1-Key Enhanced Touch I/O Flash MCU

BS83A01C

Revision: V1.20   Date: May 29, 2023

Table of Contents

Features

CPU Features

- Operating voltage
    - f_{SYS}=8MHz: 1.8V~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
- Oscillator types
    - Internal High Speed 8MHz RC – HIRC
    - Internal Low Speed 32kHz RC – LIRC
- Multi-mode operation: FAST, SLOW, IDLE and SLEEP
- Fully integrated internal oscillators require no external components
- All instructions executed in one or two instruction cycles
- Table read instructions
- 61 powerful instructions
- 2-level subroutine nesting
- Bit manipulation instruction

Peripheral Features

- Flash Program Memory: 512×14
- RAM Data Memory: 32×8
- Watchdog Timer function
- 4 bidirectional I/O lines
- Single Time-Base function for generation of fixed time interrupt signal
- 1 Touch Key function
- Low voltage reset function
- Package types: SOT23-6 and 8-pin SOP

General Description

This device is a Flash Memory I/O type 8-bit high performance RISC architecture microcontroller with fully integrated touch key function. With the touch key function provided internally and with the convenience of Flash Memory multi-programming features, this device has all the features to offer designers a reliable and easy means of implementing touch key within their product applications.

The touch key function is completely integrated eliminating the need for external components. In addition to the flash program memory, other memory includes an area of Data Memory. Protective features such as an internal Watchdog Timer and Low Voltage Reset coupled with excellent noise immunity and ESD protection ensure that reliable operation is maintained in hostile electrical environments.

The device includes fully integrated low and high speed oscillators which require no external components for their implementation. The ability to operate and switch dynamically between a range of operating modes using different clock sources gives users the ability to optimise microcontroller operation and minimise power consumption. The inclusion of flexible I/O programming features, Time Base and many other features further enhance device functionality and flexibility.

This touch key device will find excellent use in a huge range of modern touch key product applications such as single key switch and system wake-up applications.

Block Diagram

☐ : Pin-Shared Node

Pin Assignment

Top View

BS83A01C
SOT23-6-A

BS83A01C
8 SOP-A
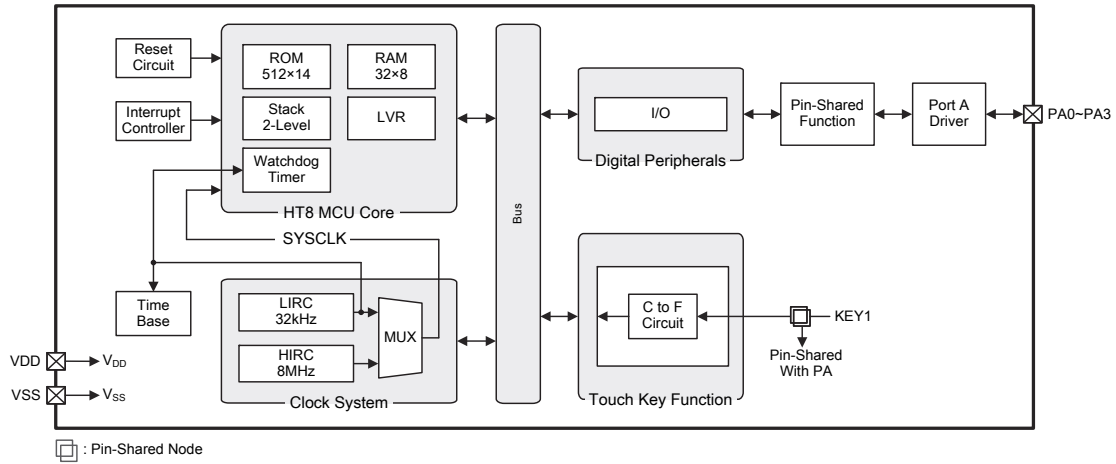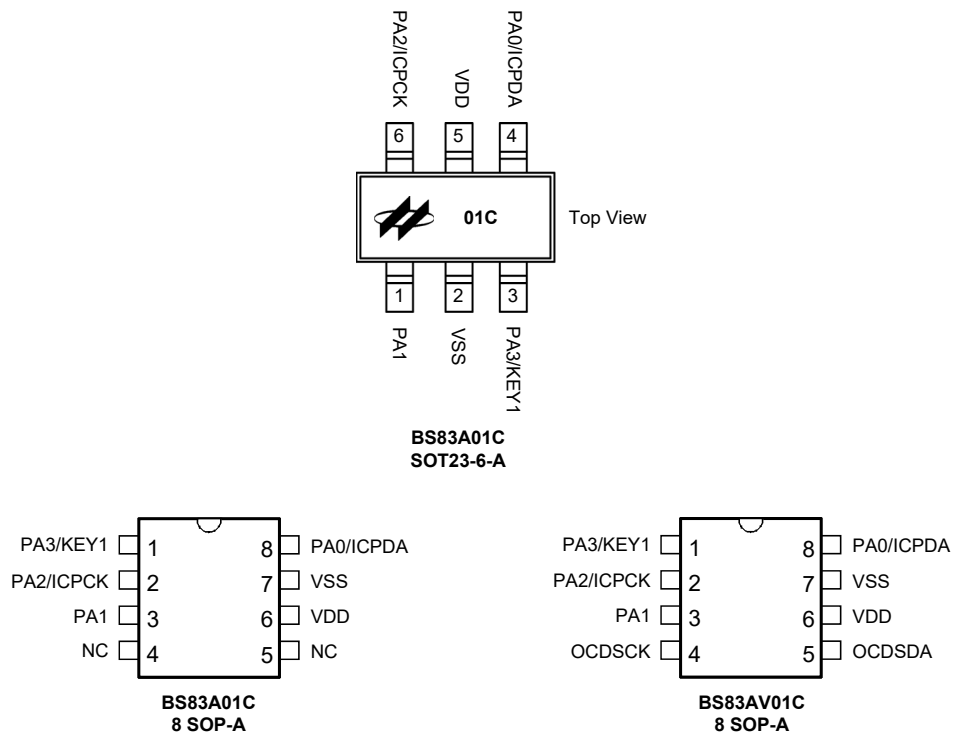
BS83AV01C
8 SOP-A

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 BS83AV01C is only available for the OCDS EV chips. The OCDSCK and OCDSDA pins are supplied for the OCDS dedicated pins.

3. SOT23-6 does not support OCDS EV Flexible Cable Converter.

Pin Description

The function of each pin is listed in the following table, however the details of each pin is contained in other sections of the datasheet.

Pin Name	Function	OPT	I/T	O/T	Description
PA0/ICPDA	PA0	PAWU PAPU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	ICPDA	—	ST	CMOS	In-circuit programming data/address pin
PA1	PA1	PAWU PAPU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
PA2/ICPCK	PA2	PAWU PAPU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	ICPCK	—	ST	—	In-circuit programming clock pin
PA3/KEY1	PA3	PAWU PAPU PAS	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	KEY1	TKM0C1 PAS	NSI	—	Touch key 1
NC	NC	—	—	—	No connection
VDD	VDD	—	PWR	—	Positive power supply
VSS	VSS	—	PWR	—	Negative power supply, ground
The following pins are only for the BS83AV01C					
OCDSCK	OCDSCK	—	ST	—	On-chip debug clock pin, for EVchip only
OCDSDA	OCDSDA	—	ST	CMOS	On-chip debug data/address pin, for EVchip only

Legend: I/T: Input type;   O/T: Output type;
     OPT: Optional by register option;   PWR: Power;
     ST: Schmitt Trigger input;   CMOS: CMOS output;
     NSI: Non-standard input.

Absolute Maximum Ratings

Supply Voltage .... V_{SS}-0.3V to 6.0V

Input Voltage .... V_{SS}-0.3V to V_{DD}+0.3V

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

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

I_{OH} Total .... -80mA

I_{OL} Total 80mA

Total Power Dissipation 500mW

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

D.C. Characteristics

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

Operating Voltage Characteristics

Ta=-40°C~85°C

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_{DD}	Operating Voltage – HIRC	f_{SYS}=8MHz	1.8	—	5.5	V
	Operating Voltage – LIRC	f_{SYS}=32kHz	1.8	—	5.5	

Standby Current Characteristics

Ta=25°C, unless otherwise specify

Symbol	Standby Mode	Test Conditions		Min.	Typ.	Max.	Max. @85°C	Unit
		V_{DD}	Conditions					
I_{STB}	SLEEP Mode	1.8V	WDT on	—	1.2	2.4	3.0	µA
		3V		—	1.5	3.0	3.7	
		5V		—	5	9	10	
	IDLE0 Mode – LIRC	1.8V	$f_{SUB}=f_{LIRC}$=on	—	2.4	4.0	4.6	µA
		3V		—	3.0	5.0	5.7	
		5V		—	5	10	11	
	IDLE1 Mode – HIRC	1.8V	f_{SUB} on, f_{SYS}=8MHz	—	0.6	1.0	1.0	mA
		3V		—	0.8	1.2	1.2	
		5V		—	1.0	2.0	2.0	

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

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

Operating Current Characteristics

Ta=25°C

Symbol	Operating Mode	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
I_{DD}	SLOW Mode – LIRC	1.8V	f_{SYS}=32kHz	—	4	8	µA
		3V		—	5	10	
		5V		—	15	30	
	FAST Mode – HIRC	1.8V	f_{SYS}=8MHz	—	0.6	1.0	mA
		3V		—	0.8	1.2	
		5V		—	1.6	2.4	

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

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

A.C. Characteristics

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

High Speed Internal Oscillator – HIRC – Frequency Accuracy

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

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Temp.				
f_{HIRC}	8MHz Writer Trimmed HIRC Frequency	1.8V~5.5V	25°C	-30%	8	+30%	MHz
			0°C~70°C	-40%	8	+40%	
			-40°C~85°C	-50%	8	+50%	
		2.2V~5.5V	25°C	-10%	8	+10%	
			0°C~70°C	-13%	8	+13%	
			-40°C~85°C	-18%	8	+18%	
		3V	25°C	-2%	8	+2%	
			0°C~70°C	-5%	8	+5%	
			-40°C~85°C	-7%	8	+7%	
		5V	25°C	-2%	8	+2%	
			0°C~70°C	-4%	8	+4%	
			-40°C~85°C	-5%	8	+5%	

