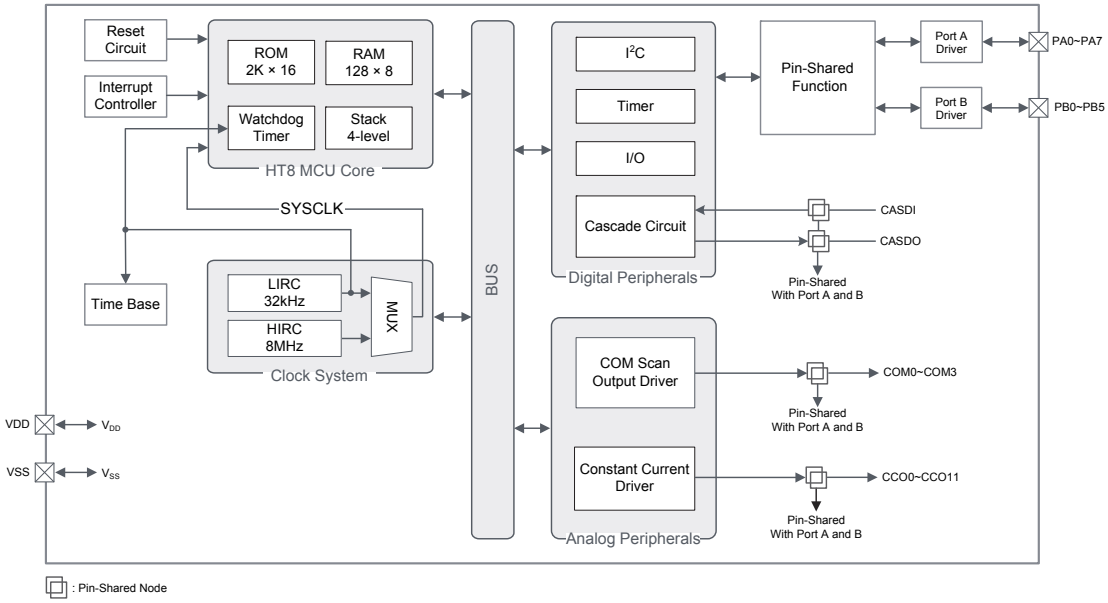
The HT45F0062 device is a Flash Memory type 8-bit high performance RISC architecture microcontroller dedicated for use in multi-channel RGB dimming LED control applications. Offering users the convenience of Flash Memory multi-programming features, the device also includes a wide range of functions and features. Other memory includes an area of RAM Data Memory.

One extremely flexible Timer Module provides timing, compare match output and PWM generation functions. Communication with the outside world is catered for by including a fully integrated I^2C interface function, which provides designers with a means of easy communication with external peripheral hardware. Protective features such as an internal Watchdog Timer coupled with excellent noise immunity and ESD protection ensure that reliable operation is maintained in hostile electrical environments.

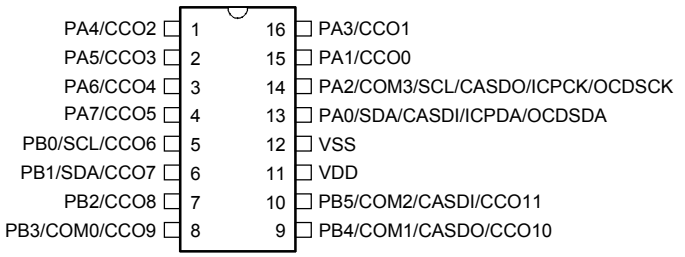
A full choice of internal high speed and low speed oscillator functions is provided including two fully integrated system oscillator which require no external components for its implementation. The ability to operate and switch dynamically between a range of operating modes using different clock sources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

Circuits specific to RGB dimming applications are also fully integrated within the device, which include PWM function, constant current LED driver and cascading transceiver interface. The inclusion of flexible I/O programming features, Time-Base function along with many other features ensure that the device will find excellent use in applications such as breathing lights, christmas lights, light strips and mood lights etc.

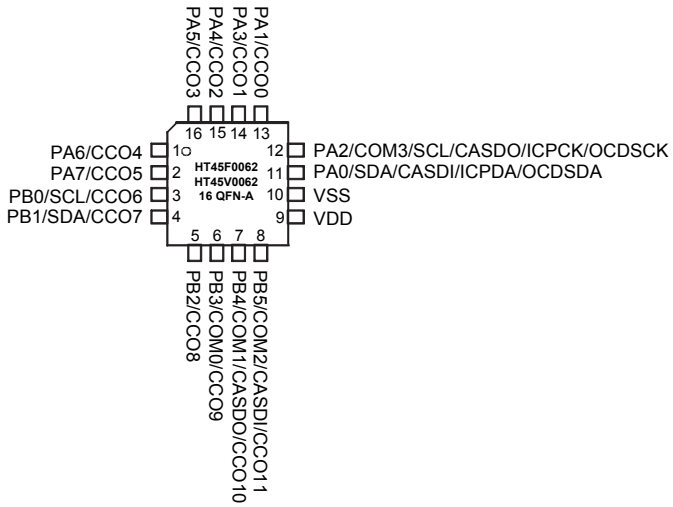
Block Diagram



Pin Assignment



HT45F0062/HT45V0062
16 NSOP-EP-A



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

2. The OCDSDA and OCDSCK pins are used as the OCDS dedicated pins and only available for the HT45V0062 device which is the OCDS EV chip of the HT45F0062.

Pin Description

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.

Pin Name	Function	OPT	I/T	O/T	Description
PA0/SDA/ CASDI/ICPDA/ OCDSDA	PA0	PAPU PAWU PAS0	ST	CMOS	General purpose I/O. Register enabled pull-high and wake-up.
	SDA	PAS0 IFS	ST	NMOS	I²C data line
	CASDI	PAS0 IFS	ST	—	Cascade transceiver interface input
	ICPDA	—	ST	CMOS	ICP data/address
	OCDSDA	—	ST	CMOS	OCDS data/address pin, for EV chip only

Pin Name	Function	OPT	I/T	O/T	Description
PA1/CCO0	PA1	PAPU PAWU PAS0	ST	CMOS	General purpose I/O. Register enabled pull-high and wake-up.
	CCO0	PAS0	—	AN	LED PWM constant current output
PA2/COM3/ SCL/CASDO/ ICPCK/CDSCK	PA2	PAPU PAWU PAS0	ST	CMOS	General purpose I/O. Register enabled pull-high and wake-up.
	COM3	PAS0	—	AN	LED PWM COM scan output
	SCL	PAS0 IFS	ST	NMOS	I²C clock line
	CASDO	PAS0	—	CMOS	Cascade transceiver interface output
	ICPCK	—	ST	—	ICP Clock pin
	OCDSCK	—	ST	—	OCDS clock pin, for EV chip only
PA3/CCO1	PA3	PAPU PAWU PAS0	ST	CMOS	General purpose I/O. Register enabled pull-high and wake-up.
	CCO1	PAS0	—	AN	LED PWM constant current output
PA4/CCO2	PA4	PAPU PAWU PAS1	ST	CMOS	General purpose I/O. Register enabled pull-high and wake-up.
	CCO2	PAS1	—	AN	LED PWM constant current output
PA5/CCO3	PA5	PAPU PAWU PAS1	ST	CMOS	General purpose I/O. Register enabled pull-high and wake-up.
	CCO3	PAS1	—	AN	LED PWM constant current output
PA6/CCO4	PA6	PAPU PAWU PAS1	ST	CMOS	General purpose I/O. Register enabled pull-high and wake-up.
	CCO4	PAS1	—	AN	LED PWM constant current output
PA7/CCO5	PA7	PAPU PAWU PAS1	ST	CMOS	General purpose I/O. Register enabled pull-high and wake-up.
	CCO5	PAS1	—	AN	LED PWM constant current output
PB0/SCL/CCO6	PB0	PBPU PBS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	SCL	PBS0 IFS	ST	NMOS	I²C clock line
	CCO6	PBS0	—	AN	LED PWM constant current output
PB1/SDA/ CCO7	PB1	PBPU PBS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	SDA	PBS0 IFS	ST	NMOS	I²C data line
	CCO7	PBS0	—	AN	LED PWM constant current output
PB2/CCO8	PB2	PBPU PBS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	CCO8	PBS0	—	AN	LED PWM constant current output
PB3/COM0/ CCO9	PB3	PBPU PBS0	ST	CMOS	General purpose I/O. Register enabled pull-up.
	COM0	PBS0	—	AN	LED PWM COM scan output
	CCO9	PBS0	—	AN	LED PWM constant current output

Pin Name	Function	OPT	I/T	O/T	Description
PB4/COM1/ CASDO/CCP10	PB4	PBPU PBS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	COM1	PBS1	—	AN	LED PWM COM scan output
	CASDO	PBS1	—	CMOS	Cascade transceiver interface output
	CCO10	PBS1	—	AN	LED PWM constant current output
PB5/COM2/ CASDI/CCO11	PB5	PBPU PBS1	ST	CMOS	General purpose I/O. Register enabled pull-up.
	COM2	PBS1	—	AN	LED PWM COM scan output
	CASDI	PBS1 IFS	ST	—	Cascade transceiver interface input
	CCO11	PBS1	—	AN	LED PWM constant current output
VDD	VDD	—	PWR	—	Positive power supply
VSS	VSS	—	PWR	—	Negative power supply, ground

Legend: I/T: Input type;
O/T: Output type;
OPT: Optional by register option;
PWR: Power;
ST: Schmitt Trigger input;
CMOS: CMOS output;
AN: Analog signal.

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

Operating Temperature ..... -40°C to 85°C

Storage Temperature ..... -60°C to 150°C

Thermal Resistance (Rth) (16 NSOP-EP) ..... 35.4˚C/W

Thermal Resistance (Rth) (16 QFN) ..... 68˚C/W

Max junction Temperature (Tj) ..... 125˚C

Power Dissipation (PD):

(@Ta=25˚C) (16 NSOP-EP) 2.83W

(@Ta=85˚C) (16 NSOP-EP) 1.13W

(@Ta=25˚C) (16 QFN) 1.47W

(@Ta=85˚C) (16 QFN) 0.95W

CCOn Output Sink Current (Single pin) 80mA

Total Power Line Current (Ta=25°C) 1000mA

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 the 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

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	Operating Voltage (HIRC)	—	$f_{SYS}=f_{HIRC}=8MHz$	2.2	—	5.5	V
I_{DD}	Operating Current (LIRC)	3V	No load, all peripherals off, $f_{SYS}=f_{LIRC}=32kHz$	—	10	20	μA
		5V		—	30	50	
	Operating Current (HIRC)	3V	No load, all peripherals off, $f_{SYS}=f_{HIRC}=8MHz$	—	1.0	2.0	mA
		5V		—	2.0	3.0	
		3V	No load, all peripherals off, $f_{SYS}=f_{HIRC}/2$, $f_{HIRC}=8MHz$	—	1.0	1.5	mA
		5V		—	1.5	2.0	
		3V	No load, all peripherals off, $f_{SYS}=f_{HIRC}/4$, $f_{HIRC}=8MHz$	—	0.9	1.3	mA
		5V		—	1.3	1.8	
		3V	No load, all peripherals off, $f_{SYS}=f_{HIRC}/8$, $f_{HIRC}=8MHz$	—	0.8	1.1	mA
		5V		—	1.1	1.6	
		3V	No load, all peripherals off, $f_{SYS}=f_{HIRC}/16$, $f_{HIRC}=8MHz$	—	0.7	1.0	mA
		5V		—	1.0	1.4	
		3V	No load, all peripherals off, $f_{SYS}=f_{HIRC}/32$, $f_{HIRC}=8MHz$	—	0.6	0.9	mA
		5V		—	0.9	1.2	
		3V	No load, all peripherals off, $f_{SYS}=f_{HIRC}/64$, $f_{HIRC}=8MHz$	—	0.5	0.8	mA
		5V		—	0.8	1.1	
I_{STB}	Standby Current (SLEEP Mode)	3V	No load, all peripherals off, WDT off	—	0.2	0.8	μA
		5V		—	0.5	1.0	
	Standby Current (SLEEP Mode)	3V	No load, all peripherals off, WDT on	—	1.3	5.0	μA
		5V		—	2.2	10	
	Standby Current (IDLE0 Mode)	3V	No load, all peripherals off, f_{SUB} on	—	1.3	3.0	μA
		5V		—	5.0	10	
	Standby Current (IDLE1 Mode, HIRC)	3V	No load, all peripherals off, f_{SUB} on, $f_{SYS}=f_{HIRC}=8MHz$	—	0.8	1.6	mA
		5V		—	1.0	2.0	
V_{IL}	Input Low Voltage for I/O Ports	5V	—	0	—	1.5	V
		—	—	0	—	$0.2V_{DD}$	
V_{IH}	Input High Voltage for I/O Ports	5V	—	3.5	—	5	V
		—	—	$0.8V_{DD}$	—	V_{DD}	
I_{OL}	Sink Current for I/O Pins	3V	$V_{OL}=0.1V_{DD}$	16	32	—	mA
		5V		32	65	—	
I_{OH}	Source Current for I/O Pins	3V	$V_{OH}=0.9V_{DD}$	-4	-8	—	mA
		5V		-8	-16	—	
R_{PH}	Pull-high Resistance for I/O Ports	3V	—	20	60	100	kΩ
		5V	—	10	30	50	
I_{LEAK}	Input Leakage Current	5V	$V_{IN}=V_{DD}$ or $V_{IN}=V_{SS}$	—	—	±1	μA
I_{OCDS}	Operating Current (Used for OCDS EV and Connected to e-Link, Normal Mode)	3V	No load, $f_{SYS}=f_{HIRC}=8MHz$, WDT enable	—	1.4	2.0	mA

A.C. Characteristics

Ta=25°C, unless otherwise specified

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
f_{SYS}	System Clock (HIRC)	2.2V~5.5V	$f_{SYS}=f_{HIRC}$=8MHz	—	8	—	MHz
	System Clock (LIRC)	2.2V~5.5V	$f_{SYS}=f_{LIRC}$=32kHz	—	32	—	kHz
f_{HIRC}	High Speed Internal RC Oscillator (HIRC)	3V/5V	Ta=25°C	-2%	8	+2%	MHz
		3V/5V	Ta=0°C~70°C	-5%	8	+5%	
		2.2V~5.5V	Ta=0°C~70°C	-8%	8	+8%	
		2.2V~5.5V	Ta=-40°C~85°C	-12%	8	+12%	
f_{LIRC}	Low Speed Internal RC Oscillator (LIRC)	5V	Ta=25°C	25.6	32	38.4	kHz
		2.2V~5.5V	Ta=25°C	12.8	32	41.6	
			Ta=-40°C~85°C	8	32	50	
t_{START}	LIRC Start Up Time	—	—	—	—	100	μs
t_{RSTD}	System Reset Delay Time (Power-on Reset, WDT Software Reset)	—	—	25	50	150	ms
	System Reset Delay Time (WDT Time-out Hardware Cold Reset)	—	—	8.3	16.7	50	ms
t_{SST}	System Start-up Time (Wake-up from Condition Where f_{SYS} is Off)	—	$f_{SYS}=f_H$~f_H/64, $f_H=f_{HIRC}$	—	16	—	t_{HIRC}
		—	$f_{SYS}=f_{SUB}=f_{LIRC}$	—	2	—	t_{LIRC}
	System Start-up Time (Wake-up from Condition Where f_{SYS} is On)	—	$f_{SYS}=f_H$~f_H/64, $f_H=f_{HIRC}$	—	2	—	t_H
		—	$f_{SYS}=f_{SUB}=f_{LIRC}$	—	2	—	t_{SUB}
	System Speed Switch Time (Slow Mode ↔ Normal Mode)	—	f_{HIRC} off → on	—	16	—	t_{HIRC}
t_{SRESET}	Minimum Software Reset Width to Reset	—	—	45	90	250	μs
t_{CASDI}	CASDI Input Pin Minimum Pulse Width	—	—	0.3	—	—	μs
$f_{CASCLKI}$	Cascade Transceiver Interface Clock Input Maximum Frequency	5V	—	—	—	8	MHz

Constant Current Characteristics

Operating Temperature: -40°C~85°C, unless otherwise specified

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	Operating Voltage	—	—	3.0	—	5.5	V
I_{CCS}	Additional Current for Constant Current Function Enable	3V	CCOn=off	—	3.8	5.0	mA
		5V		—	5.0	6.5	
I_{CCO}	CCOn Output Sink Current	5V	Current range, V_{CCOn}=1.5V, CCG[1:0]=00	Typ. -5%	5	Typ. +5%	mA
			Current range, V_{CCOn}=1.5V, CCG[1:0]=01	Typ. -10%	14	Typ. +10%	mA
			Current range, V_{CCOn}=1.5V, CCG[1:0]=10	Typ. -10%	33	Typ. +10%	mA
			Current range, V_{CCOn}=1.5V, CCG[1:0]=11	Typ. -10%	55	Typ. +10%	mA
dI_{CCO1}	Current Skew (Channel)	3V/5V	I_{CCOn}=5mA, V_{CCOn}=0.7V	—	±1.5	±3.0	%
dI_{CCO2}	Current Skew (IC)	3V/5V	I_{CCOn}=5mA, V_{CCOn}=0.7V	—	±3.0	±6.0	%
$\%/dV_{CCO}$	Output Current vs. Output Voltage Regulation	3V/5V	V_{CCOn}=0.7V~3.0V, I_{CCOn}=5mA	—	±0.1	—	%/V
$\%/dV_{DD}$	Output Current vs. Supply Voltage Regulation	—	V_{DD}=3.0V~5.5V, V_{CCOn}=0.7V	—	±1.0	±8.0	%/V

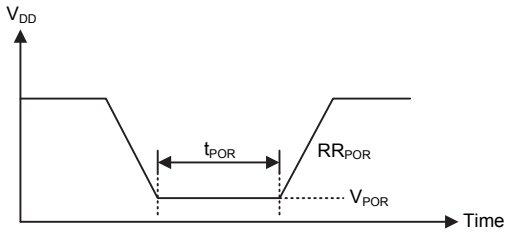
Note: $\%/dV_{CCO} = \{[I_{CCOn}$ (at V_{CCOn}=3.0V) – I_{CCOn} (at V_{CCOn}=0.7V)] / I_{CCOn} (at V_{CCOn}=1.5V)$\}$ ×100% / (3.0V – 0.7V)

$\%/dV_{DD} = \{[I_{CCOn}$ (at V_{DD}=5.5V) – I_{CCOn} (at V_{DD}=3.0V)] / I_{CCOn} (at V_{DD}=4.0V)$\}$ ×100% / (5.5V – 3.0V)

Power on Reset Characteristics

Ta=25°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

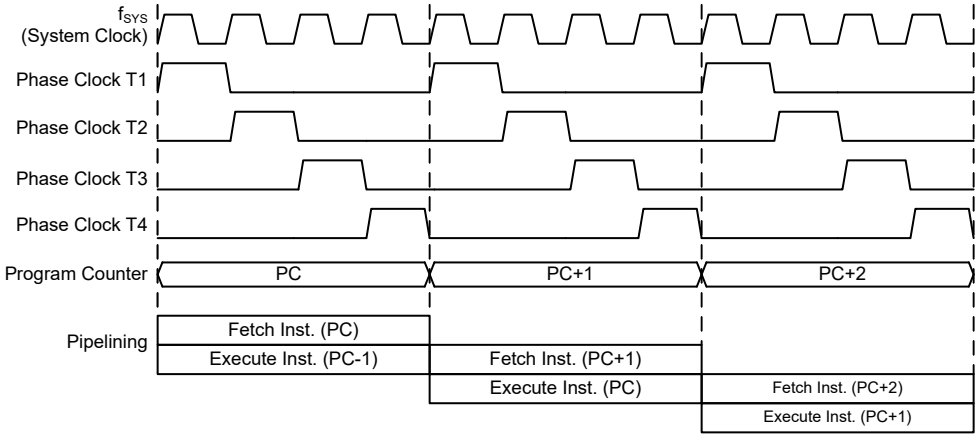


System Architecture

A key factor in the high-performance features of the Holtek 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 cycle, with the exception of branch or call instructions. 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 control system with maximum reliability and flexibility. This makes the device suitable for low-cost, high-volume production for controller applications.

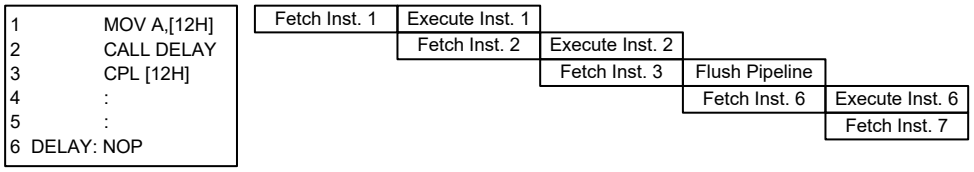
Clocking and Pipelining

The main system clock, derived from either an 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.



System Clocking and Pipelining

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.



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. 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**
PC10~PC8	PCL7~PCL0

Program Counter

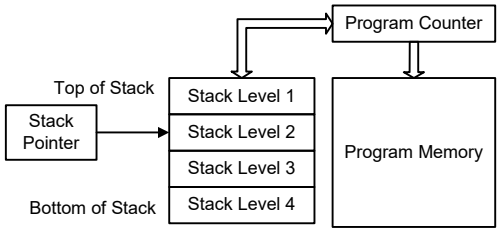
The lower byte of the Program Counter, known as the Program Counter Low register or PCL, is available for program control and is a readable and writeable register. By transferring data directly into this register, a short program jump can be executed directly. However, as only this low byte is available 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 organized into multiple levels and 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:

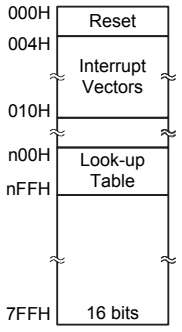
- Arithmetic operations: ADD, ADDM, ADC, ADCM, SUB, SUBM, SBC, SBCM, DAA
- Logic operations: AND, OR, XOR, ANDM, ORM, XORM, CPL, CPLA
- Rotation: RRA, RR, RRCA, RRC, RLA, RL, RLCA, RLC
- Increment and Decrement: INCA, INC, DECA, DEC
- Branch decision: JMP, SZ, SZA, SNZ, SIZ, SDZ, SIZA, SDZA, CALL, RET, RETI

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 2K×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 arranged 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 configured by placing the address of the look up data to be retrieved in the table pointer registers, 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 "TABRD [m]" or "TABRDL[m]" instructions 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. Any unused bits in this transferred higher order byte will be read as "0".

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

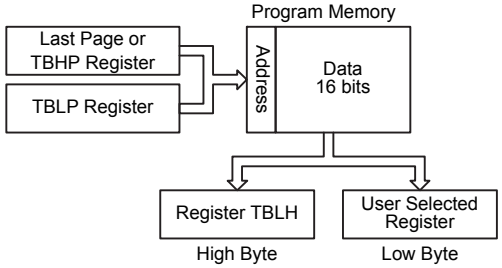


Table Program Example

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 "700H" which refers to the start address of the last page within the 2K words Program Memory of the device. The table pointer low byte register is set 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 "706H" or 6 locations after the start of the last page. Note that the value for the table pointer is referenced to the specific address pointed by the TBLP and TBHP registers if the "TABRD [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]" instruction is executed.

Because the TBLH register is a read-only register and cannot 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

```
tempreg1 db?        ; temporary register #1
tempreg2 db?        ; temporary register #2
:
:
mov a,06h           ; initialise table pointer - note that this address is referenced
mov tblp,a          ; to the last page or the page that tbhp pointed
mov a,07h           ; initialise high table pointer
mov tbhp,a          ; it is not necessary to set tbhp if executing tabrdl
:
:
tabrd tempreg1      ; transfers value in table referenced by table pointer
                    ; data at program memory address "706H" 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 "705H" transferred to tempreg2 and TBLH
                    ; in this example the data "1AH" is transferred to tempreg1 and data "0FH"
                    ; to tempreg2
                    ; the value "00H" will be transferred to the high byte register TBLH
:
:
org 700h            ; sets initial address of last page
dc 00Ah,00Bh,00Ch,00Dh,00Eh,00Fh,01Ah,01Bh
```

In Circuit Programming – ICP

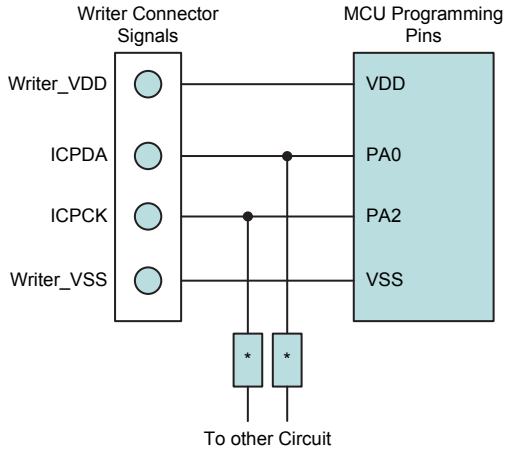
The provision of Flash type Program Memory provides the user with a means of convenient and easy upgrades and modifications to their programs on the same device.

As an additional convenience, Holtek has provided a means of programming the microcontroller in-circuit 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.

Holtek 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, the user can take care of the ICPDA and ICPCK pins for data and clock programming purposes 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

There is an EV chip named HT45V0062 which is used to emulate the real MCU device named HT45F0062. The EV chip device also provides an "On-Chip Debug" function to debug the real MCU device during the development process. The EV chip and the real MCU device are almost functionally compatible except for "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 Holtek 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 device will have no effect in the EV chip. For more detailed OCDS information, refer to the corresponding document named "Holtek e-Link for 8-bit MCU OCDS User's Guide".

Holtek 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

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.

