HT95R25
Phone 8-bit MCU

Technical Document

- Application Note
  - HA0075E MCU Reset and Oscillator Circuits Application Note

Features

- Operating voltage at f_{SYS}= 3.58MHz: 2.2V~5.5V
- 16K×16 OTP type Program Memory
- 2112×8 Data Memory
- 50 bidirectional I/Os with pull-high options
- 2 NMOS output-only lines
- External interrupt input
- Dual 16-bit timers with interrupts
- Timer external input
- 8-level stack
- 32768Hz system oscillator
- 32768Hz to 3.58MHz frequency-up PLL circuit
- Real time clock function
- Watchdog timer function
- PFD driver output
- DTMF generator
- Power-down and wake-up feature for power-saving operation: Idle mode, Sleep mode, Green mode and normal mode
- Up to 1.117μs instruction cycle with 3.58MHz system clock at V_{DD}=2.2V~5.5V
- Bit manipulation instructions
- Table read function
- 63 powerful instructions
- All instructions executed in 1 or 2 machine cycles
- Supported by comprehensive suite of hardware and software tools
- 64-pin LQFP package

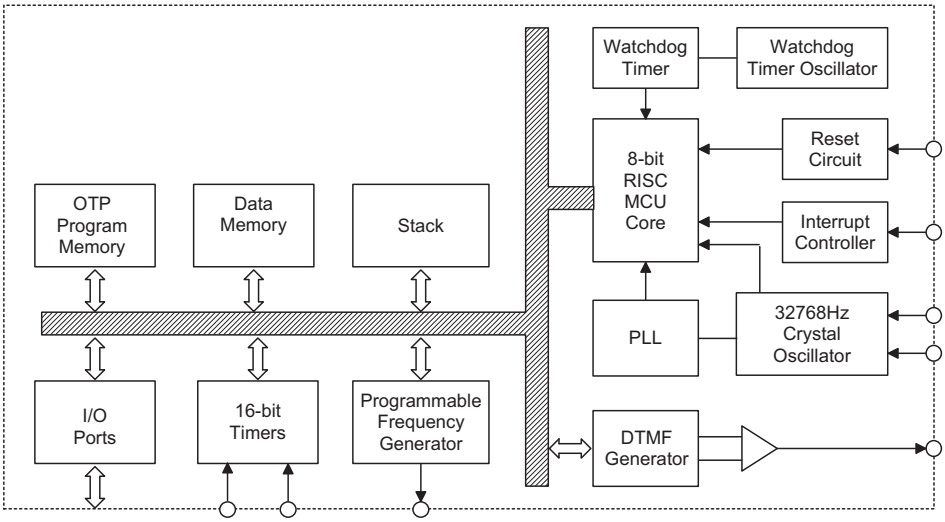
General Description

The HT95R25 MCU is an 8-bit high performance, RISC architecture microcontroller device specially designed for telephone applications. Device flexibility is enhanced with their internal special features such as power-down and wake-up functions, DTMF generator, PFD driver, etc. These features combine to ensure applications require a minimum of external components and therefore reduce overall product costs.

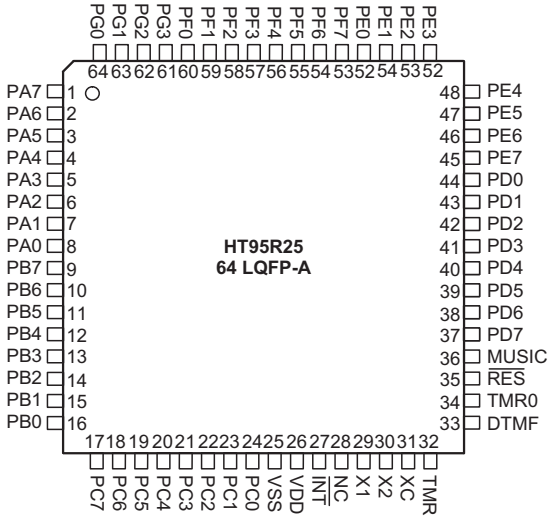
Having the advantages of low-power consumption, high-performance, I/O flexibility as well as low-cost, these devices have the versatility to suit a wide range of application possibilities such as deluxe feature phone, cordless phones, fax and answering machines, etc.

The device is best suited for phone products that comply with versatile dialer specification requirements for different areas or countries. The device is fully supported by the Holtek range of fully functional development and programming tools, providing a means for fast and efficient product development cycles.

Block Diagram



Pin Assignment



Pin Description

Pad Name	I/O	Options	Description
PA0~PA7	I/O	Pull-high Wake-up	Bidirectional 8-bit input/output port. Each individual pin on this port can be configured as a wake-up input by a configuration option. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine which pins on the port have pull-high resistors.
PB0~PB7	I/O	Pull-high	Bidirectional 8-bit input/output port.Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine which pins on the port have pull-high resistors.
PC0, PC5, PC7	I/O	Pull-high	Bidirectional input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine which pins on the port have pull-high resistors. When the multi-function interrupt is enabled an interrupt will be generated whenever PC0 or PC5 has a falling edge, or PC7 has a rising edge.When in the idle mode such an interrupt will wake up the device.
PC1, PC4, PC6	I/O	Pull-high	Bidirectional input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine which pins on the port have pull-high resistors.
PC2, PC3	O	—	NMOS output structures
PD0~PD7 PE0~PE7 PF0~PF7	I/O	Pull-high	Bidirectional 8-bit input/output port.Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine which nibble on the port have pull-high resistors.
PG0~PG3	I/O	Pull-high	Bidirectional 4-bit input/output port.Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if all the pins on the port have pull-high resistors.
$\overline{INT}$	I	—	External interrupt Schmitt trigger input. Edge trigger activated on high to low transition. No pull-high resistor.
TMR0	I	—	Timer/Event Counter 0 Schmitt trigger input . No pull-high resistor.
TMR1	I	—	Timer/Event Counter 1 Schmitt trigger input . No pull-high resistor.
DTMF	O	—	Dual Tone Multi Frequency Output
MUSIC	O	—	CMOS output structure Programmable Frequency Divider pin.
X1 X2	I O	—	X1 and X2 are connected to an external 32768Hz crystal or resonator for the system clock.
XC	—	—	External low pass filter pin used for the frequency up conversion circuit.
$\overline{RES}$	I	—	Schmitt trigger reset input. Active low.
VDD	—	—	Positive power supply
VSS	—	—	Negative power supply, ground.

Note: Each pin on PA can be programmed through a configuration option to have a wake-up function.

Absolute Maximum Ratings

Supply Voltage ..... V_{SS}–0.3V to V_{SS}+6.0V
Input Voltage ..... V_{SS}–0.3V to V_{DD}+0.3V
I_{OL} Total 150mA
Total Power Dissipation 500mW

Storage Temperature ..... –50°C to 125°C
Operating Temperature ..... –40°C to 85°C
I_{OH} Total ..... –100mA

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

D.C. Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
General							
V_{DD}	Operating Voltage	—	—	2.2	—	5.5	V
CPU							
I_{IDL1}	Idle Mode Current 1	3V	32768Hz and 3.58MHz oscillator off, system HALT, WDT off, no load	—	—	1.5	μA
		5V		—	—	2	
I_{IDL2}	Idle Mode Current 2	3V	32768Hz and 3.58MHz oscillator off, system HALT, WDT on, no load	—	—	5	μA
		5V		—	—	10	
I_{SLP}	Sleep Mode Current	3V	32768Hz on, 3.58MHz oscillator off, system HALT, no load	—	—	15	μA
		5V		—	—	30	
I_{GRN}	Green Mode Current	3V	32768Hz on, 3.58MHz oscillator off, system on, no load	—	—	25	μA
		5V		—	—	50	
I_{NOR}	Normal Mode Current	3V	32768Hz and 3.58MHz oscillator on, system on, DTMF generator off, no load	—	—	2	mA
		5V		—	—	3	
R_{PH}	Pull-high Resistor	3V	—	66	200	330	kΩ
		5V		33	100	166	
V_{IL}	I/O Port Input Low Voltage	—	—	0	—	$0.3V_{DD}$	V
V_{IH}	I/O Port Input High Voltage	—	—	$0.7V_{DD}$	—	V_{DD}	V
I_{OL1}	I/O Port Sink Current	3V	V_{OL}= $0.1V_{DD}$	3	4	—	mA
		5V		4	6	—	
I_{OL2}	PC2, PC3 Sink Current	5V	PC2/PC3= 0.5V	2.5	—	—	mA
I_{OH1}	I/O Port Source Current	3V	V_{OH}= $0.9V_{DD}$	–1	–2	—	mA
		5V		–2	–3	—	
I_{OH2}	PC2, PC3 Leakage Current	5V	PC2/PC3= 5V	—	—	2.5	μA
DTMF Generator (Operating Temperature: –20°C ~ 85°C)							
V_{TDC}	DTMF Output DC Level	—	—	$0.45V_{DD}$	—	$0.7V_{DD}$	V
V_{TOL}	DTMF Sink Current	—	V_{DTMF}= 0.5V	0.1	—	—	mA

A.C. Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
General							
f_{SYS1}	System Clock 1	—	Normal mode 32768Hz crystal oscillator	—	3.5795	—	MHz
f_{SYS2}	System Clock 2	—	Green mode 32768Hz crystal oscillator	—	32	—	kHz
t_{SST}	System Start-up Timer Period	—	Power-up, Reset or wake-up from HALT	—	1024	—	t_{SYS}
t_{WAKE}	Wake-up Time for 32768Hz Crystal OSC	3V	32kHz oscillator OFF → ON	—	—	200	ms
t_{FUP}	Settling Time for 32768Hz to 3.58MHz PLL (Frequency Up Conversion)	3V	32kHz oscillator is ON; 3.58MHz oscillator OFF → ON	—	—	20	ms
t_{S2G}	Time from Sleep Mode to Green Mode	—	Wake-up from Sleep Mode	—	0	—	ms
MCU							
t_{WDTOSC}	Watchdog Oscillator Period	3V	—	45	90	180	μs
		5V	—	32	65	130	
t_{RES}	External Reset Low Pulse Width	—	—	1	—	—	μs
t_{INT}	Interrupt Pulse Width	—	—	1	—	—	μs
DTMF Generator (Operating Temperature: −20°C ~ 85°C)							
f_{DTMFO}	Single Tone Output Frequency	2.5V	Microcontroller normal mode; DTMF generator single tone test mode	690	—	704	Hz
				762	—	778	
				843	—	861	
				932	—	950	
				1197	—	1221	
				1323	—	1349	
				1462	—	1492	
				1617	—	1649	
V_{TAC}	DTMF Output AC Level	—	Row group, R_L= 5kΩ	120	155	180	mV_{rms}
R_L	DTMF Output Load	—	T.H.D. ≤ −23dB	5	—	—	kΩ
A_{CR}	Column Pre-emphasis	—	Row group= 0dB	1	2	3	dB
THD	Tone Signal Distortion	—	R_L= 5kΩ	—	−30	−23	dB

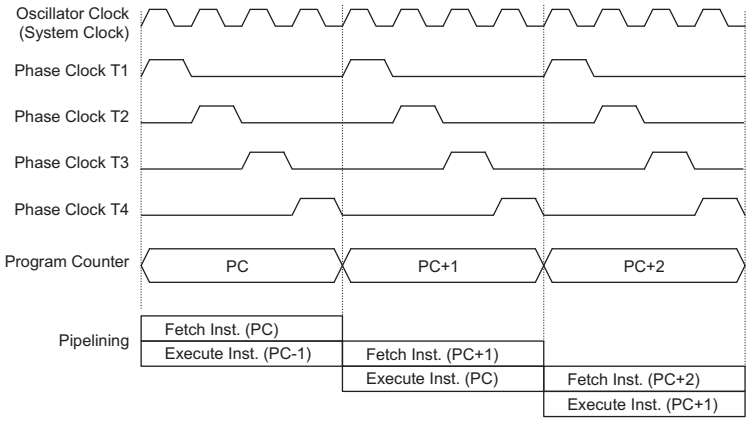
System Architecture

A key factor in the high-performance features of the Holtek range of microcontrollers is attributed to the internal system architecture. The range of devices 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 operations of the instruction set. It 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 these devices suitable for low-cost, high-volume production for phone controller applications requiring up to 16K words of Program Memory and 2112 bytes of Data Memory storage.