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

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

Low Speed Internal Oscillator Characteristics – LIRC

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Temp.				
f_{LIRC}	LIRC Frequency	1.8V~5.5V	25°C	12.8	32	41.6	kHz
			-40°C~85°C	8	32	60	
		5V	25°C	25.6	32	38.4	kHz
t_{START}	LIRC Start Up Time	—	25°C	—	—	100	μs

Operating Frequency Characteristic Curves

System Start Up Time Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
t_{SST}	System Start-up Time Wake-up from Condition where f_{SYS} is Off	—	$f_{SYS}=f_H\sim f_H/64$, $f_H=f_{HIRC}$	—	16	—	t_{HIRC}
	System Start-up Time Wake-up from Condition where f_{SYS} is On	—	$f_{SYS}=f_H\sim f_H/64$, $f_H=f_{HIRC}$	—	2	—	t_H
t_{RSTD}	System Reset Delay Time Reset Source from Power-on Reset or LVR Hardware Reset	—	RR_{POR}=5V/ms	25	50	150	ms
	System Reset Delay Time WDTC Software Reset	—	—				
	System Reset Delay Time Reset Source from WDT Overflow	—	—	8.3	16.0	50.0	ms
t_{SRESET}	Minimum Software Reset Width to Reset	—	—	40	90	375	µs

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

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

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

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

Input/Output Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{IL}	Input Low Voltage for I/O Ports or Input Pins	5V	—	0	—	1.5	V
		—		0	—	$0.2V_{DD}$	
V_{IH}	Input High Voltage for I/O Ports or Input Pins	5V	—	3.5	—	5.0	V
		—		$0.8V_{DD}$	—	V_{DD}	
I_{OL}	Sink Current for I/O Pins	3V	V_{OL}=0.1V_{DD}	5	10	—	mA
		5V		10	20	—	
I_{OH}	Source Current for I/O Pins	3V	V_{OH}=0.9V_{DD}	-4	-5	—	mA
		5V		-8	-10	—	
R_{PH}	Pull-high Resistance for I/O Ports [Note]	3V	LVPU=0, PAPU=0FH	20	60	100	kΩ
		5V		10	30	50	
		3V	LVPU=1, PAPU=0FH	6.67	15.00	23.00	kΩ
		5V		3.5	7.5	12.0	
I_{LEAK}	Input Leakage Current	5V	$V_{IN}=V_{DD}$ or $V_{IN}=V_{SS}$	—	—	±1	µA

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

LVR Electrical Characteristics

Ta=-40°C~85°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{LVR}	Low Voltage Reset Voltage	—	LVR enable, voltage select 1.7V	-5%	1.7	+5%	V
I_{LVR}	Operating Current	3V	LVR enable, V_{LVR} = 1.7V	—	—	15	μA
		5V		—	15	25	
t_{LVR}	Minimum Low Voltage Width to Reset	—	—	140	600	1000	μs

Memory Characteristics

Ta=-40°C~85°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
Flash Program Memory							
V_{DD}	V_{DD} for Read	—	—	1.8	—	5.5	V
	V_{DD} for Erase / Write	—	—	3.0	—	5.5	
t_{DEW}	Erase / Write Cycle Time – Flash Program Memory	5.0V	—	—	2	3	ms
I_{DDPGM}	Programming / Erase Current on V_{DD}	5.0V	—	—	—	5.0	mA
E_P	Cell Endurance	—	—	100K	—	—	E/W
t_{RETD}	ROM Data Retention Time	—	Ta=25°C	—	40	—	Year
RAM Data Memory							
V_{DD}	V_{DD} for Read / Write	—	—	V_{DDmin}	—	V_{DDmax}	V
V_{DR}	RAM Data Retention Voltage	—	—	1.0	—	—	V

Note: "E/W" means Erase/Write times.

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
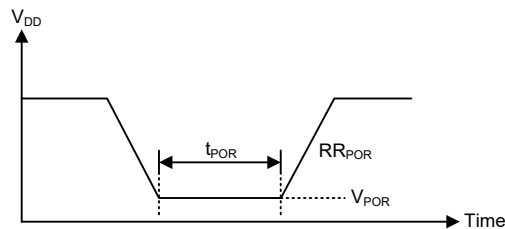
System Architecture

A key factor in the high-performance features of the Holtek range of microcontrollers is attributed to their internal system architecture. The range of the device take 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.
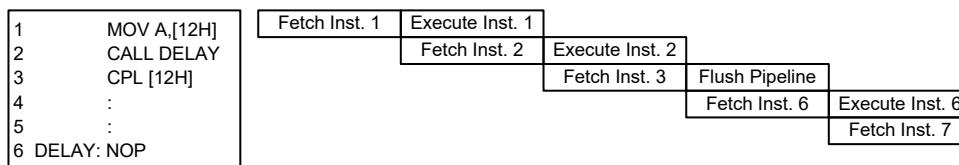
Clocking and Pipelining

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.
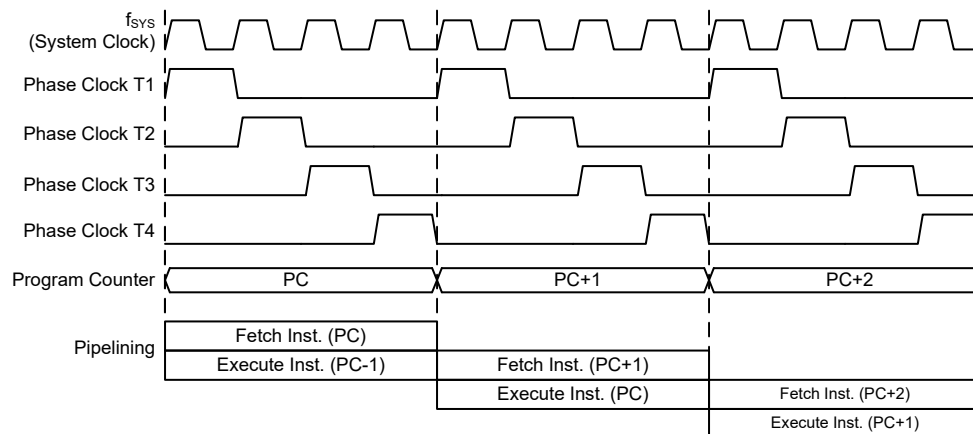
System Clocking and Pipelining

For instructions involving branches, such as jump or call instructions, two instruction 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.

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

Fetch Inst. 1	Execute Inst. 1
Fetch Inst. 2	Execute Inst. 2
Fetch Inst. 3	Flush Pipeline
Fetch Inst. 6	Execute Inst. 6
Fetch Inst. 7

Instruction Fetching

Program Counter – PC

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 demand 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	
High Byte	**Low Byte (PCL)**
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 2 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.
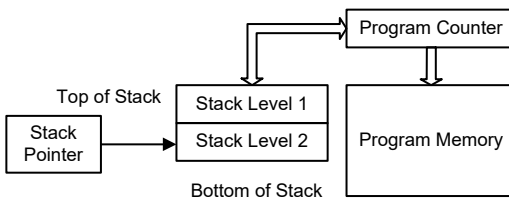
Arithmetic and Logic Unit – ALU

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 the 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 devices 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 512×14 bits. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries. Table data, which can be setup in any location within the Program Memory, is addressed by a separate table pointer register.
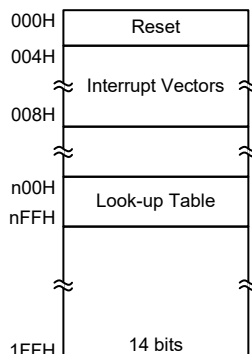
Program Memory Structure

Special Vectors

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

Look-up Table

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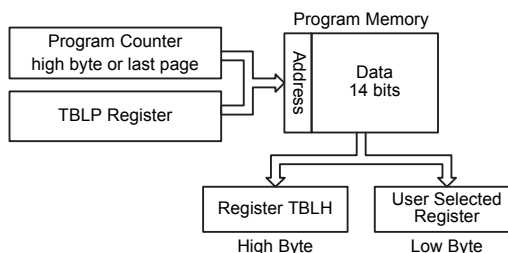
Table Program Example

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 "100H" which refers to the start address of the last page within the 512×14 Program Memory of the device. The table pointer is setup here to have an initial value of "06H". This will ensure that the first data read from the data table will be at the Program Memory address "106H" or 6 locations after the start of the last page. Note that the value for the table pointer is referenced to the specific page pointed by the TBLP register 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/write register and can be restored, care should be taken to ensure its protection if both the main routine and Interrupt Service Routine use table read instructions. If using the table read instructions, the Interrupt Service Routines may change the value of the TBLH and subsequently cause errors if used again by the main routine. As a rule it is recommended that simultaneous use of the table read instructions should be avoided. However, in situations where simultaneous use cannot be avoided, the interrupts should be disabled prior to the execution of any main routine table-read instructions. Note that all table related instructions require two instruction cycles to complete their operation.

Table Read Program Example

```
tempreg1 db ?        ; temporary register #1
tempreg2 db ?        ; temporary register #2
:
:
mov a,06h            ; initialize low table pointer - note that this address is referenced
mov tblp,a           ; to the last page or present page
:
:
tabrdl tempreg1      ; transfers value in table referenced by table pointer data at program
                     ; memory address "106H" transferred to tempreg1 and TBLH
dec tblp             ; reduce value of table pointer by one
tabrdl tempreg2      ; transfers value in table referenced by table pointer data at program
                     ; memory address "105H" transferred to tempreg2 and TBLH in this example the
                     ; data "1AH" is transferred to tempreg1 and data "0FH" to register tempreg2
:
:
org 100h             ; sets initial address of program memory
db 000Ah, 000Bh, 000Ch, 000Dh, 000Eh, 000Fh, 001Ah, 001Bh
:
:
```

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.