Categorised into two types, the first of these is an area of RAM, known as the Special Function Data Memory. 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 known as the General Purpose Data Memory, which is reserved for general purpose use. All locations within this area are read and write accessible under program control.

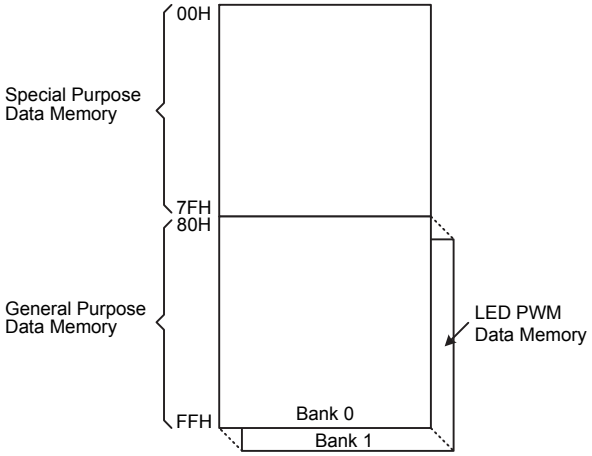
Switching between the different Data Memory banks must be achieved by properly setting the Bank Pointer to correct value.

Structure

The Data Memory is subdivided into two banks, which are implemented in 8-bit wide Memory. The Data Memory Bank 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 Memory address range is from 80H to FFH. Data Memory addresses from 80H to FFH in Bank 1 are used as the LED PWM data buffer.

Special Purpose Data Memory	General Purpose Data Memory		LED PWM Data Memory		
Located Bank	Capacity	Bank: Address	Capacity		Bank: Address
0	128×8	0: 80H~FFH	128×8	Buffer_A	1: 80H~BFH
				Buffer_B	1: C0H~FFH

Data Memory Summary



Data Memory Structure

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”.

	Bank 0
00H	IAR0
01H	MP0
02H	IAR1
03H	MP1
04H	BP
05H	ACC
06H	PCL
07H	TBLP
08H	TBLH
09H	TBHP
0AH	STATUS
0BH	
0CH	SCC
0DH	HIRCC
0EH	WDTC
0FH	RSTFC
10H	INTC0
11H	INTC1
12H	MFI
13H	
14H	PA
15H	PAC
16H	PAPU
17H	PAWU
18H	PB
19H	PBC
1AH	PBPU
1BH	
1CH	PSCR
1DH	TBC
1EH	
1FH	PAS0
20H	PAS1
21H	PBS0
22H	PBS1
23H	IFS
24H	
25H	CTMC0
26H	CTMC1
27H	CTMDL
28H	CTMDH
29H	CTMAL
2AH	CTMAH
2BH	COM_PWM
2CH	PWM0DATA
2DH	PWM1DATA
2EH	PWM2DATA
2FH	PWM3DATA
30H	PWM4DATA
31H	PWM5DATA
32H	PWM6DATA
33H	PWM7DATA
34H	PWM8DATA
35H	PWM9DATA
36H	PWM10DATA
37H	PWM11DATA
38H	CASCON
39H	CASPRE
3AH	CASTH
3BH	D0CNT
3CH	D1CNT
3DH	PCNT
3EH	RCNT
3FH	CASDNB

	Bank 0
40H	CASD0
41H	CASD1
42H	CASD2
43H	CASD3
44H	CASD4
45H	CASD5
46H	CASD6
47H	CASD7
48H	CASD8
49H	CASD9
4AH	CASD10
4BH	CASD11
4CH	CASD12
4DH	CASD13
4EH	CASD14
4FH	CASD15
50H	CASD16
51H	CASD17
52H	CASD18
53H	CASD19
54H	CASD20
55H	CASD21
56H	CASD22
57H	CASD23
58H	CASD24
59H	CASD25
5AH	CASD26
5BH	CASD27
5CH	CASD28
5DH	CASD29
5EH	CASD30
5FH	CASD31
60H	CASD32
61H	CASD33
62H	CASD34
63H	CASD35
64H	INTCON0
65H	INTCON1
66H	CCS
67H	IICC0
68H	IICC1
69H	IICD
6AH	IICA
6BH	IICTOC
6CH	CCOPU0
6DH	CCOPU1
7FH	

: Unused, read as 00H

Special Purpose Data Memory

Special Function Register Description

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

Indirect Addressing Registers – IAR0, IAR1

The Indirect Addressing Registers, IAR0 and IAR1, 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 and IAR1 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 or MP1. Acting as a pair, IAR0 and MP0 can together access data from Bank 0 while the IAR1 register together with the MP1 register can access data from any Data Memory Bank. 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, MP1

Two Memory Pointers, known as MP0 and MP1 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 Bank 0, while MP1 together with IAR1 are used to access data from all banks.

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

```
data .section ´data´
adres1   db ?
adres2   db ?
adres3   db ?
adres4   db ?
block    db ?
code .section at 0 ´code´
org 00h
start:
     mov a,04h            ; set size of block
     mov block,a
     mov a,offset adres1  ; Accumulator loaded with first RAM address
     mov mp0,a            ; set 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:
```

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

Bank Pointer – BP

The Data Memory is divided into two banks. Selecting the required Program Memory area is achieved using the Bank Pointer. The Data Memory is initialised to Bank 0 after a reset, except for a WDT time-out reset in the IDLE or SLEEP Mode, in which case, the Data Memory bank remains unaffected. Directly addressing the Data Memory will always result in Bank 0 being accessed irrespective of the value of the Bank Pointer. Accessing data from banks other than Bank 0 must be implemented using Indirect addressing.

• **BP Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	—	DMBP0
R/W	—	—	—	—	—	—	—	R/W
POR	—	—	—	—	—	—	—	0

Bit 7~1 Unimplemented, read as "0"

Bit 0 **DMBP0**: Data Memory Bank selection
0: Bank 0
1: Bank 1

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 set 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 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, and C 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.

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	—	—	TO	PDF	OV	Z	AC	C
R/W	—	—	R	R	R/W	R/W	R/W	R/W
POR	—	—	0	0	x	x	x	x

"x": unknown

Bit 7~6 Unimplemented, read as "0"

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.

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 operations are selected through the application program and relevant control registers.

Oscillator Overview

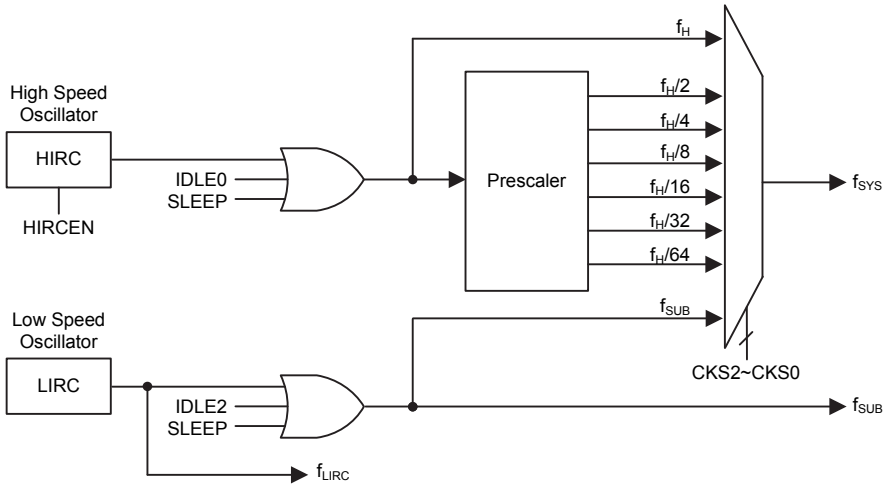
In addition to being the source of the main system clock the oscillators also provide clock sources for the Watchdog Timer and Time Base Interrupts. Two fully integrated internal oscillators, requiring no external components, are provided to form a range of both fast and slow system oscillators. The higher frequency oscillator provides higher performance but carry with it the disadvantage of higher power requirements, while the opposite is of course true for the lower frequency oscillator. 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	Frequency
Internal High Speed RC	HIRC	8MHz
Internal Low Speed RC	LIRC	32kHz

Oscillator Types

System Clock Configurations

There are two oscillator sources, a high speed oscillator and a low speed oscillator. The high speed oscillator is the internal 8MHz RC oscillator, HIRC. The low speed oscillator is the internal 32kHz RC oscillator, LIRC. 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 low speed or high speed system clock can be dynamically selected.



System Clock Configurations

High Speed Internal RC Oscillator – HIRC

The high speed internal RC oscillator is a fully integrated system oscillator requiring no external components. The internal RC oscillator has a fixed frequency of 8MHz. 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. This internal system clock requires no external pins for its operation.

Internal 32kHz Oscillator – LIRC

The internal 32kHz System Oscillator is a fully integrated RC oscillator with a typical frequency of 32kHz, 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.

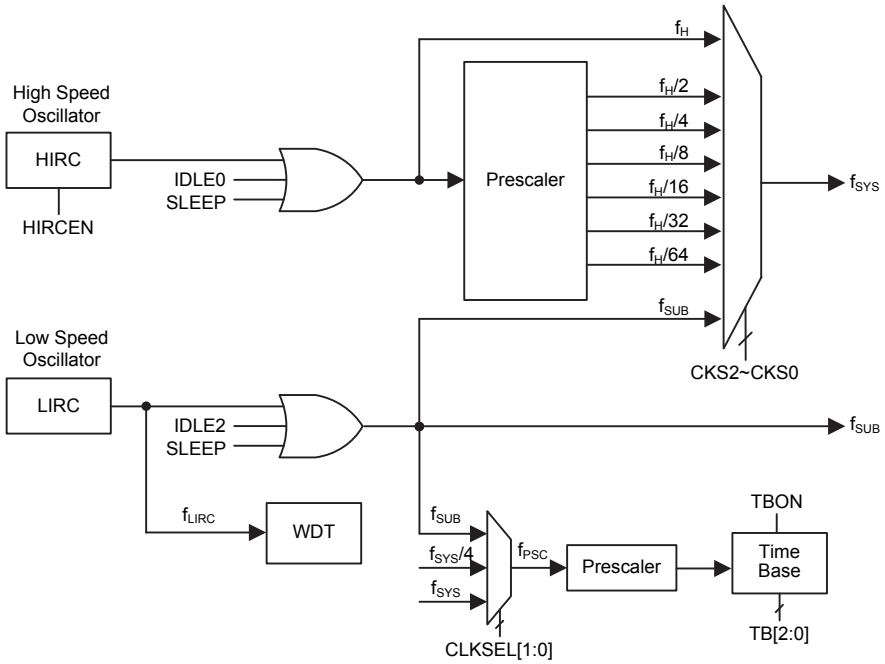
Operating Modes and System Clocks

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 Holtek has provided the device with both high and low speed clock sources and 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 many 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 HIRC oscillator. The low speed system clock source can be sourced from the 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 NORMAL 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				
NORMAL Mode	On	x	x	000~110	f_H~f_H/64	On	On	On
SLOW Mode	On	x	x	111	f_{SUB}	On/Off(1)	On	On
IDLE0 Mode	Off	0	1	000~110	Off	Off	On	On
				111	On			
IDLE1 Mode	Off	1	1	xxx	On	On	On	On
IDLE2 Mode	Off	1	0	000~110	On	On	Off	On
				111	Off			
SLEEP Mode	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.

NORMAL Mode

As the name suggests 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 the high speed oscillator. This mode operates allowing the microcontroller to operate normally with a clock source will come from the high speed oscillator, HIRC. 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 LIRC oscillator.

SLEEP Mode

The SLEEP Mode is entered when an HALT instruction is executed and when the FHIDEN and FSIDEN bit are low. In the SLEEP mode the CPU will be stopped. The f_{SUB} clock provided to the peripheral function will also be stopped, too. 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 an 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 an 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 an 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 Register

The SCC and HIRCC registers 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	—	—	—	FHIDEN	FSIDEN
HIRCC	—	—	—	—	—	—	HIRCF	HIRCEN

System Operating Mode Control Register List

• **SCC Register**

Bit	7	6	5	4	3	2	1	0
Name	CKS2	CKS1	CKS0	—	—	—	FHIDEN	FSIDEN
R/W	R/W	R/W	R/W	—	—	—	R/W	R/W
POR	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 4~2 Unimplemented, read as "0"

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	—	—	—	—	—	—	HIRCF	HIRCEN
R/W	—	—	—	—	—	—	R	R/W
POR	—	—	—	—	—	—	0	1

Bit 7~2 Unimplemented, read as "0"

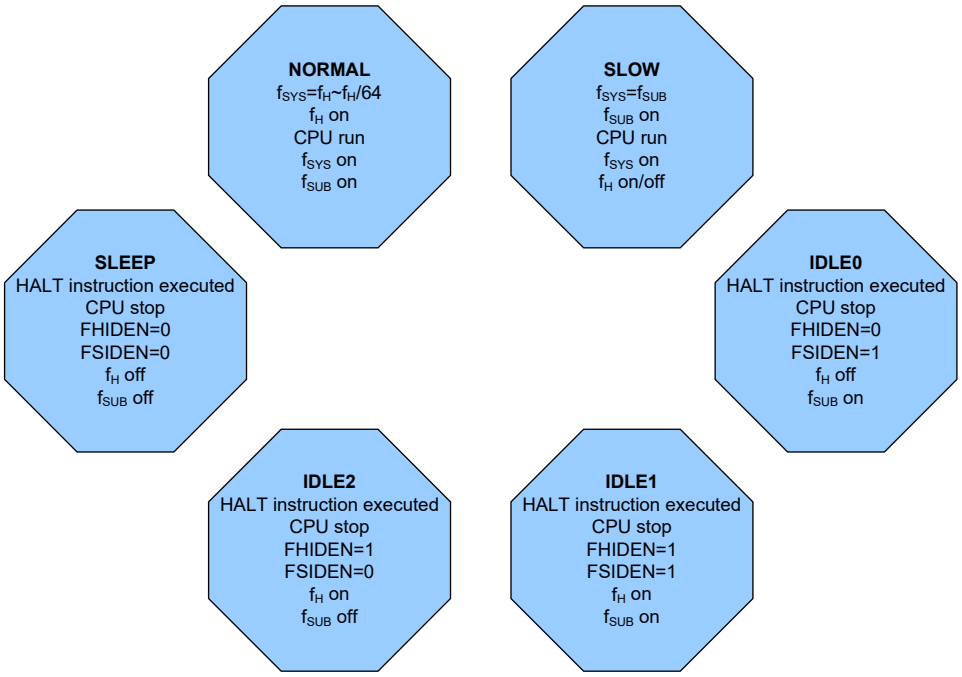
Bit 1 **HIRCF**: HIRC oscillator stable flag
0: Unstable
1: Stable
This bit is used to indicate whether the HIRC oscillator is stable or not. When the HIRCEN bit is set to 1 to enable the HIRC oscillator, the HIRCF bit will first be cleared to 0 and then set to 1 after the HIRC oscillator is stable.

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

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.

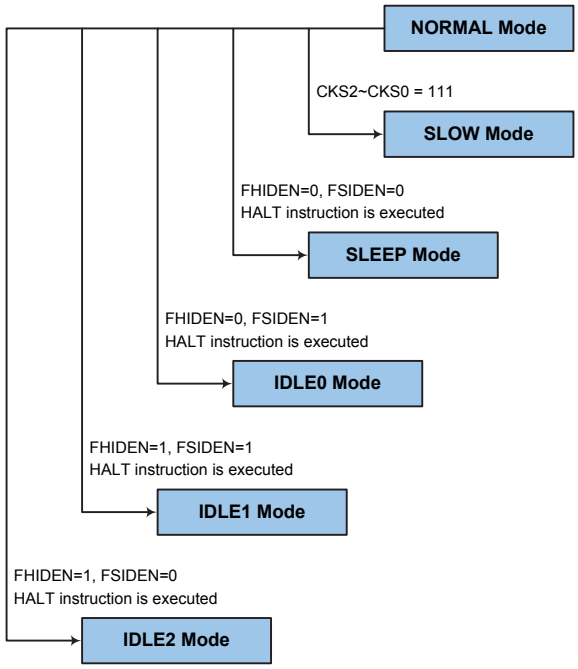
In simple terms, mode switching between the NORMAL Mode and SLOW Mode is executed using the CKS2~CKS0 bits in the SCC register while mode switching from the NORMAL/SLOW Modes to the SLEEP/IDLE Modes is executed via the HALT instruction. When an HALT instruction is executed, whether the device enters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



NORMAL Mode to SLOW Mode Switching

When running in the NORMAL 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.

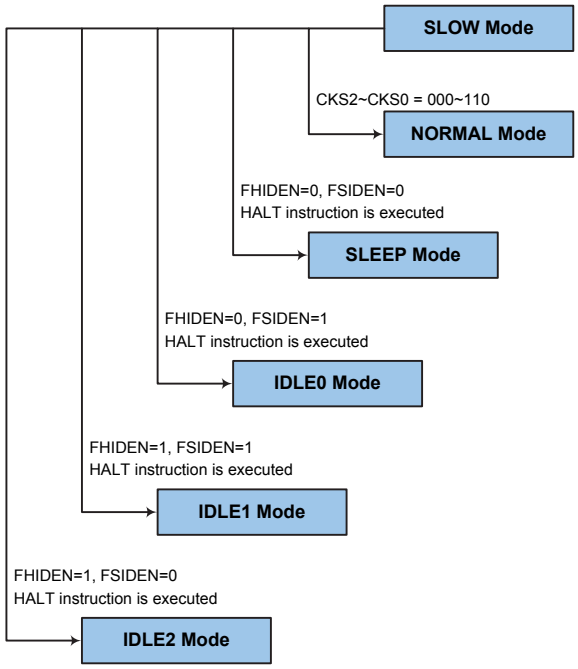
The SLOW Mode system clock is sourced from the LIRC oscillator and therefore requires this oscillator to be stable before full mode switching occurs.



SLOW Mode to NORMAL Mode Switching

In SLOW mode the system clock is derived from f_{SUB}. When system clock is switched back to the NORMAL 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 NORMAL mode from the SLOW Mode. This is monitored using the HIRCF bit in the HIRCC register. The time duration required for the high speed system oscillator stabilization is specified in the A.C. 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", 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 the Watchdog time-out flag, TO, will be cleared.
- The WDT will be cleared and resume counting if the WDT function is enabled. If the WDT function is disabled, the WDT will be cleared and then stopped.

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 the Watchdog time-out flag, TO, will be cleared.
- The WDT will be cleared and resume counting if the WDT function is enabled. If the WDT function is disabled, the WDT will be cleared and then stopped.

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 the Watchdog time-out flag, TO, will be cleared.
- The WDT will be cleared and resume counting if the WDT function is enabled. If the WDT function is disabled, the WDT will be cleared and then stopped.

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 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 the Watchdog time-out flag, TO, will be cleared.
- The WDT will be cleared and resume counting if the WDT function is enabled. If the WDT function is disabled, the WDT will be cleared and then stopped.

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. These must either be set as outputs or if set 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 set 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 IDLE2 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 system interrupt
- A WDT overflow

When the device executes the "HALT" instruction, it will enter the IDLE or SLEEP mode and the PDF flag will be set to "1". The PDF flag will be cleared to 0 if the device experiences a system power-up or executes the clear Watchdog Timer instruction. If the system is woken up by a WDT overflow, a Watchdog Timer reset will be initiated and the TO flag will be set to "1". 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 set 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 sourced from the LIRC oscillator. The LIRC internal oscillator has an approximate frequency of 32kHz and this specified internal clock period can vary with V_{DD}, temperature and process variations. The Watchdog Timer source clock is then subdivided by a ratio of 2^8 to 2^{15} 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 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
Other values: Reset MCU
When these bits are changed to any other values due to environmental noise the microcontroller will be reset; this 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: $2^9/f_{LIRC}$
010: $2^{10}/f_{LIRC}$
011: $2^{11}/f_{LIRC}$
100: $2^{12}/f_{LIRC}$
101: $2^{13}/f_{LIRC}$
110: $2^{14}/f_{LIRC}$
111: $2^{15}/f_{LIRC}$
These three bits determine the division ratio of the Watchdog Timer source clock, which in turn determines the timeout period.

• **RSTFC Register**

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

Bit 7~1 Unimplemented, read as "0"

Bit 0 **WRF**: WDT control register software reset flag
0: Not occurred
1: Occurred
This bit is set to 1 by the WDT control register software reset and cleared to zero 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 additional enable/disable 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. The WDT function will be enabled if the WE4~WE0 bits value is 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 the value of 01010B.

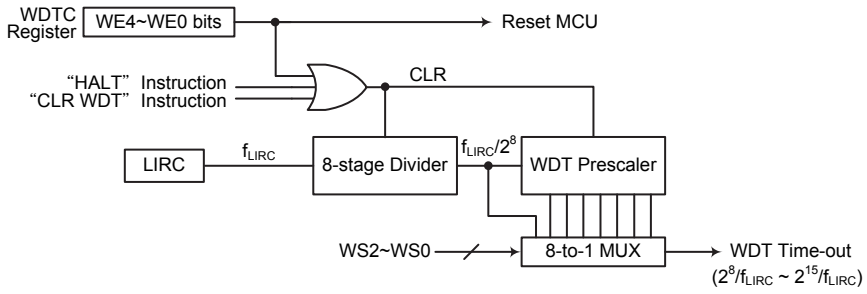
WE4~WE0 Bits	WDT Function
10101B	Disable
01010B	Enable
Any other value	Reset MCU

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. Three methods can be adopted to clear the contents of the Watchdog Timer. The first is a WDTC software 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.

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^{15} 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 1 seconds for the 2^{15} 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.

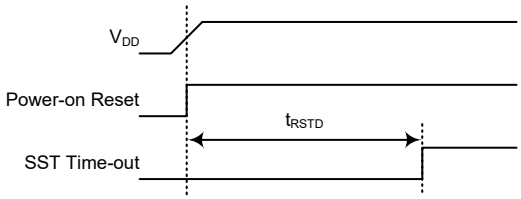
Another type of reset is when the Watchdog Timer overflows and resets the microcontroller. All types of reset operations result in different register conditions being set.

Reset Functions

There are several ways in which a microcontroller reset can occur, each of which will be described as follows.

Power-on Reset

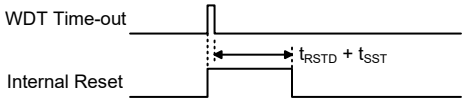
The most fundamental and unavoidable reset is the one that occurs after power is first applied to the microcontroller. As well as ensuring that the Program Memory begins execution from the first memory address, a power-on reset also ensures that certain other registers are preset to known conditions.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



Power-On Reset Timing Chart

Watchdog Time-out Reset during Normal Operation

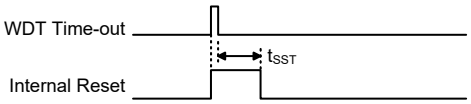
The Watchdog time-out Reset during normal operations in the NORMAL or SLOW mode is the same as a Power On reset except that the Watchdog time-out flag TO will be set to "1".



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 "0" and the TO flag will be set to "1". Refer to the A.C. Characteristics for t_{SST} details.



WDT Time-out Reset during SLEEP or IDLE 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
1	u	WDT time-out reset during NORMAL or SLOW Mode operation
1	1	WDT time-out reset during IDLE or SLEEP Mode operation

Note: "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 Base	Clear after reset, WDT begins counting
Timer Module	Timer Module will be turned off
Input/Output Ports	I/O ports will be set 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.