Clocking and Pipelining

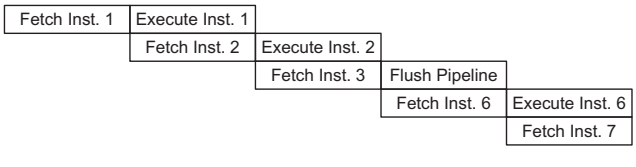
The system clock is derived from an external 32768Hz Crystal/ Resonator which then generates a 3.58MHz system clock using internal frequency-up converter circuitry. This internal clock is subdivided into four internally generated non-overlapping clocks, T1~T4. The Program Counter is incremented at the beginning of the T1 clock during which time a new instruction is fetched. The remaining T2~T4 clocks carry out the decoding and execution functions. In this way, one T1~T4 clock cycle forms one instruction cycle. Although the fetching and execution of instructions takes place in consecutive instruction cycles, the pipelining structure of the microcontroller ensures that instructions are effectively executed in one instruction cycle. The exception to this are instructions where the contents of the Program Counter are changed, such as subroutine calls or jumps, in which case the instruction will take one more instruction cycle to execute.

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



System Clocking and Pipelining

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



Instruction Fetching

Program Counter

During program execution, the Program Counter is used to keep track of the address of the next instruction to be executed. It is automatically incremented by one each time an instruction is executed except for instructions, such as ″JMP″ or ″CALL″ that demand a jump to a non-consecutive Program Memory address. Note that the Program Counter width varies with the Program Memory capacity depending upon which device is selected. However, it must be noted that only the lower 8 bits, known as the Program Counter Low Register, are directly addressable by user.

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.

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 writable 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.

Stack

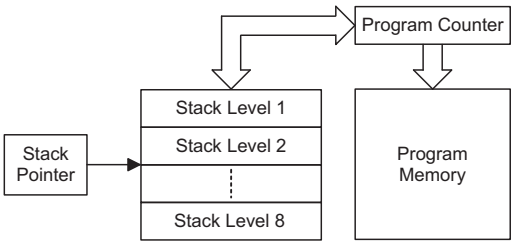
This is a special part of the memory which is used to save the contents of the Program Counter only. The stack has 8 levels and is neither part of the data nor part of the program space, and can neither be read from nor written to. The activated level is indexed by the Stack Pointer, SP, which can also neither be read from nor written to. 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 , 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.

Mode	Program Counter Bits													
	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Initial Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0
External Interrupt	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Timer/Event Counter 0 Overflow	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Timer/Event Counter 1 Overflow	0	0	0	0	0	0	0	0	0	0	1	1	0	0
Peripheral Interrupt	0	0	0	0	0	0	0	0	0	1	0	0	0	0
RTC Interrupt	0	0	0	0	0	0	0	0	0	1	0	1	0	0
Multi-Function Interrupt	0	0	0	0	0	0	0	0	0	1	1	0	0	0
Skip	Program Counter + 2 (Within current bank)													
Loading PCL	PC13	PC12	PC11	PC10	PC9	PC8	@7	@6	@5	@4	@3	@2	@1	@0
Jump, Call Branch	#13	#12	#11	#10	#9	#8	#7	#6	#5	#4	#3	#2	#1	#0
Return from Subroutine	S13	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0

Program Counter

Note: PC13~PC8: Current Program Counter bits
@7~@0: PCL bits
#13~#0: Instruction code address bits
S13~S0: Stack register bits



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

Program Memory

The Program Memory is the location where the user code or program is stored. For these devices the Program Memory is an OTP type, which means it can be programmed once.

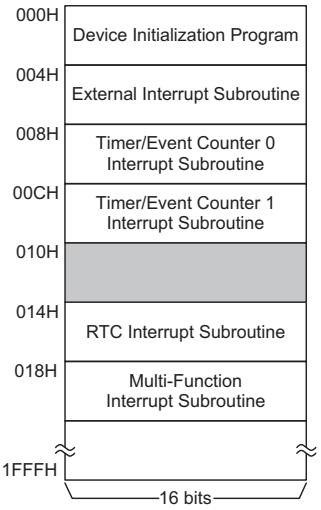
Structure

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

Special Vectors

Within the Program Memory, certain locations are reserved for special usage such as reset and interrupts.

- Location 000H
  This vector 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.
- Location 004H
  This vector is used by the external interrupt. If the external interrupt pin on the device goes low, the program will jump to this location and begin execution if the external interrupt is enabled and the stack is not full.
- Location 008H
  This internal vector is used by the Timer/Event Counter 0. If a counter overflow occurs, the program will jump to this location and begin execution if the timer/event counter 0 interrupt is enabled and the stack is not full.
- Location 00CH
  This internal vector is used by the Timer/Event Counter 1. If a counter overflow occurs, the program will jump to this location and begin execution if the timer/event counter 1 interrupt is enabled and the stack is not full.
- Location 014H
  This location is used by the RTC. When the RTC is enabled and a time-out occurs, the program will jump to this location and begin execution if the RTC interrupt is enabled and the stack is not full.
- Location 018H
  This location is used by the Multi-function Interrupt. If a falling edge transition is detected on PC0 or PC5, or a rising edge transition detected on PC7, the program will jump to this location and begin execution if the multi-function interrupt is enabled and the stack is not full.



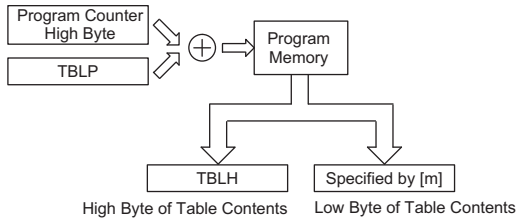
Program Memory Structure

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 lower order address of the look up data to be retrieved in the table pointer register, TBLP. This register defines the lower 8-bit address of the look-up table.

After setting up the table pointer, the table data can be retrieved from the current Program Memory page or last Program Memory page using the ″TABRDC[m]″ or ″TABRDL [m]″ instructions, respectively. When these instructions are 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 have uncertain values.

The following diagram illustrates the addressing/data flow of the look-up table:



Look-up Table

Table Program Example

The following example shows how the table pointer and table data is defined and retrieved from the device. This example uses raw table data located in the last page which is stored there using the ORG statement. The value at this ORG statement is ″3F00H″ which refers to the start address of the last page within the 16K Program Memory of the microcontroller. 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 ″3F06H″ or 6 locations after the start of the last page. Note that the value for the table pointer is referenced to the first address of the present page if the ″TABRDC [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 ″TABRDL [m]″ instruction is executed.

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

Instruction	Table Location Bits													
	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
TABRDC [m]	PC13	PC12	PC11	PC10	PC9	PC8	@7	@6	@5	@4	@3	@2	@1	@0
TABRDL [m]	1	1	1	1	1	1	@7	@6	@5	@4	@3	@2	@1	@0

Table Location

Note: PC13~PC8: Current Program Counter bits
@7~@0: Table Pointer Lower-order bits (TBLP)

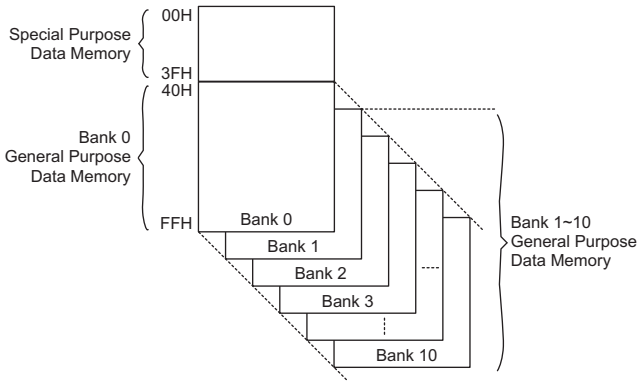
```
tempreg1            db  ?       ; temporary register #1
tempreg2            db  ?       ; temporary register #2
    :
    :

mov a,06h                       ; initialise table pointer - note that this address
                                ; is referenced

mov tblp,a                      ; to the last page or present page
    :
    :

tabrdl              tempreg1    ; transfers value in table referenced by table pointer
                                ; to tempreg1
                                ; data at prog. memory address "3F06H" transferred to
                                ; tempreg1 and TBLH

dec tblp                        ; reduce value of table pointer by one

tabrdl              tempreg2    ; transfers value in table referenced by table pointer
                                ; to tempreg2
                                ; data at prog.memory address "3F05H" transferred to
                                ; tempreg2 and TBLH
                                ; in this example the data "1AH" is transferred to
                                ; tempreg1 and data "0FH" to register tempreg2
                                ; the value "0FH" will be transferred to the high byte
                                ; register TBLH
    :
    :

org 3F00h                       ; sets initial address of HT95R25 last page

dc  00Ah, 00Bh, 00Ch, 00Dh, 00Eh, 00Fh, 01Ah, 01Bh
    :
    :
```

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

Structure

The Special Purpose and General Purpose Data Memory are located at consecutive locations. All are implemented in RAM and are 8 bits wide. The start address of the Data Memory is the address 00H. Registers which are common to all microcontrollers, such as ACC, PCL, etc., have the same Data Memory address. Note that after power-on, the contents of the Data Memory, will be in an unknown condition, the programmer must therefore ensure that the Data Memory is properly initialised. The Special Purpose Data Memory is located in Bank 0 while the General Purpose Data Memory is divided into 11 individual areas or Banks known as Bank 0 to Bank 10. Switching between different banks is achieved by setting the Bank Pointer to the correct value.



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 program for both read and write operations. By using the "SET [m].i" and "CLR [m].i" 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. As the General Purpose Data Memory is located within 11 different banks, it is first necessary to ensure that the Bank Pointer is properly set to the correct value before accessing the General Purpose Data Memory. Only Bank 0 data can be read directly. Indirect Addressing of Bank 0 is executed using Indirect Addressing Register IAR0 and Memory Pointer MP0. Data in Banks 1~10 can only be read indirectly using Indirect Addressing Register IAR1 and Memory Pointer MP1.

Special Purpose Data Memory

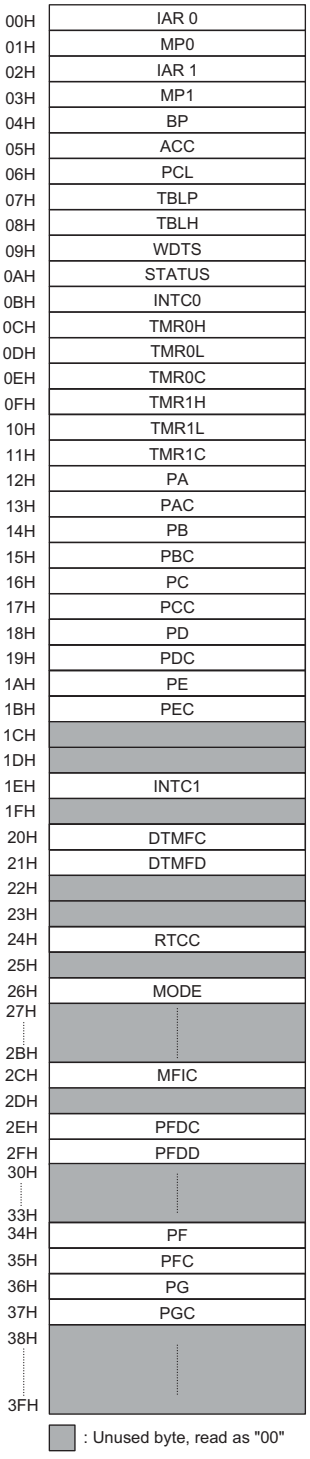
This area of Data Memory, is located in Bank 0, where registers, necessary for the correct operation of the microcontroller, are stored. Most of the registers are both readable and writeable but some are protected and are readable only, the details of which are located under the relevant Special Function Register section. Note that for locations that are unused, any read instruction to these addresses will return the value "00H". Although the Special Purpose Data Memory registers are located in Bank 0, they will still be accessible even if the Bank Pointer has selected Banks 1~10.