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

Holtek Write 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 PA0 and PA2 I/O pins for data and clock programming purposes. The user must there take care to ensure that no other outputs are connected to these two pins.
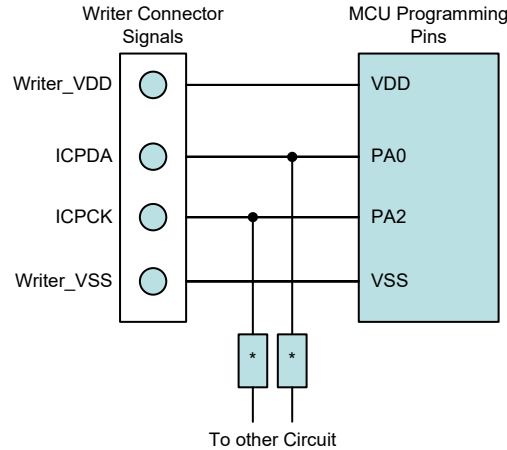
Note: * may be resistor or capacitor. The resistance of * must be greater than 1kΩ or the capacitance of * must be less than 1nF.

On-Chip Debug Support – OCDS

There is an EV chip named BS83AV01C which is used to emulate the real MCU device named BS83A01C. This EV chip device also provides an "On-Chip Debug" function to debug the device during the development process. The EV chip and the actual MCU device are almost functionally compatible except for the "On-Chip Debug" function and package type. 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 actual MCU device will have no effect in the EV chip. For a more detailed OCDS description, 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.

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

Structure

The Data Memory has a bank named Bank 0, which is implemented in 8-bit wide Memory. The Data Memory is subdivided into two sections, 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 3FH while the General Purpose Data Memory address range is from 40H to 5FH.

Special Purpose Data Memory		General Purpose Data Memory	
Located Bank	Bank: Address	Capacity	Bank: Address
0	0: 00H~3FH	32×8	0: 40H~5FH

Data Memory Summary
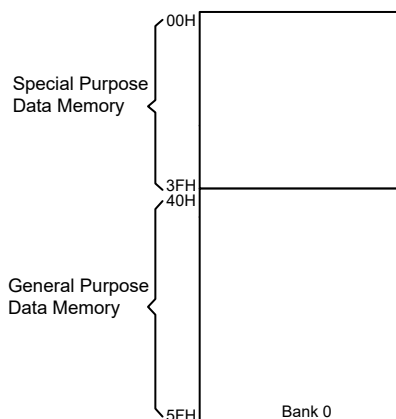
Data Memory Structure

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			Bank 0
00H	IAR0		20H	
01H	MP0		21H	
02H			22H	
03H			23H	
04H			24H	
05H	ACC		25H	
06H	PCL		26H	
07H	TBLP		27H	
08H	TBLH		28H	
09H			29H	
0AH	STATUS		2AH	
0BH	SCC		2BH	
0CH	HIRCC		2CH	
0DH			2DH	
0EH	INTC		2EH	
0FH	RSTFC		2FH	
10H			30H	
11H			31H	
12H			32H	
13H			33H	
14H	PA		34H	
15H	PAC		35H	TKTMR
16H	PAPU		36H	TKC0
17H	PAWU		37H	TK16DL
18H	LVPUC		38H	TK16DH
19H			39H	TKC1
1AH	WDTC		3AH	TKM016DL
1BH	TBC		3BH	TKM016DH
1CH	PSCR		3CH	TKM0ROL
1DH	PAS		3DH	TKM0ROH
1EH			3EH	TKM0C0
1FH			3FH	TKM0C1

☐ : Unused, read as 00H

Special Purpose Data Memory Structure

Special Function Register Description

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

Indirect Addressing Register – IAR0

The Indirect Addressing Register, IAR0, although having its location in normal RAM register, do not actually physically exist as normal register. The method of indirect addressing for RAM data manipulation is using this Indirect Addressing Register and Memory Pointer, in contrast to direct memory addressing, where the actual memory address is specified. Actions on the IAR0 register will result in no actual read or write operation to this register but rather to the memory location specified by its corresponding Memory Pointer, MP0. Acting as a pair, IAR0 and MP0 can together access data from Bank 0. As the Indirect Addressing Register is not physically implemented, reading the Indirect Addressing Register indirectly will return a result of "00H" and writing to the register directly will result in no operation.

Memory Pointer – MP0

The Memory Pointer, known as MP0 is provided. This Memory Pointer are physically implemented in the Data Memory and can be manipulated in the same way as normal register providing a convenient way with which to address and track data. When any operation to the relevant Indirect Addressing Register 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 only used to access data from Bank 0.

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                ; setup size of block
    mov block,a
    mov a,offset adres1      ; Accumulator loaded with first RAM address
    mov mp0,a                ; setup memory pointer with first RAM address
loop:
    clr IAR0                 ; clear the data at address defined by mp0
    inc mp0                  ; increment memory pointer
    sdz block                ; check if last memory location has been cleared
    jmp loop
continue:
```

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

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 Register – TBLP, TBLH

The TBLP and TBLH special function registers are used to control operation of the look-up table which is stored in the Program Memory. TBLP is the table pointer and indicates the location where the table data is located. Its value must be setup before any table read commands are executed. The 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 after a table read data instruction has been executed.

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 selections and operation 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 Interrupt. Two fully integrated internal oscillators, requiring no external components, are provided to form a wide range of both fast and slow system oscillators. The higher frequency oscillators provide higher performance but carry with it the disadvantage of higher power requirements, while the opposite is of course true for the lower frequency oscillators. With the capability of dynamically switching between fast and slow system clock, the device has the flexibility to optimize the performance/power ratio, a feature especially important in power sensitive portable applications.

Type	Name	Frequency
Internal High Speed RC	HIRC	8MHz
Internal Low Speed RC	LIRC	32kHz

Oscillator Types

System Clock Configurations

There are two methods of generating the system clock, one high speed oscillator and one low speed oscillator. The high speed oscillator is the internal 8MHz RC oscillator. The low speed oscillator is the internal 32kHz RC oscillator. 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 as the system clock can be dynamically selected.
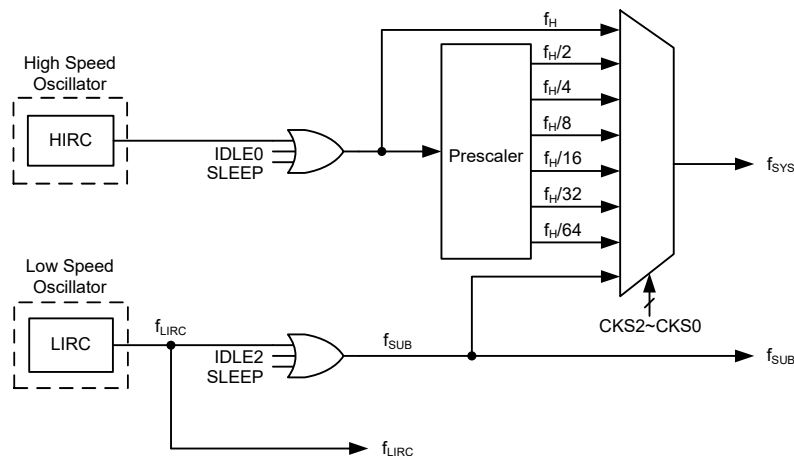
System Clock Configurations

Internal RC Oscillator – HIRC

The internal RC oscillator is a fully integrated system oscillator requiring no external components. The internal RC oscillator has a fixed frequency of 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.

Internal 32kHz Oscillator – LIRC

The LIRC is a fully self-contained free running on-chip RC oscillator with a typical frequency of 32kHz at 5V, 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 application. 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 this 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 options using register programming, a clock system can be configured to obtain maximum application performance.

The main system clock, can come from 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 the HIRC oscillator. The low speed system clock source is sourced from the LIRC oscillator. The other choice, which is a divided version of the high speed system oscillator has a range of $f_H/2 \sim f_H/64$.
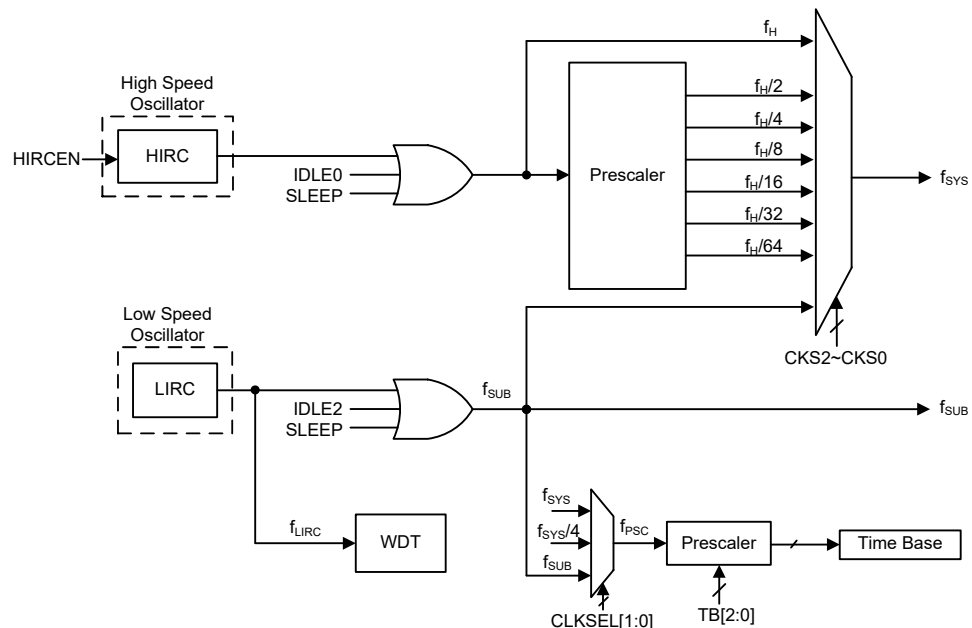
Device Clock Configurations

Note: When the system clock source f_{SYS} is switched to f_{SUB} from f_H, the high speed oscillator will be stopped to conserve the power. Thus there is no $f_H \sim f_H/64$ for peripheral circuit to use. which is determined by configuring the corresponding high speed oscillator enable control bit.

System Operation Modes

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

Operation Mode	CPU	Register Setting			f_{SYS}	f_H	f_{SUB}	f_{LIRC}
		FHIDEN	FSIDEN	CKS[2:0]				
FAST	On	x	x	000~110	$f_H \sim f_H/64$	On	On	On
SLOW	On	x	x	111	f_{SUB}	On/Off[1]	On	On
IDLE0	Off	0	1	000~110	Off	Off	On	On
				111	On			
IDLE1	Off	1	1	xxx	On	On	On	On
IDLE2	Off	1	0	000~110	On	On	Off	On
				111	Off			
SLEEP	Off	0	0	xxx	Off	Off	Off	On[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 will be switched on since the WDT function is always enabled in the SLEEP mode.

FAST Mode

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

SLOW Mode

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

SLEEP Mode

The SLEEP Mode is entered when an HALT instruction is executed and when the FHIDEN and FSIDEN bits are low. In the SLEEP mode the CPU will be stopped, and the f_{SUB} clock to peripheral will be stopped too. However the f_{LIRC} clock still continues to operate since the WDT function is always enabled.

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 system oscillator will be inhibited from driving the CPU but if the system oscillator is high speed system oscillator, it may continue to provide a clock source to keep some peripheral functions operational.

Control Register

The registers, SCC and HIRCC 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: HIRC unstable
　　　　1: HIRC stable

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

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

Operating Mode Switching

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

In simple terms, mode switching between the FAST Mode and SLOW Mode is executed using the CKS2~CKS0 bits in the SCC register while mode switching from the FAST/SLOW Modes to the SLEEP/IDLE Modes is executed via the HALT instruction. When 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.

FAST
$f_{SYS}=f_H{\sim}f_H/64$
f_H on
CPU run
f_{SYS} on
f_{SUB} on

SLOW
$f_{SYS}=f_{SUB}$
f_{SUB} on
CPU run
f_{SYS} on
f_H on/off

SLEEP
HALT instruction executed
CPU stop
FHIDEN=0
FSIDEN=0
f_H off
f_{SUB} off

IDLE0
HALT instruction executed
CPU stop
FHIDEN=0
FSIDEN=1
f_H off
f_{SUB} on

IDLE2
HALT instruction executed
CPU stop
FHIDEN=1
FSIDEN=0
f_H on
f_{SUB} off

IDLE1
HALT instruction executed
CPU stop
FHIDEN=1
FSIDEN=1
f_H on
f_{SUB} on
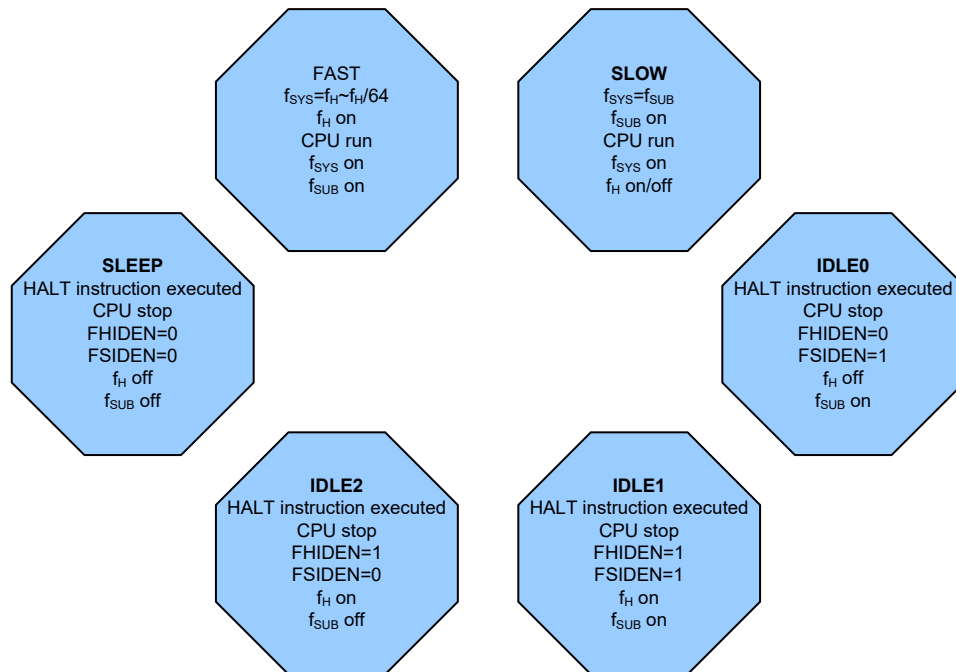
FAST Mode to SLOW Mode Switching

When running in the FAST Mode, which uses the high speed system oscillator, and therefore consumes more power, the system clock can switch to run in the SLOW Mode by set 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.