Register	Power On Reset	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
IAR0	xxxx xxxx	xxxx xxxx	xuuu uuuu
MP0	xxxx xxxx	xxxx xxxx	xuuu uuuu
IAR1	xxxx xxxx	xxxx xxxx	xuuu uuuu
MP1	xxxx xxxx	xxxx xxxx	xuuu uuuu
BP	---- ---0	---- ---0	---- ---u
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCL	0000 0000	0000 0000	0000 0000
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu
TBLH	xxxx xxxx	uuuu uuuu	uuuu uuuu
TBHP	---- -xxx	---- -uuu	---- -uuu
STATUS	--00 xxxx	--1u uuuu	--11 uuuu
SCC	000- --00	000- --00	uuu- --uu
HIRCC	---- --01	---- --01	---- --uu
WDTC	0101 0011	0101 0011	uuuu uuuu
RSTFC	---- ---0	---- ---u	---- ---u
INTC0	-000 0000	-000 0000	-uuu uuuu
INTC1	---0 ---0	---0 ---0	---u ---u
MFI	--00 --00	--00 --00	--uu --uu
PA	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	uuuu uuuu
PAPU	0000 0000	0000 0000	uuuu uuuu
PAWU	0000 0000	0000 0000	uuuu uuuu
PB	--11 1111	--11 1111	--uu uuuu
PBC	--11 1111	--11 1111	--uu uuuu
PBPU	--00 0000	--00 0000	--uu uuuu
PSCR	---- --00	---- --00	---- --uu
TBC	0--- -000	0--- -000	u--- -uuu
PAS0	0000 0000	0000 0000	uuuu uuuu
PAS1	0000 0000	0000 0000	uuuu uuuu
PBS0	0000 0000	0000 0000	uuuu uuuu
PBS1	---- 0000	---- 0000	---- uuuu
IFS	---- -000	---- -000	---- -uuu
CTMC0	0000 0000	0000 0000	uuuu uuuu

Register	Power On Reset	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
CTMC1	0000 0000	0000 0000	uuuu uuuu
CTMDL	0000 0000	0000 0000	uuuu uuuu
CTMDH	---- --00	---- --00	---- --uu
CTMAL	0000 0000	0000 0000	uuuu uuuu
CTMAH	---- --00	---- --00	---- --uu
COM_PWM	0-00 0000	0-00 0000	u-uu uuuu
PWM0DATA	0000 0000	0000 0000	uuuu uuuu
PWM1DATA	0000 0000	0000 0000	uuuu uuuu
PWM2DATA	0000 0000	0000 0000	uuuu uuuu
PWM3DATA	0000 0000	0000 0000	uuuu uuuu
PWM4DATA	0000 0000	0000 0000	uuuu uuuu
PWM5DATA	0000 0000	0000 0000	uuuu uuuu
PWM6DATA	0000 0000	0000 0000	uuuu uuuu
PWM7DATA	0000 0000	0000 0000	uuuu uuuu
PWM8DATA	0000 0000	0000 0000	uuuu uuuu
PWM9DATA	0000 0000	0000 0000	uuuu uuuu
PWM10DATA	0000 0000	0000 0000	uuuu uuuu
PWM11DATA	0000 0000	0000 0000	uuuu uuuu
CASCON	-100 0000	-100 0000	-uuu uuuu
CASPRE	---- -000	---- -000	---- -uuu
CASTH	---0 0111	---0 0111	---u uuuu
D0CNT	---0 0100	---0 0100	---u uuuu
D1CNT	---0 1010	---0 1010	---u uuuu
PCNT	0001 1000	0001 1000	uuuu uuuu
RCNT	1000 0000	1000 0000	uuuu uuuu
CASDNB	---- 0011	---- 0011	---- uuuu
CASD0	0000 0000	0000 0000	uuuu uuuu
CASD1	0000 0000	0000 0000	uuuu uuuu
CASD2	0000 0000	0000 0000	uuuu uuuu
CASD3	0000 0000	0000 0000	uuuu uuuu
CASD4	0000 0000	0000 0000	uuuu uuuu
CASD5	0000 0000	0000 0000	uuuu uuuu
CASD6	0000 0000	0000 0000	uuuu uuuu
CASD7	0000 0000	0000 0000	uuuu uuuu
CASD8	0000 0000	0000 0000	uuuu uuuu
CASD9	0000 0000	0000 0000	uuuu uuuu
CASD10	0000 0000	0000 0000	uuuu uuuu
CASD11	0000 0000	0000 0000	uuuu uuuu
CASD12	0000 0000	0000 0000	uuuu uuuu
CASD13	0000 0000	0000 0000	uuuu uuuu
CASD14	0000 0000	0000 0000	uuuu uuuu
CASD15	0000 0000	0000 0000	uuuu uuuu
CASD16	0000 0000	0000 0000	uuuu uuuu
CASD17	0000 0000	0000 0000	uuuu uuuu
CASD18	0000 0000	0000 0000	uuuu uuuu
CASD19	0000 0000	0000 0000	uuuu uuuu
CASD20	0000 0000	0000 0000	uuuu uuuu

Register	Power On Reset	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
CASD21	0000 0000	0000 0000	uuuu uuuu
CASD22	0000 0000	0000 0000	uuuu uuuu
CASD23	0000 0000	0000 0000	uuuu uuuu
CASD24	0000 0000	0000 0000	uuuu uuuu
CASD25	0000 0000	0000 0000	uuuu uuuu
CASD26	0000 0000	0000 0000	uuuu uuuu
CASD27	0000 0000	0000 0000	uuuu uuuu
CASD28	0000 0000	0000 0000	uuuu uuuu
CASD29	0000 0000	0000 0000	uuuu uuuu
CASD30	0000 0000	0000 0000	uuuu uuuu
CASD31	0000 0000	0000 0000	uuuu uuuu
CASD32	0000 0000	0000 0000	uuuu uuuu
CASD33	0000 0000	0000 0000	uuuu uuuu
CASD34	0000 0000	0000 0000	uuuu uuuu
CASD35	0000 0000	0000 0000	uuuu uuuu
INTCON0	0000 0000	0000 0000	uuuu uuuu
INTCON1	---0 ---0	---0 ---0	---u ---u
CCS	0010 --00	0010 --00	uuuu --uu
IICC0	---- 000-	---- 000-	---- uuu-
IICC1	1000 0001	1000 0001	uuuu uuuu
IICD	xxxx xxxx	xxxx xxxx	xuuu uuuu
IICA	0000 000-	0000 000-	uuuu uuu-
IICTOC	0000 0000	0000 0000	uuuu uuuu
CCOPU0	0000 0000	0000 0000	uuuu uuuu
CCOPU1	---- 0000	---- 0000	---- uuuu

Note: “u” stands for unchanged
“x” stands for unknown
“-” stands for unimplemented

Input/Output Ports

Holtek 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 name PA and PB. 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	—	—	PB5	PB4	PB3	PB2	PB1	PB0
PBC	—	—	PBC5	PBC4	PBC3	PBC2	PBC1	PBC0
PBPU	—	—	PBPU5	PBPU4	PBPU3	PBPU2	PBPU1	PBPU0

"—": Unimplemented, read as "0"

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 input have the capability of being connected to an internal pull-high resistor. These pull-high resistors are selected using the relevant pull-high control registers PAPU and PBPU, and are implemented using weak PMOS transistors.

Note that the pull-high resistor can be controlled by the relevant pull-high control registers 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 and B. 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 IDLE or SLEEP 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

Bit 7~0 **PAWU7~PAWU0**: PA7~PA0 wake-up function control
0: Disable
1: Enable

I/O Port Control Registers

Each I/O port has its own control register known as PAC~PBC, to control 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 set as a CMOS output. If the pin is currently set 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	PxC6	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 and B. However, the actual available bits for each I/O Port may be different.

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 IFS, which can select the desired functions of the multi-function pin-shared pins.

When the pin-shared input function is selected to be used, the corresponding input and output functions selection should be properly managed. For example, if the cascading transceiver interface is used, the corresponding pin-shared function should be configured as the cascading transceiver interface function by configuring the PxSn register and the cascading transceiver interface signal input pin should be properly selected using the IFS register.

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, a special point must be noted for some digital input pins, which share the same pin-shared control configuration with their corresponding general purpose I/O functions when setting the relevant pin-shared control bit fields. To select these pin functions, in addition to the necessary pin-shared control and peripheral functional setup aforementioned, they must also be set as an 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	PAS05	PAS04	PAS03	PAS02	PAS01	PAS00
PAS1	PAS17	PAS16	PAS15	PAS14	PAS13	PAS12	PAS11	PAS10
PBS0	PBS07	PBS06	PBS05	PBS04	PBS03	PBS02	PBS01	PBS00
PBS1	—	—	—	—	PBS13	PBS12	PBS11	PBS10
IFS	—	—	—	—	—	IFS2	IFS1	IFS0

Pin-shared Function Selection Register List

- **PAS0 Register**

Bit	7	6	5	4	3	2	1	0
Name	PAS07	PAS06	PAS05	PAS04	PAS03	PAS02	PAS01	PAS00
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 **PAS07~PAS06**: PA3 Pin-shared function selection
00: PA3
01: PA3
10: PA3
11: CCO1

Bit 5~4 **PAS05~PAS04**: PA2 Pin-shared function selection
00: PA2
01: COM3
10: SCL
11: CASDO

Bit 3~2 **PAS03~PAS02**: PA1 Pin-shared function selection
00: PA1
01: PA1
10: PA1
11: CCO0

Bit 1~0 **PAS01~PAS00**: PA0 Pin-shared function selection
00: PA0
01: PA0
10: SDA
11: CASDI

• **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: PA7
01: PA7
10: PA7
11: CCO5

Bit 5~4 **PAS15~PAS14**: PA6 Pin-shared function selection
00: PA6
01: PA6
10: PA6
11: CCO4

Bit 3~2 **PAS13~PAS12**: PA5 Pin-shared function selection
00: PA5
01: PA5
10: PA5
11: CCO3

Bit 1~0 **PAS11~PAS10**: PA4 Pin-shared function selection
00: PA4
01: PA4
10: PA4
11: CCO2

• **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: PB3
01: COM0
10: PB3
11: CCO9

Bit 5~4 **PBS05~PBS04**: PB2 Pin-shared function selection
00: PB2
01: PB2
10: PB2
11: CCO8

Bit 3~2 **PBS03~PBS02**: PB1 Pin-shared function selection
00: PB1
01: PB1
10: SDA
11: CCO7

Bit 1~0 **PBS01~PBS00**: PB0 Pin-shared function selection
00: PB0
01: PB0
10: SCL
11: CCO6

• **PBS1 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	PBS13	PBS12	PBS11	PBS10
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 **PBS13~PBS12**: PB5 Pin-shared function selection
00: PB5
01: COM2
10: CASDI
11: CCO11

Bit 1~0 **PBS11~PBS10**: PB4 Pin-shared function selection
00: PB4
01: COM1
10: CADO
11: CCO10

• **IFS Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	IFS2	IFS1	IFS0
R/W	—	—	—	—	—	R/W	R/W	R/W
POR	—	—	—	—	—	0	0	0

Bit 7~3 Unimplemented, read as "0"

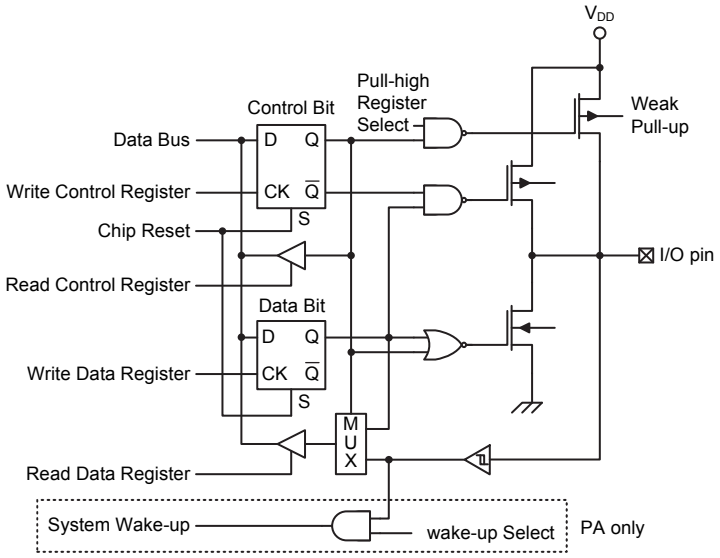
Bit 2 **IFS2**: SDA input source pin selection
0: PA0
1: PB1

Bit 1 **IFS1**: SCL input source pin selection
0: PA2
1: PB0

Bit 0 **IFS0**: CASDI input source pin selection
0: PA0
1: PB5

I/O Pin Structures

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



Logic Function Input/Output Structure

Programming Considerations

Within the user program, one of the first things to consider is port initialisation. After a reset, all of the I/O data and port control registers will be set high. This means that all I/O pins will default 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 set 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 set to have this function.

Timer Modules – TM

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

Introduction

The device contains one Compact type TM having a reference name of CTM. The general features to the Compact TM will be described in this section and the detailed operation will be described in the Compact type TM section. The main features of the CTM are summarised in the accompanying table.

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

TM Function Summary

TM Operation

The Compact type TM offers 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 counter whose value is then compared with the value of pre-programmed internal comparators. When the free running 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.

TM Clock Source

The clock source which drives the main counter in the TM can originate from various sources. The selection of the required clock source is implemented using the CTCK2~CTCK0 bits in the CTM control register. 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.

TM Interrupts

The Compact type TM has two internal interrupts, 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 signal.

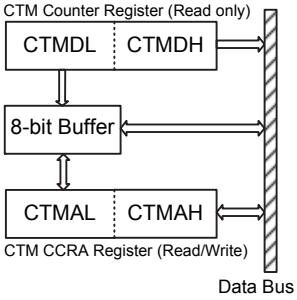
TM Output Signals

The TM has two output signals, CTP and CTPB. The CTPB is the inverted signal of the CTP output. When the TM is in the Compare Match Output Mode, these signals can be controlled by the TM to switch to a high or low level or to toggle when a compare match situation occurs. The PWM output waveforms are also reflected on CTP and CTPB.

Programming Considerations

The TM Counter Registers and the Capture/Compare CCRA register, being 10-bit, all have a low and high byte structure. The high byte can be directly accessed, but as the low byte can only be accessed via an internal 8-bit buffer, reading or writing to this register pair 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 register is implemented in the way shown in the following diagram and accessing these registers is carried out in a specific way described above, it is recommended to use the "MOV" instruction to access the CCRA low byte register, named CTMAL, using the following access procedures. Accessing the CCRA low byte register without following these access procedures will result in unpredictable values.

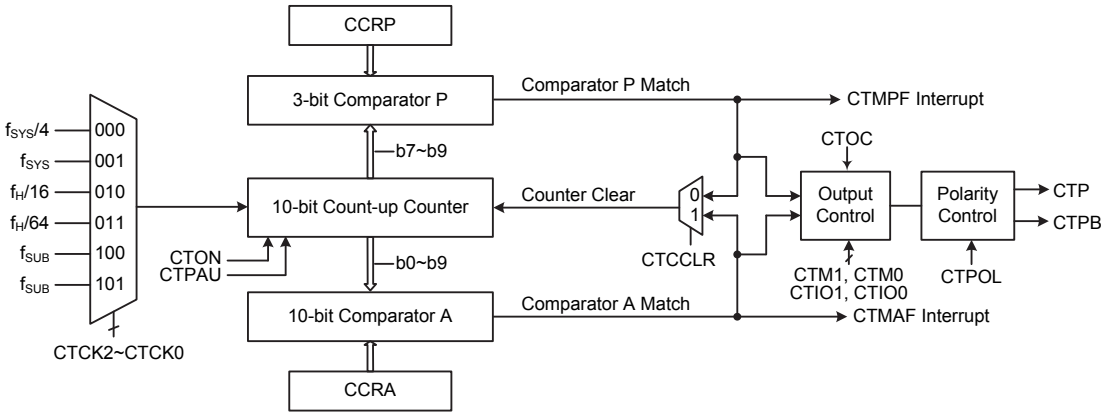


The following steps show the read and write procedures:

- Writing Data to CCRA
  - Step 1. Write data to Low Byte CTMAL
    - Note that here data is only written to the 8-bit buffer.
  - Step 2. Write data to High Byte CTMAH
    - 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 or CCRA
  - Step 1. Read data from the High Byte CTMDH or CTMAH
    - 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 CTMDL or CTMAL
    - This step reads data from the 8-bit buffer.

Compact Type TM – CTM

Although a simple TM type, the Compact TM type still contains three operating modes, which are Compare Match Output, Timer/Event Counter and PWM Output modes.



Note: CTPB is the inverted signal of CTP. Both CTP and CTPB are not bounded to the external package pins.

10-bit Compact Type TM Block Diagram

Compact Type TM Operation

At its core is a 10-bit count-up counter which is driven by a user selectable internal 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 is 3-bit wide whose value is compared with the highest three bits in the counter while the CCRA is 10-bit wide and therefore compares with all counter bits.

The only way of changing the value of the 10-bit counter using the application program, is to clear the counter by changing the CTON 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 CTM interrupt signal will also usually be generated. The Compact Type TM can operate in a number of different operational modes, can be driven by different clock sources. All operating setup conditions are selected using relevant internal registers.

Compact Type TM Register Description

Overall operation of the Compact TM is controlled using a series of registers. A read only register pair exists to store the internal counter 10-bit value, while a read/write register pair exists to store the internal 10-bit CCRA value. The remaining two registers are control registers which configure the different operating and control modes as well as the three CCRP bits.

Register Name	Bit							
	7	6	5	4	3	2	1	0
CTMC0	CTPAU	CTCK2	CTCK1	CTCK0	CTON	CTRP2	CTRP1	CTRP0
CTMC1	CTM1	CTM0	CTIO1	CTIO0	CTOC	CTPOL	CTDPX	CTCCLR
CTMDL	D7	D6	D5	D4	D3	D2	D1	D0
CTMDH	—	—	—	—	—	—	D9	D8
CTMAL	D7	D6	D5	D4	D3	D2	D1	D0
CTMAH	—	—	—	—	—	—	D9	D8

10-bit Compact TM Register List

• **CTMC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	CTPAU	CTCK2	CTCK1	CTCK0	CTON	CTRP2	CTRP1	CTRP0
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 **CTPAU**: CTM 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 CTM 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 **CTCK2~CTCK0**: CTM counter clock selection
000: $f_{SYS}/4$
001: f_{SYS}
010: $f_H/16$
011: $f_H/64$
100: f_{SUB}
101: f_{SUB}
110: Undefined, cannot be used
111: Undefined, cannot be used

These three bits are used to select the clock source for the CTM. 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 **CTON**: CTM counter on/off control
0: Off
1: On

This bit controls the overall on/off function of the CTM. Setting the bit high enables the counter to run, clearing the bit disables the CTM. Clearing this bit to zero will stop the counter from counting and turn off the CTM 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 CTM is in the Compare Match Output Mode or the PWM Output Mode then the CTM output will be reset to its initial condition, as specified by the CTOC bit, when the CTON bit changes from low to high.

Bit 2~0 **CTRP2~CTRP0**: CTM CCRP 3-bit register, compared with the CTM Counter bit 9 ~ bit 7
Comparator P Match Period=
000: 1024 CTM clocks
001: 128 CTM clocks
010: 256 CTM clocks
011: 384 CTM clocks
100: 512 CTM clocks
101: 640 CTM clocks
110: 768 CTM clocks
111: 896 CTM clocks

These three bits are used to set the value on the internal CCRP 3-bit register, which are then compared with the internal counter's highest three bits. The result of this comparison can be selected to clear the internal counter if the CTCCLR bit is set to zero. Setting the CTCCLR 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 three counter bits, the compare values exist in 128 clock cycle multiples. Clearing all three bits to zero is in effect allowing the counter to overflow at its maximum value.

• **CTMC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	CTM1	CTM0	CTIO1	CTIO0	CTOC	CTPOL	CTDPX	CTCCLR
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 **CTM1~CTM0**: CTM operating mode selection
00: Compare Match Output Mode
01: Undefined
10: PWM Output Mode
11: Timer/Counter Mode

These bits set the required operating mode for the CTM. To ensure reliable operation the CTM should be switched off before any changes are made to the CTM1 and CTM0 bits. In the Timer/Counter Mode, the CTM output state is undefined.

Bit 5~4 **CTIO1~CTIO0**: CTM output function selection

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

PWM Output Mode
00: PWM output inactive state
01: PWM output active state
10: PWM output
11: Undefined

Timer/Counter Mode
Unused

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

In the Compare Match Output Mode, the CTIO1 and CTIO0 bits determine how the CTM output changes state when a compare match occurs from the Comparator A. The CTM output can be set 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 CTM output should be configured using the CTOC bit in the CTMC1 register. Note that the output level requested by the CTIO1 and CTIO0 bits must be different from the initial value setup using the CTOC bit otherwise no change will occur on the CTM output when a compare match occurs. After the CTM output changes state, it can be reset to its initial level by changing the level of the CTON bit from low to high.

In the PWM Output Mode, the CTIO1 and CTIO0 bits determine how the CTM output 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 CTIO1 and CTIO0 bits only after the CTM has been switched off. Unpredictable PWM outputs will occur if the CTIO1 and CTIO0 bits are changed when the CTM is running.

Bit 3 **CTOC**: CTM CTP output control

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

PWM Output Mode
0: Active low
1: Active high

This is the output control bit for the CTM output. Its operation depends upon whether CTM is being used in the Compare Match Output Mode or in the PWM Output Mode.

It has no effect if the CTM is in the Timer/Counter Mode. In the Compare Match Output Mode it determines the logic level of the CTM output before a compare match occurs. In the PWM Output Mode it determines if the PWM signal is active high or active low.

Bit 2 **CTPOL**: CTM CTP output polarity control
0: Non-invert
1: Invert
This bit controls the polarity of the CTP output. When the bit is set high the CTM output will be inverted and not inverted when the bit is zero. It has no effect if the TM is in the Timer/Counter Mode.

Bit 1 **CTDPX**: CTM 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 **CTCCLR**: CTM Counter Clear condition selection
0: CTM Comparator P match
1: CTM Comparator A match
This bit is used to select the method which clears the counter. Remember that the Compact TM contains two comparators, Comparator A and Comparator P, either of which can be selected to clear the internal counter. With the CTCCLR 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 CTCCLR bit is not used in the PWM Output Mode.

• CTMDL 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 **D7~D0**: CTM Counter Low Byte Register bit 7 ~ bit 0
CTM 10-bit Counter bit 7 ~ bit 0

• CTMDH Register

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 **D9~D8**: CTM Counter High Byte Register bit 1 ~ bit 0
CTM 10-bit Counter bit 9 ~ bit 8

• CTMAL 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**: CTM CCRA Low Byte Register bit 7 ~ bit 0
CTM 10-bit CCRA bit 7 ~ bit 0

• **CTMAH Register**

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**: CTM CCRA High Byte Register bit 1 ~ bit 0
CTM 10-bit CCRA bit 9 ~ bit 8

Compact Type TM Operating Modes

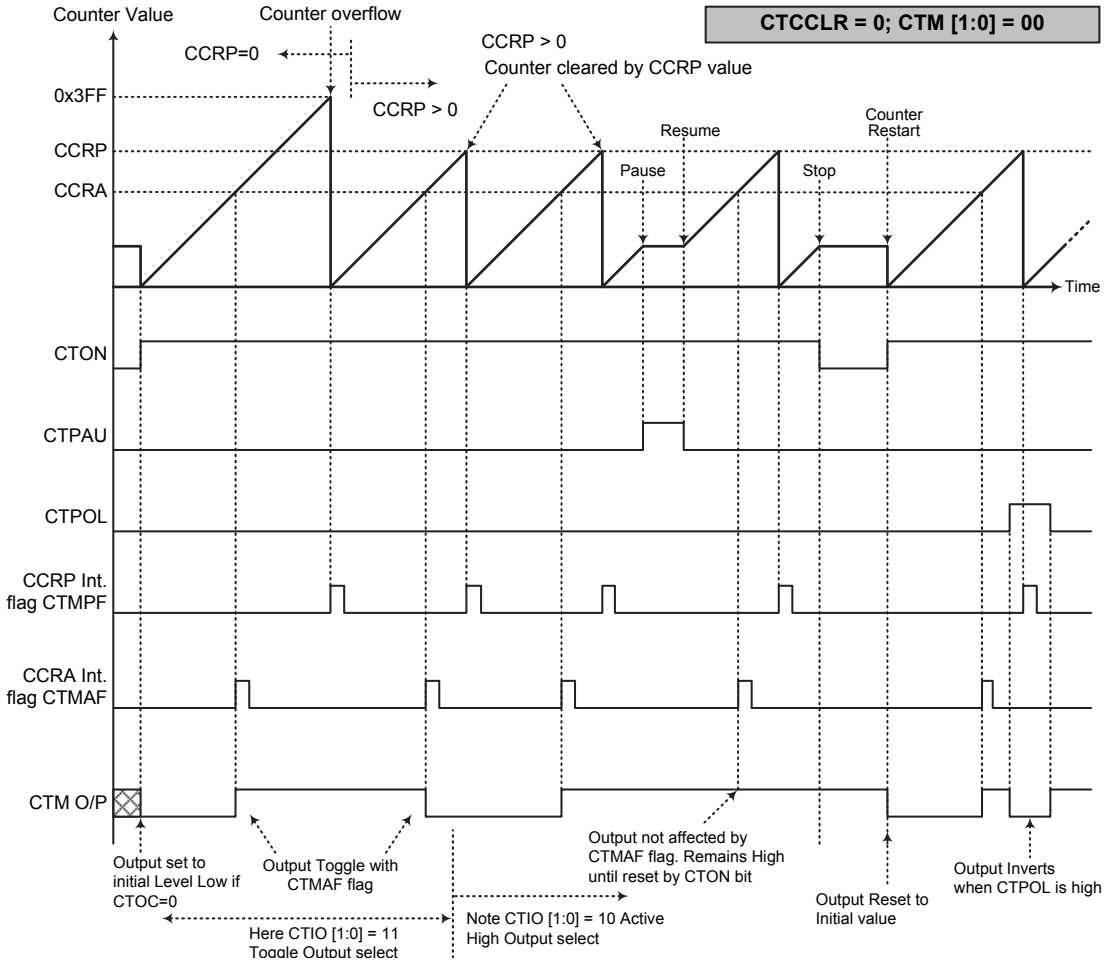
The Compact Type TM can operate in one of three operating modes, Compare Match Output Mode, PWM Output Mode or Timer/Counter Mode. The operating mode is selected using the CTM1 and CTM0 bits in the CTMC1 register.

Compare Match Output Mode

To select this mode, bits CTM1 and CTM0 in the CTMC1 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 CTCCLR bit is low, there are two ways in which the counter can be cleared. One is when a compare match occurs from Comparator P, the other is when the CCRP bits are all zero which allows the counter to overflow. Here both CTMAF and CTMPF interrupt request flags for the Comparator A and Comparator P respectively, will both be generated.

If the CTCCLR bit in the CTMC1 register is high then the counter will be cleared when a compare match occurs from Comparator A. However, here only the CTMAF interrupt request flag will be generated even if the value of the CCRP bits is less than that of the CCRA registers. Therefore when CTCCLR is high no CTMPF interrupt request flag will be generated. If the CCRA bits are all zero, the counter will overflow when it reaches its maximum 10-bit, 3FF Hex, value, however here the CTMAF interrupt request flag will not be generated.

As the name of the mode suggests, after a comparison is made, the CTM output will change state. The CTM output condition however only changes state when a CTMAF interrupt request flag is generated after a compare match occurs from Comparator A. The CTMPF interrupt request flag, generated from a compare match occurs from Comparator P, will have no effect on the CTM output. The way in which the CTM output changes state are determined by the condition of the CTIO1 and CTIO0 bits in the CTMC1 register. The CTM output can be selected using the CTIO1 and CTIO0 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 CTM output, which is configured after the CTON bit changes from low to high, is configured using the CTOC bit. Note that if the CTIO1 and CTIO0 bits are zero then no output change will take place.