Special Function Registers

To ensure successful operation of the microcontroller, certain internal registers are implemented in the RAM Data Memory area. These registers ensure correct operation of internal functions such as timers, interrupts, watchdog, etc., as well as external functions such as I/O data control. The location of these registers within the RAM Data Memory begins at the address "00H". Any unused Data Memory locations between these special function registers and the point where the General Purpose Memory begins is reserved for future expansion purposes, attempting to read data from these locations will return a value of "00H".

Indirect Addressing Register – IAR0, IAR1

The Indirect Addressing Registers, IAR0 and IAR1, although having their locations in normal RAM register space, do not actually physically exist as normal registers. The method of indirect addressing for RAM data manipulation uses these Indirect Addressing Registers and Memory Pointers, in contrast to direct memory addressing, where the actual memory address is specified. Actions on the IAR0 and IAR1 registers will result in no actual read or write operation to these registers but rather to the memory location specified by their corresponding Memory Pointer, MP0 or MP1. Acting as a pair, IAR0 and MP0 can together only access data from Bank 0, while the IAR1 and MP1 register pair can access data from both Bank 0 and Bank 1 to Bank 10. As



Special Purpose Data Memory Structure

the Indirect Addressing Registers are not physically implemented, reading the Indirect Addressing Registers indirectly will return a result of "00H" and writing to the registers indirectly will result in no operation.

Memory Pointer – MP0, MP1

For all devices, two Memory Pointers, known as MP0 and MP1 are provided. These Memory Pointers are physically implemented in the Data Memory and can be manipulated in the same way as normal registers providing a convenient way with which to address and track data. When any operation to the relevant Indirect Addressing Registers is carried out, the actual address that the microcontroller is directed to, is the address specified by the related Memory Pointer. MP0, together with Indirect Addressing Register, IAR0, are used to access data from Bank 0 only, while MP1 and IAR1 are used to access data from Banks 0~10.