```
                                                              ┌─────────────┐
                                                              │  FAST Mode  │
                                                              └─────────────┘
                                          CKS2~CKS0=111
                                                              ┌─────────────┐
                                                              │  SLOW Mode  │
                                                              └─────────────┘
                              FHIDEN=0, FSIDEN=0
                              HALT instruction is executed
                                                              ┌─────────────┐
                                                              │ SLEEP Mode  │
                                                              └─────────────┘
                      FHIDEN=0, FSIDEN=1
                      HALT instruction is executed
                                                              ┌─────────────┐
                                                              │ IDLE0 Mode  │
                                                              └─────────────┘
              FHIDEN=1, FSIDEN=1
              HALT instruction is executed
                                                              ┌─────────────┐
                                                              │ IDLE1 Mode  │
                                                              └─────────────┘
      FHIDEN=1, FSIDEN=0
      HALT instruction is executed
                                                              ┌─────────────┐
                                                              │ IDLE2 Mode  │
                                                              └─────────────┘
```
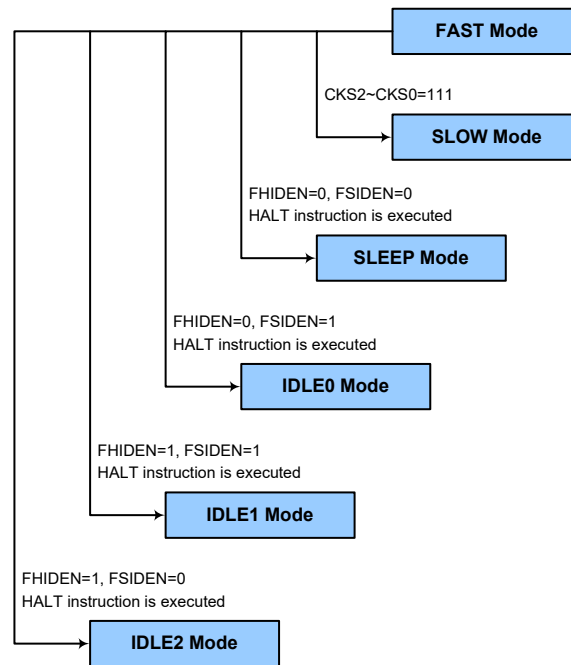
SLOW Mode to FAST Mode Switching

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

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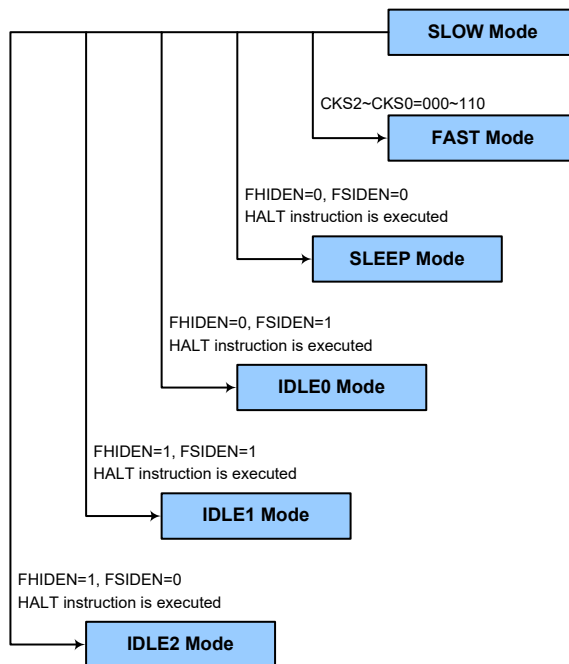
Entering the SLEEP Mode

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

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

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

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

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

Standby Current Considerations

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

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

In the IDLE1 and IDLE2 Mode the system oscillator is on, if the system oscillator is from the high speed system 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 IDLE Mode, where the system clock source to the CPU will be stopped. However when the device is woken up again, it can take a considerable time for the original system oscillator to restart, stabilise 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 SLEEP or IDLE 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 setup using the PAWU register to permit a negative transition on the pin to wake up the system. When a Port A pin wake-up occurs, the program will resume execution at the instruction following the "HALT" instruction. If the system is woken up by an interrupt, then two possible situations may occur. The first is where the related interrupt is disabled or the interrupt is enabled but the stack is full, in which case the program will resume execution at the instruction following the "HALT" instruction. In this situation, the interrupt which woke up the device will not be immediately serviced, but will rather be serviced later when the related interrupt is finally enabled or when a stack level becomes free. The other situation is where the related interrupt is enabled and the stack is not full, in which case the regular interrupt response takes place. If an interrupt request flag is set high before entering the SLEEP or IDLE Mode, the wake-up function of the related interrupt will be disabled.

Watchdog Timer

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

Watchdog Timer Clock Source

The Watchdog Timer clock source is provided by the internal clock f_{LIRC}, which is sourced from the LIRC oscillator. The LIRC internal oscillator has an approximate period frequency of 32kHz , it should be noted that 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 and reset operation.

• WDTC Register

A single register, WDTC, controls the required time-out period as well as the enable and reset MCU operation.

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	1	1	1

Bit 7~3   **WE4~WE0**: WDT function control
      10101B/01010B: 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**: Select WDT time-out period selection
      000: $[(2^8-2^0)\sim2^8]/f_{LIRC}$
      001: $[(2^9-2^1)\sim2^9]/f_{LIRC}$
      010: $[(2^{10}-2^2)\sim2^{10}]/f_{LIRC}$
      011: $[(2^{11}-2^3)\sim2^{11}]/f_{LIRC}$
      100: $[(2^{12}-2^4)\sim2^{12}]/f_{LIRC}$
      101: $[(2^{13}-2^5)\sim2^{13}]/f_{LIRC}$
      110: $[(2^{14}-2^6)\sim2^{14}]/f_{LIRC}$
      111: $[(2^{15}-2^7)\sim2^{15}]/f_{LIRC}$

These three bits determine the division ratio of the Watchdog Timer source clock, which in turn determines the time-out period.

• RSTFC Register

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

"x": unknown

Bit 7~3   Unimplemented, read as "0"

Bit 2   **LVRF**: LVR function reset flag
      Refer to the Low Voltage Reset section

Bit 1   Unimplemented, read as "0"

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

          0: Not occur
          1: Occurred

This bit is set high by the WDT Control register software reset and cleared by the application program. Note that this bit can only be cleared to zero by the application program.

Watchdog Timer Operation

The Watchdog Timer operates by providing a device reset when its timer overflows. This means that in the application program and during normal operation the user has to strategically clear the Watchdog Timer before it overflows to prevent the Watchdog Timer from executing a reset. This is done using the clear watchdog instruction.

If the program malfunctions for whatever reason, jumps to an unknown location, or enters an endless loop, the clear instruction will not be executed in the correct manner, in which case the Watchdog Timer will overflow and reset the device. With regard to the Watchdog Timer enable and reset function, there are five bits, WE4~WE0, in the WDTC register to offer the enable and reset control of the Watchdog Timer. The WDT function will be enabled if the WE4~WE0 bits are equal to 01010B or 10101B. If the WE4~WE0 bits are set to any other values, other than 01010B and 10101B, it will reset the device after t_{SRESET} time. After power on these bits will have a value of 01010B.

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

Watchdog Timer Function 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 WDT software reset, which means a certain value except 01010B and 10101B written into the WE4~WE0 field. The second is using the Watchdog Timer software clear instruction. The third is via an HALT instruction.

To clear the Watchdog Timer is to use the single "CLR WDT" instruction. A simple execution of "CLR WDT" will clear the WDT.

The maximum time out period is when the 2^{15} division ratio is selected. As an example, with a LIRC oscillator as its source clock, this will give a maximum watchdog period of around 1 second for the 2^{15} division ratio, and a minimum timeout of 8ms for the 2^8 division ration.

Watchdog Timer
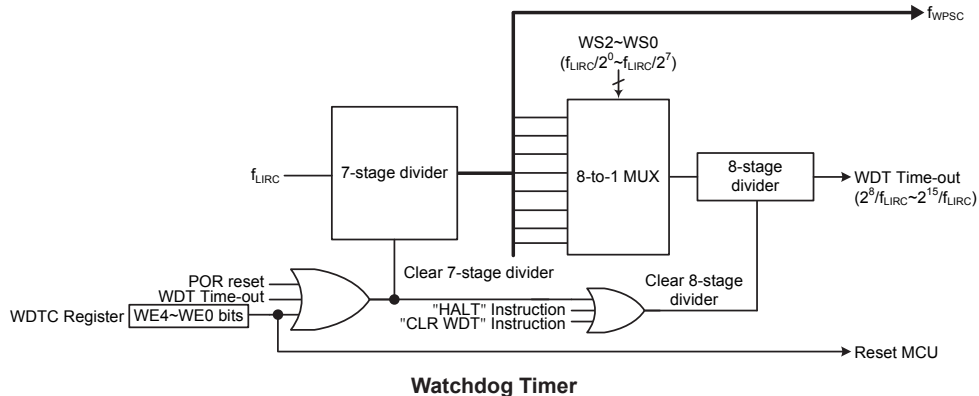
Reset and Initialisation