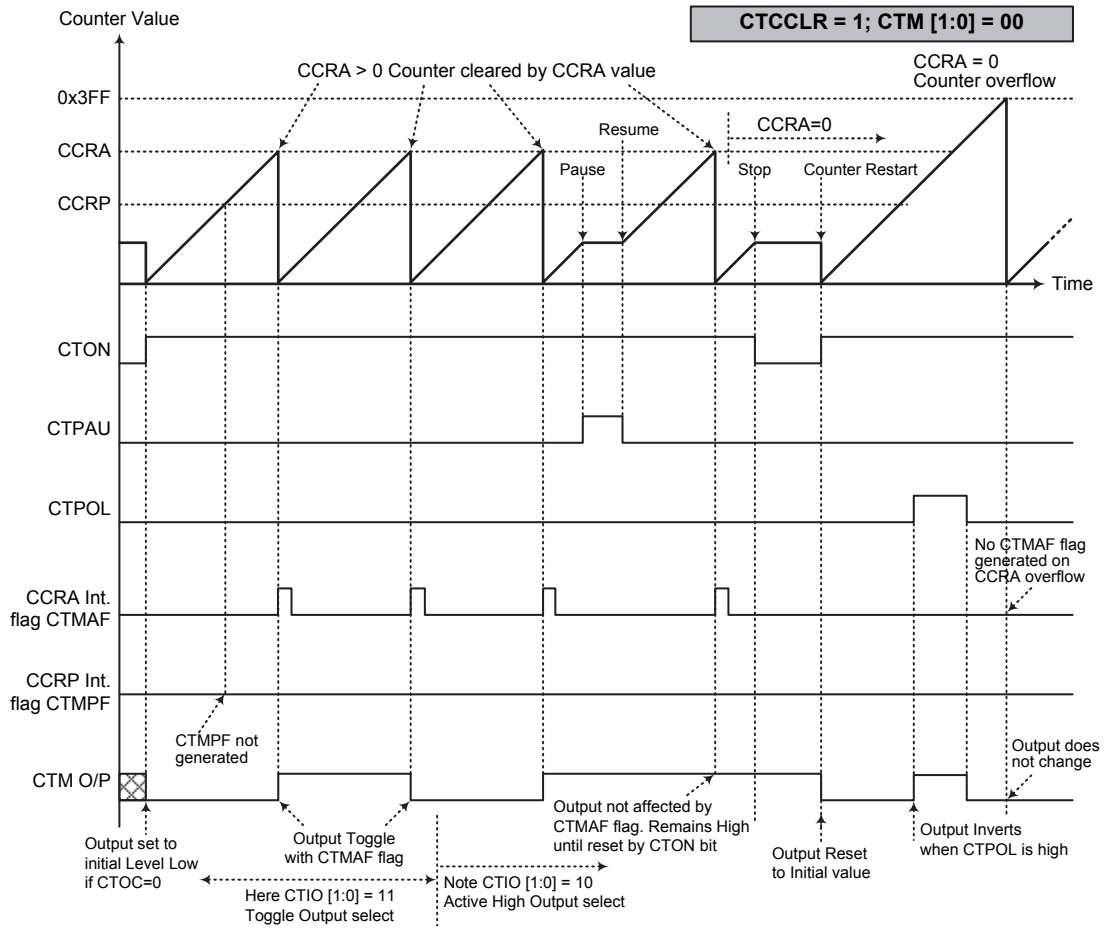


Compare Match Output Mode – CTCCLR=0

Note: 1. With CTCCLR=0, a Comparator P match will clear the counter

2. The CTM output is controlled only by the CTMAF flag

3. The output is reset to its initial state by a CTON bit rising edge



Compare Match Output Mode – CTCCLR=1

Note: 1. With CTCCLR=1, a Comparator A match will clear the counter

2. The CTM output is controlled only by the CTMAF flag
3. The output is reset to its initial state by a CTON bit rising edge
4. The CTMPF flag is not generated when CTCCLR=1

Timer/Counter Mode

To select this mode, bits CTM1 and CTM0 in the CTMC1 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 CTM output is not used. Therefore the above description and Timing Diagrams for the Compare Match Output Mode can be used to understand its function.

PWM Output Mode

To select this mode, bits CTM1 and CTM0 in the CTMC1 register should be set to "10" respectively. The PWM function within the CTM 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 CTM output, 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 CTCCLR bit has no effect on the PWM operation. 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 CTDPX bit in the CTMC1 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 CTOC bit in the CTMC1 register is used to select the required polarity of the PWM waveform while the two CTIO1 and CTIO0 bits are used to enable the PWM output or to force the CTM output to a fixed high or low level. The CTPOL bit is used to reverse the polarity of the PWM output waveform.

• 10-bit CTM, PWM Output Mode, Edge-aligned Mode, CTDPX=0

CCRP	001b	010b	011b	100b	101b	110b	111b	000b
Period	128	256	384	512	640	768	896	1024
Duty	CCRA							

If f_{SYS}=8MHz, CTM clock source is f_{SYS}/4, CCRP=100b, CCRA=128,

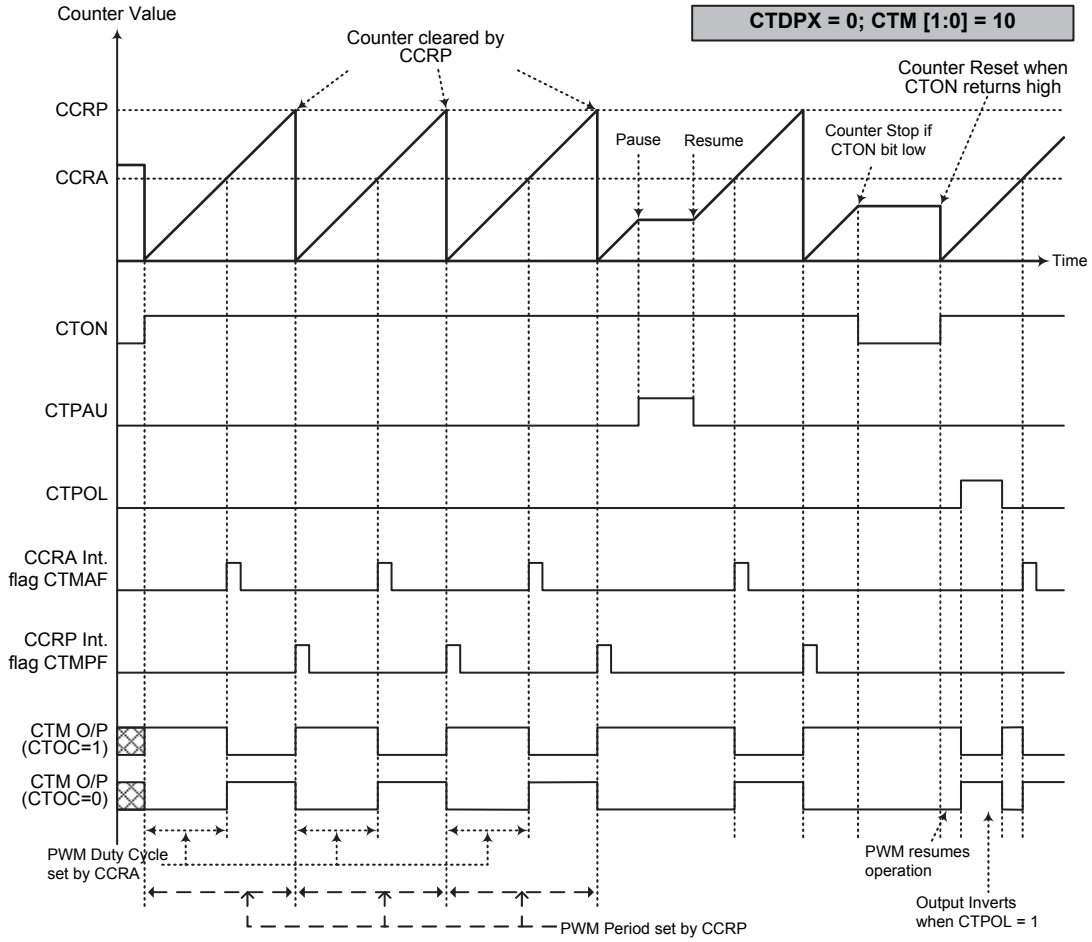
The CTM PWM output frequency=(f_{SYS}/4)/512=f_{SYS}/2048≈4kHz, duty=128/512=25%.

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%.

• 10-bit CTM, PWM Output Mode, Edge-aligned Mode, CTDPX=1

CCRP	001b	010b	011b	100b	101b	110b	111b	000b
Period	CCRA							
Duty	128	256	384	512	640	768	896	1024

The PWM output period is determined by the CCRA register value together with the CTM clock while the PWM duty cycle is defined by the CCRP register value.



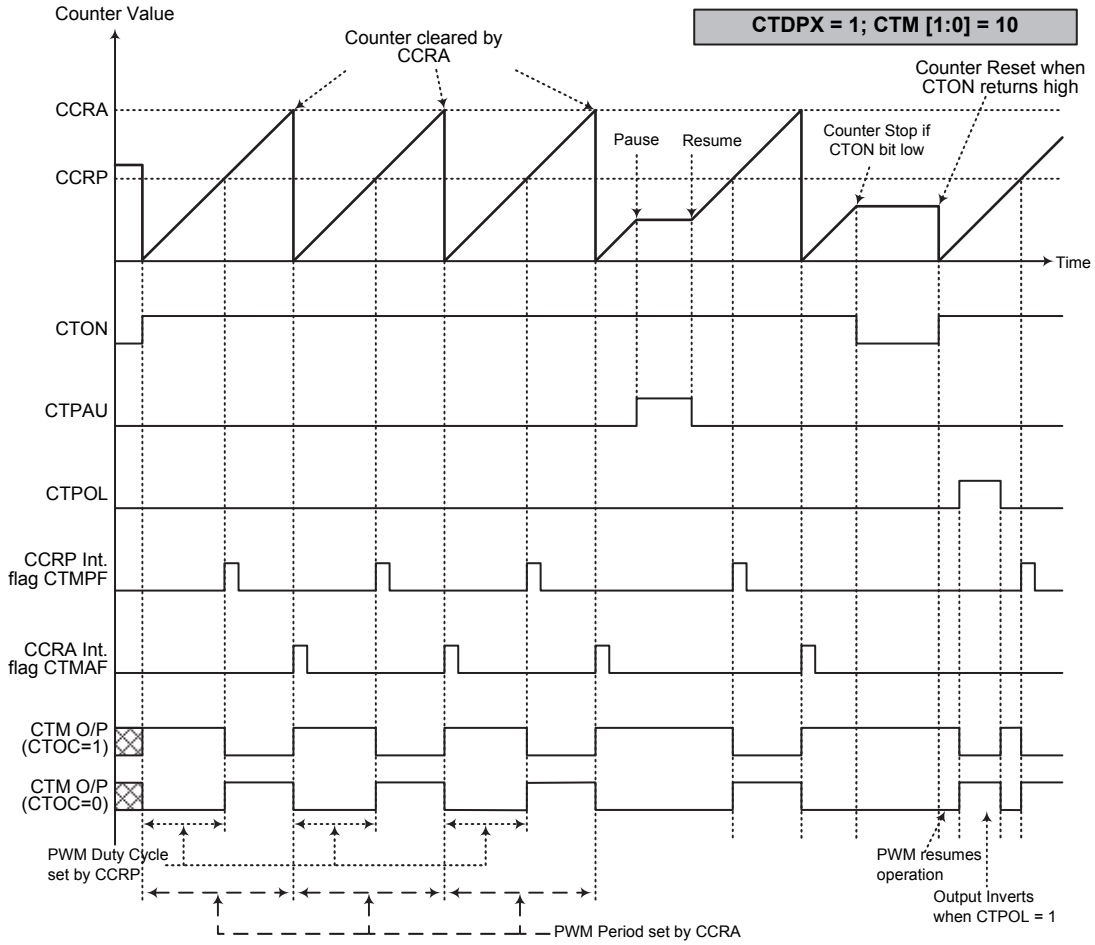
PWM Output Mode – CTDPX=0

Note: 1. Here CTDPX=0 – Counter cleared by CCRP

2. A counter clear sets the PWM Period

3. The internal PWM function continues even when CTIO [1:0]=00 or 01

4. The CTCCLR bit has no influence on PWM operation



PWM Output Mode – CTDPX=1

Note: 1. Here CTDPX=1 – Counter cleared by CCRA

2. A counter clear sets the PWM Period

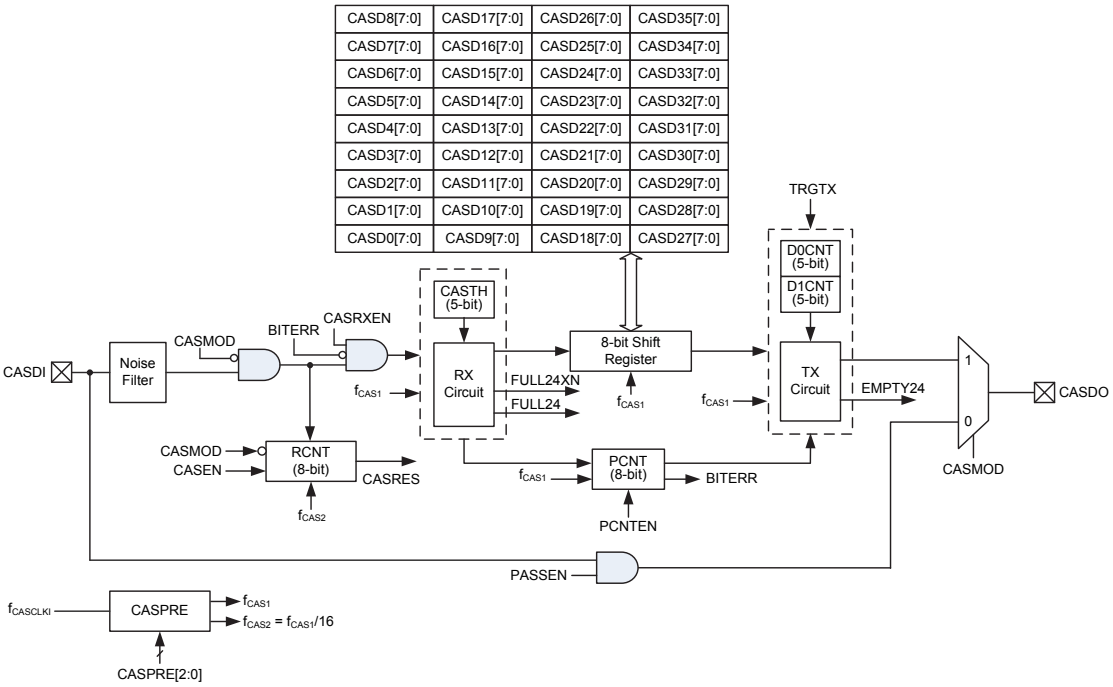
3. The internal PWM function continues even when CTIO [1:0]=00 or 01

4. The CTCCLR bit has no influence on PWM operation

Cascading Transceiver Interface

The cascade function is a feature of the LED tape light. It can be issued from a master MCU, and transfer PWM data for RGB color LED by single line cascading transceiver interface. The transfer rate is as soon as possible to make RGB LED change color smoothly. It should be noted that when the cascading transceiver is the master device, the TX function is active.

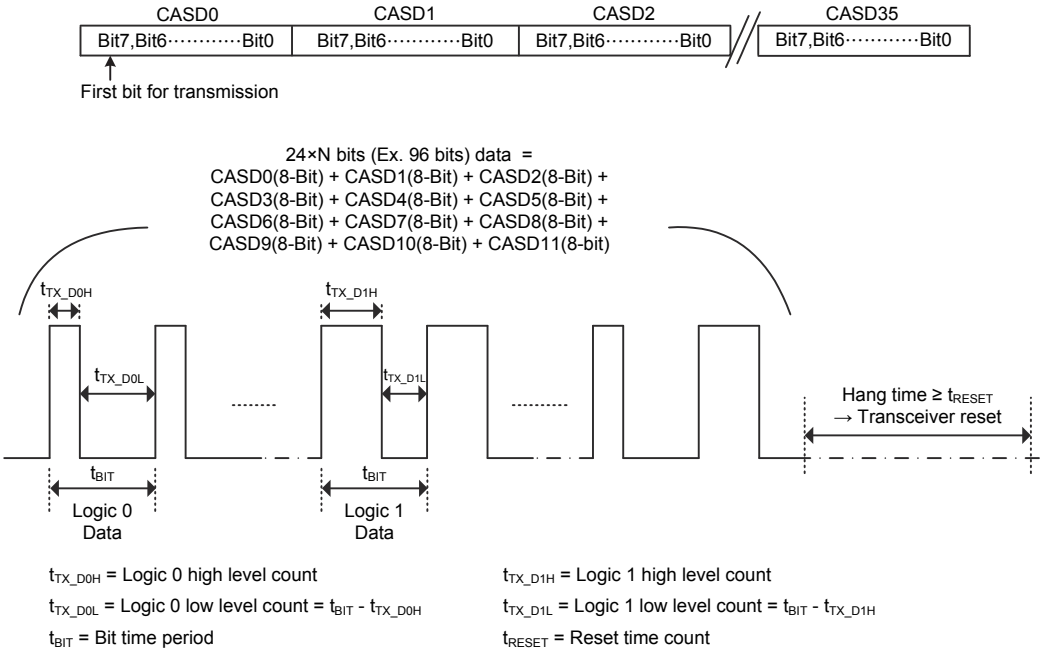
The Single Line Cascading Transceiver interface is a one-way transmission interface which contains an input CASDI pin and an output CASDO pin. Every 24 bits of data, once being received by the RX circuit, can be used to turn on a group of RGB LEDs. When the RX circuit has received full 24×N bits of data, where N ranges from 1 to 12 and is determined by application requirements and selected by the CASDNB register, the transceiver will enable the bypass path to the next device, and the next device will continue to read the following 24×N bits of data. The interception of 24×N bits of data in this sequence is called cascade function.



Note: 1. The cascade clock $f_{CASCLKI}$ is sourced from the system clock f_{SYS}.

2. TX buffer: CASD0~CASD2;
   RX buffer for direct driving: CASD0~CASD11;
   RX buffer for 3×COM scanning: CASD0~CASD26;
   RX buffer for 4×COM scanning: CASD0~CASD35.

Cascading Transceiver Interface Block Diagram



Single Line Cascading Transceiver Interface Data Sequence

Cascading Transceiver Interface Register Description

Overall operation of the Cascading Transceiver Interface is controlled using a series of registers. The CASCON, INTCON0 and INTCON1 registers are used for various cascading transceiver RX and TX function controls and interrupt control. The CASPRE register is used to select cascading transceiver clock. The CASTH register is used to specify the cascading transceiver RX function input data judgment threshold value. The D0CNT and D1CNT registers are used to control the cascading transceiver TX function logic 0 and logic 1 output data. The PCNT and RCNT registers are used to control the cascading transceiver data bit time period and reset time period. The CASDNB register is used to select the data length to be received in the RX mode. The CASD0~CASD35 data registers are used to store the data received or to be transmitted.

Register Name	Bit							
	7	6	5	4	3	2	1	0
CASCON	—	PCNTEN	CASRXEN	D4	TRGTX	PASSEN	CASMOD	CASEN
CASPRE	—	—	—	—	—	CASPRE2	CASPRE1	CASPRE0
CASTH	—	—	—	THS4	THS3	THS2	THS1	THS0
D0CNT	—	—	—	LCNT4	LCNT3	LCNT2	LCNT1	LCNT0
D1CNT	—	—	—	HCNT4	HCNT3	HCNT2	HCNT1	HCNT0
PCNT	PS7	PS6	PS5	PS4	PS3	PS2	PS1	PS0
RCNT	RS7	RS6	RS5	RS4	RS3	RS2	RS1	RS0
CASDNB	—	—	—	—	D3	D2	D1	D0
CASDn	D7	D6	D5	D4	D3	D2	D1	D0
INTCON0	BITERR	CASRES	EMPTY24	FULL24XN	BERINTEN	RESINTEN	EPTINTEN	FULINTEN
INTCON1	—	—	—	FULL24	—	—	—	INT24EN

Cascading Transceiver Interface Register List (n=0~35)

• **CASCON Register**

Bit	7	6	5	4	3	2	1	0
Name	—	PCNTEN	CASRXEN	D4	TRGTX	PASSEN	CASMOD	CASEN
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 Unimplemented, read as "0"

Bit 6 **PCNTEN**: Cascading transceiver RX/TX transfer bit time counter control
0: Disable
1: Enable

This bit is used to count the RX or TX data transfer bit time period. When the PCNTEN bit is cleared to 0, the bit time counter PCNT function will be disabled in the RX mode. It means that the bit time will not be checked and then the BITERR flag will always be 0. For the TX mode the PCNT is always enabled and the PCNTEN bit has no effect on the PCNT function.

When the PCNTEN bit is set to 1, the bit time counter PCNT function will be enabled. For the TX function the bit time counter is used to define the transmit bit time. For the RX function the bit time counter is used to define the maximum bit time. If the bit time counter underflows and no second rising edge appears on the CASDI line, the BITERR flag will be set to 1.

Bit 5 **CASRXEN**: Cascading transceiver RX function enable control
0: Disable
1: Enable

This bit is used to control the cascading transceiver RX function. When this bit is set to 1, the cascading transceiver RX function will be enabled. This bit will automatically be cleared to 0 when the RX shift register full flag, FULL24XN, is set high. Then it will automatically be set to 1 again when the cascading transceiver reset flag, CASRES, is set high. When this bit is set to 0, the cascading transceiver RX function will be disabled. However, the cascading transceiver reset signal can still be decoded and recognized as the RX function is disabled. Note that the PASSEN bit will automatically be cleared to 0 by hardware when the CASRXEN bit is set to 1 by hardware and vice versa.

Bit 4 **D4**: Reserved bit, should be fixed at "0"

Bit 3 **TRGTX**: Cascading transceiver TX output buffer trigger control
0: No action or data is transmitted completely
1: TX buffer output is triggered and data is transmitting continuously

This bit can only be written with a value of "1" when the CASEN bit is high. The TRGTX bit will be cleared to 0 by hardware when the TX shift register empty flag, EMPTY24, is set high. It will also be cleared to 0 when the CASEN bit is set low. Note that the EMPTY24 bit will be cleared to 0 by hardware when the TRGTX bit is set high by software regardless of the transceiver operation modes. Setting the TRGTX bit in the RX mode will have no operation.

Bit 2 **PASSEN**: Cascading transceiver input signal bypass RX circuit enable control
0: Disable – Not bypass the RX circuit
1: Enable – Bypass the RX circuit

This bit controls the cascading transceiver input signal bypass function. It will automatically be set and cleared by hardware. When the CASRXEN bit is set to 1 by hardware to enable the cascading transceiver RX function, the PASSEN bit will automatically be set to 0 by hardware and the cascading transceiver input signal will be decoded by the RX circuit. When the CASRXEN bit is cleared to 0 by hardware to disable the cascading transceiver RX function, the PASSEN bit will automatically be set to 1 by hardware. At the same time the cascading transceiver input signal will bypass the RX circuit and directly be connected to the CASDO line.

Bit 1 **CASMOD**: Cascading transceiver TX or RX mode selection
0: RX mode
1: TX mode
This bit can only be modified when the CASEN bit is set low. The cascading transceiver operating mode should first be selected followed by setting the CASEN bit high.

Bit 0 **CASEN**: Cascading transceiver enable control
0: Disable
1: Enable
This bit is used to enable or disable the cascading transceiver function. When it is cleared to 0 to disable the cascading transceiver function, only the internal control circuit and corresponding read-only flags in the INTCON0 and INTCON1 registers together with the TRGTX bit will be reset. Other registers contents will be kept unchanged. Note that in the RX mode the contents of the CASD0~CASD35 registers will be unknown when the CASEN bit is set low. When the CASEN bit is set low, the CASDI input path will be switched off and the CASDO output will be floating.

• CASPRE Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	CASPRE2	CASPRE1	CASPRE0
R/W	—	—	—	—	—	R/W	R/W	R/W
POR	—	—	—	—	—	0	0	0

Bit 7~3 Unimplemented, read as "0"

Bit 2~0 **CASPRE2~CASPRE0**: Cascading transceiver clock f_{CAS1} division ratio selection
000: $f_{CAS1}=f_{CASCLKI}$
001: $f_{CAS1}=f_{CASCLKI}/2$
010: $f_{CAS1}=f_{CASCLKI}/4$
011: $f_{CAS1}=f_{CASCLKI}/8$
100: $f_{CAS1}=f_{CASCLKI}/16$
101: $f_{CAS1}=f_{CASCLKI}/32$
110: $f_{CAS1}=f_{CASCLKI}/64$
111: $f_{CAS1}=f_{CASCLKI}/128$
These bits are used to select the cascading transceiver clock f_{CAS1} division ratio. The $f_{CASCLKI}$ clock is the cascading transceiver input clock which is sourced from the system clock f_{SYS}. The f_{CAS1} clock is used to drive the whole cascading transceiver circuits except the reset time counter, RCNT. The reset time counter, RCNT, is driven by the clock, f_{CAS2}, where $f_{CAS2}=f_{CAS1}/16$.

• CASTH Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	THS4	THS3	THS2	THS1	THS0
R/W	—	—	—	R/W	R/W	R/W	R/W	R/W
POR	—	—	—	0	0	1	1	1

Bit 7~5 Unimplemented, read as "0"

Bit 4~0 **THS4~THS0**: Cascading transceiver RX function data judgment threshold
Logic 0 threshold: $t_{RX\_DT0HS} = (THS[4:0] - 2) \times t_{CAS1}$,
Logic 1 threshold: $t_{RX\_DT1HS} = (THS[4:0] + 1) \times t_{CAS1}$,
where $t_{CAS1} = 1/f_{CAS1}$
These bits are used to specify the cascading transceiver RX function input data judgment threshold value. When the input signal high level period is equal to or greater than the t_{RX_DT1HS} threshold, the input data will be recognized as a logic 1. However, the input data will be recognized as a logic 0 if the input signal high level period is equal to or less than the t_{RX_DT0HS} threshold. The received data will be stored in the CASD0~CASD35 registers.