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

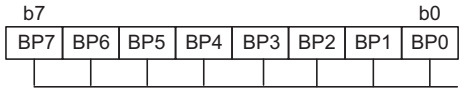
```
data .section  'data'
adres1              db ?
adres2              db ?
adres3              db ?
adres4              db ?
block               db ?
code .section at 0 'code'
org   00h

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 RAM addresses.

Bank Pointer – BP

The Data Memory RAM is divided into eleven banks, known as Bank 0~Bank 10. All of the Special Purpose Registers are contained in Bank 0. Selecting the required Data Memory area is achieved using the Bank Pointer. If data in Bank 0 is to be accessed, then the BP register must be loaded with the value "00", while if data in Bank 1 is to be accessed, then the BP register must be loaded with the value "01" and so on for the other registers. Using Memory Pointer MP0 and Indirect Addressing Register IAR0 will always access data from Bank 0, irrespective of the value of the Bank Pointer.

The Data Memory is initialised to Bank 0 after a reset, except for a WDT time-out reset in the Power Down Mode, in which case, the Data Memory bank remains unaffected. It should be noted that Special Function Data Memory is not affected by the bank selection, which means that the Special Function Registers can be accessed from Bank 0 to Bank 10. Directly addressing the Data Memory will always result in Bank 0 being accessed irrespective of the value of the Bank Pointer.



BP7~BP0	Data Memory
0	Bank 0
1	Bank 1
2	Bank 2
3	Bank 3
4	Bank 4
5	Bank 5
6	Bank 6
7	Bank 7
8	Bank 8
9	Bank 9
10	Bank 10

Accumulator – ACC

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

Program Counter Low Register – PCL

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

Look-up Table Registers – TBLP, TBLH

These two 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. Its 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.

Watchdog Timer Register – WDTS

The Watchdog feature of the microcontroller provides an automatic reset function giving the microcontroller a means of protection against spurious jumps to incorrect Program Memory addresses. To implement this, a timer is provided within the microcontroller which will issue a reset command when its value overflows. To provide variable Watchdog Timer reset times, the Watchdog Timer clock source can be divided by various division ratios, the value of which is set using the WDTS register. By writing directly to this register, the appropriate division ratio for the Watchdog Timer clock source can be setup. Note that only the lower 3 bits are used to set division ratios between 1 and 128.

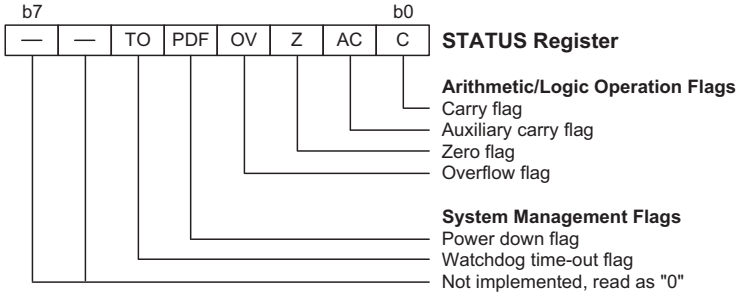
Status Register – STATUS

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.



Status Register

- **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.

Interrupt Control Register – INTC0, INTC1

These two 8-bit register, known as the INTC0 and INTC1 registers, control the operation of both external and internal timer, interrupts, etc. By setting various bits within this register using standard bit manipulation instructions, the enable/disable function of the external and timer interrupts can be independently controlled. A master interrupt bit within this register, the EMI bit, acts like a global enable/disable and is used to set all of the interrupt enable bits on or off. This bit is cleared when an interrupt routine is entered to disable further interrupt and is set by executing the ″RETI″ instruction.

Timer/Event Counter Registers

This device contains two 16-bit Timer/Event Counters, which have associated register pairs known as TMR0L/TMR0H and TMR1L/TMR1H. These are the locations where the timers 16-bit value is located. Two associated control registers, known as TMR0C and TMR1C, contain the setup information for these two timers.

Input/Output Ports and Control Registers

Within the area of Special Function Registers, the I/O registers and their associated control registers play a prominent role. All I/O ports have a designated register correspondingly labeled as PA, PB, PC, PD, PE, PF and PG. These labeled I/O registers are mapped to specific addresses within the Data Memory as shown in the Data Memory table, which are used to transfer the appropriate output or input data on that port. with each I/O port there is an associated control register labeled PAC, PBC, PCC, PDC, PEC, PFC and PGC, also mapped to specific addresses with the Data Memory. The control register specifies which pins of that port are set as inputs and which are set as outputs. To setup a pin as an input, the corresponding bit of the control register must be set high, for an output it must be set low. During program initialization, it is important to first setup the control registers to specify which pins are outputs and which are inputs before reading data from or writing data to the I/O ports. One flexible feature of these registers is the ability to directly program single bits using the ″SET [m].i″ and ″CLR [m].i″ instructions. The ability to change I/O pins from output to input and vice versa by manipulating specific bits of the I/O control registers during normal program operation is a useful feature of these devices.

DTMF Registers – DTMFC, DTMFD

The device contains fully integrated DTMF generator circuitry for generation of DTMF signals. The DTMF generator also requires two registers for its operation, a DTMFC register for its overall control and DTMFD register to store the digital codes that are to be generated as DMTF signals.

Mode Register – MODE

The device supports two system clock and four operation modes. The system clock could be 32768Hz or 3.58MHz and operation mode could be Normal, Green, Sleep or Idle mode. These are all selected by the software.

MFIC Register – MFIC

PC0, PC5 and PC7 could be used to trigger an extra interrupt. They are enabled or disabled individually by bit0~bit2 of MFIC. When a multi-function interrupt occurs, the programmer should check bit4~bit6 of MFIC to determine the cause of the interrupt.

PFD Registers – PFDC/PFDD

The device contains a Programmable Frequency Divider function which can generate accurate frequencies based on the system clock. The clock source, enable function and output frequency is controlled using these two registers.

RTCC Register

The device contains a Real Time Clock function otherwise known as the RTC. To control this function a register known as the RTCC register is provided which provides the overall on/off control and time out flag.

Input/Output Ports

Holtek microcontrollers offer considerable flexibility on their I/O ports. With the input or output designation of every pin fully under user program control, pull-high options for all ports and wake-up options on certain pins, the user is provided with an I/O structure to meet the needs of a wide range of application possibilities. The device provides 34 bidirectional input/output lines labeled with port names PA, PB, PC, PD, PE, PF and PG. These I/O ports are mapped to the 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.

Unlike other port lines, PC2 and PC3 are supplied as NMOS output-only lines.

Pull-high Resistors

Many product applications require pull-high resistors for their switch inputs usually requiring the use of an external resistor. To eliminate the need for these external resistors, all I/O pins, when configured as an input have the capability of being connected to an internal pull-high resistor. These pull-high resistors are selectable via configuration options and are implemented using a weak PMOS transistor. Note that if the pull-high option is selected, then all I/O pins on that port will be connected to pull-high resistors, individual pins cannot be selected for pull-high resistor options.

Port A Wake-up

Each device has a HALT instruction enabling the microcontroller to enter a Power Down Mode and preserve 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. After a "HALT" instruction forces the microcontroller into entering the Power Down mode, the device will remain idle or in a low-power state until the logic condition of the selected wake-up pin on Port A changes from high to low. This function is especially suitable for applications that can be woken up via external switches. Note that each pin on Port A can be selected individually to have this wake-up feature.

I/O Port Control Registers

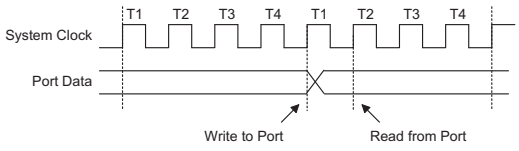
Each I/O port has its own control register PAC, PBC, PCC, PDC, PEC, PFC and PGC, to control the input/output configuration. with this control register, each CMOS output or input with or without pull-high resistor structures can be reconfigured dynamically under software control. Each pin of the I/O ports 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.

I/O Pin Structures

The following diagrams illustrate the I/O pin internal structures. As the exact logical construction of the I/O pin may differ from these drawings, they are supplied as a guide only to assist with the functional understanding of the I/O pins.

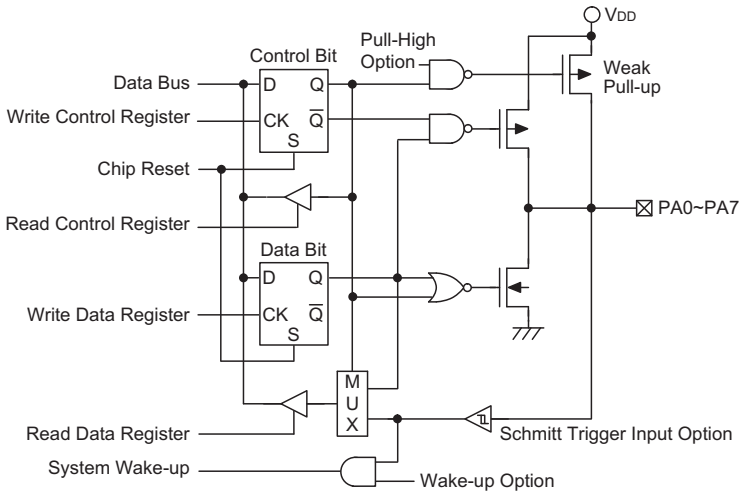
Programming Considerations

Within the user program, one of the first things to consider is port initialization. After a reset, all of the I/O data and port control registers will be set high. This means that all I/O pins will default to an input state, the level of which depends on the other connected circuitry and whether pull-high options have been selected. If the port control registers, PAC, PBC, PCC, PDC, PEC, PFC and PGC, 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, PA, PB, PC, PD, PE, PF and PG, 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.

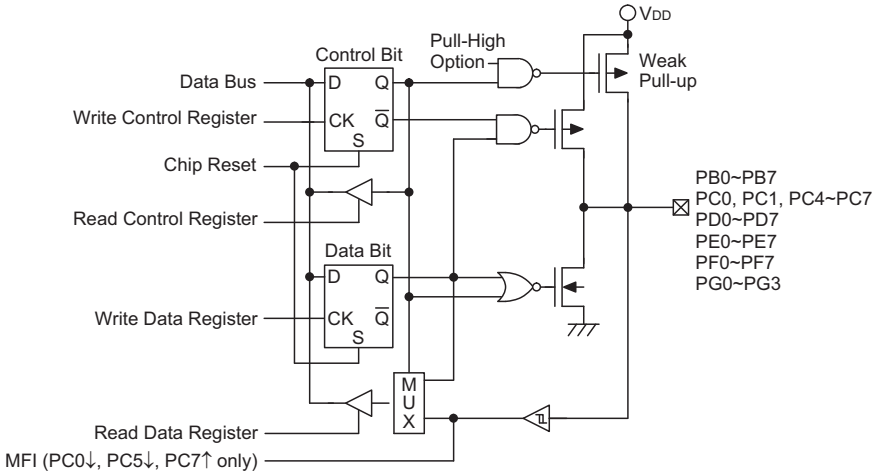


Read/Write Timing

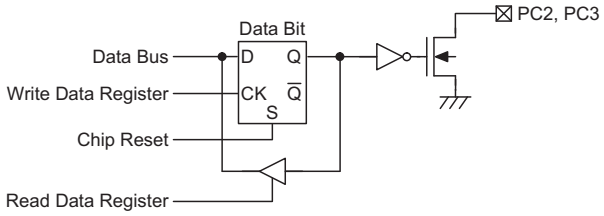
Port A has the additional capability of providing wake-up functions. When the device is in the Power Down Mode, various methods are available to wake the device up. One of these is a high to low transition of any of the Port A pins. Single or multiple pins on Port A can be setup to have this function.



PA Input/Output Port



PB, PC (Except for PC2 and PC3), PD, PE, PF and PG Input/Output Ports



PC2, PC3 NMOS Output Port

Timer/Event Counters

The provision of timers form an important part of any microcontroller, giving the designer a means of carrying out time related functions. The device contains two count-up timers of 16-bit capacity. As each timer has three different operating modes, they can be configured to operate as a general timer, an external event counter or as a pulse width measurement device.

There are two types of registers related to the Timer/Event Counters. The first are the registers that contains the actual value of the Timer/Event Counter and into which an initial value can be preloaded, and are known as TMR0L/TMR0H and TMR1L/TMR1H. Reading these register pairs retrieves the contents of the Timer/Event Counters. The second type of associated register are the Timer Control Registers, which defines the timer options and determines how the Timer/Event Counters are to be used, and have the name TMR0C and TMT1C. The device can have the timer clock configured to come from the internal clock source or from an external timer pin.

An external clock source is used when the Timer/Event Counter is in the event counting mode, with the clock source being provided on the external timer pins. These pins have the name TMR0 and TMR1. Depending upon the condition of the T0E or T1E bit in the Timer Control Register, each high to low, or low to high transition on the external timer input pin will increment the Timer/Event Counter by one.

Configuring the Timer/Event Counter Input Clock Source

For Timer/Event Counter 0, the internal timer clock source can originate from either the system clock/4 system clock or from an external clock source. For Timer/Event Counter 1, the internal timer clock source can originate from the 32768Hz system clock or from an external clock source. The system clock input timer source is used when the timer is in the timer mode or in the pulse width measurement mode.

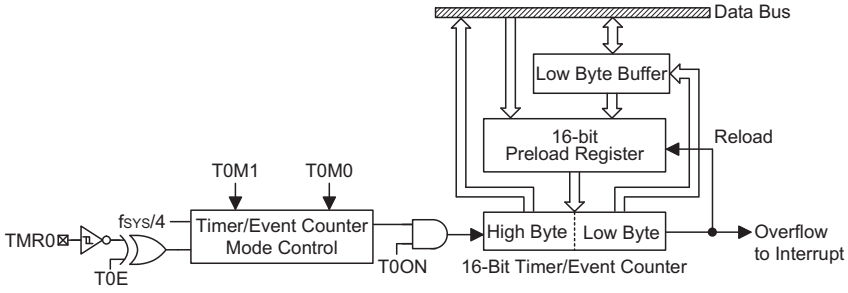
An external clock source is used when the timer is in the event counting mode, the clock source being provided on the external timer pins TMR0 or TMR1. Depending upon the condition of the T0E or T01 bit, each high to low, or low to high transition on the external timer pin will increment the counter by one.

Timer Registers – TMR0L/TMR0H, TMR1L/TMR1H

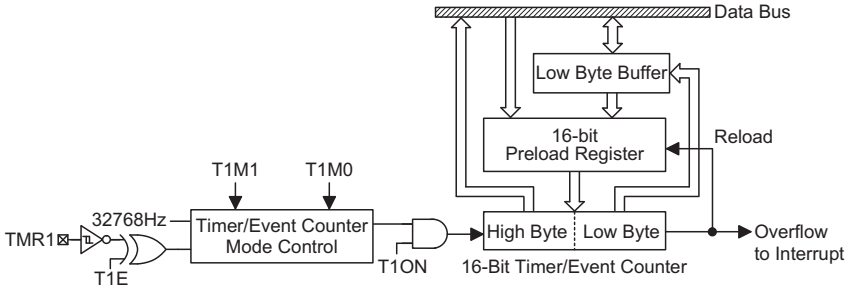
The timer registers are special function register pairs located in the Special Purpose Data Memory and is the place where the 16-bit actual timer value is stored. These register pairs are known as TMR0L/TMR0H and TMR1L/TMR1H. The value in the timer register pair increases by one each time an internal clock pulse is received or an external transition occurs on the external timer pin. The timer will count from the initial value loaded by the preload register to the full count of FFFFH at which point the timer overflows and an internal interrupt signal is generated. The timer value will then be reset with the initial preload register value and continue counting.