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 reset exists in the form of a Low Voltage Reset, LVR, where a full reset is implemented in situations where the power supply voltage falls below a certain threshold.

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

Reset Functions

There are three ways in which a microcontroller reset can occur, through events occurring internally.

Power-on Reset

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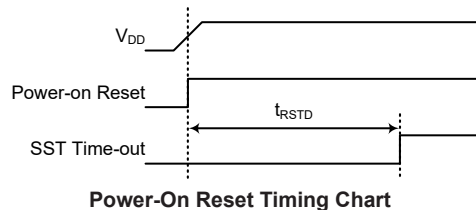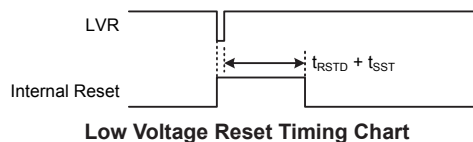

Power-On Reset Timing Chart

Low Voltage Reset – LVR

The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device and provides an MCU reset should the value fall below a certain predefined level. If the supply voltage of the device drops to within a range of $0.9V \sim V_{LVR}$ such as might occur when changing the battery in battery powered applications, the LVR will automatically reset the device internally and the LVRF bit in the RSTFC register will also be set high. For a valid LVR signal, a low supply voltage, i.e., a voltage in the range between $0.9V \sim V_{LVR}$ must exist for a time greater than that specified by t_{LVR} in the LVR characteristics. If the low supply voltage state does not exceed this value, the LVR will ignore the low supply voltage and will not perform a reset function.

Note that the LVR function will be automatically disabled when the device enters the IDLE or SLEEP mode.


Low Voltage Reset Timing Chart

- **RSTFC Register**

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

"x": unknown

Bit 7~3   Unimplemented, read as "0"

Bit 2   **LVRF**: LVR function reset flag
0: Not occurred
1: Occurred
This bit is set high when a specific Low Voltage Reset situation condition occurs. This bit can only be cleared to zero by the application program.

Bit 1   Unimplemented, read as "0"

Bit 0   **WRF**: WDT control register software reset flag
Refer to the Watchdog Timer Control Register section.

Watchdog Time-out Reset during Normal Operation

When the Watchdog time-out Reset during normal operations in the FAST or SLOW mode occurs, the Watchdog time-out flag TO will be set to "1".

WDT Time-out Reset during Normal Operation Timing Chart
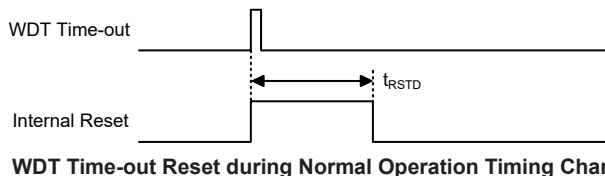
Watchdog Time-out during SLEEP or IDLE Mode

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 System Start Up Time Characteristics for t_{SST} details.

WDT Time-out Reset during SLEEP or IDLE Timing Chart
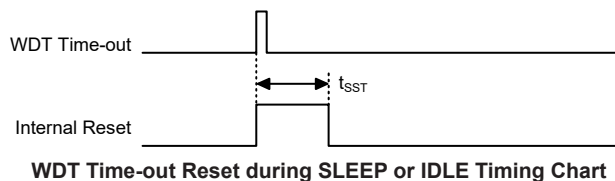
Reset Initial Conditions

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
u	u	LVR reset during FAST or SLOW Mode operation
1	u	WDT time-out reset during FAST or SLOW Mode operation
1	1	WDT time-out reset during IDLE or SLEEP Mode operation

"u" stands for unchanged

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

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

The different kinds of resets all affect the internal registers of the microcontroller in different ways. To ensure reliable continuation of normal program execution after a reset occurs, it is important to know what condition the microcontroller is in after a particular reset occurs. The following table describes how each type of reset affects each of the microcontroller internal registers.

Register	Reset (Power On)	WDT Time-out (Normal Operation)	WDT Time-out (IDLE/SLEEP)
IAR0	xxxx xxxx	uuuu uuuu	uuuu uuuu
MP0	1xxx xxxx	1uuu uuuu	1uuu uuuu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu
PCL	0000 0000	0000 0000	0000 0000
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu
TBLH	--xx xxxx	--uu uuuu	--uu uuuu
STATUS	--00 xxxx	--1u uuuu	--11 uuuu
SCC	000- --00	000- --00	uuu- --uu
HIRCC	---- --01	---- --01	---- --uu
INTC	--00 -000	--00 -000	--uu -uuu
RSTFC	---- -x-0	---- -u-u	---- -u-u
PA	---- 1111	---- 1111	---- uuuu
PAC	---- 1111	---- 1111	---- uuuu
PAPU	---- 0000	---- 0000	---- uuuu
PAWU	---- 0000	---- 0000	---- uuuu
LVPUC	---- ---0	---- ---0	---- ---u
WDTC	0101 0111	0101 0111	uuuu uuuu
TBC	0--- -000	0--- -000	u--- -uuu
PSCR	---- --00	---- --00	---- --uu
PAS	00-- ----	00-- ----	uu-- ----
TKTMR	0000 0000	0000 0000	uuuu uuuu
TKC0	-000 0000	-000 0000	-uuu uuuu
TK16DL	0000 0000	0000 0000	uuuu uuuu
TK16DH	0000 0000	0000 0000	uuuu uuuu
TKC1	---- --11	---- --11	---- --uu
TKM016DL	0000 0000	0000 0000	uuuu uuuu
TKM016DH	0000 0000	0000 0000	uuuu uuuu
TKM0ROL	0000 0000	0000 0000	uuuu uuuu
TKM0ROH	---- --00	---- --00	---- --uu
TKM0C0	--00 0000	--00 0000	--uu uuuu
TKM0C1	0-00 ---0	0-00 ---0	u-uu ---u

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

Input/Output Ports

The microcontrollers offer considerable flexibility on their I/O ports. Most pins can have either an input or output designation under user program control. Additionally, as there are pull-high resistors and wake-up software configurations, 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. 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	—	—	—	—	PA3	PA2	PA1	PA0
PAC	—	—	—	—	PAC3	PAC2	PAC1	PAC0
PAPU	—	—	—	—	PAPU3	PAPU2	PAPU1	PAPU0
PAWU	—	—	—	—	PAWU3	PAWU2	PAWU1	PAWU0
LVPUC	—	—	—	—	—	—	—	LVPU

"—": Unimplemented, read as "0"

I/O Logic Function Registers List

Pull-high Resistors

Many product applications require pull-high resistors for their switch inputs usually requiring the use of an external resistor. To eliminate the need for these external resistors, all I/O pins, when configured as an digital input have the capability of being connected to an internal pull-high resistor. These pull-high resistors are selected using registers PAPU and LVPUC etc. and are implemented using weak PMOS transistors. The PAPU register is used to determine whether the pull-high function is enabled or not while the LVPUC register is used to select the pull-high resistors value for low voltage power supply applications.

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.

• PAPU Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	PAPU3	PAPU2	PAPU1	PAPU0
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   **PAPU3~PAPU0**: PA3~PA0 pull-high resistor control
    0: Disable
    1: Enable

• **LVPUC Register**

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

Bit 7~1   Unimplemented, read as "0"

Bit 0   **LVPU**: Pull-high resistor select when low voltage power supply
        0: All pin pull high resistor is 60kΩ (typ.) @ 3V
        1: All pin pull high resistor is 15kΩ (typ.) @ 3V
     This bit is used to select the pull-high resistor value for low voltage power supply applications. The LVPU bit is only available when the corresponding pin pull-high function is enabled by setting the relevant pull-high control bit high. This bit will have no effect when the pull-high function is disabled.

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	—	—	—	—	PAWU3	PAWU2	PAWU1	PAWU0
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   **PAWU3~PAWU0**: PA3~PA0 wake-up function control
        0: Disable
        1: Enable

I/O Port Control Register

The I/O port has its own control register known as PAC, to control the input/output configuration. With this control register, each CMOS output or input with or without pull-high resistor structures can be reconfigured dynamically under software control. Each pin of the I/O ports is directly mapped to a bit in its associated port control register. For the I/O pin to function as an input, the corresponding bit of the control register must be written as a "1". This will then allow the logic state of the input pin to be directly read by instructions. When the corresponding bit of the control register is written as a "0", the I/O pin will be setup as a CMOS output. If the pin is currently setup as an output, instructions can still be used to read the output register. However, it should be noted that the program will in fact only read the status of the output data latch and not the actual logic status of the output pin.