• **D0CNT Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	LCNT4	LCNT3	LCNT2	LCNT1	LCNT0
R/W	—	—	—	R/W	R/W	R/W	R/W	R/W
POR	—	—	—	0	0	1	0	0

Bit 7~5 Unimplemented, read as "0"

Bit 4~0 **LCNT4~LCNT0**: Cascading transceiver TX function output data logic 0 high pulse count value, t_{TX_D0H}

t_{TX_D0H} = LCNT[4:0] × t_{CAS1}, where t_{CAS1} = 1/f_{CAS1}

These bits are used to specify the high pulse count value of the cascading transceiver TX function logic 0 output data, which is driven by the f_{CAS1} clock. When the transmitted data stored in the CASDn register is logic 0, a signal with a high pulse width of t_{TX_D0H} and a low pulse width of (t_{BIT} – t_{TX_D0H}) will be output on the CASDO line, where the bit time t_{BIT} is specified by the bit time counter PCNT. The LCNT field minimum value should be properly configured according to the corresponding cascading transceiver input clock frequency, $f_{CASCLKI}$, for applications.

• **D1CNT Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	HCNT4	HCNT3	HCNT2	HCNT1	HCNT0
R/W	—	—	—	R/W	R/W	R/W	R/W	R/W
POR	—	—	—	0	1	0	1	0

Bit 7~5 Unimplemented, read as "0"

Bit 4~0 **HCNT4~HCNT0**: Cascading transceiver TX function output data logic 1 high pulse count value, t_{TX_D1H}

t_{TX_D1H} = HCNT[4:0] × t_{CAS1}, where t_{CAS1} = 1/f_{CAS1}

These bits are used to specify the high pulse count value of the cascading transceiver TX function logic 1 output data, which is driven by the f_{CAS1} clock. When the transmitted data stored in the CASDn register is logic 1, a signal with a high pulse width of t_{TX_D1H} and a low pulse width of (t_{BIT} – t_{TX_D1H}) will be output on the CASDO line, where the bit time t_{BIT} is specified by the bit time counter PCNT. The HCNT field minimum value should be properly configured according to the corresponding cascading transceiver input clock frequency, $f_{CASCLKI}$, for applications.

• **PCNT Register**

Bit	7	6	5	4	3	2	1	0
Name	PS7	PS6	PS5	PS4	PS3	PS2	PS1	PS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	1	1	0	0	0

Bit 7~0 **PS7~PS0**: Cascading transceiver bit time counter

TX mode: t_{BIT} = PS[7:0] × t_{CAS1},

RX mode: t_{BIT} < (PS[7:0] – 1) × t_{CAS1},

where t_{CAS1} = 1/f_{CAS1}

This counter is used to specify the count value of the cascading transceiver data bit time period. The bit time counter is driven by the f_{CAS1} clock. For the TX function the bit time counter is always enabled regardless of the PCNTEN bit status. In the TX mode the bit time is calculated as the above formula shown. When the TRGTX bit is set to 1 in the TX mode, the PCNT and D0CNT or D1CNT counters will start to count. A signal with a high pulse of t_{TX_D0H} or t_{TX_D1H} and a low pulse of (t_{BIT} – t_{TX_D0H}) or (t_{BIT} – t_{TX_D1H}) representing a logic data 0 or 1 respectively will be output on the CASDO line.

For the RX function the bit time counter is also used to check whether the error condition on the received data occurs or not other than to specify the bit time. When the PCNT function is enabled by setting the PCNTEN bit high and there is a rising edge on the CASDI line, the bit time counter PCNT will start to count down. If there is no second rising edge on the CASDI line before the PCNT counter counts down to zero for the received data bit 0 ~ bit (24×N-2), the bit error flag, BITERR, will automatically be set to 1, which means a bit transfer error occurs. For the data bit (24×N-1) reception if a falling edge appears on the CASDI line before the PCNT counter counts down to zero, the RX shift register full flags, FULL24XN and FULL24, will be set to 1. It means that the whole 24×N bits of data has been completely received. Then the PASSEN bit will automatically be set to 1 by hardware. If there is no falling edge on the CASDI line when receiving the data bit (24×N-1) before the PCNT counter counts down to zero, the bit error flag, BITERR, will also be set to 1 by hardware to indicate that a bit transfer error occurs. In this case the FULL24XN and FULL24 flags are cleared. The PASSEN bit will then be kept unchanged with a low level state.

• RCNT Register

Bit	7	6	5	4	3	2	1	0
Name	RS7	RS6	RS5	RS4	RS3	RS2	RS1	RS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	0	0	0	0	0	0	0

Bit 7~0 **RS7~RS0**: Cascading transceiver reset time counter

t_{RESET} = RS[7:0] × t_{CAS2}, where $t_{CAS2} = 1/f_{CAS2} = 16/f_{CAS1} = 16 \times t_{CAS1}$

This down-counter is used to specify the count value of the cascading transceiver reset time period in the RX function. The reset time counter is driven by the f_{CAS2} clock where the f_{CAS2} clock is equal to the f_{CAS1} clock divided by 16. When the CASMOD bit is set to 0 to select the RX mode and the CASEN bit is set to 1 to enable the cascading transceiver function, the RCNT counter will start to count. If the CASDI line signal is kept at a high or low level for a certain time period and the RCNT counts down to zero, the cascading transceiver reset flag, CASRES, will be set to 1 to indicate that a cascading transceiver reset condition occurs. When the CASRES bit is set to 1, the BITERR, FULL24XN, FULL24, PASSEN bits will be cleared to 0 and the CASRXEN bit will be set to 1.

• CASDNB Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	D3	D2	D1	D0
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	0	1	1

Bit 7~4 Unimplemented, read as "0"

Bit 3~0 **D3~D0**: 24×N bits for cascading, RX mode only

0000: N=1
0001: N=2
0010: N=3
0011: N=4
0100: N=5
0101: N=6
0110: N=7
0111: N=8
1000: N=9
1001: N=10
1010: N=11
1011~1111: N=12

• **CASDn Register (n=0~2)**

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**: Data byte n

This register is used to store the data byte n received in the RX mode or to be transmitted in the TX mode.

• **CASDn Register (n=3~35)**

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**: Data byte n

This register is used to store the data byte n received in the RX mode.

• **INTCON0 Register**

Bit	7	6	5	4	3	2	1	0
Name	BITERR	CASRES	EMPTY24	FULL24XN	BERINTEN	RESINTEN	EPTINTEN	FULINTEN
R/W	R	R	R	R	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **BITERR**: RX received data bit time-out flag
0: No bit time-out error condition occurs
1: Bit time-out error condition occurs

The bit is used to indicate whether a received bit time-out condition occurs or not. When a received data bit time is greater than the t_{BIT} time specified by the PCNT register, the BITERR bit will be set to 1 by hardware to indicate that a received bit time-out error condition occurs. When the bit time-out error condition occurs, the RX input data will not be decoded until the CASRES bit is set high which means that a cascading transceiver reset condition occurs. If the CASRES bit is set high, the BITERR bit will automatically be cleared to 0 by hardware.

Bit 6 **CASRES**: Cascading transceiver reset flag
0: No reset condition occurs
1: Reset condition occurs

The bit is used to indicate whether a cascading transceiver reset condition occurs or not. If the CASDI line signal is kept unchanged at a high or low level for a certain time period greater than the t_{RESET} time specified by the RCNT register, the CASRES bit will be set to 1 by hardware to indicate that a RX reset condition occurs. When the CASRES bit is set high, the BITERR, FULL24XN, FULL24 and PASSEN bits will automatically be cleared to 0 and the CASRXEN bit will be set high by hardware. The CASRES bit will be cleared to 0 if a rising edge signal on the CASDI line appears.

Bit 5 **EMPTY24**: Cascading transceiver 24-bit TX shift register empty flag
0: TX shift register is not empty
1: TX shift register is empty

The bit is used to indicate whether the cascading transceiver 24-bit TX shift register is empty or not. When the TX circuit transmits 24 bits of data completely, the 24-bit shift register will be empty and the EMPTY24 bit will be set to 1. If the EMPTY24 bit is set high, the TRGTX bit will automatically be cleared to 0 by hardware. When the TRGTX bit is set high to initiate a transmission, the EMPTY24 bit will automatically be cleared to 0.

Bit 4 **FULL24XN**: Cascading transceiver 24×N bits RX shift register full flag
0: 24×N bits RX shift register is not full
1: 24×N bits RX shift register is full
The bit is used to indicate whether the cascading transceiver 24×N bits RX shift register is full or not. When the RX circuit receives 24×N bits of data completely, the 24×N bits shift register will be full and the FULL24XN bit will be set to 1 with the BITERR bit cleared. If the FULL24XN bit is set high, the PASSEN bit will be set to 1 and CASRXEN bit will be cleared to 0 by hardware. This makes that the CASDI signal is directly output to the CASDO line bypassing the cascading transceiver. The FULL24XN bit will automatically be cleared to 0 when the CASRES bit is set high.

Bit 3 **BERINTEN**: RX received data bit error interrupt control
0: Disable
1: Enable
The bit is used to control the RX received data bit error interrupt function. When the BITERR bit is set high as the BERINTEN bit is set high, the cascading transceiver will generate a bit error interrupt signal to inform the microcontroller.

Bit 2 **RESINTEN**: Cascading transceiver reset interrupt control
0: Disable
1: Enable
The bit is used to control the cascading transceiver reset interrupt function. When the CASRES bit is set high as the RESINTEN bit is set high, the cascading transceiver will generate a reset interrupt signal to inform the microcontroller.

Bit 1 **EPTINTEN**: Cascading transceiver 24-bit TX shift register empty interrupt control
0: Disable
1: Enable
The bit is used to control the cascading transceiver TX shift register empty interrupt function. When the EMPTY24 bit is set high as the EPTINTEN bit is set high, the cascading transceiver will generate a TX shift register empty interrupt signal to inform the microcontroller.

Bit 0 **FULINTEN**: Cascading transceiver 24×N bits RX shift register full interrupt control
0: Disable
1: Enable
The bit is used to control the cascading transceiver 24×N bits RX shift register full interrupt function. When the FULL24XN bit is set high as the FULINTEN bit is set high, the cascading transceiver will generate a 24×N bits RX shift register full interrupt signal to inform the microcontroller.

• INTCON1 Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	FULL24	—	—	—	INT24EN
R/W	—	—	—	R/W	—	—	—	R/W
POR	—	—	—	0	—	—	—	0

Bit 7~5 Unimplemented, read as “0”

Bit 4 **FULL24**: Cascading transceiver 24-bit RX shift register full flag
0: 24-bit RX shift register is not full
1: 24-bit RX shift register is full
The bit is used to indicate whether the cascading transceiver 24-bit RX shift register is full or not. When the RX circuit has received 24 bits of data completely, the 24-bit shift register will be full and the FULL24 bit will be set to 1 by hardware. This flag should be cleared to zero by software if more data are required to be received. The FULL24 bit will automatically be cleared to 0 when the CASRES bit is set high. Note that this flag cannot be set high by software.

Bit 3~1 Unimplemented, read as “0”

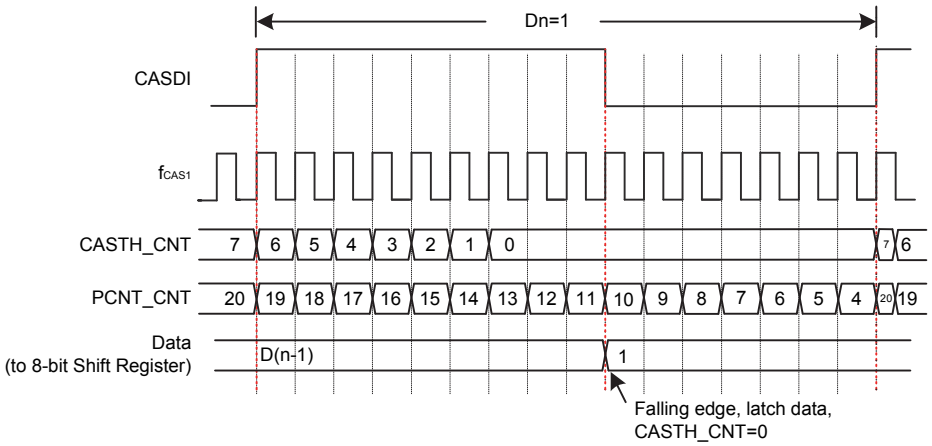
Bit 0 **INT24EN**: RX received 24-bit data interrupt control
0: Disable
1: Enable
The bit is used to control the cascading transceiver 24-bit RX shift register full interrupt function. When the FULL24 bit is set high as the INT24EN bit is set high, the cascading transceiver will generate a 24-bit RX shift register full interrupt signal to inform the microcontroller.

Cascade RX Function Operation

The cascade RX function is used to decode the pulse from the CASDI line to be logic high or logic low.

Cascade Function Logic High

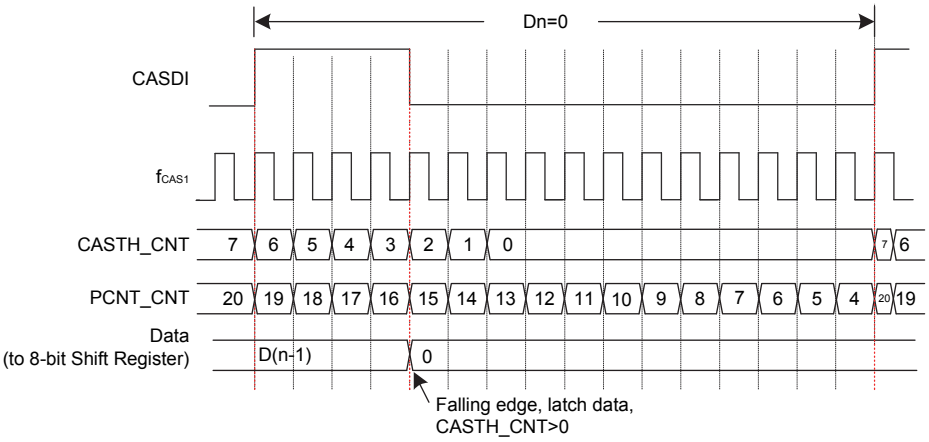
If the cascading transceiver clock high level count value is greater than or equal to (CASTH+1), it means logic high.



Data=1 in RX Mode (Ex. CASTH=7)

Cascade Function Logic Low

If the cascading transceiver clock high level count value is less than or equal to (CASTH–2), it means logic low.



Data=0 in RX Mode (Ex. CASTH=7)

Cascade RX Procedure

Before the cascade function is carried out, if the CASDI line signal is kept at a high or low level for a certain time period and the RCNT counts down to zero, the cascading transceiver reset flag, CASRES, will be set to 1 to indicate that a cascading transceiver reset condition occurs.

If the CASRES bit is set high, the bus for bypass (MUX to CASDO) will be disabled, it is only ready for the RX circuit to decode the CASDI line signal.

Step 1: When the master MCU resets the cascade single line bus, the FULL24XN, FULL24, BITERR and PASSEN bits are cleared to 0, the CASRES and CASRXEN bits are set to 1, the RX circuit is only ready for the CASDI line signal.

Step 2: When a rising edge appears on the CASDI line, the RX circuit begins to count high level. If the high level count value is less than or equal to (CASTH-2) before a falling edge appears on the CASDI line, it means logic low. If the high level count value is greater than or equal to (CASTH+1) before a falling edge appears on the CASDI line, it means logic high.

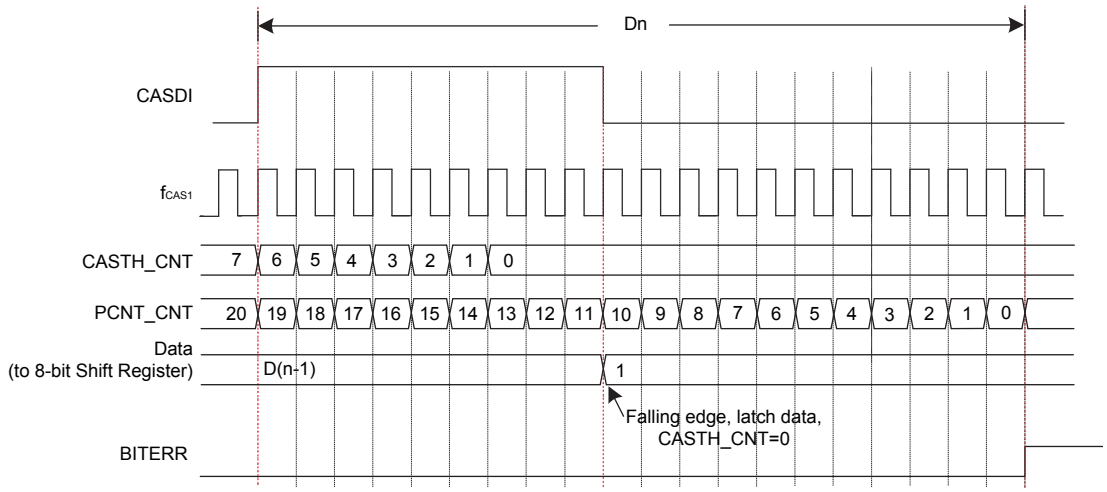
Step 3: When the 24×N bits shift register is full, the FULL24XN bit is set to 1, the bus for bypass path will be automatically enabled. So the PASSEN bit is set to 1, the CASRXEN bit is cleared to 0 and the system will generate a CASINT interrupt.

Step 4: The input signal from CASDI line will be transferred to CASDO line through Mux, and can be passed to the next device.

Step 5: When the master MCU resets the cascade bus again, the process will begin from step1 again.

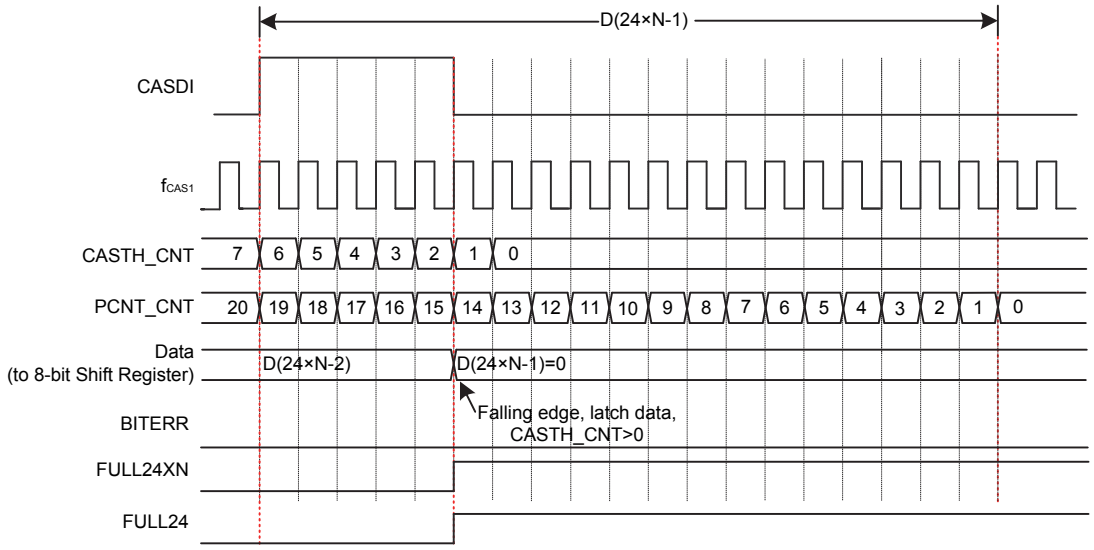
Step 6: When the first rising edge appears on the CASDI line, the CASRES bit will be cleared to zero automatically.

Note: The cascade clock f_{CAS1} can be adjusted for different baud rates.



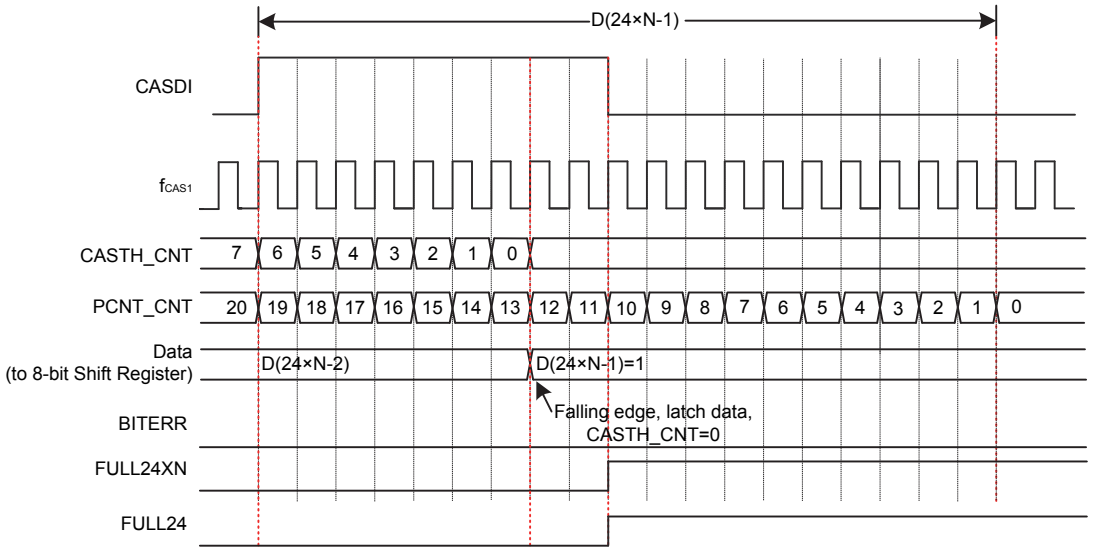
Bit Error in RX mode for bit 0~bit (24×N-2)

Note: If there is no second rising edge on the CASDI line before the PCNT counter counts down to zero for the received data bit 0 ~ bit (24×N-2), the bit error flag, BITERR, will automatically be set to 1.



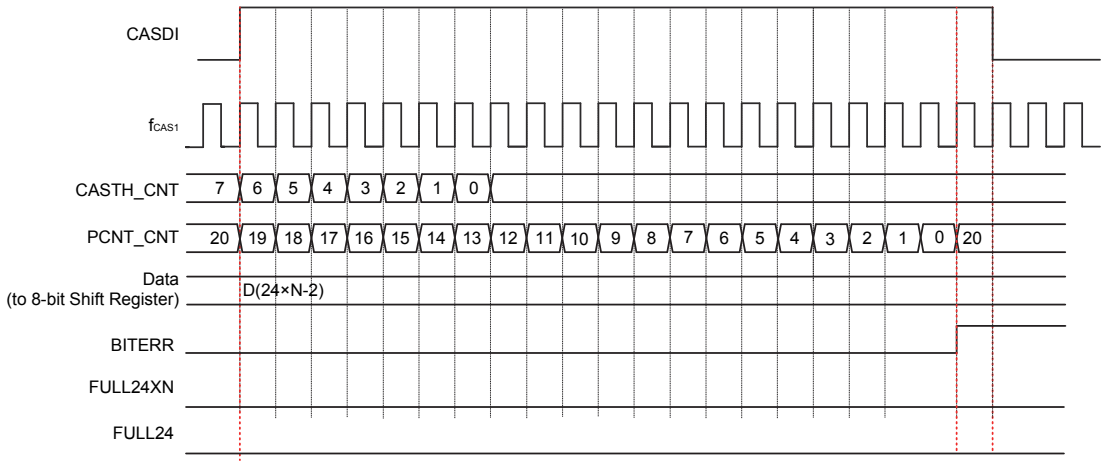
Data=0 in RX Mode for bit (24×N-1)

Note: For the data bit (24×N-1) reception if a falling edge appears on the CASDI line before the PCNT counter counts down to zero, the RX shift register full flags, FULL24XN and FULL24, will be set to 1. Before the CASTH counter counts down to zero, the data logic 0 will be read out when a falling edge appears on the CASDI line.



Data=1 in RX Mode for bit (24×N-1)

Note: For the data bit (24×N-1) reception if a falling edge appears on the CASDI line before the PCNT counter counts down to zero, the RX shift register full flags, FULL24XN and FULL24, will be set to 1. When the CASTH counter counts down to zero, the data logic 1 will be stored in the 8-bit shift register and be read out until a falling edge appears on the CASDI line.



Abnormal Situation in RX mode for bit (24×N-1)

Note: This is an abnormal situation. If the CASDI line signal is kept at a high level for the data bit (24×N-1) reception period. If no falling edge appears on the CASDI line until the PCNT counter underflows, it means that the whole 24×N bits of data have not been completely received, the BITERR bit will be set high, the FULL24XN and FULL24 flags will be cleared to 0, and the PASSEN bit will have no change.

Cascade TX Procedure

Step 1: Firstly clear the CASEN bit to 0 to disable RX and TX functions.

Step 2: Set the CASMOD bit high and clear the PASSEN bit to 0 to enable TX function and pass path to the TX circuit. Then set the CASEN bit high.

Step 3: Write 3 bytes of data into the CASD0~CASD2 registers respectively.