To achieve a maximum full range count of FFFFH the preload register must first be cleared to all zeros. It should be noted that after power-on, the preload register will be in an unknown condition. Note that if the Timer/Event Counter is switched off and data is written to its preload register, this data will be immediately written into the actual timer register. However, if the Timer/Event Counter is enabled and counting, any new data written into the preload data register during this period will remain in the preload register and will only be written into the timer register the next time an overflow occurs.

Reading from and writing to these registers is carried out in a specific way. It must be noted that when using instructions to preload data into the low byte register, namely TMR0L or TMR1L, the data will only be placed in a low byte buffer and not directly into the low byte register. The actual transfer of the data into the low byte register is only carried out when a write to its associated high byte register, namely TMR0H or TMR1H, is executed. Also, using instructions to preload data into the high byte timer register will result in the data being directly written to the high byte register. At the same time the data in the low byte buffer will be transferred into its associated low byte register. For this reason, when preloading data into the 16-bit timer registers, the low byte should be written first. It must also be noted that to read the contents of the low byte register, a read to the high byte register must first be executed to latch the contents of the low byte buffer from its associated low byte register. After this has been done, the low byte register can be read in the normal way. Note that reading the low byte timer register directly will only result in reading the previously latched contents of the low byte buffer and not the actual contents of the low byte timer register.



16-bit Timer/Event Counter 0 Structure

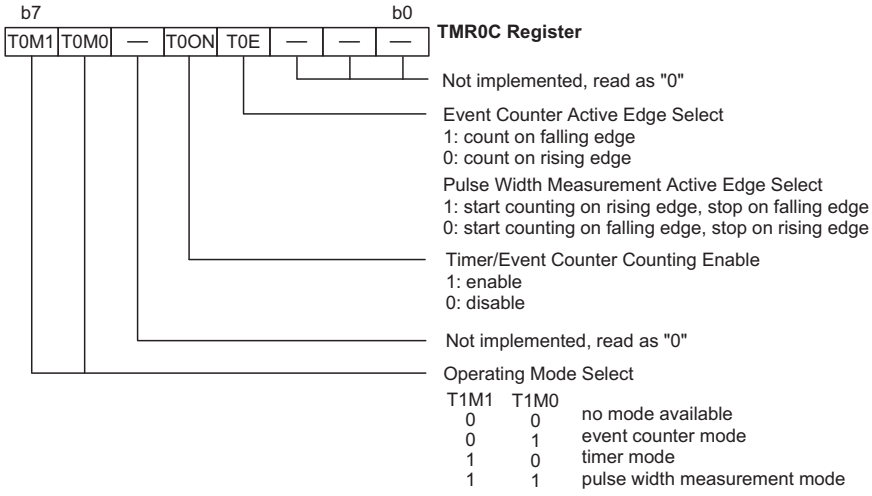


16-bit Timer/Event Counter 1 Structure

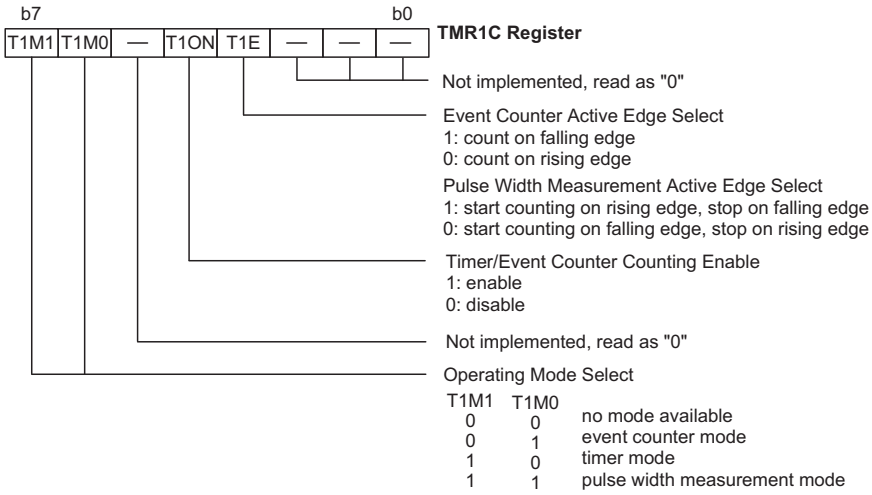
Timer Control Registers – TMR0C, TMR1C

The flexible features of the Holtek microcontroller Timer/Event Counters enable them to operate in three different modes, the options of which are determined by the contents of their control register, which has the name TMR0C/TMR1C. It is the Timer Control Register together with its corresponding timer register pair that control the full operation of each Timer/Event Counter. Before the Timer/Event Counter can be used, it is essential that the Timer Control Register is fully programmed with the right data to ensure its correct operation, a process that is normally carried out during program initialisation.

To choose which of the three modes the Timer/Event Counter is to operate in, either in the timer mode, the event counting mode or the pulse width measurement mode, bits 7 and 6 of the Timer Control Register, which are known as the bit pair T0M1/T0M0 and T1M1/T1M0, must be set to the required logic levels. The Timer/Event Counter on/off bit, which is bit 4 of the Timer Control Register, and known as T0ON and T1ON, provides the basic on/off control of the Timer/Event Counter. Setting the bit high allows the Timer/Event Counter to run, clearing the bit stops it running. If the Timer/Event Counter is in the event count or pulse width measurement mode, the active transition edge level type is selected by the logic level of bit 3 of the Timer Control Register which is known as T0E and T1E.



Timer/Event Counter 0 Control Register



Timer/Event Counter 1 Control Register

Configuring the Timer Mode

In this mode, the Timer/Event Counters can be utilised to measure fixed time intervals, providing an internal interrupt signal each time the Timer/Event Counter overflows. To operate in this mode, the Operating Mode Select bit pair in the Timer Control Register must be set to the correct value as shown.

Control Register Operating Mode Select Bits for the Timer Mode	Bit7	Bit6
	1	0

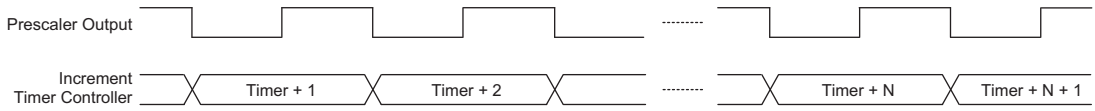
In this mode the internal clock, is used as the Timer/Event Counter clock. After the other bits in the Timer Control Register have been setup, the enable bit, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter to run. Each time an internal clock cycle occurs, the Timer/Event Counter increments by one. When it is full and overflows, an interrupt signal is generated and the Timer/Event Counter will reload the value already loaded into the preload register and continue counting. The interrupt can be disabled by ensuring that the Timer/Event Counter Interrupt Enable bit in the Interrupt Control Register, is reset to zero.

Configuring the Event Counter Mode

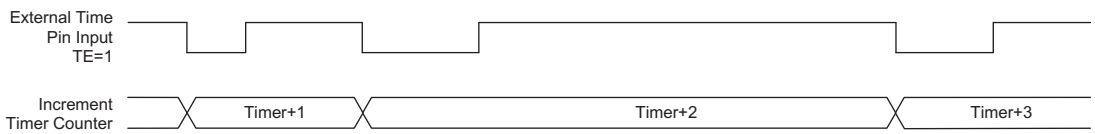
In this mode, a number of externally changing logic events, occurring on the external timer pin, can be recorded by the Timer/Event Counter. To operate in this mode, the Operating Mode Select bit pair in the Timer Control Register must be set to the correct value as shown.

Control Register Operating Mode Select Bits for the Event Counter Mode	Bit7	Bit6
	0	1

In this mode the external timer pin is used as the Timer/Event Counter clock source, however it is not divided by the internal prescaler. After the other bits in the Timer Control Register have been setup, the enable bit, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter to run. If the Active Edge Select bit, which is bit 3 of the Timer Control Register, is low, the Timer/Event Counter will increment each time the external timer pin receives a low to high transition. If the Active Edge Select bit is high, the counter will increment each time the external timer pin receives a high to low transition. When it is full and overflows, an interrupt signal is generated and the Timer/Event Counter will reload the value already loaded into the preload register and continue counting. The interrupt can be disabled by ensuring that the Timer/Event Counter Interrupt Enable bit in the Interrupt Control Register, is reset to zero.



Timer Mode Timing Diagram



Event Counter Mode Timing Diagram

To ensure that the timer pin is configured to operate as an event counter input pin the Timer Control Register must place the Timer/Event Counter in the Event Counting Mode. It should be noted that in the event counting mode, even if the microcontroller is in the Power Down Mode, the Timer/Event Counter will continue to record externally changing logic events on the timer input pin. As a result when the timer overflows it will generate a timer interrupt and corresponding wake-up source.

Configuring the Pulse Width Measurement Mode

In this mode, the Timer/Event Counter can be utilised to measure the width of external pulses applied to the external timer pin. To operate in this mode, the Operating Mode Select bit pair in the Timer Control Register must be set to the correct value as shown.

Control Register Operating Mode Select Bits for the Pulse Width Measurement Mode	Bit7	Bit6
	1	1

In this mode the internal clock is used as the Timer/Event Counter clock. After the other bits in the Timer Control Register have been setup, the enable bit, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter, however it will not actually start counting until an active edge is received on the external timer pin.

If the Active Edge Select bit, which is bit 3 of the Timer Control Register, is low, once a high to low transition has been received on the external timer pin, the Timer/Event Counter will start counting until the external timer pin returns to its original high level. At this point the enable bit will be automatically reset to zero and the Timer/Event Counter will stop counting. If the Active Edge Select bit is high, the Timer/Event Counter will begin counting once a low to high transition has been received on the external timer pin and stop counting when the external timer pin returns to its original low level. As before, the enable bit will be automatically reset to zero and the Timer/Event Counter will stop counting. It is important to note that in the Pulse Width Measurement Mode, the enable bit is automatically reset to zero when the external control signal on the external timer pin returns to its original level, whereas in the other two modes the enable bit can only be reset to zero under program control.

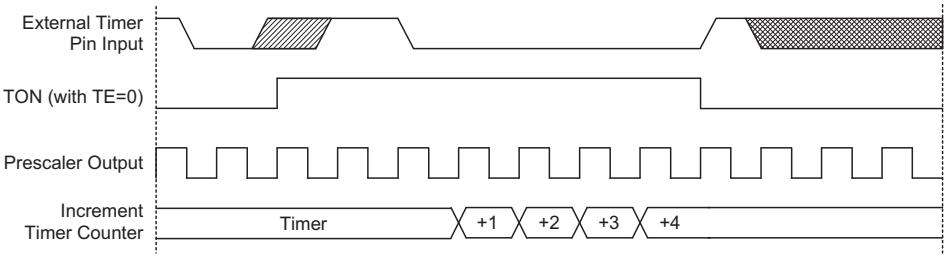
The residual value in the Timer/Event Counter, which can now be read by the program, therefore represents the length of the pulse received on the external timer pin. As the enable bit has now been reset, any further transitions on the external timer pin will be ignored. Not until the enable bit is again set high by the program can the timer begin further pulse width measurements. In this way, single shot pulse measurements can be easily made.

It should be noted that in this mode the Timer/Event Counter is controlled by logical transitions on the external timer pin and not by the logic level. When the Timer/Event Counter is full and overflows, an interrupt signal is generated and the Timer/Event Counter will reload the value already loaded into the preload register and continue counting. The interrupt can be disabled by ensuring that the Timer/Event Counter Interrupt Enable bit in the Interrupt Control Register, is reset to zero.

To ensure that the timer pin is configured to operate as a pulse width measurement pin the Timer Control Register must place the Timer/Event Counter in the Pulse Width Measurement Mode.

Programming Considerations

When configured to run in the timer mode, the internal system clock is used as the timer clock source and is therefore synchronized with the overall operation of the microcontroller. In this mode, when the appropriate timer register is full, the microcontroller will generate an internal interrupt signal directing the program flow to the respective internal interrupt vector. For the pulse width measurement mode, the internal system clock is also used as the timer clock source but the timer will only run when the correct logic condition appears on the external timer input pin. As this is an external event and not synchronised with the internal timer clock, the microcontroller will only see this external event when the next timer clock pulse arrives. As a result there may be small differences in measured values requiring programmers to take this into account during programming. The same applies if the timer is configured to be in the event counting mode which again is an external event and not synchronised with the internal system or timer clock.



Prescaler Output is sampled at every falling edge of T1.

Pulse Width Measure Mode Timing Diagram

When the Timer/Event Counter is read or if data is written to the preload registers, the clock is inhibited to avoid errors, however as this may result in a counting error, this should be taken into account by the programmer. Care must be taken to ensure that the timers are properly initialised before using them for the first time. The associated timer enable bits in the interrupt control register must be properly set otherwise the internal interrupt associated with the timer will remain inactive. The edge select, timer mode and clock source control bits in timer control register must also be correctly set to ensure the timer is properly configured for the required application. It is also important to ensure that an initial value is first loaded into the timer register before the timer is switched on; this is because after power-on the initial value of the timer register is unknown. After the timer has been initialised the timer can be turned on and off by controlling the enable bit in the timer control register. Note that setting the timer enable bit high to turn the timer on, should only be executed after the timer mode bits have been properly setup. Setting the timer enable bit high together with a mode bit modification, may lead to improper timer operation if executed as a single timer control register byte write instruction.

When the Timer/Event counter overflows, its corresponding interrupt request flag in the interrupt control register will be set. If the timer interrupt is enabled this will in turn generate an interrupt signal. However irrespective of whether the interrupts are enabled or not, a Timer/Event counter overflow will also generate a wake-up signal if the device is in a Power-down condition. This situation may occur if the Timer/Event Counter is in the Event Counting Mode and if the external signal continues to change state. In such a case, the Timer/Event Counter will continue to count these external events and if an overflow occurs the device will be woken up from its Power-down condition. To prevent such a wake-up from occurring, the timer interrupt request flag should first be set high before issuing the HALT instruction to enter the Power Down Mode.

Timer Program Example

This program example shows how the Timer/Event Counter registers are setup, along with how the interrupts are enabled and managed. Note how the Timer/Event Counter is turned on, by setting bit 4 of the Timer Control Register. The Timer/Event Counter can be turned off in a similar way by clearing the same bit.

This example program sets the Timer/Event Counter tobe in the timer mode, which uses the internal system clock as the clock source.