• PAC Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	PAC3	PAC2	PAC1	PAC0
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	1	1	1	1

Bit 7~4   Unimplemented, read as "0"

Bit 3~0   **PAC3~PAC0**: PA3~PA0 Input/Output Control
    0: Output
    1: Input

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 "A" pin shared function selection registers, labeled as PAS, which can select the desired functions of the multi-function pin-shared pins.

The most important point to note is to make sure that the desired pin-shared function is properly selected and also deselected. For most pin-shared functions, to select the desired pin-shared function, the pin-shared function should first be correctly selected using the corresponding pin-shared control register. After that the corresponding peripheral functional setting should be configured and then the peripheral function can be enabled. However, 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 setup 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.

• PAS Register

Bit	7	6	5	4	3	2	1	0
Name	PAS7	PAS6	—	—	—	—	—	—
R/W	R/W	R/W	—	—	—	—	—	—
POR	0	0	—	—	—	—	—	—

Bit 7~6   **PAS7~PAS6**: Pin-shared function selection
    00: PA3
    01: PA3
    10: PA3
    11: KEY1

Bit 5~0   Unimplemented, read as "0"

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
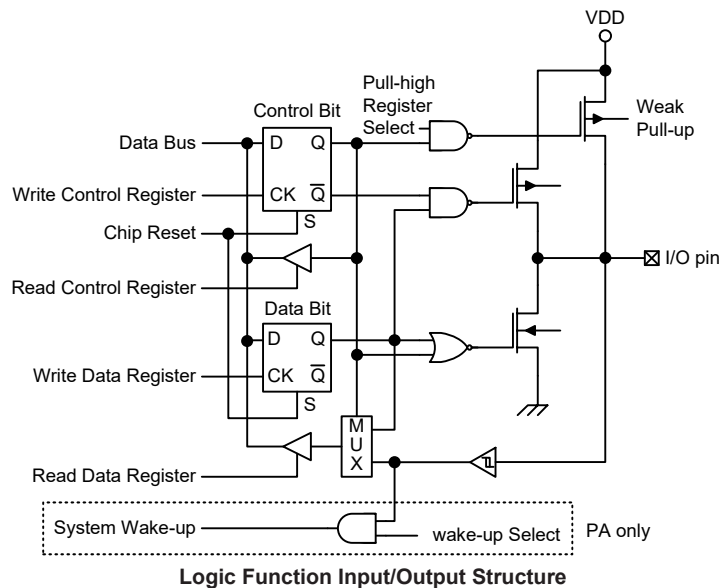
Programming Considerations

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

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

Touch Key Function

The device provides one Touch Key function. The Touch Key function is fully integrated and requires no external components, allowing Touch Key functions to be implemented by the simple manipulation of internal registers.

Touch Key Structure

The touch key is pin-shared with the I/O pins, with the desired function chosen via the corresponding selection register bits. Key is organised into one group, known as a module. The module is a fully independent set of one Touch Key has its own oscillator. The module contains its own control logic circuits and register set.

Total Key Number	Touch Key Module 0	Touch Key	Shared I/O Pin
1	M0	Key 1	PA3

Touch Key Structure

Touch Key Register Definition

The touch key module, which contains one Touch Key function, has its own suite registers. The following table shows the register set for the Touch Key module.

Register Name	Description
TKTMR	Touch key time slot 8-bit counter preload register
TKC0	Touch key function control register 0
TKC1	Touch key function control register 1
TK16DL	Touch key 16-bit counter low byte
TK16DH	Touch key 16-bit counter high byte
TKM016DL	Touch key module 0 16-bit counter C/F low byte
TKM016DH	Touch key module 0 16-bit counter C/F high byte
TKM0ROL	Touch key module 0 reference oscillator capacitor select low byte
TKM0ROH	Touch key module 0 reference oscillator capacitor select high byte
TKM0C0	Touch key module 0 control register 0
TKM0C1	Touch key module 0 control register 1

Touch Key Function Register Definition

Register Name	Bit							
	7	6	5	4	3	2	1	0
TKTMR	D7	D6	D5	D4	D3	D2	D1	D0
TKC0	—	TKRCOV	TKST	TKCFOV	TK16OV	D2	TK16S1	TK16S0
TK16DL	D7	D6	D5	D4	D3	D2	D1	D0
TK16DH	D15	D14	D13	D12	D11	D10	D9	D8
TKC1	—	—	—	—	—	—	TKFS1	TKFS0
TKM016DL	D7	D6	D5	D4	D3	D2	D1	D0
TKM016DH	D15	D14	D13	D12	D11	D10	D9	D8
TKM0ROL	D7	D6	D5	D4	D3	D2	D1	D0
TKM0ROH	—	—	—	—	—	—	D9	D8
TKM0C0	—	—	M0DFEN	M0FILEN	M0SOFC	M0SOF2	M0SOF1	M0SOF0
TKM0C1	M0TSS	—	M0ROEN	M0KOEN	—	—	—	M0K1EN

Touch Key Function Register List

• **TKTMR 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**: Touch Key time slot 8-bit counter preload register

The touch key time slot counter preload register is used to determine the touch key time slot overflow time. The time slot unit period is obtained by a 5-bit counter and equal to 32 time slot clock cycles. Therefore, the time slot counter overflow time is equal to the following equation shown.

Time slot counter overflow time=$(256-TKTMR[7:0])\times32t_{TSC}$, where t_{TSC} is the time slot counter clock period.

• **TKC0 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	TKRCOV	TKST	TKCFOV	TK16OV	D2	TK16S1	TK16S0
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   **TKRCOV**: Time slot counter overflow flag
      0: No overflow
      1: Overflow

This bit can be accessed by application program. When this bit is set by touch key time slot counter overflow, the corresponding touch key interrupt request flag will be set. However, if this bit is set by application program, the touch key interrupt request flag will not be affected. Therefore, this bit cannot be set by application program but must be cleared to 0 by application program.

If the module time slot counter overflows, the TKRCOV bit and the Touch Key Interrupt request flag, TKMF, will be set and the module key oscillators and reference oscillators will automatically stop. The touch key module 16-bit C/F counter, touch key function 16-bit counter, 5-bit time slot unit period counter and 8-bit time slot counter will be automatically switched off.

Bit 5   **TKST**: Touch key detection Start control
      0: Stopped
      0→1: Started

The touch key module 16-bit C/F counter, touch key function 16-bit counter and 5-bit time slot unit period counter will automatically be cleared when this bit is cleared to zero. However, the 8-bit programmable time slot counter will not be cleared. When this bit is changed from low to high, the touch key module 16-bit C/F counter, touch key function 16-bit counter, 5-bit time slot unit period counter and 8-bit time slot counter will be switched on together with the key and reference oscillators to drive the corresponding counters.

Bit 4   **TKCFOV**: Touch Key module 16-bit C/F counter overflow flag
      0: Not overflow
      1: Overflow

This bit is set high by the touch key module 16-bit C/F counter overflow and must be cleared to 0 by application programs.

Bit 3   **TK16OV**: Touch Key module 16-bit counter overflow flag
      0: Not overflow
      1: Overflow

This bit is set high by the touch key function 16-bit counter overflow and must be cleared to 0 by application programs.

Bit 2   **D2**: Reserved, cannot be used

Bit 1~0   **TK16S1~TK16S0**: Touch Key function 16-bit counter clock source select
00: f_{SYS}
01: $f_{SYS}/2$
10: $f_{SYS}/4$
11: $f_{SYS}/8$

• **TKC1 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	TKFS1	TKFS0
R/W	—	—	—	—	—	—	R/W	R/W
POR	—	—	—	—	—	—	1	1

Bit 7~2   Unimplemented, read as "0"

Bit 1~0   **TKFS1~TKFS0**: Touch key oscillator and Reference oscillator frequency select
00: 1MHz
01: 3MHz
10: 7MHz
11: 11MHz

• **TK16DH/TK16DL – Touch Key Function 16-bit Counter Register Pair**

Register	TK16DH								TK16DL							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register pair is used to store the touch key function 16-bit counter value. This 16-bit counter can be used to calibrate the reference or key oscillator frequency. When the touch key time slot counter overflows, this 16-bit counter will be stopped and the counter content will be unchanged. This register pair will be cleared to zero when the TKST bit is set low.

• **TKM016DH/TKM016DL – Touch Key Module 0 16-bit C/F Counter Register Pair**