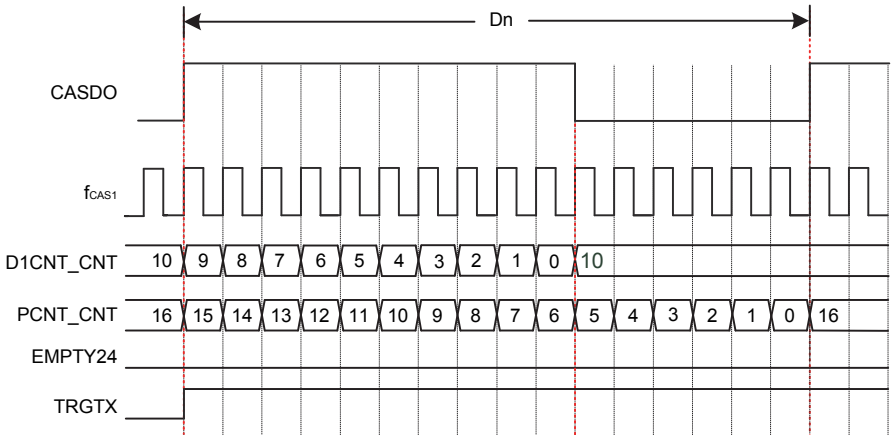
Step 4: Set the TRGTX bit high to begin to shift data in the CASD0~CASD2 registers and output.

Step 5: When the EMPTY24 flag is set high which occurs when the 24-bit TX shift register is empty, fill value into the CASD0, CASD1 and CASD2 registers again. Repeat from Step 4.

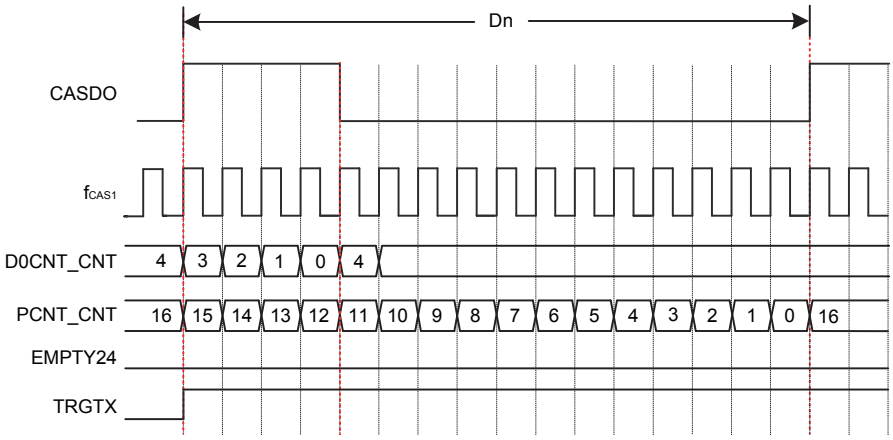
Step 6: When every 24×N×M bits of data have been shifted out, where M stands for the number of the RX devices cascaded, the TX device can send a RESET command to the slave device allowing the RX circuit to set the CASRES bit high by software.

Step 7: Repeat from Step 3 for next frame data (24×N×M bits) again.

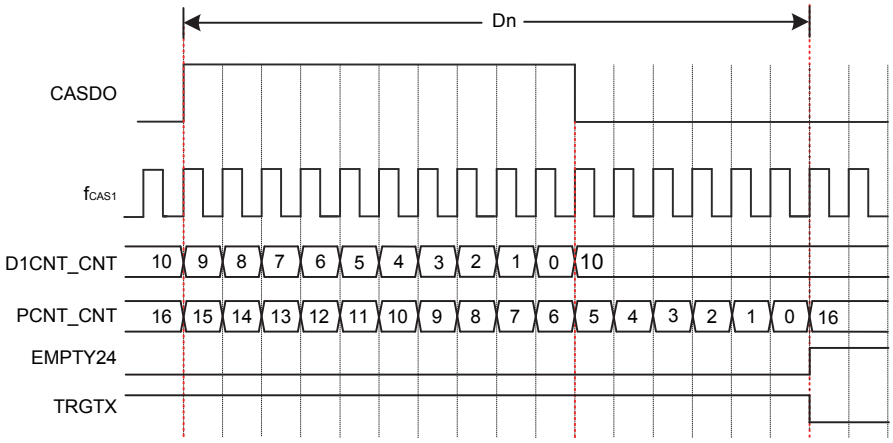
Note: The cascade clock fCAS1 can be adjusted for different baud rates.



Data=1 in TX Mode (n=0~24×N-2, D1CNT=10)



Data=0 in TX Mode (n=0~24×N-2, D0CNT=4)

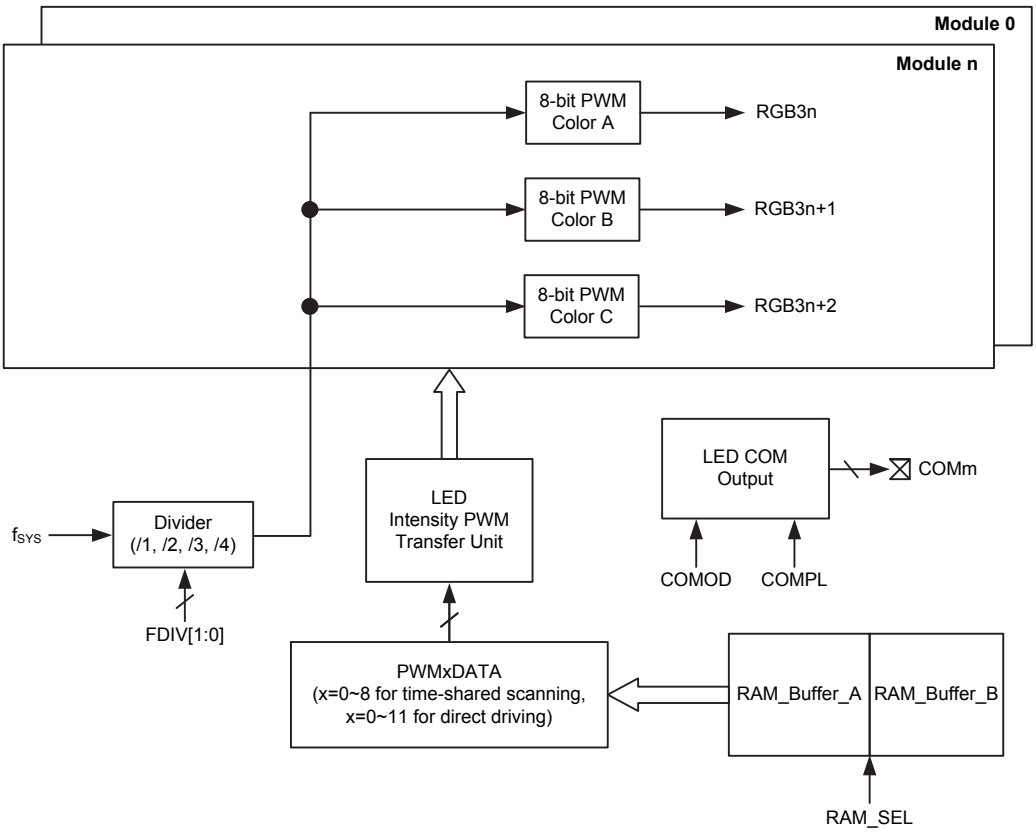


Data=1 in TX Mode (n=24×N-1, D1CNT=10)

Note: The TRGTX bit will be cleared to 0 by hardware when the TX shift register empty flag, EMPTY24, is set high.

PWM for RGB LED

The device includes a PWM function for RGB LED applications, which is composed of four LED PWM modules, two LED RAM buffers and a Register transfer unit as well as an LED COM output unit.



Note: n=0~3 and m=0~3.

PWM for RGB LED Block Diagram

PWM Register Description

Overall operation of the PWM function for RGB LED is controlled using a series of registers. The COM_PWM register is used for PWM function control, PWM clock divider selection, scan mode selection and configuration. The PWM0DATA~PWM11DATA registers are used to define the duty of the PWM outputs.

Register Name	Bit							
	7	6	5	4	3	2	1	0
COM_PWM	PWMEN	—	FDIV1	FDIV0	RAM_SEL	COMPL	COMOD	SCANMOD
PWM0DATA	D7	D6	D5	D4	D3	D2	D1	D0
PWM1DATA	D7	D6	D5	D4	D3	D2	D1	D0
PWM2DATA	D7	D6	D5	D4	D3	D2	D1	D0
PWM3DATA	D7	D6	D5	D4	D3	D2	D1	D0
PWM4DATA	D7	D6	D5	D4	D3	D2	D1	D0
PWM5DATA	D7	D6	D5	D4	D3	D2	D1	D0
PWM6DATA	D7	D6	D5	D4	D3	D2	D1	D0
PWM7DATA	D7	D6	D5	D4	D3	D2	D1	D0
PWM8DATA	D7	D6	D5	D4	D3	D2	D1	D0
PWM9DATA	D7	D6	D5	D4	D3	D2	D1	D0
PWM10DATA	D7	D6	D5	D4	D3	D2	D1	D0
PWM11DATA	D7	D6	D5	D4	D3	D2	D1	D0

PWM Register List

- **COM_PWM Register**

Bit	7	6	5	4	3	2	1	0
Name	PWMEN	—	FDIV1	FDIV0	RAM_SEL	COMPL	COMD	SCANMOD
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 **PWMEN**: PWM enable or disable control
0: PWM all disable
1: PWM all enable
When the PWMEN is set high, the FDIV1~FDIV0, COMPL, COMOD and SCANMOD bits are forbidden to be changed with the exception of the RAM_SEL bit. When the bit is cleared to zero, the COM0~COM3 pins will output inactive data and RGB0~RGB11 are fixed at "1".

Bit 6 Unimplemented, read as "0"

Bit 5~4 **FDIV1~FDIV0**: PWM clock divider selection
f_{PWMCLK}=
00: f_{SYS} clock divided by 1
01: f_{SYS} clock divided by 2
10: f_{SYS} clock divided by 3
11: f_{SYS} clock divided by 4
When these bits are set to 10, positive half cycle = 2×PWMCLK, negative half cycle = 1×PWMCLK.

Bit 3 **RAM_SEL**: RAM buffer selection for COM scan display (only available for time-shared scanning)
0: Select RAM buffer A for COM scan
1: Select RAM buffer B for COM scan
Note that the RAM buffer change only takes effect when the current frame is finished. At the end of the current frame, the RAM_SEL bit value will be captured and the new RAM buffer selection will take effect. In the time-shared scanning mode, general read or write operations to the selected buffer are not available.

Bit 2 **COMPL**: COM polarity selection (only available for time-shared scanning)
0: Low active
1: High active
When in the direct driving mode, the COM pins will output inactive data. Pay attention to the COMPL bit setup when in the PWM disable status. If PWMEN=0 and COMPL=0, the COM pins will output a high level (inactive data); if PWMEN=0 and COMPL=1, the COM pins will output a low level (inactive data). It is suggested to first select the COM pins by configuring the pin-shared control bits and configure the COMPL bit before enabling the PWM function.

Bit 1 **COMOD**: COM mode selection (only available for time-shared scanning)
0: 3×COM
1: 4×COM

Bit 0 **SCANMOD**: PWM time-shared scanning or direct driving mode selection
0: Direct driving mode
1: Time-shared scanning mode

• PWM0DATA 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**: LED0 color A
This register is read only when in the time-shared scanning mode.

• PWM1DATA 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**: LED0 color B
This register is read only when in the time-shared scanning mode.

• PWM2DATA 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**: LED0 color C
This register is read only when in the time-shared scanning mode.

• PWM3DATA 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**: LED1 color A
This register is read only when in the time-shared scanning mode.

• **PWM4DATA 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**: LED1 color B
This register is read only when in the time-shared scanning mode.

• **PWM5DATA 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**: LED1 color C
This register is read only when in the time-shared scanning mode.

• **PWM6DATA 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**: LED2 color A
This register is read only when in the time-shared scanning mode.

• **PWM7DATA 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**: LED2 color B
This register is read only when in the time-shared scanning mode.

• **PWM8DATA 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**: LED2 color C
This register is read only when in the time-shared scanning mode.

• **PWM9DATA 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**: LED3 color A
This register is read only when in the time-shared scanning mode.

• **PWM10DATA 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**: LED3 color B
This register is read only when in the time-shared scanning mode.

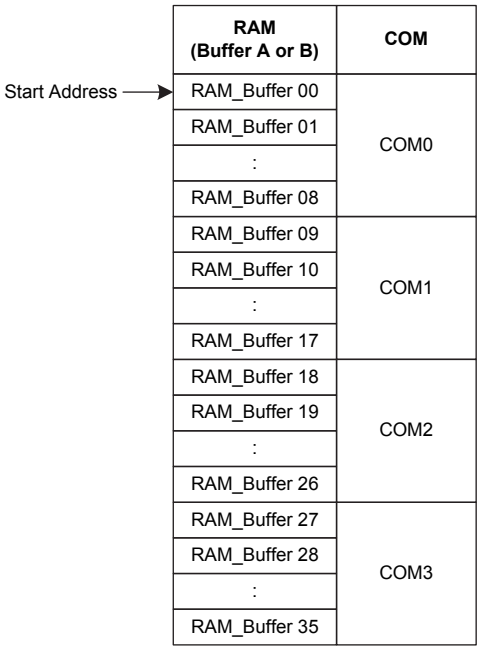
• **PWM11DATA Register**

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**: LED3 color C
This register is read only when in the time-shared scanning mode.

PWM RAM Buffer

There are two LED PWM RAM buffers, buffer A and B, with each containing 64 bytes. The buffers are located in Bank 1, 80H~BFH for buffer A and C0H~FFH for buffer B. However, the actual available size for the PWM function is 36 bytes, as shown below. More details regarding the RAM buffer usage will be described in the PWM Operation section.



PWM Operation

The overall PWM function is controlled by the PWMEN bit in the COM_PWM register. If the PWM function is not used, the PWMEN bit should be cleared to disable the PWM function for power saving consideration. During the PWM operations, either in direct driving mode or time-shared scanning mode, if a HALT instruction is executed and the f_{PWMCLK} clock is still on, the PWM operations and waveform output will continue. However, if the f_{PWMCLK} clock is off after executing the HALT instruction, the PWM function will stay in the instantaneous state of entering the HALT status.

The PWM clock is derived from the system clock f_{SYS} and the clock divider used to generate the f_{PWMCLK} clock is selected by the FDIV1~FDIV0 bits.

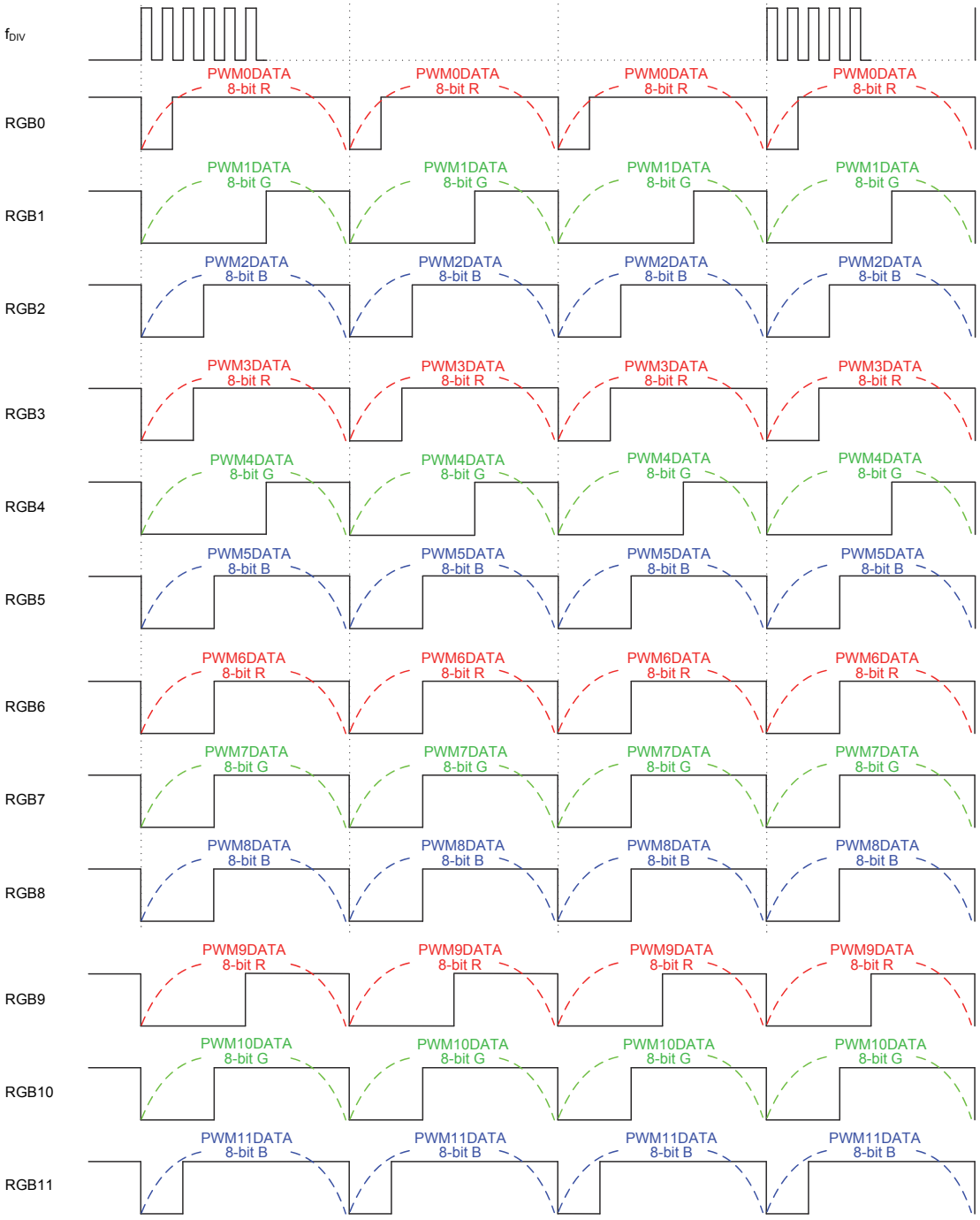
The PWM function supports two operating modes, which are direct driving mode and time-shared scanning mode, selected by the SCANMOD bit in the COM_PWM register.

Direct Driving Mode

If the direct driving mode is selected by clearing the SCANMOD bit to zero, all of the 12 PWM sets will be used which can drive 4 RGB LEDs. And the output waveform is determined by the LED PWM data registers, PWM0DATA~PWM11DATA. The PWM data register contents, once being changed, will directly react on the RGBn outputs and the RGB dimming will take effect immediately.

If the PWM data registers are modified, the PWM function will not be reset to output a new PWM duty cycle but will output the new PWM waveform directly after the old PWM waveform at the moment of data change. It means that the PWM waveforms before and after the data change share the same ($256\times f_{PWMCLK}$) cycle.

In this mode, both RAM buffer A and B can be used as the general purpose data memory and are read/write available.



LED PWM Timing for Direct Driving Mode

Time-shared Scanning Mode

If the time-shared scanning mode is selected by setting the SCANMOD bit high, only 9 PWM sets will be used to drive RGB LEDs. And the output waveform is determined by the LED PWM data registers, PWM0DATA~PWM8DATA. The PWM9DATA~PWM11DATA registers take no effect in this mode and the corresponding PWM outputs are floating regardless of the register contents.

In this mode, the COM mode, 3×COM or 4×COM, is selected using the COMOD bit and the COM polarity is determined by the COMPL bit. The RAM buffer A or B for COM scan display is determined by the RAM_SEL bit. The 27 bytes (3×COM) or 36 bytes (4×COM) of data in the RAM_Buffer_A/B will be transferred to the PWM0DATA~PWM2DATA & PWM3DATA~PWM5DATA & PWM6DATA~PWM8DATA registers by hardware with 9 bytes each time, to generate the corresponding PWM waveforms.

If RAM_SEL is equal to 0, the RAM buffer A is selected and cannot be read or written by the application program, however the RAM buffer B is read/write available. On the contrary, if RAM_SEL is equal to 1, the RAM buffer B is selected and cannot be read or written by the application program, however the RAM buffer A is read/write available. It should be noted that if the RAM buffer selection is changed, the new RAM buffer data will be loaded to the PWM data registers only when the current frame is completed. The unused bytes in the RAM buffer A or B have no effect on the PWM outputs.

The time for loading data from RAM buffer to the PWM data register is:

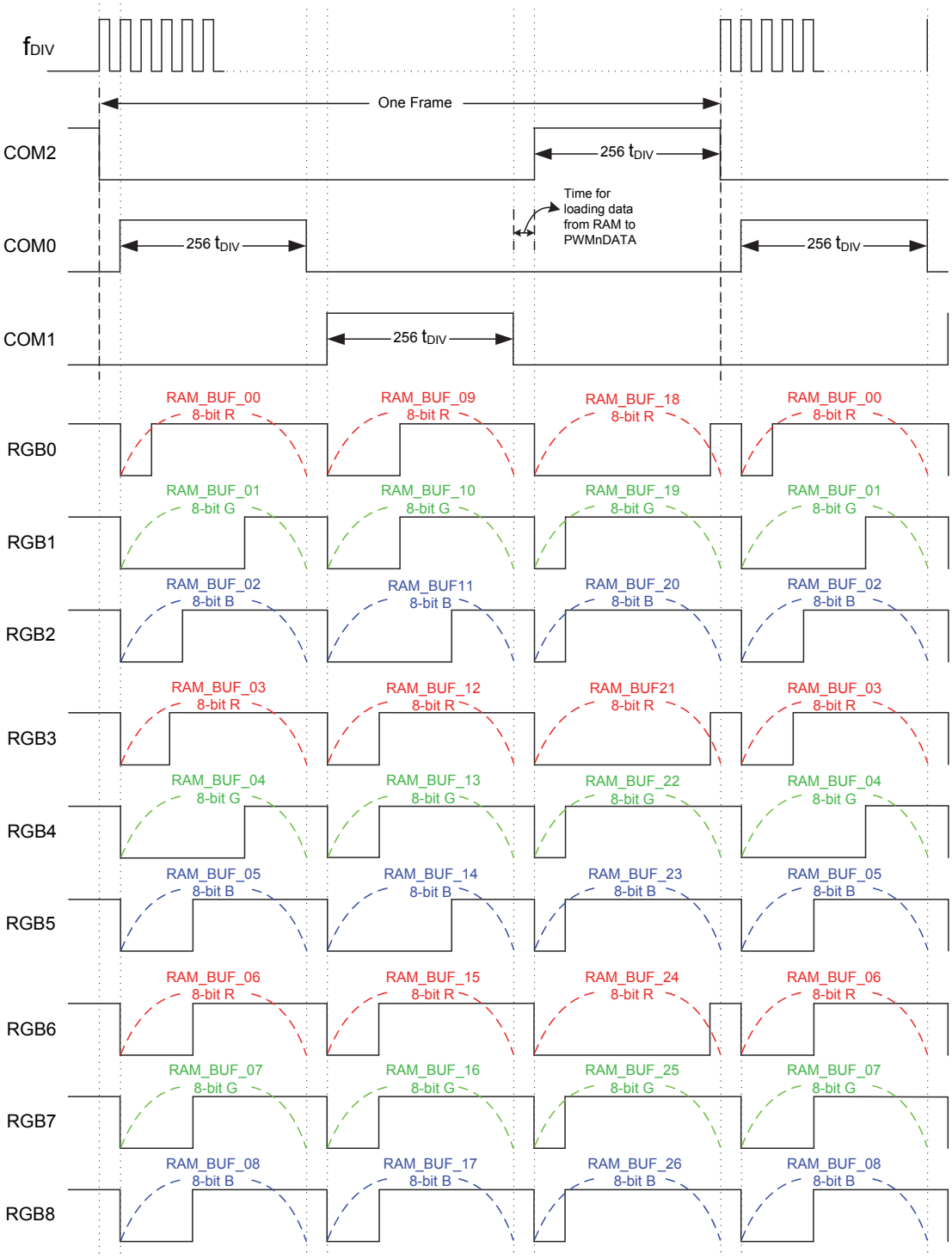
$$4 \times t_{PWMCLK} \times 9 + (0\sim1) \times t_{PWMCLK} + (0\sim1) \times t_{DIV}$$

The frame cycle for the 3×COM mode is:

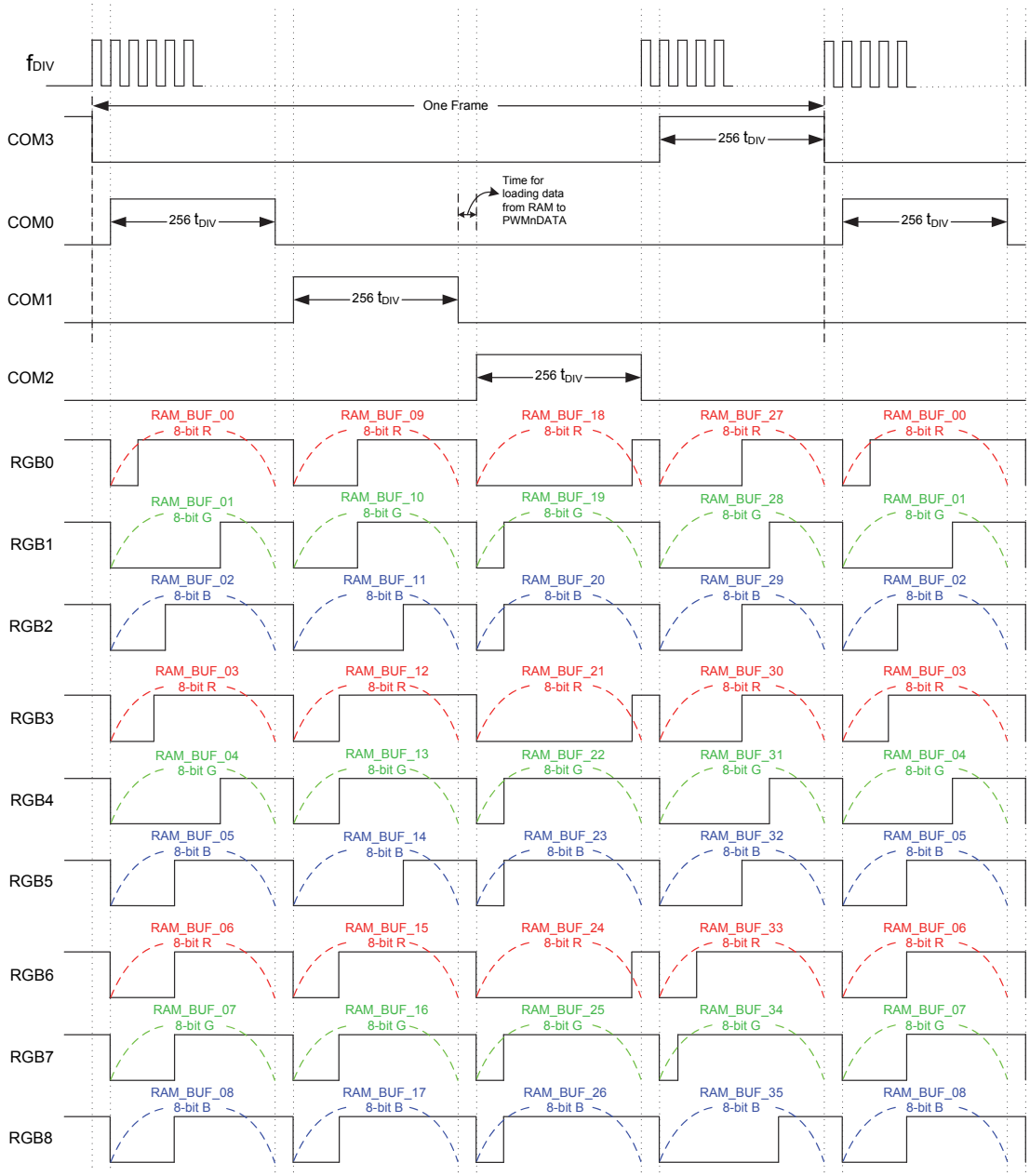
$$3 \times \{[4 \times t_{PWMCLK} \times 9 + (0\sim1) \times t_{PWMCLK} + (0\sim1) \times t_{DIV}] + 256 \times t_{DIV}\}$$