```
Org 04h                 ; external interrupt vector
reti
Org 08h                 ; Timer/Event Counter 0 interrupt vector
jmp tmr0nt              ; jump here when Timer 0 overflows
    :
org 20h                 ; main program
;internal Timer/Event Counter interrupt routine
tmr0nt:
    :
; Timer/Event Counter main program placed here
    :
 reti
    :
    :
begin:
;setup Timer registers
    mov a,01fh          ; setup preload value – timer counts from this value to FFFFH
    mov tmr01,a
    mov a,09bh
    mov tmr0h,a
    mov a,080h          ; setup Timer control register
    mov tmr0c,a         ; timer mode
; setup interrupt register
    mov a,005h          ; enable master interrupt and timer interrupt
    mov intc0,a
    set tmrc0.4         ; start Timer – note mode bits must be previously setup
    :
    :
```

Interrupts

Interrupts are an important part of any microcontroller system. When an external event or an internal function such as a Timer/Event Counter 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. Each device contains a single external interrupt and single internal timer interrupt functions. The external interrupt is controlled by the action of the external $\overline{INT}$ pin, while the internal interrupt are controlled by Timer/Event Counter overflow, a Real Time Clock overflow, a DTMF reciever valid character reception etc.

Interrupt Register

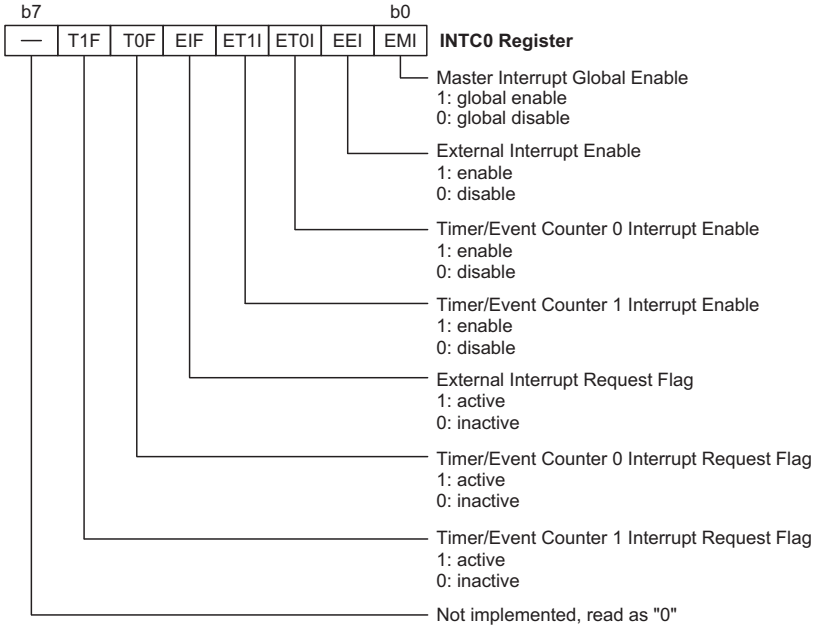
Overall interrupt control, which means interrupt enabling and request flag setting, is controlled by two interrupt control registers, INTC0 and INTC1, located in the Data Memory. By controlling the appropriate enable bits in this register each individual interrupt can be enabled or disabled. Also when an interrupt occurs, the corresponding request flag will be set by the microcontroller. The global enable flag if cleared to zero will disable all interrupts.

Interrupt Operation

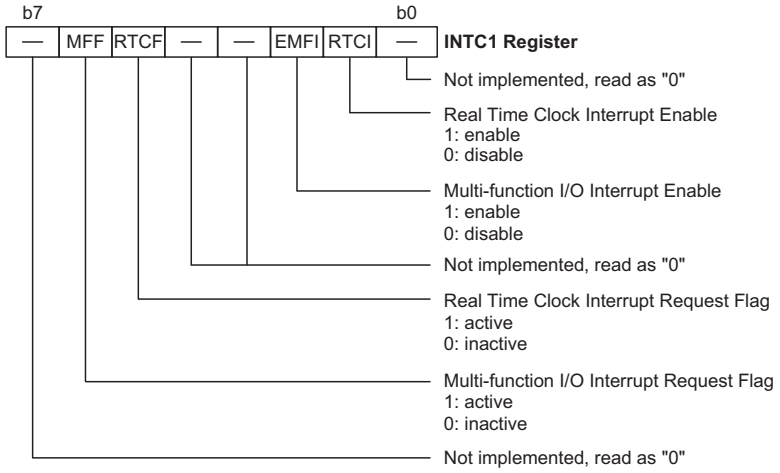
A Timer/Event Counter overflow, the reception of a valid DTMF character etc. or the external interrupt line being pulled low will all generate an interrupt request by setting their corresponding request flag, if their appropriate interrupt enable bit is set. When this happens, the Program Counter, which 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 statement which will take program execution 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 statement, 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 diagram with their order of priority.

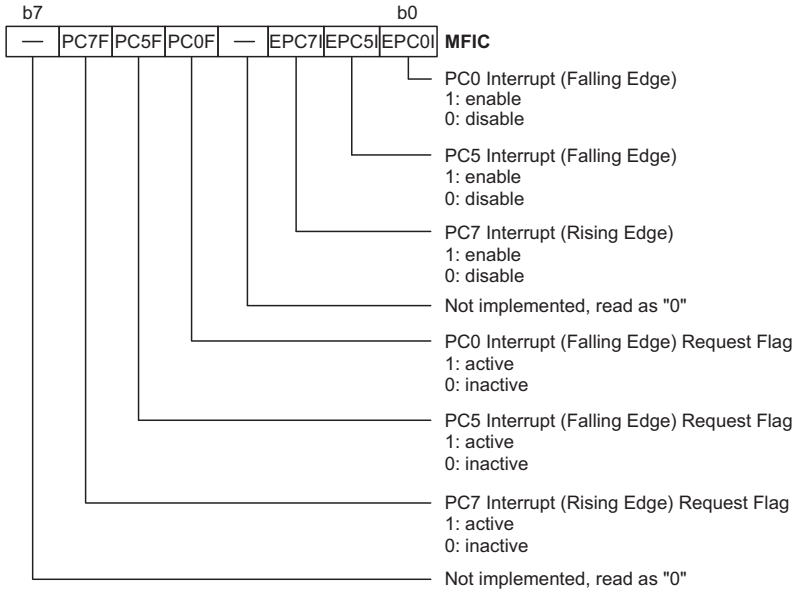
Once an interrupt subroutine is serviced, all the other interrupts will be blocked, as the EMI bit will be cleared automatically. This will prevent any further interrupt nesting from occurring. However, if other interrupt requests occur during this interval, although the interrupt will not be immediately 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.



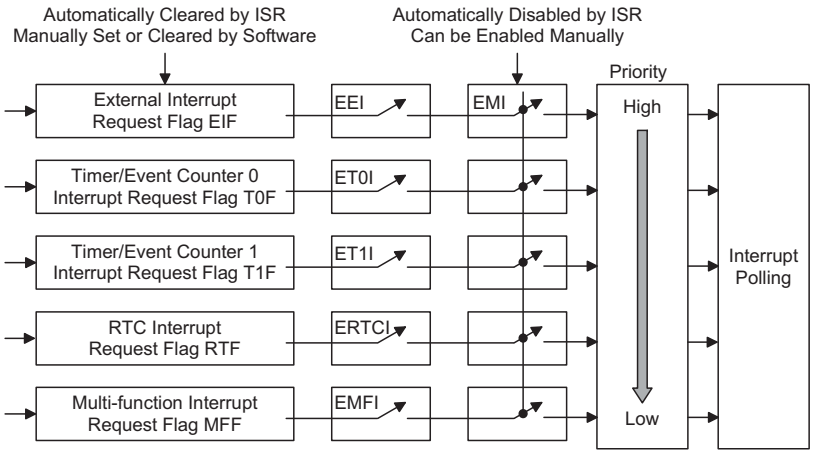
Interrupt Control 0 Register



Interrupt Control 1 Register



MFIC Register



Interrupt Structure

Interrupt Priority

Interrupts, occurring in the interval between the rising edges of two consecutive T2 pulses, will be serviced on the latter of the two T2 pulses, if the corresponding interrupts are enabled. In case of simultaneous requests, the following table shows the priority that is applied. These can be masked by resetting the EMI bit.

Interrupt Source	All Devices Priority
Reset	1
External Interrupt	2
Timer 0 Interrupt	3
Timer 1 Interrupt	4
Real Time Clock Interrupt	5
Multi-function Interrupt	6

In cases where both external and internal interrupts are enabled and where an external and internal interrupt occurs simultaneously, the external interrupt will always have priority and will therefore be serviced first. Suitable masking of the individual interrupts using the INTC register can prevent simultaneous occurrences.

External Interrupt

For an external interrupt to occur, the global interrupt enable bit, EMI, and external interrupt enable bit, EEI, must first be set. An actual external interrupt will take place when the external interrupt request flag, EIF, is set, a situation that will occur when a high to low transition appears on the $\overline{INT}$ line. When the interrupt is enabled, the stack is not full and a high to low transition appears on the external interrupt pin, a subroutine call to the external interrupt vector at location 04H, will take place. When the interrupt is serviced, the external interrupt request flag, EIF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

Timer/Event Counter Interrupt

For a Timer/Event Counter interrupt to occur, the global interrupt enable bit, EMI, and the corresponding timer interrupt enable bit, ET0I or ET1I, must first be set. An actual Timer/Event Counter interrupt will take place when the Timer/Event Counter request flag, T0F or T1F, is set, a situation that will occur when the Timer/Event Counter overflows. When the interrupt is enabled, the stack is not full and a Timer/Event Counter overflow occurs, a subroutine call to the timer interrupt vector at location 08H or 0CH, will take place. When the interrupt is serviced, the timer interrupt request flag, T0F or T1F, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

Real Time Clock Interrupt

For a Real Time Clock interrupt to occur, the global interrupt enable bit, EMI, and the corresponding real timer clock interrupt enable bit, RTCI, must first be set. An actual Real Time Clock interrupt will take place when the Real Time Clock request flag, RTCF, is set, a situation that will occur when the RTC times out which will occur every second. When the interrupt is enabled, the stack is not full and a Real Time Clock interrupt request occurs, a subroutine call to the real time clock interrupt vector at location 14H, will take place. When the interrupt is serviced, the timer interrupt request flag, RTCF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

Multi-function Interrupt

For a Multi-function interrupt to occur, the global interrupt enable bit, EMI, and the corresponding multi-function interrupt enable bit, EMFI, must first be set. An actual Multi-function interrupt will take place when the Multi-function interrupt request flag, MFF, is set, a situation that will occur when PC0 or PC5 has a falling edge or PC7 has a rising edge. When the interrupt is enabled, the stack is not full and a Multi-function

interrupt request occurs, a subroutine call to the multi-function interrupt vector at location 18H, will take place. When the interrupt is serviced, the multi-function interrupt request flag, MFF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

Programming Considerations

By disabling the 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 INTC register until the corresponding interrupt is serviced or until the request flag is cleared by a software instruction.

It is recommended that programs do not use the "CALL subroutine" instruction within the interrupt subroutine. Interrupts often occur in an unpredictable manner or need to be serviced immediately in some applications. 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.

All of these interrupts have the capability of waking up the processor when in the Power Down Mode. Only the Program Counter is pushed onto the stack. If the contents of the register or status register are altered by the interrupt service program, which may corrupt the desired control sequence, then the contents should be saved in advance.

Reset and Initialisation

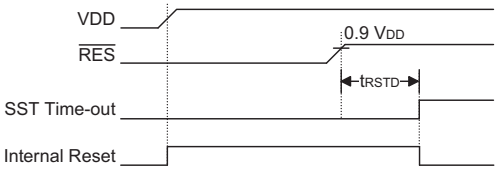
A reset function is a fundamental part of any microcontroller ensuring that the device can be set to some predetermined condition irrespective of outside parameters. The most important reset condition is after power is first applied to the microcontroller. In this case, internal circuitry will ensure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

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

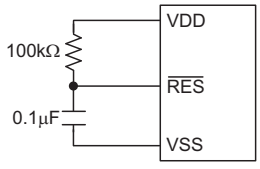
There are five ways in which a microcontroller reset can occur, through events occurring both internally and externally:

- Power-on Reset
  The most fundamental and unavoidable reset is the one that occurs after power is first applied to the microcontroller. As well as ensuring that the Program Memory begins execution from the first memory address, a power-on reset also ensures that certain other registers are preset to known conditions. All the I/O port and port control registers will power up in a high condition ensuring that all pins will be first set to inputs.
  Although the microcontroller has an internal RC reset function, if the VDD power supply rise time is not fast enough or does not stabilise quickly at power-on, the internal reset function may be incapable of providing proper reset operation. For this reason it is recommended that an external RC network is connected to the $\overline{\text{RES}}$ pin, whose additional time delay will ensure that the $\overline{\text{RES}}$ pin remains low for an extended period to allow the power supply to stabilise. During this time delay, normal operation of the microcontroller will be inhibited. After the $\overline{\text{RES}}$ line reaches a certain voltage value, the reset delay time t_{RSTD} is invoked to provide an extra delay time after which the microcontroller will begin normal operation. The abbreviation SST in the figures stands for System Start-up Timer.



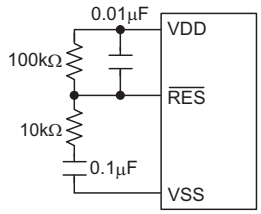
Power-On Reset Timing Chart

For most applications a resistor connected between VDD and the $\overline{\text{RES}}$ pin and a capacitor connected between VSS and the $\overline{\text{RES}}$ pin will provide a suitable external reset circuit. Any wiring connected to the $\overline{\text{RES}}$ pin should be kept as short as possible to minimise any stray noise interference.