Register	TKM016DH								TKM016DL							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Name	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This register pair is used to store the touch key module 16-bit C/F counter value. This 16-bit C/F counter will be stopped and the counter content will be kept unchanged when the touch key time slot counter overflows. This register pair will be cleared to zero when the TKST bit is set low.

• **TKM0ROH/TKM0ROL – Touch Key Module 0 Reference Oscillator Capacitor Select Register Pair**

Register	TKM0ROH								TKM0ROL							
Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	—	D9	D8	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	R/W	R/W
POR	—	—	—	—	—	—	0	0	0	0	0	0	0	0	0	0

This register pair is used to store the touch key module reference oscillator capacitor value.

The reference oscillator internal capacitor value=(TKM0RO[9:0]×30pF)/1024

• **TKM0C0 Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	M0DFEN	M0FILEN	M0SOFC	M0SOF2	M0SOF1	M0SOF0
R/W	—	—	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	—	0	0	0	0	0	0

Bit 7~6   Unimplemented,read as "0"

Bit 5   **M0DFEN**: Touch key module 0 multi-frequency control
　　　　0: Disable
　　　　1: Enable

Bit 4   **M0FILEN**: Touch key module 0 filter function control
　　　　0: Disable
　　　　1: Enable

Bit 3   **M0SOFC**: Touch key module 0 C/F oscillator frequency hopping function control select
　　　　0: Controlled by the M0SOF2~M0SOF0
　　　　1: Controlled by hardware circuit
　　　This bit is used to select the touch key oscillator frequency hopping function control method. When this bit is set to 1, the key oscillator frequency hopping function is controlled by the hardware circuit regardless of the M0SOF2~M0SOF0 bits value.

Bit 2~0   **M0SOF2~M0SOF0**: Touch key module 0 Reference and Key oscillators hopping frequency select (M0SOFC=0)
　　　　000: 1.1020MHz
　　　　001: 1.040MHz
　　　　010: 1.059MHz
　　　　011: 1.074MHz
　　　　100: 1.085MHz
　　　　101: 1.099MHz
　　　　110: 1.111MHz
　　　　111: 1.125MHz
　　　These bits are used to select the touch key oscillator frequency for the hopping function. Note that these bits are only available when the M0SOFC bit is cleared to 0. The frequency mentioned here will be changed when the external or internal capacitor is with different values. If the touch key operates at 1MHz frequency, users can adjust the frequency in scale when any other frequency is selected.

• **TKM0C1 Register**

Bit	7	6	5	4	3	2	1	0
Name	M0TSS	—	M0ROEN	M0KOEN	—	—	—	M0K1EN
R/W	R/W	—	R/W	R/W	—	—	—	R/W
POR	0	—	0	0	—	—	—	0

Bit 7   **M0TSS**: Touch key module 0 time slot counter clock source select
　　　　0: Touch key module 0 reference oscillator
　　　　1: $f_{SYS}/4$

Bit 6   Unimplemented, read as "0"

Bit 5   **M0ROEN**: Touch key module 0 reference oscillator enable control
　　　　0: Disable
　　　　1: Enable

Bit 4   **M0KOEN**: Touch key module 0 Key oscillator enable control
　　　　0: Disable
　　　　1: Enable

Bit 3~1   Unimplemented, read as "0"

Bit 0   **M0K1EN**: Touch key module 0 Key 1 enable control
　　　　0: Disable
　　　　1: Enable

Touch Key Operation

When a finger touches or is in proximity to a touch pad, the capacitance of the pad will increase. By using this capacitance variation to change slightly the frequency of the internal sense oscillator, touch actions can be sensed by measuring these frequency changes. Using an internal programmable divider the reference clock is used to generate a fixed time period. By counting the number of generated clock cycles from the sense oscillator during this fixed time period touch key actions can be determined.
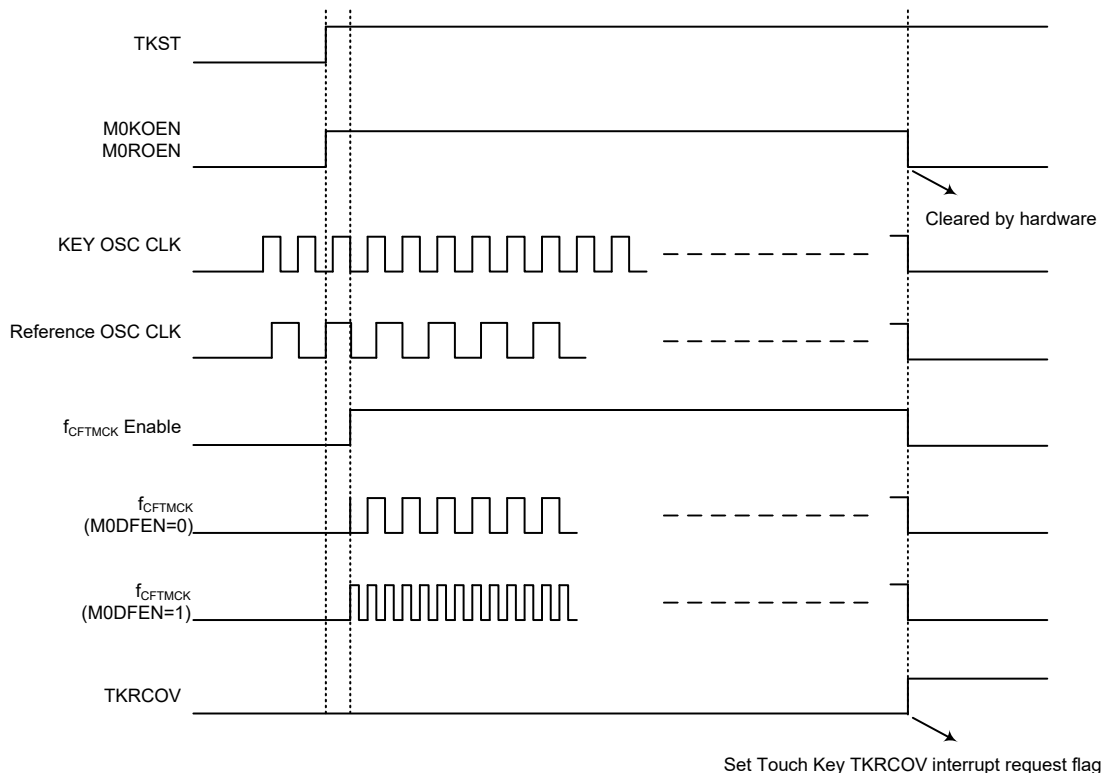
Touch Key Scan Mode Timing Diagram

The touch key module 0 contains one touch key input, namely KEY1, which is shared with logical I/O pin, and the desired function is selected using register bit. The touch key has its own independent sense oscillator. Therefore, there is one sense oscillator within the touch key module 0.

During this reference clock fixed interval, the number of clock cycles generated by the sense oscillator is measured, and it is this value that is used to determine if a touch action has been made or not. At the end of the fixed reference clock time interval a Touch Key interrupt signal will be generated.

The touch key module use the same started signal, TKST, in the TKC0 register. The touch key module 16-bit C/F counter, 16-bit counter, 5-bit time slot unit period counter in the module will be automatically cleared when the TKST bit is cleared to zero, but the 8-bit programmable time slot counter will not be cleared. The overflow time is setup by user. When the TKST bit changes from low to high, the 16-bit C/F counter, 16-bit counter, 5-bit time slot unit period counter and 8-bit time slot timer counter will be automatically switched on.

The key oscillator and reference oscillator in the module will be automatically stopped and the 16-bit C/F counter, 16-bit counter, 5-bit time slot unit period counter and 8-bit time slot timer counter will be automatically switched off when the time slot counter overflows. The clock source for the time slot counter is sourced from the reference oscillator or $f_{SYS}/4$ which is selected using the M0TSS bit

in the TKM0C1 register. The reference oscillator and key oscillator will be enabled by setting the M0ROEN bit and M0KOEN bits in the TKM0C1 register.

When the time slot counter in the touch key module 0 overflows, an actual touch key interrupt will take place.

Touch Key Interrupt

The Touch Key only has single interrupt, when the time slot counter in the Touch Key module 0 overflows, an actual Touch Key interrupt will take place. The 16-bit C/F counter, 16-bit counter, 5-bit time slot counter and 8-bit time slot counter in all modules will be automatically cleared. More details regarding the Touch Key interrupt is located in the interrupt section of the datasheet.

Programming Considerations

After the relevant registers are setup, the touch key detection process is initiated by changing the TKST bit from low to high. This will enable and synchronise all relevant oscillators. The TKRCOV flag which is the time slot counter flag will go high when the counter overflows. When this happens an interrupt signal will be generated. As the TKRCOV flag will not be automatically cleared, it has to be cleared by the application program.

The TKCFOV flag which is the 16-bit C/F counter overflow flag will go high when any of the Touch Key Module 16-bit C/F counter overflows. As this flag will not be automatically cleared, it has to be cleared by the application program. The TK16OV flag which is the 16-bit counter overflow flag will go high when the 16-bit counter overflows. As this flag will not be automatically cleared, it has to be cleared by the application program.

When the external touch key size and layout are defined, their related capacitances will then determine the sensor oscillator frequency.

Interrupts

Interrupts are an important part of any microcontroller system. When an external event or an internal function such as a touch action requires microcontroller attention, their corresponding interrupt will enforce a temporary suspension of the main program allowing the microcontroller to direct attention to their respective needs. The device contains several internal interrupt function. The internal interrupts are generated by various internal functions such as the Touch Key and Time base, etc.

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 the INTC register, located in the Special Purpose Data Memory.

The 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 interrupt followed by either an "E" for enable/disable bit or "F" for request flag.