The frame cycle for the 4×COM mode is:

$$4 \times \{[4 \times t_{PWMCLK} \times 9 + (0\sim1) \times t_{PWMCLK} + (0\sim1) \times t_{DIV}] + 256 \times t_{DIV}\}$$



LED PWM Timing for Time-shared Scanning Mode – 3×COM



LED PWM Timing for Time-shared Scanning Mode – 4×COM

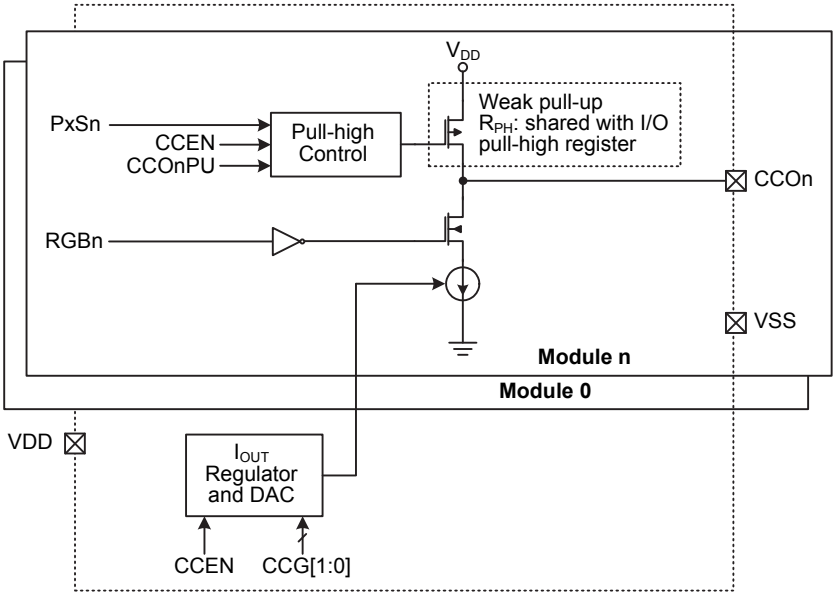
Constant Current LED Driver

The device includes an accurate constant current driver which is specifically designed for LED display applications. The device provides 12-channel stable and constant current outputs for driving LEDs.

The output constant current is determined by the current gain selection bits CCG1~CCG0, RGBn PWM input and CCOn pull-high function. The current variation between channels is ±1.5% while the current variation between different devices is ±3%. The output current remains constant regardless of the LED forward voltage value.

The constant current can be calculated using the following formula:

$$I_{CCOn} = 5mA \times Gain$$



Note: 1. n=0~11.
2. Module n stands for Module 0 ~ Module 11.
3. RGBn is sourced from the PWM output.

Constant Current LED Driver Block Diagram

Constant Current LED Driver Register Description

There are three control registers associated with the constant current LED driver, CCS, CCOPU0 and CCOPU1. The CCS register is used for constant current function control and gain selection. The CCOPU0 and CCOPU1 registers are used to control the pull-high function of the CCOn pin.

Register Name	Bit							
	7	6	5	4	3	2	1	0
CCS	CCEN	D6	D5	D4	—	—	CCG1	CCG0
CCOPU0	CCO7PU	CCO6PU	CCO5PU	CCO4PU	CCO3PU	CCO2PU	CCO1PU	CCO0PU
CCOPU1	—	—	—	—	CCO11PU	CCO10PU	CCO9PU	CCO8PU

Constant Current LED Driver Register List

• **CCS Register**

Bit	7	6	5	4	3	2	1	0
Name	CCEN	D6	D5	D4	—	—	CCG1	CCG0
R/W	R/W	R/W	R/W	R/W	—	—	R/W	R/W
POR	0	0	1	0	—	—	0	0

Bit 7 **CCEN**: Constant Current function enable or disable control
0: Disable
1: Enable
This bit is used to control the constant current function. When the CCEN bit is cleared to zero, the constant current function will be disabled and the CCOn output will be in a floating state regardless of the RGBn signal. The pull-high resistor shared with the I/O pin cannot be controlled by the CCOnPU bit.
When the pin is configured as a CCOn output function and the CCEN bit is set high, the pull-high resistor can be controlled using the corresponding CCOnPU bit. When the RGBn signal is in a logic low state with the CCO function enabled, the CCOn output will be a constant current driver. If the RGBn signal is in a logic high state with the CCO function enabled, the CCOn output will be in a high or floating state depending upon whether the pull-high function is enabled or not.

I/O Mode	CCEN	RGBn	CCOnPU	Pull-high Enable/Disable	CCOn Status
Px	X	X	X	Controlled by PxPU	General I/O status
CCOn	0	X	X	Disable	Floating
CCOn	1	0	0	Disabled by CCOnPU	Output low (constant current)
CCOn	1	0	1	Enabled by CCOnPU	Output low (affect constant current value)
CCOn	1	1	0	Disabled by CCOnPU	Floating
CCOn	1	1	1	Enabled by CCOnPU	Pull-high
Note: Px: Port A or B; X: Don't care.					

Bit 6~4 **D6~D4**: Reserved bits. These bits cannot be used and must be fixed as "010".

Bit 3~2 Unimplemented, read as "0"

Bit 1~0 **CCG1~CCG0**: Constant current selection
00: I_{CCOn}=5mA
01: I_{CCOn}=14mA
10: I_{CCOn}=32mA
11: I_{CCOn}=53mA

• **CCOPU0 Register**

Bit	7	6	5	4	3	2	1	0
Name	CCO7PU	CCO6PU	CCO5PU	CCO4PU	CCO3PU	CCO2PU	CCO1PU	CCO0PU
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 **CCO7PU~CCO0PU**: CCO7~CCO0 pull-high control
0: Disable
1: Enable

• **CCOPU1 Register**

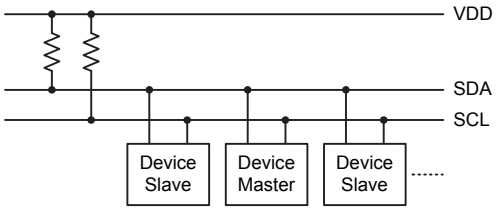
Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	CCO11PU	CCO10PU	CCO9PU	CCO8PU
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	0	0	0

Bit 7~4 Unimplemented, read as "0"

Bit 3~0 **CCO11PU~CCO8PU**: CCO11~CCO8 pull-high control
0: Disable
1: Enable

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

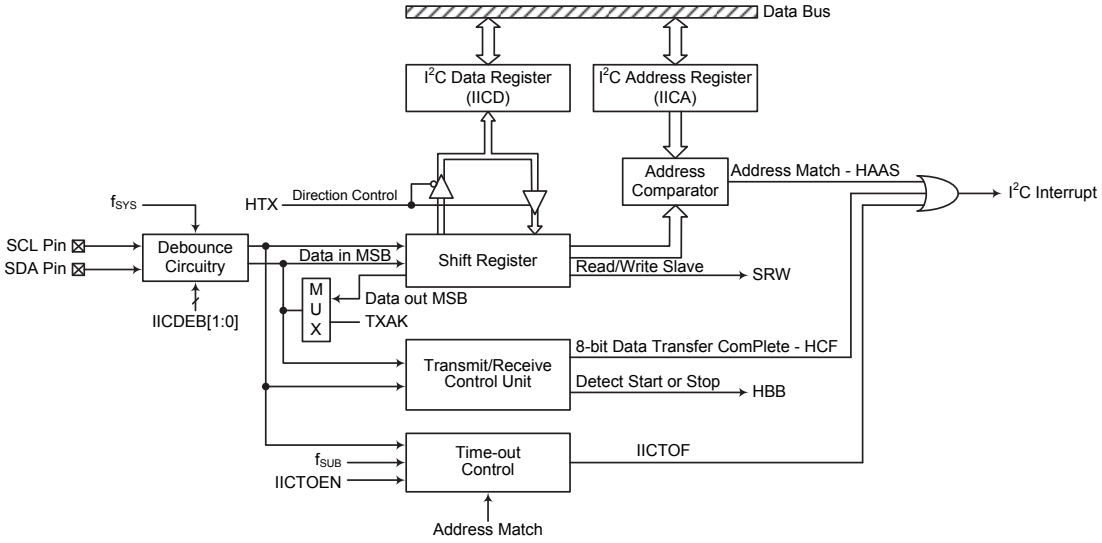
I²C Interface Operation

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.

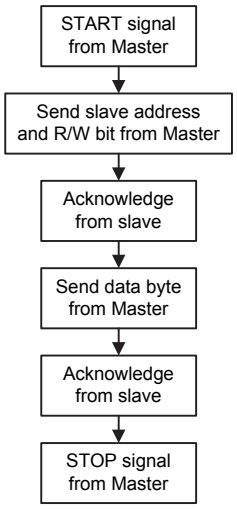
When two devices communicate with each other on the bidirectional I²C bus, one is known as the master device and one as the slave device. Both master and slave can transmit and receive data, however, it is the master device that has overall control of the bus. For the device, which only operates in slave mode, there are two methods of transferring data on the I²C bus, the slave transmit mode and the slave receive mode.

It is suggested that the device shall not enter the IDLE or SLEEP mode during the I²C communication is in progress.

If the related pin is configured as SDA or SCL function, the pin must be configured to open-collect input/output port and its pull-high function can be enabled by programming the related pull-high control register.



I²C Block Diagram



I²C Interface Operation

The IICDEB1 and IICDEB0 bits determine the debounce time of the I²C interface. This uses the internal 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 the I²C Standard 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 Debounce	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 Requirements

I²C Registers

There are three control registers associated with the I²C bus, IICC0, IICC1 and IICTOC, one address register IICA and one data register, IICD.

Register Name	Bit							
	7	6	5	4	3	2	1	0
IICC0	—	—	—	—	IICDEB1	IICDEB0	IICEN	—
IICC1	HCF	HAAS	HBB	HTX	TXAK	SRW	IAMWU	RXAK
IICD	D7	D6	D5	D4	D3	D2	D1	D0
IICA	IICA6	IICA5	IICA4	IICA3	IICA2	IICA1	IICA0	—
IICTOC	IICTOEN	IICTOF	IICTOS5	IICTOS4	IICTOS3	IICTOS2	IICTOS1	IICTOS0

I²C Register List

I²C Data Register

The IICD register is used to store the data being transmitted and received. Before the device writes data to the I²C bus, the actual data to be transmitted must be placed in the IICD register. After the data is received from the I²C bus, the device can read it from the IICD register. Any transmission or reception of data from the I²C bus must be made via the IICD register.

- **IICD 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**: I²C data register bit 7 ~ bit 0

I²C Address Register

The IICA register is the location where the 7-bit slave address of the slave device is stored. Bits 7~1 of the IICA register define the device slave address. Bit 0 is not defined. When a master device, which is connected to the I²C bus, sends out an address, which matches the slave address in the IICA register, the slave device will be selected.

- **IICA Register**

Bit	7	6	5	4	3	2	1	0
Name	IICA6	IICA5	IICA4	IICA3	IICA2	IICA1	IICA0	—
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 7~1 **IICA6~IICA0**: I²C slave address
IICA6~IICA0 is the I²C slave address bit 6~bit 0.

Bit 0 Unimplemented, read as "0"

I²C Control Registers

There are three control registers for the I²C interface, IICC0, IIC1 and IICTOC. The IICC0 register is used to control the enable/disable function and to set the data transmission clock frequency. The IICC1 register contains the relevant flags which are used to indicate the I²C communication status. Another register, IICTOC, is used to control the I²C time-out function and will be described in the I²C Time-out section.

- **IICC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	IICDEB1	IICDEB0	IICEN	—
R/W	—	—	—	—	R/W	R/W	R/W	—
POR	—	—	—	—	0	0	0	—

Bit 7~4 Unimplemented, read as "0"

Bit 3~2 **IICDEB1~IICDEB0**: I²C Debounce Time Selection
00: No debounce
01: 2 system clock debounce
1x: 4 system clock debounce
Note that the I²C debounce circuit will operate normally if the system clock, f_{SYS}, is derived from the f_H clock or the IAMWU bit is equal to 0. Otherwise, the debounce circuit will have no effect and be bypassed.

Bit 1 **IICEN**: I²C Enable Control
0: Disable
1: Enable
The bit is the overall on/off control for the I²C interface. When the IICEN bit is cleared to zero to disable the I²C interface, the SDA and SCL lines will lose their I²C function and the I²C operating current will be reduced to a minimum value. When the bit is high the I²C interface is enabled. If the IICEN 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 Unimplemented, read as "0"

- **IICC1 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	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. Below is an example of the flow of a two-byte I²C data transfer.
First, the I²C slave device receives a start signal from the I²C master and then the HCF bit is automatically cleared to zero. Second, the I²C slave device finishes receiving the 1st data byte and then the HCF bit is automatically set to one. Third, users read the 1st data byte from the IICD register by the application program and then the HCF bit is automatically cleared to zero. Fourth, the I²C slave device finishes receiving the 2nd data byte and then the HCF bit is automatically set high and so on. Finally, the I²C slave device receives a stop signal from the I²C master and then the HCF bit is automatically set high.

Bit 6 **HAAS**: I²C Bus address match 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 is transmitter or receiver selection
0: Slave device is the receiver
1: Slave device is the transmitter

Bit 3 **TXAK**: I²C Bus transmit acknowledge flag
0: Slave send acknowledge flag
1: Slave do 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**: I²C 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 I²C Slave Read/Write flag. This flag determines whether the master device wishes to transmit or receive data from the I²C 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.

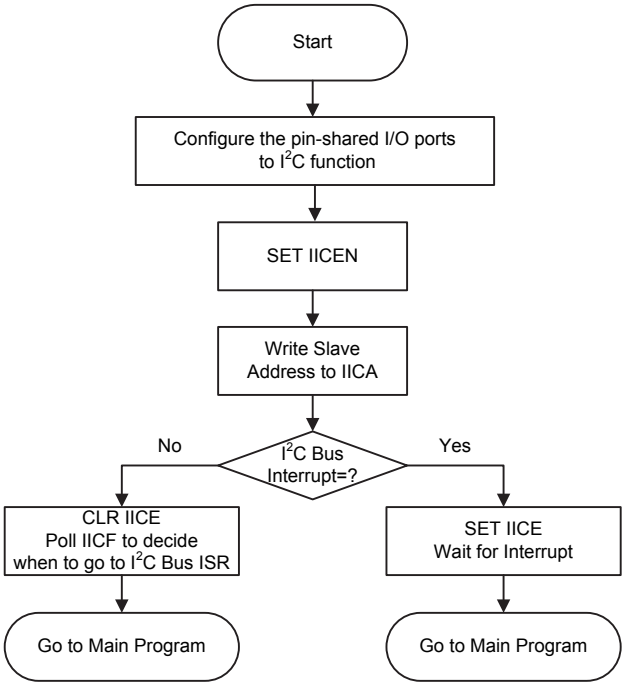
Bit 1 **IAMWU**: I²C Address Match Wake-up control
0: Disable
1: Enable
This bit should be set to 1 to enable the I²C address match wake up from the SLEEP or IDLE Mode. If the IAMWU bit has been set before entering either the SLEEP or IDLE mode to enable the I²C address match wake up, then this bit must be cleared by the application program after wake-up to ensure correction device operation.

Bit 0 **RXAK**: I²C Bus Receive acknowledge flag
0: Slave receive 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 I²C Bus.

I²C Bus Communication

Communication on the I²C 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 I²C 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 IICC1 register will be set and an I²C interrupt will be generated. After entering the interrupt service routine, the slave device must first check the condition of the HAAS and IICTOF bits to determine whether the interrupt source originates from an address match or from the completion of an 8-bit data transfer completion or from the I²C 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 I²C bus, the microcontroller must initialise the bus, the following are steps to achieve this:

- Step 1
  Configure the related pin-shared I/O pins as I²C pin functions. Set the IICEN bits in the IICC0 register to "1" to enable the I²C bus.
- Step 2
  Write the slave address of the device to the I²C bus address register IICA.
- Step 3
  Set the IICE interrupt enable bit of the interrupt control register to enable the I²C 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 IICC1 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 I²C bus interrupt can come from three sources, when the program enters the interrupt subroutine, the HAAS and IICTOF bits should be examined to see whether the interrupt source comes from a matching slave address or from the completion of a data byte transfer or from the I²C bus time-out occurrence. When a slave address is matched, the device must be placed in either the transmit mode and then write data to the IICD register, or in the receive mode where it must implement a dummy read from the IICD register to release the SCL line.

I²C Bus Read/Write Signal

The SRW bit in the IICC1 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 configured 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 configured to read data from the I²C bus as a receiver.

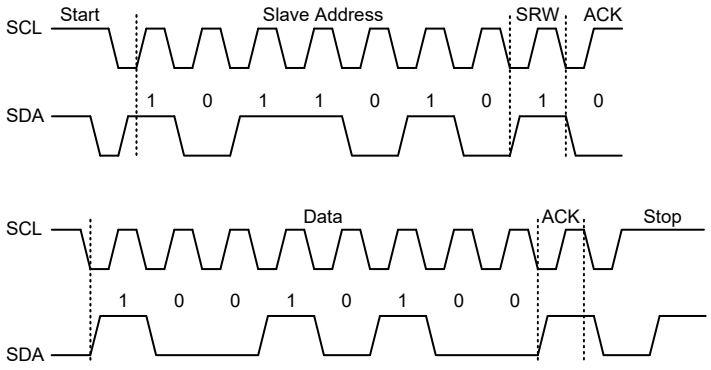
I²C Bus Slave Address Acknowledge Signal

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 configured to be a transmitter so the HTX bit in the IICC1 register should be set to “1”. If the SRW flag is low, then the microcontroller slave device should be configured as a receiver and the HTX bit in the IICC1 register should be set to “0”.

I²C Bus Data and Acknowledge Signal

The transmitted data is 8-bit 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 IICD register. If configured as a transmitter, the slave device must first write the data to be transmitted into the IICD register. If configured as a receiver, the slave device must read the transmitted data from the IICD 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 configured as a transmitter will check the RXAK bit in the IICC1 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.

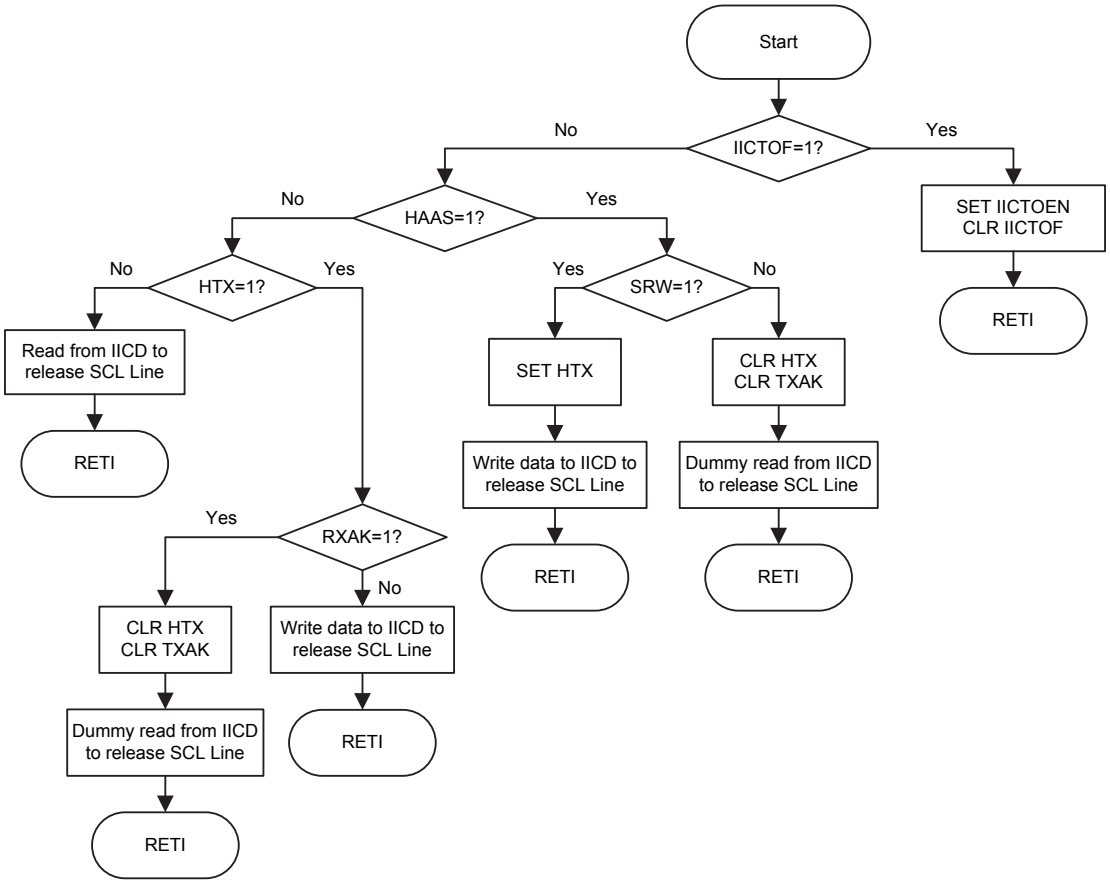


S=Start (1 bit)
SA=Slave Address (7 bits)
SR=SRW bit (1 bit)
M=Slave device send acknowledge bit (1 bit)
D=Data (8 bits)
A=ACK (RXAK bit for transmitter, TXAK bit for receiver, 1 bit)
P=Stop (1 bit)

S	SA	SR	M	D	A	D	A	·····	S	SA	SR	M	D	A	D	A	·····	P

I²C Communication Timing Diagram

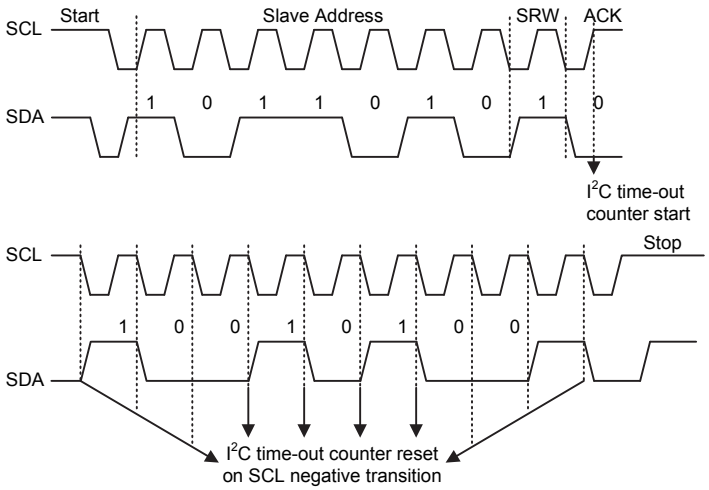
Note: When a slave address is matched, the device must be placed in either the transmit mode and then write data to the IICD register, or in the receive mode where it must implement a dummy read from the IICD register to release the SCL line.



I²C Bus ISR Flow Chart

I²C Time-out Control

In order to reduce the problem of I²C lockup due to reception of erroneous clock sources, a time-out function is provided. If the clock source to the I²C 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 counting 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 setup by the IICTOC register, then a time-out condition will occur. The time-out function will stop when an I²C "STOP" condition occurs.



I²C Time-out

When an I²C time-out counter overflow occurs, the counter will stop and the IICTOEN bit will be cleared to zero and the IICTOF 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 I²C 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:

Registers	After I²C Time-out
IICD, IICA, IICC0	No change
IICC1	Reset to POR condition

I²C Registers after Time-out

The IICTOF flag can be cleared by the application program. There are 64 time-out periods which can be selected using IICTOS bit field in the IICTOC register. The time-out time is given by the formula: ((1~64) × 32) / f_{SUB}. This gives a time-out period which ranges from about 1ms to 64ms.