Basic Reset Circuit

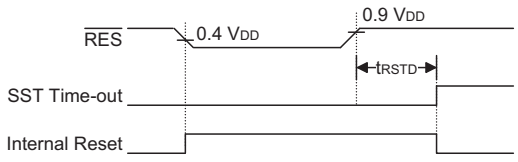
For applications that operate within an environment where more noise is present the Enhanced Reset Circuit shown is recommended.



Enhanced Reset Circuit

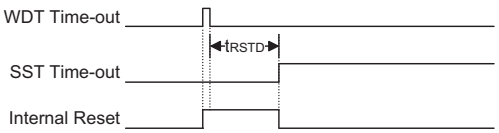
More information regarding external reset circuits is located in Application Note HA0075E on the Holtek website.

- $\overline{RES}$ Pin Reset
  This type of reset occurs when the microcontroller is already running and the $\overline{RES}$ pin is forcefully pulled low by external hardware such as an external switch. In this case as in the case of other reset, the Program Counter will reset to zero and program execution initiated from this point.



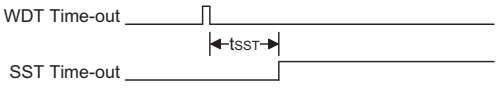
$\overline{RES}$ Reset Timing Chart

- Watchdog Time-out Reset during Normal Operation
  The Watchdog time-out Reset during normal operation is the same as a hardware $\overline{RES}$ pin reset except that the Watchdog time-out flag TO will be set to "1".



WDT Time-out Reset during Normal Operation Timing Chart

- Watchdog Time-out Reset during Power Down
  The Watchdog time-out Reset during Power Down is a little different from other kinds of reset. Most of the conditions remain unchanged except that the Program Counter and the Stack Pointer will be cleared to "0" and the TO flag will be set to "1". Refer to the A.C. Characteristics for t_{SST} details.



WDT Time-out Reset during Power Down 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 Power Down function or Watchdog Timer. The reset flags are shown in the table:

TO	PDF	RESET Conditions
0	0	$\overline{RES}$ reset during power-on
u	u	$\overline{RES}$ reset during normal operation
1	u	WDT time-out reset during normal operation
1	1	WDT time-out reset during Power Down

Note: "u" stands for unchanged

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

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

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

Register	Reset (Power-on)	$\overline{\text{RES}}$ Reset (Normal/Green)	$\overline{\text{RES}}$ Reset (Sleep/Idle)	WDT Time-out (Normal/Green)	WDT Time-out (Sleep/Idle)
IAR0	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
MP0	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
IAR1	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
MP1	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
BP	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCL	0000H	0000H	0000H	0000H	0000H
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
WDTS	0000 0111	0000 0111	0000 0111	0000 0111	uuuu uuuu
STATUS	--00 xxxx	--uu uuuu	--01 uuuu	--1u uuuu	--11 uuuu
INTC0	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
TMR0H	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TMR0L	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TMR0C	00-0 1---	00-0 1---	00-0 1---	00-0 1---	uu-u u---
TMR1H	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TMR1L	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TMR1C	00-0 1---	00-0 1---	00-0 1---	00-0 1---	uu-u u---
PA	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PB	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PCC	1111 --11	1111 --11	1111 --11	1111 --11	uuuu --uu
PD	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PDC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PE	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PEC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PF	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PFC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PG	---- 1111	---- 1111	---- 1111	---- 1111	---- uuuu
PGC	---- 1111	---- 1111	---- 1111	---- 1111	---- uuuu
INTC1	-00- -00-	-00- -00-	-00- -00-	-00- -00-	-uu- -uu-
DTMFC	---- 0--1	---- 0--1	---- 0--1	---- 0--1	---- -u-u
DTMFD	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
RTCC	0-0- ----	u-u- ----	u-u- ----	u-u- ----	u-u- ----

Register	Reset (Power-on)	$\overline{RES}$ Reset (Normal/Green)	$\overline{RES}$ Reset (Sleep/Idle)	WDT Time-out (Normal/Green)	WDT Time-out (Sleep/Idle)
MODE	000– ----	00u– ----	00u– ----	00u– ----	00u– ----
MFIC	–000 –000	–000 –000	–000 –000	–000 –000	–uuu –uuu
PFDC	0000 ----	0000 ----	0000 ----	0000 ----	uuuu ----
PFDD	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu

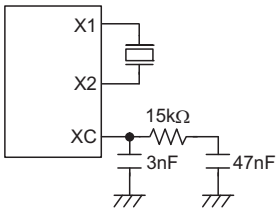
Note: "u" stands for unchanged
"x" stands for unknown
"–" stands for unimplemented

Oscillator

There are two oscillator circuits within the controller. One is for the system clock which uses an externally connected 32768Hz crystal. The other is an internal watchdog oscillator.

System Crystal/Ceramic Oscillator

The system clock is generated using an external 32768Hz crystal or ceramic resonator connected between pins X1 and X2. From this clock source an internal circuit generates a 3.58MHz clock source which is also required by the system. This frequency generator circuit requires the addition of externally connected RC components to pin XC to form a low pass filter for the 3.58MHz output frequency stabilisation.



Crystal/Ceramic Oscillator

Watchdog Timer Oscillator

The WDT oscillator is a fully integrated free running RC oscillator with a typical period of 65μs at 5V, requiring no external components. It is selected via configuration option. If selected, when the device enters the Power Down Mode, the system clock will stop running, however the WDT oscillator will continue to run and keep the watchdog function active. However, as the WDT will consume a certain amount of power when in the Power Down Mode, for low power applications, it may be desirable to disable the WDT oscillator by configuration option.

Operation Mode, Power-down and Wake-up

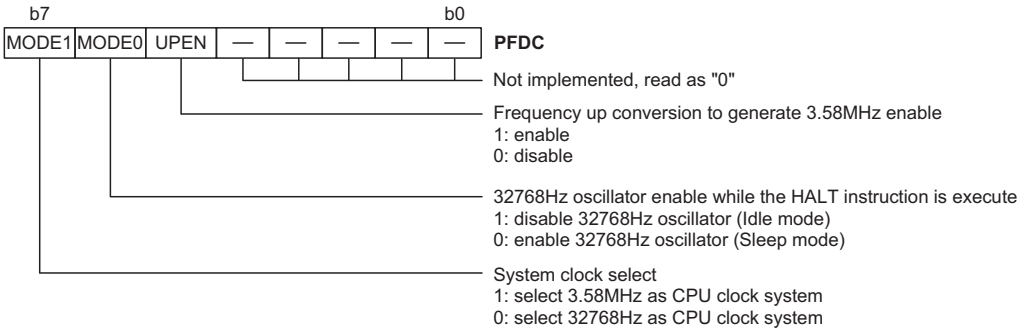
There are four operational modes, known as the idle mode, sleep mode, green mode, and normal mode. The chosen mode is selected using the MODE0, MODE1 and UPEN bits in the PFDC register but also depends upon whether the HALT instruction has been executed or not.

Idle Mode

When the device enters this mode, the normal operating current, will be reduced to an extremely low standby current level. This occurs because when the device enters the Power Down Mode, both the 3.58MHz and 32768Hz system oscillators are stopped which reduces the power consumption to extremely low levels, however, as the device maintains its present internal condition, it can be woken up at a later stage and continue running, without requiring a full reset. This feature is extremely important in application areas where the microcontroller must have its power supply constantly maintained to keep the device in a known condition but where the power supply capacity is limited such as in battery applications.

HALT Instruction	MODE1	MODE0	UPEN	Operation Mode	32768Hz	3.58MHz	System Clock
Not executed	1	X	1	Normal	ON	ON	3.58MHz
Not executed	0	X	0	Green	ON	OFF	32768Hz
Executed	0	0	0	Sleep	ON	OFF	Stopped
Executed	0	1	0	Idle	OFF	OFF	Stopped

Note: "X" means don't care
MODE0 will be cleared to 0 automatically after wake-up from Idle Mode.



PFDC Register

Sleep Mode

In the Sleep Mode is similar to the mode, except here the 32768Hz oscillator continues running after after the HALT instruction has been executed. This feature enables the device to continue with instruction execution immediately after wake-up.

Green Mode

In the Green Mode, the 32768Hz oscillator is used as the system clock for instruction execution. The following conditions will force the microcontroller enter the Green Mode:

- Any reset condition from any operational mode
- Any interrupt occurring during the Sleep Mode or Idle mode
- A Port A Wake-up from the Sleep Mode or Idle Mode

Normal Mode

In the Normal mode the device uses the 3.58MHz generated by the frequency-up conversion circuit as the system clock for instruction execution.

Changing the Operational Mode

Holtek's telephone controllers support two system clocks and four operational modes. The system clock can be either 32768Hz or 3.58MHz and the operational mode can be either Normal, Green, Sleep or Idle mode. The operation mode is selected using software in the following way:

- Normal mode to Green mode:
  Clear bit MODE1 to 0, which will change the operational mode to the Green mode.
  The UPEN bit status is not changed. However, the UPEN bit can be cleared by software.

- Normal mode or Green mode to Sleep mode:
  Step 1: Clear bit MODE0 to 0
  Step 2: Clear bit MODE1 to 0
  Step 3: Clear bit UPEN to 0
  Step 4: Execute the HALT instruction
  After Step 4, the operational mode will be changed to the Sleep mode.

- Normal mode or Green mode to Idle mode:
  Step 1: Set bit MODE0 to 1
  Step 2: Clear bit MODE1 to 0
  Step 3: Clear bit UPEN to 0
  Step 4: Execute the HALT instruction
  After Step 4, the operational mode will be changed to the Idle mode.

- Green mode to Normal mode:
  Step 1: Set bit UPEN to 1
  Step 2: Execute a 20ms software delay
  Step 3: Set bit MODE1 to 1
  After Step 3, the operational mode will be changed to the Normal mode.

- Sleep mode or Idle mode to Green mode:
  Method 1: The occurrence of any reset condition
  Method 2: Any active interrupt
  Method 3: A Port A wake-up

Note that a Timer/Event Counter 0/1 and RTC interrupt will not be generated when in the Idle mode as the 32768Hz crystal oscillator is stopped.

Standby Current Considerations