Function	Enable Bit	Request Flag
Global	EMI	—
Touch Key Module	TKME	TKMF
Time Base	TBE	TBF

Interrupt Register Bit Naming Conventions

• **INTC Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	TKMF	TBF	—	TKME	TBE	EMI
R/W	—	—	R/W	R/W	—	R/W	R/W	R/W
POR	—	—	0	0	—	0	0	0

Bit 7~6   Unimplemented, read as "0"

Bit 5   **TKMF**: Touch Key Module 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   Unimplemented, read as "0"

Bit 2   **TKME**: Touch Key Module 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
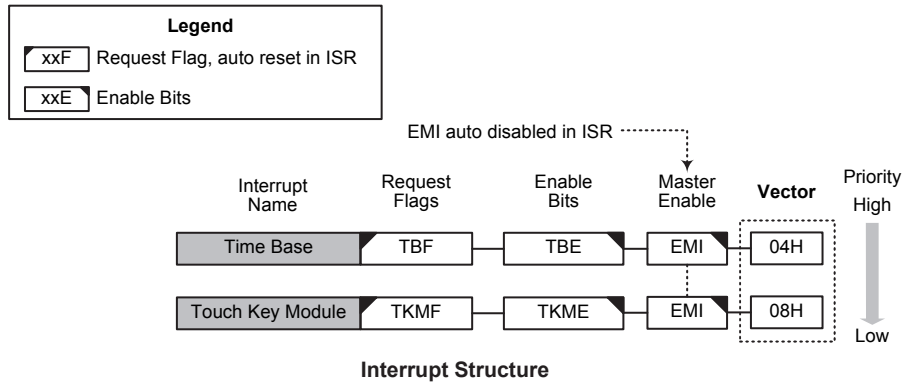   1: Enable

Interrupt Operation

When the conditions for an interrupt event occur, such as a Touch Key counter overflow, 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. These interrupt sources have their own individual 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

Touch Key Interrupt

For a Touch Key interrupt to occur, the global interrupt enable bit, EMI, and the Touch Key interrupt enable TKME must be first set. An actual Touch Key interrupt will take place when the Touch Key request flag. TKMF, is set, a situation that will occur when the time slot counter overflows. When the interrupt is enabled, the stack is not full and the Touch Key time slot counter overflow occurs, a subroutine call to the relevant timer interrupt vector, will take place. When the interrupt is serviced, the Touch Key interrupt request flag, TKMF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

Time Base Interrupt

Time Base Interrupt function is to provide regular time signal in the form of an internal interrupt. It is controlled by the overflow signal from its respective timer function. When this happens its respective interrupt request flag, TBF, will be set. To allow the program to branch to their 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 their respective vector locations will take place. When the interrupt is serviced, the respective 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 CLKSEL1~CLKSEL0 bits in the PSCR register.
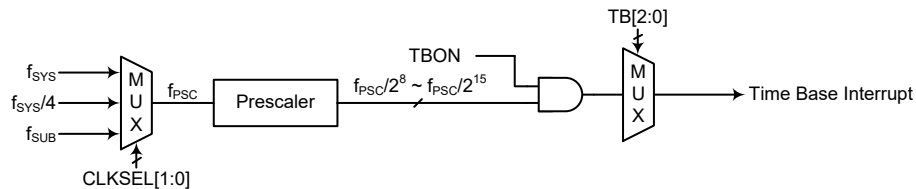
Time Base Interrupt

- **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 f_{PSC} 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 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: $2^{15}/f_{PSC}$

Interrupt Wake-up Function

Each of the interrupt functions has the capability of waking up the microcontroller when in the SLEEP or IDLE Mode. A wake-up is generated when an interrupt request flag changes from low to high and is independent of whether the interrupt is enabled or not. Therefore, even though the device is in the SLEEP or IDLE Mode and its system oscillator stopped. 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.

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 the 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
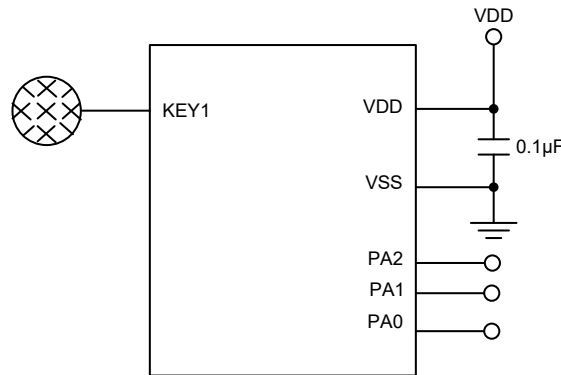
Instruction Set Summary

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	1[Note]	None
MOV A,x	Move immediate data to ACC	1	None
Bit Operation			
CLR [m].i	Clear bit of Data Memory	1[Note]	None
SET [m].i	Set bit of Data Memory	1[Note]	None
Branch Operation			
JMP addr	Jump unconditionally	2	None
SZ [m]	Skip if Data Memory is zero	1[Note]	None
SZA [m]	Skip if Data Memory is zero with data movement to ACC	1[Note]	None
SZ [m].i	Skip if bit i of Data Memory is zero	1[Note]	None
SNZ [m].i	Skip if bit i of Data Memory is not zero	1[Note]	None
SIZ [m]	Skip if increment Data Memory is zero	1[Note]	None
SDZ [m]	Skip if decrement Data Memory is zero	1[Note]	None
SIZA [m]	Skip if increment Data Memory is zero with result in ACC	1[Note]	None
SDZA [m]	Skip if decrement Data Memory is zero with result in ACC	1[Note]	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	2[Note]	None
TABRDL [m]	Read table (last page) to TBLH and Data Memory	2[Note]	None
Miscellaneous			
NOP	No operation	1	None
CLR [m]	Clear Data Memory	1[Note]	None
SET [m]	Set Data Memory	1[Note]	None
CLR WDT	Clear Watchdog Timer	1	TO, PDF
SWAP [m]	Swap nibbles of Data Memory	1[Note]	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	$ACC \leftarrow ACC + [m] + C$
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	$[m] \leftarrow ACC + [m] + C$
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	$ACC \leftarrow ACC + [m]$
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	$ACC \leftarrow ACC + x$
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	$[m] \leftarrow ACC + [m]$
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	$ACC \leftarrow ACC\ ''AND''\ [m]$
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	$ACC \leftarrow ACC\ ''AND''\ x$
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	$[m] \leftarrow ACC\ ''AND''\ [m]$
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	$[m] \leftarrow [m] - 1$
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	$ACC \leftarrow [m] - 1$
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	$TO \leftarrow 0$ $PDF \leftarrow 1$
Affected flag(s)	TO, PDF

INC [m]	Increment Data Memory
Description	Data in the specified Data Memory is incremented by 1.
Operation	$[m] \leftarrow [m] + 1$
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	$ACC \leftarrow [m] + 1$
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 $\leftarrow$ 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	$ACC \leftarrow [m]$
Affected flag(s)	None

MOV A,x	Move immediate data to ACC
Description	The immediate data specified is loaded into the Accumulator.
Operation	$ACC \leftarrow x$
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	$[m] \leftarrow ACC$
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	$ACC \leftarrow ACC - [m] - \overline{C}$
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	$[m] \leftarrow ACC - [m] - \overline{C}$
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	$[m] \leftarrow [m] + 1$ 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	$ACC \leftarrow [m] + 1$ 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 $[m].i \neq 0$
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	$ACC \leftarrow ACC - [m]$
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	$[m] \leftarrow ACC - [m]$
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	$ACC \leftarrow ACC - x$
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	$[m].3 \sim [m].0 \leftrightarrow [m].7 \sim [m].4$
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	The contents of the specified Data Memory are read out and then written to the specified Data Memory again. 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 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

6-pin SOT23-6 Outline Dimensions
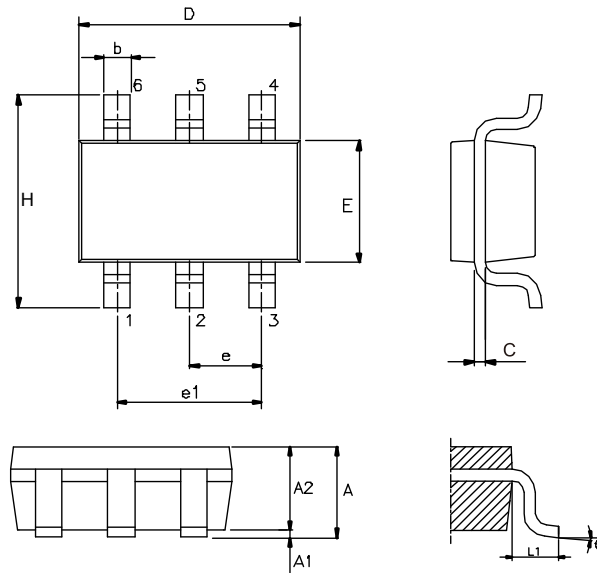
Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	—	—	0.057
A1	—	—	0.006
A2	0.035	0.045	0.051
b	0.012	—	0.020
C	0.003	—	0.009
D	—	0.114 BSC	—
E	—	0.063 BSC	—
e	—	0.037 BSC	—
e1	—	0.075 BSC	—
H	—	0.110 BSC	—
L1	—	0.024 BSC	—
θ	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	—	1.45
A1	—	—	0.15
A2	0.90	1.15	1.30
b	0.30	—	0.50
C	0.08	—	0.22
D	—	2.90 BSC	—
E	—	1.60 BSC	—
e	—	0.95 BSC	—
e1	—	1.90 BSC	—
H	—	2.80 BSC	—
L1	—	0.60 BSC	—
θ	0°	—	8°

8-pin SOP (150mil) Outline Dimensions
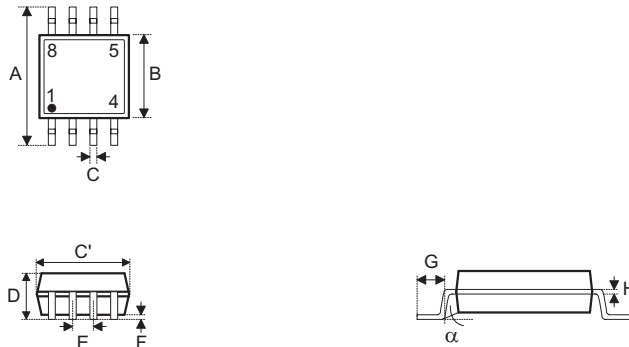
Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	—	0.236 BSC	—
B	—	0.154 BSC	—
C	0.012	—	0.020
C'	—	0.193 BSC	—
D	—	—	0.069
E	—	0.050 BSC	—
F	0.004	—	0.010
G	0.016	—	0.050
H	0.004	—	0.010
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	6.00 BSC	—
B	—	3.90 BSC	—
C	0.31	—	0.51
C'	—	4.90 BSC	—
D	—	—	1.75
E	—	1.27 BSC	—
F	0.10	—	0.25
G	0.40	—	1.27
H	0.10	—	0.25
α	0°	—	8°