• **IICTOC Register**

Bit	7	6	5	4	3	2	1	0
Name	IICTOEN	IICTOF	IICTOS5	IICTOS4	IICTOS3	IICTOS2	IICTOS1	IICTOS0
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 **IICTOEN**: I²C Time-out control
0: Disable
1: Enable

Bit 6 **IICTOF**: I²C Time-out flag
0: No time-out occurred
1: Time-out occurred
This bit is set high when time-out occurs and can only be cleared to zero by application program.

Bit 5~0 **IICTOS5~IICTOS0**: I²C Time-out period selection
I²C time-out clock source is $f_{SUB}/32$.
I²C time-out time is equal to (IICTOS[5:0]+1) × $(32/f_{SUB})$.

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 only contains internal interrupts functions. The internal interrupts are generated by various internal functions such as TM, Time Base, I²C interface and cascading transceiver interface.

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, as shown in the accompanying table. The registers fall into two categories. The first is the INTC0~INTC1 registers which configure the primary interrupts, the second is the MFI register which configures the Multi-function interrupt.

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
Global	EMI	—
Multi-function	MFE	MFF
Time Base	TBE	TBF
I²C Interface	IICE	IICF
Cascading transceiver interface	CASINTE	CASINTF
CTM	CTMPE	CTMPF
	CTMAE	CTMAF

Interrupt Register Bit Naming Conventions

Register Name	Bit							
	7	6	5	4	3	2	1	0
INTC0	—	CASINTF	MFF	TBF	CASINTE	MFE	TBE	EMI
INTC1	—	—	—	IICF	—	—	—	IICE
MFI	—	—	CTMAF	CTMPF	—	—	CTMAE	CTMPE

Interrupt Register List

- **INTC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	CASINTF	MFF	TBF	CASINTE	MFE	TBE	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 **CASINTF**: Cascading transceiver interface interrupt request flag
0: No request
1: Interrupt request

Bit 5 **MFF**: Multi-function interrupt request flag
0: No request
1: Interrupt request

Bit 4 **TBF**: Time Base interrupt request flag
0: No request
1: Interrupt request

Bit 3 **CASINTE**: Cascading transceiver interface interrupt control
0: Disable
1: Enable

Bit 2 **MFE**: Multi-function interrupt control
0: Disable
1: Enable

Bit 1 **TBE**: Time Base interrupt 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	—	—	—	IICF	—	—	—	IICE
R/W	—	—	—	R/W	—	—	—	R/W
POR	—	—	—	0	—	—	—	0

Bit 7~5 Unimplemented, read as "0"

Bit 4 **IICF**: I^2C interface interrupt request flag
0: No request
1: Interrupt request

Bit 3~1 Unimplemented, read as "0"

Bit 0 **IICE**: I^2C interface interrupt control
0: Disable
1: Enable

• **MFI Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	CTMAF	CTMPF	—	—	CTMAE	CTMPE
R/W	—	—	R/W	R/W	—	—	R/W	R/W
POR	—	—	0	0	—	—	0	0

Bit 7~6 Unimplemented, read as "0"

Bit 5 **CTMAF**: CTM Comparator A match interrupt request flag
0: No request
1: Interrupt request

Bit 4 **CTMPF**: CTM Comparator P match interrupt request flag
0: No request
1: Interrupt request

Bit 3~2 Unimplemented, read as "0"

Bit 1 **CTMAE**: CTM Comparator A match interrupt control
0: Disable
1: Enable

Bit 0 **CTMPE**: CTM Comparator P match interrupt control
0: Disable
1: Enable

Interrupt Operation

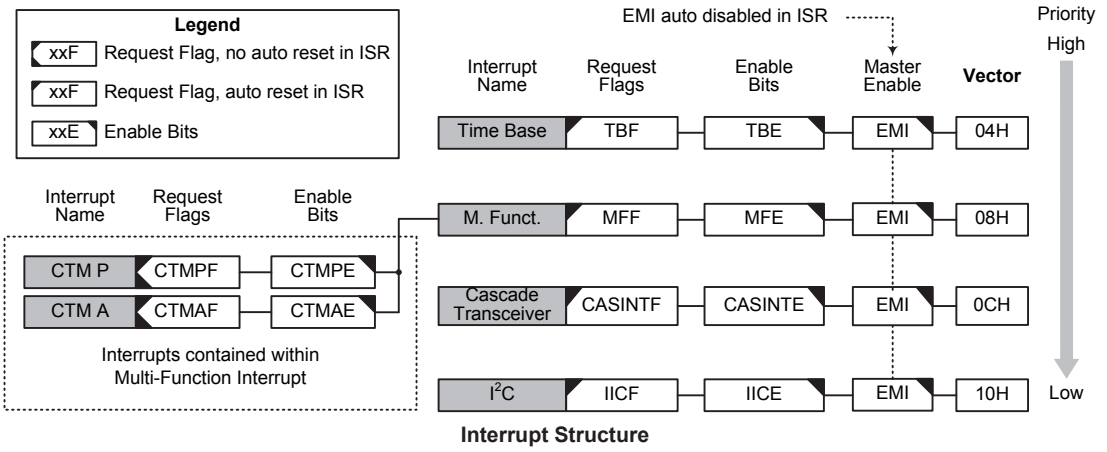
When the conditions for an interrupt event occur, such as a TM Comparator P or Comparator A match etc., the relevant interrupt request flag will be set. Whether the request flag actually generates a program jump to the relevant interrupt vector is determined by the condition of the interrupt enable bit. 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

Multi-function Interrupt

Within the device there is a Multi-function interrupt. Unlike the other independent interrupts, the interrupt has no independent source, but rather are formed from other existing interrupt sources, namely the TM interrupts.

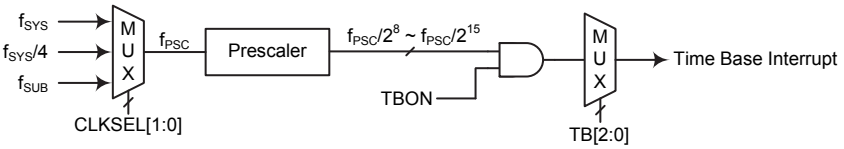
A Multi-function interrupt request will take place when any of the Multi-function interrupt request flag MFF is set. The Multi-function interrupt flag will be set when any of their included functions generate an interrupt request flag. To allow the program to branch to its respective interrupt vector address, when the Multi-function interrupt is enabled and the stack is not full, and any one of the interrupts contained within the Multi-function interrupt occurs, a subroutine call to the Multi-function interrupt vector 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 flag will be automatically reset when the interrupt is serviced, the request flags from the original source of the Multi-function interrupt will not be automatically reset and must be manually reset by the application program.

Time Base Interrupt

The function of the Time Base Interrupt is to provide regular time signal in the form of an internal interrupt. It is controlled by the overflow signal from its internal timer. When this happens its interrupt request flag, TBF, will be set. To allow the program to branch to its respective interrupt vector addresses, the global interrupt enable bit, EMI and Time Base enable bit, TBE, must first be set. When the interrupt is enabled, the stack is not full and the Time Base overflows, a subroutine call to its respective vector location will take place. When the interrupt is serviced, the interrupt request flag, TBF, will be automatically reset and the EMI bit will be cleared to disable other interrupts.

The purpose of the Time Base Interrupt is to provide an interrupt signal at fixed time periods. Its clock source, f_{PSC}, 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 TBC register to obtain longer interrupt periods whose value ranges. The clock source which in turn controls the Time Base interrupt period is selected using the CLKSEL [1:0] in the PSCR register respectively.



Time Base Interrupt

- **PSCR Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	CLKSEL1	CLKSEL0
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	0	0

Bit 7~2 Unimplemented, read as "0"

Bit 1~0 **CLKSEL1~CLKSEL0**: Prescaler clock source selection
00: f_{SYS}
01: $f_{SYS}/4$
1x: f_{SUB}

- **TBC Register**

Bit	7	6	5	4	3	2	1	0
Name	TBON	—	—	—	—	TB2	TB1	TB0
R/W	R/W	—	—	—	—	R/W	R/W	R/W
POR	0	—	—	—	—	0	0	0

Bit 7 **TBON**: Time Base Enable Control
0: Disable
1: Enable

Bit 6~3 Unimplemented, read as "0"

Bit 2~0 **TB2~TB0**: Time Base time-out period selection
000: $2^8/f_{PSC}$
001: $2^9/f_{PSC}$
010: $2^{10}/f_{PSC}$
011: $2^{11}/f_{PSC}$
100: $2^{12}/f_{PSC}$
101: $2^{13}/f_{PSC}$
110: $2^{14}/f_{PSC}$
111: $2^{15}/f_{PSC}$

Cascade Transceiver Interface Interrupt

A Cascade Transceiver Interface Interrupt request will take place when the Cascade Transceiver Interface Interrupt request flag, CASINTF, is set, which occurs when RX receive data format error occurs, cascade circuit reset occurs, cascading transceiver TX shift register is empty or cascading transceiver 24 bits or 24×N bits RX shift register is full. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and the Cascade Transceiver Interface Interrupt enable bit, CASINTE, must first be set. When the interrupt is enabled, the stack is not full and any one of the above four described situations occurs, a subroutine call to the respective Interrupt vector, will take place. When the Cascade Transceiver Interface Interrupt is serviced, the interrupt request flag, CASINTF, will be automatically reset and the EMI bit will be cleared to disable other interrupts.

Timer Module Interrupts

The CTM has two interrupts which are both contained within the Multi-function Interrupt. For the CTM there are two interrupt request flags CTMPF and CTMAF and two enable bits CTMPE and CTMAE. A CTM interrupt request will take place when any of the CTM request flags is set, a situation which occurs when a CTM comparator P or comparator A match situation happens.

To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and the respective CTM Interrupt enable bit, and the associated Multi-function interrupt enable bit, MFE, must first be set. When the interrupt is enabled, the stack is not full and a CTM comparator match situation occurs, a subroutine call to the relevant CTM Interrupt vector locations, will take place. When the CTM interrupt is serviced, the EMI bit will be automatically cleared to disable other interrupts, however only the related MFF flag will be automatically cleared. As the CTM interrupt request flags will not be automatically cleared, they have 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. 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, MFF, 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.

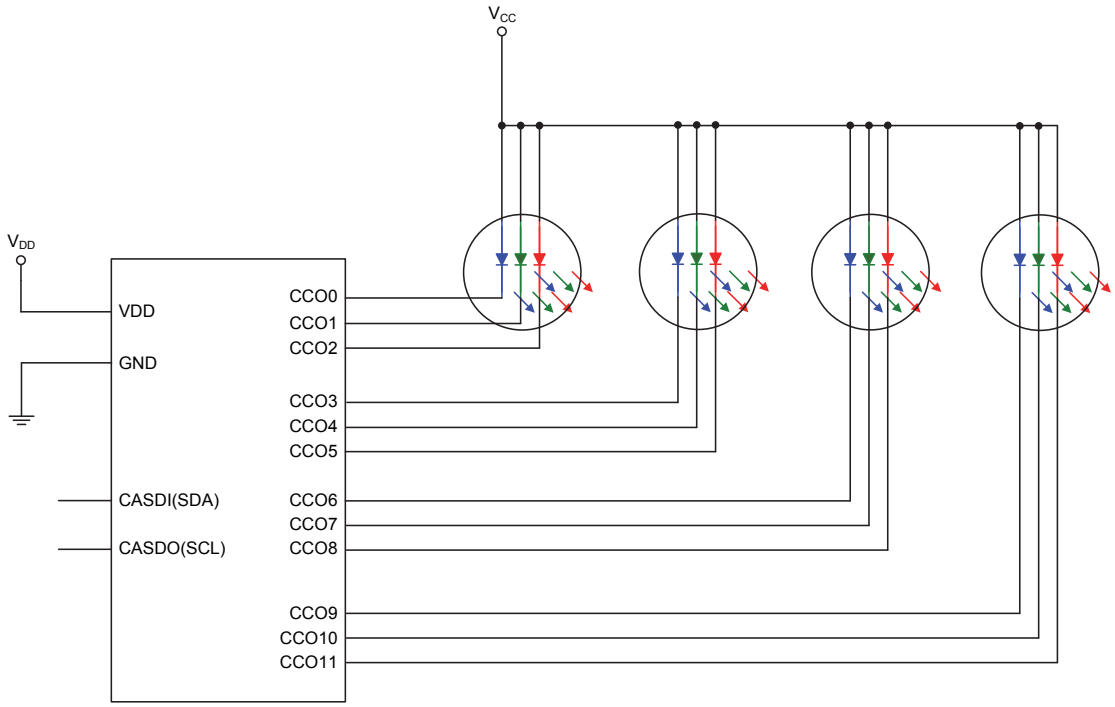
Every interrupt has the capability of waking up the microcontroller when it is in SLEEP or IDLE Mode, the wake up being generated when the interrupt request flag changes from low to high. If it is required to prevent a certain interrupt from waking up the microcontroller then its respective request flag 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.

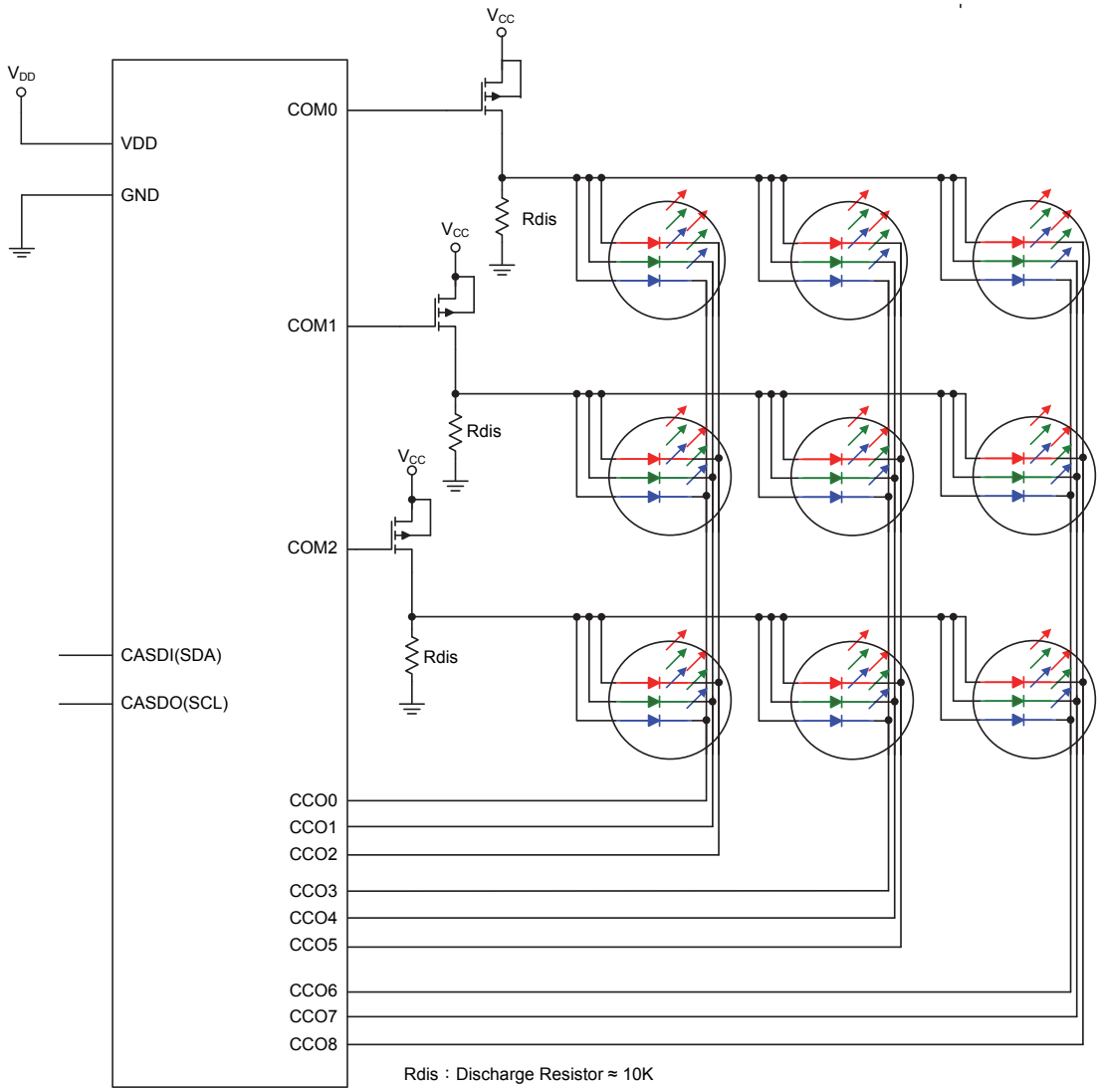
To return from an interrupt subroutine, either a RET or RETI instruction may be executed. The RETI instruction in addition to executing a return to the main program also automatically sets the EMI bit high to allow further interrupts. The RET instruction however only executes a return to the main program leaving the EMI bit in its present zero state and therefore disabling the execution of further interrupts.

Application Circuits

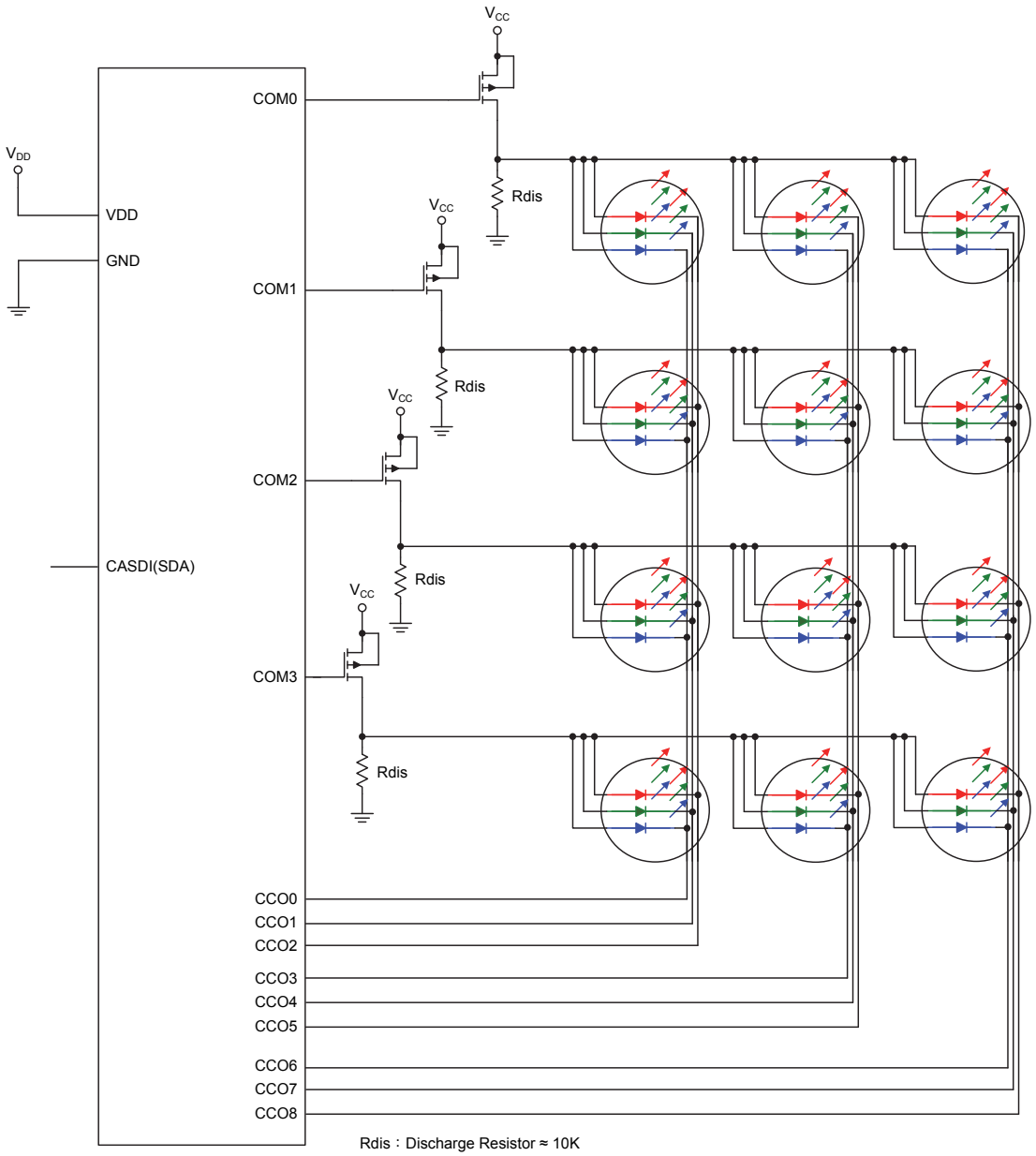
Direct Driving Mode



Time-shared Scanning Mode – 3×COM



Time-shared Scanning Mode – 4×COM



Rdis：Discharge Resistor ≈ 10K

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 three 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 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 set 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 following table depicts a summary of the instruction set categorised according to function and can be consulted as a basic instruction reference using the following listed conventions.

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
ADDM A,[m]	Add ACC to Data Memory	1[Note]	Z, C, AC, OV
ADD A,x	Add immediate data to ACC	1	Z, C, AC, OV
ADC A,[m]	Add Data Memory to ACC with Carry	1	Z, C, AC, OV
ADCM A,[m]	Add ACC to Data memory with Carry	1[Note]	Z, C, AC, OV
SUB A,x	Subtract immediate data from the ACC	1	Z, C, AC, OV
SUB A,[m]	Subtract Data Memory from ACC	1	Z, C, AC, OV
SUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory	1[Note]	Z, C, AC, OV
SBC A,[m]	Subtract Data Memory from ACC with Carry	1	Z, C, AC, OV
SBCM A,[m]	Subtract Data Memory from ACC with Carry, result in Data Memory	1[Note]	Z, C, AC, OV
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].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 or current page) to TBLH and Data Memory	2Note	None
TABRDL [m]	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 two 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.

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

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

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

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

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

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] ← $\overline{[m]}$

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 ← $\overline{[m]}$

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

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

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 bit i of Data Memory is not 0

Description If bit i 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].i ≠ 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

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

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

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 or current page) to TBLH and Data Memory

Description The low byte of the program code addressed by the table pointer (TBHP and TBLP or only TBLP if no 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

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

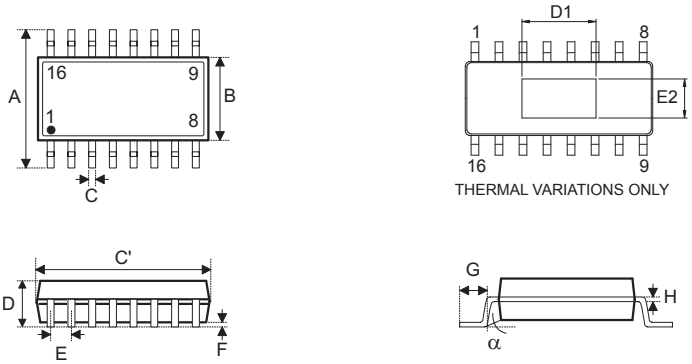
Package Information

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

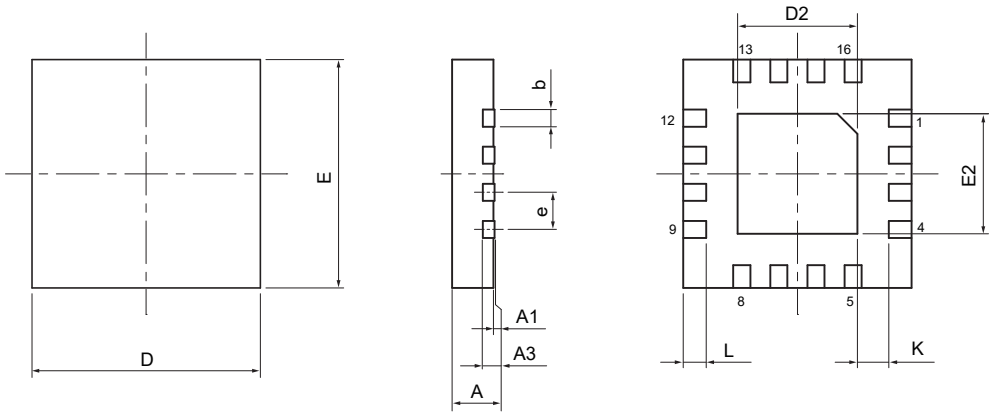
16-pin NSOP (150mil) Outline Dimensions (Exposed Pad)



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	—	0.236 BSC	—
B	—	0.154 BSC	—
D1	0.059	—	—
E2	0.039	—	—
C	0.012	—	0.020
C’	—	0.390 BSC	—
D	—	—	0.069
E	—	0.050 BSC	—
F	0.000	—	0.006
G	0.016	—	0.050
H	0.004	—	0.010
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	6.00 BSC	—
B	—	3.90 BSC	—
D1	1.50	—	—
E2	1.00	—	—
C	0.31	—	0.51
C’	—	9.90 BSC	—
D	—	—	1.75
E	—	1.27 BSC	—
F	0.00	—	0.15
G	0.40	—	1.27
H	0.10	—	0.25
α	0°	—	8°

SAW Type 16-pin QFN (3mm×3mm for FP0.25mm) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.028	0.030	0.031
A1	0.000	0.001	0.002
A3	—	0.008 BSC	—
b	0.007	0.010	0.012
D	—	0.118 BSC	—
E	—	0.118 BSC	—
e	—	0.020 BSC	—
D2	0.063	0.067	0.069
E2	0.063	0.067	0.069
L	0.008	0.010	0.012
K	0.008	—	—

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	0.700	0.750	0.800
A1	0.000	0.020	0.050
A3	—	0.200 BSC	—
b	0.180	0.250	0.300
D	—	3.000 BSC	—
E	—	3.000 BSC	—
e	—	0.50 BSC	—
D2	1.60	1.70	1.75
E2	1.60	1.70	1.75
L	0.20	0.25	0.30
K	0.20	—	—