As the main reason for entering the Power Down Mode is to keep the current consumption of the MCU to as low a value as possible, perhaps only in the order of several micro-amps, 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. Care must also be taken with the loads, which are connected to I/Os, 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 configuration options have enabled the Watchdog Timer internal oscillator.

Wake-up

A reset, interrupt or port A wake-up can all wake up the device from the Sleep Mode or the Idle Mode. A reset can include a power-on reset, an external reset or a WDT time-out reset. By examining the device status flags, PDF and TO, the program can distinguish between the different reset conditions. Refer to the Reset section for a more detailed description.

A port A wake-up and an interrupt can be considered as a continuation of normal execution. Each bit in port A can be independently selected to wake-up the device using configuration options. When awakened by a Port A stimulus, the program will resume execution at the next instruction following the HALT instruction.

Any valid interrupt during the Sleep Mode or Idle Mode may have one of two consequences. One is if the related interrupt is disabled or the interrupt is enabled but the stack is full, the program will resume execution at the next instruction. The other is if the interrupt is enabled and the stack is not full, the regular interrupt response takes place. It is necessary to mention that if an interrupt request flag is set to "1" before entering the Sleep mode or Idle mode, the Wake-up function of the related interrupt will be disabled.

Once a Sleep mode or Idle mode Wake-up event occurs, it will take an SST delay time, which is 1024 system clock periods, to resume to the Green mode. This means that a dummy period is inserted after a Wake-up. If the Wake-up results from an interrupt acknowledge signal, the actual interrupt subroutine execution will be delayed by one or more cycles. If the Wake-up results in the next instruction execution, this will be executed immediately after the dummy period has finished.

To minimise power consumption, all the I/O pins should be carefully managed before entering the Sleep Mode or Idle Mode.

The Sleep Mode or Idle Mode is initialised by a HALT instruction and results in the following.

- The system clock will be turned off.
- The WDT function will be disabled if the WDT clock source is the instruction clock.
- The WDT function will be disabled if the WDT clock source is the 32768Hz oscillator in the Idle mode.
- The WDT will still function if the WDT clock source is the WDT internal oscillator.
- If the WDT function is still enabled, the WDT counter and WDT prescaler will be cleared and resume counting.
- The contents of the on chip Data Memory and registers remain unchanged.
- All the I/O ports maintain their original status.
- The PDF flag is set and the TO flag is cleared by hardware.

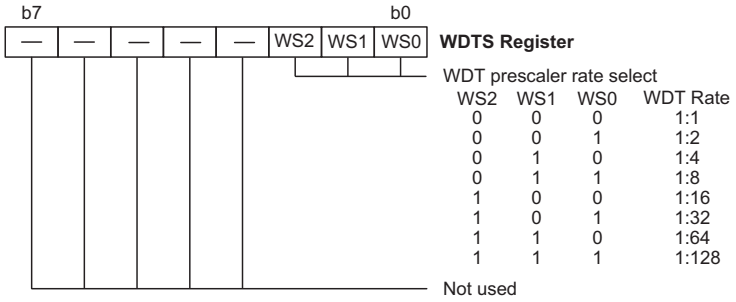
Watchdog Timer

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. It operates by providing a device reset when the WDT counter overflows. The WDT clock is supplied by one of two sources selected by a configuration option. These an be either its own self-contained dedicated internal WDT oscillator, or the instruction clock which is the system clock divided by 4. Note that if the WDT configuration option has been disabled, then any instruction relating to its operation will result in no operation.

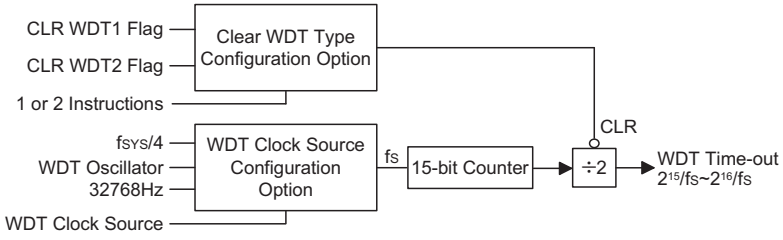
A configuration option can select the instruction clock, which is the system clock divided by 4, as the WDT clock source instead of the internal WDT oscillator. If the instruction clock is used as the clock source, it must be noted that when the system enters the Power Down Mode, as the system clock is stopped, then the WDT clock source will also be stopped. Therefore the WDT will lose its protecting purposes. In such cases the system cannot be restarted by the WDT and can only be restarted using external signals. For systems that operate in noisy environments, using the internal WDT oscillator is therefore the recommended choice.

Under normal program operation, a WDT time-out will initialise a device reset and set the status bit TO. However, if the system is in the Power Down Mode, when a WDT time-out occurs, only the Program Counter and Stack Pointer will be reset. Three methods can be adopted to clear the contents of the WDT and the WDT prescaler. The first is an external hardware reset, which means a low level on the $\overline{\text{RES}}$ pin, the second is using the watchdog software instructions and the third is via a ″HALT″ instruction.

There are two methods of using software instructions to clear the Watchdog Timer, one of which must be chosen by configuration option. The first option is to use the single ″CLR WDT″ instruction while the second is to use the two commands ″CLR WDT1″ and ″CLR WDT2″. For the first option, a simple execution of ″CLR WDT″ will clear the WDT while for the second option, both ″CLR WDT1″ and ″CLR WDT2″ must both be executed to successfully clear the WDT. Note that for this second option, if ″CLR WDT1″ is used to clear the WDT, successive executions of this instruction will have no effect, only the execution of a ″CLR WDT2″ instruction will clear the WDT. Similarly, after the ″CLR WDT2″ instruction has been executed, only a successive ″CLR WDT1″ instruction can clear the Watchdog Timer.



Watchdog Timer Register



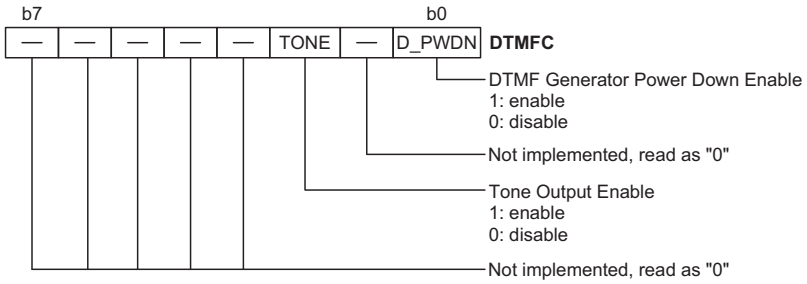
Watchdog Timer

DTMF Generator

The device includes a fully integrated DTMF, Dual-Tone Multiple-Frequency, generator function. This functional block can generate the necessary16 dual tones and 8 single tones for DTMF signal generation. The signal will be provided on the DTMF pin of the device. The DTMF generator also includes a power down and a tone on/off function. The clock source for the DTMF generator is the 3.58MHz oscillator. Before the DTMF function is used, the device must have been placed into the Normal mode.

DTMF Generator Control

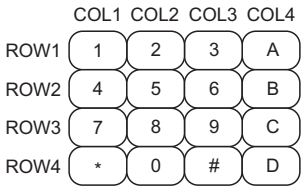
The DTMF Generator is controlled by two registers, a control register known as DTMFC and a data register known as DTMFD. The power down mode will terminate all the DTMF generator functions and can be activated by setting the D_PWDN bit in the DTMFC register to 1. These two registers, DTMFC and DTMFD are still accessible even if the DTMF function is in the power down mode. The generation duration time of the DTMF output signal should be determined by the software. The DTMFD register value can be changed as desired, at which point the DTMF pin will output the new dual-tone simultaneously.



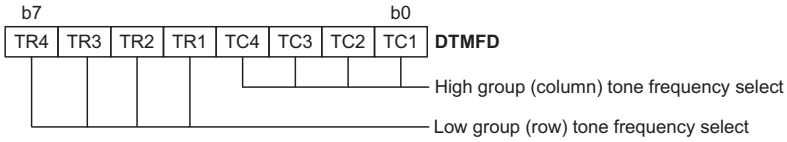
DTMF Generator Control Register

DTMF Generator Frequency Selection

The DTMF pin output is controlled using a combination of the D_PWDN, TONE, TR~TC bits.



DTMF Dialing Matrix



DTMF Generator Data Register

Control Register Bits			DTMF Pin Output Status
D_PWDN	TONE	TR4~TR1/TC4~TC1	
1	x	x	0
0	0	x	1/2 VDD
0	1	0	1/2 VDD
0	1	Any valid value	16 dual tones or 8 signal tones, bias at 1/2 VDD

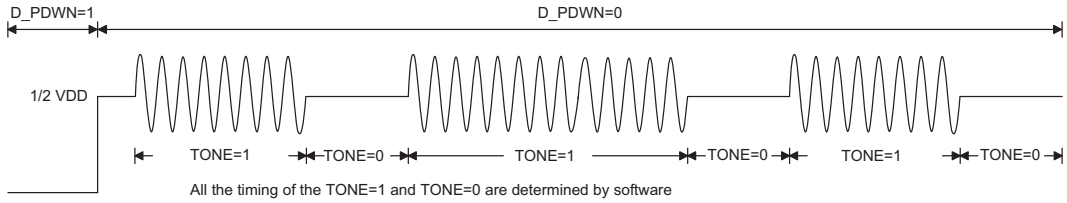
Output Frequency (Hz)		% Error
Specified	Actual	
697	699	+0.29%
770	766	–0.52%
852	847	–0.59%
941	948	+0.74%
1209	1215	+0.50%
1336	1332	–0.30%
1477	1472	–0.34%

% Error does not contain the crystal frequency shift

DTMF Frequency Selection Table

Low Group				High Group				DTMF Output		Code
TR4	TR3	TR2	TR1	TC4	TC3	TC2	TC1	Low	High	
0	0	0	1	0	0	0	1	697	1209	1
0	0	0	1	0	0	1	0	697	1336	2
0	0	0	1	0	1	0	0	697	1477	3
0	0	0	1	1	0	0	0	697	1633	A
0	0	1	0	0	0	0	1	770	1209	4
0	0	1	0	0	0	1	0	770	1336	5
0	0	1	0	0	1	0	0	770	1477	6
0	0	1	0	1	0	0	0	770	1633	B
0	1	0	0	0	0	0	1	852	1209	7
0	1	0	0	0	0	1	0	852	1336	8
0	1	0	0	0	1	0	0	852	1477	9
0	1	0	0	1	0	0	0	852	1633	C
1	0	0	0	0	0	0	1	941	1209	*
1	0	0	0	0	0	1	0	941	1336	0
1	0	0	0	0	1	0	0	941	1477	#
1	0	0	0	1	0	0	0	941	1633	D
Single tone for testing only										
0	0	0	1	0	0	0	0	697	x	x
0	0	1	0	0	0	0	0	770	x	x
0	1	0	0	0	0	0	0	852	x	x
1	0	0	0	0	0	0	0	941	x	x
0	0	0	0	0	0	0	1	x	1209	x
0	0	0	0	0	0	1	0	x	1336	x
0	0	0	0	0	1	0	0	x	1477	x
0	0	0	0	1	0	0	0	x	1633	x

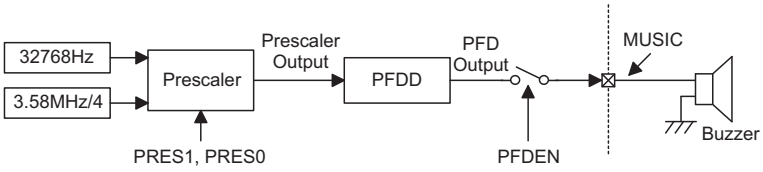
Writing other values to TR4~TR1, TC4~TC1 may generate an unpredictable tone.



DTMF Output

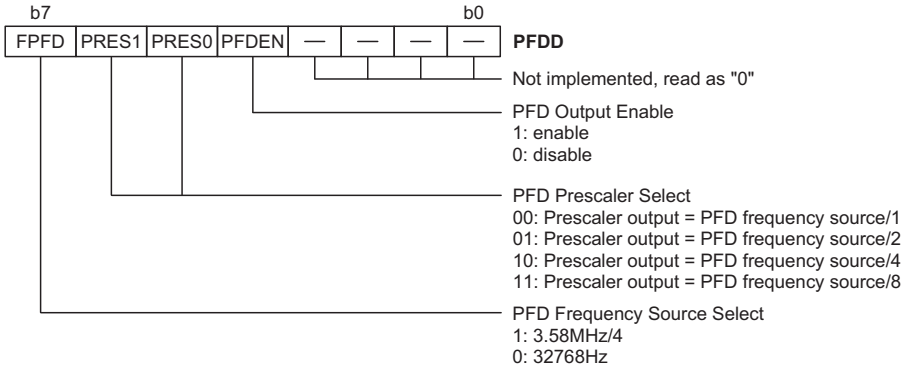
Programmable Frequency Divider (PFD) Generator – MUSIC

A Programmable Frequency Divider function, otherwise known as PFD, is integrated within the microcontroller, providing a means of accurate frequency generation. It is composed of two functional blocks: a prescaler and a general counter.



PFD Control Register

The overall PFD function is controlled using the PFDD register. The prescaler is controlled by the register bits, PRES0 and PRES1. The general counter is programmed by an 8-bit register PFDD. The clock source for the PFD can be selected to be either the 3.58MHz/4 or the 32768Hz oscillator. To enable the PFD output, the PFDEN bit should be set to 1. When the PFD is disabled the PFDD register is inhibited to be written to. To modify the PFDD contents, the PFD must be enabled. When the generator is disabled, the PFDD is cleared by hardware.



PFD Data Register

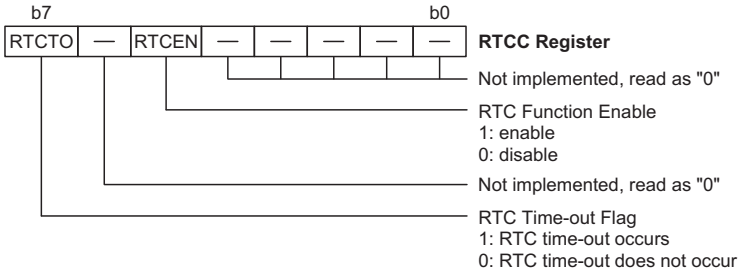
Bit No.	Label	R/W	Function
7~0	—	RW	PFD data register

PFDD (2FH) Register

$$PFD\_Output\_Frequency = \frac{Prescaler\_Output}{2x(N+1)}$$, where N = the value of the PFD Data

RTC Function

When RTC 1000ms time-out occurs, the hardware will set the interrupt request flag RTCF and the RTCTO flag to 1. When the interrupt service routine is serviced, the interrupt request flag (RTCF) will be cleared to 0, but the flag RTCTO remains in its original values. This bit (RTCTO) should be cleared only by software. However, next RTC interrupt will still occur, even though the RTCTO flag is not cleared.



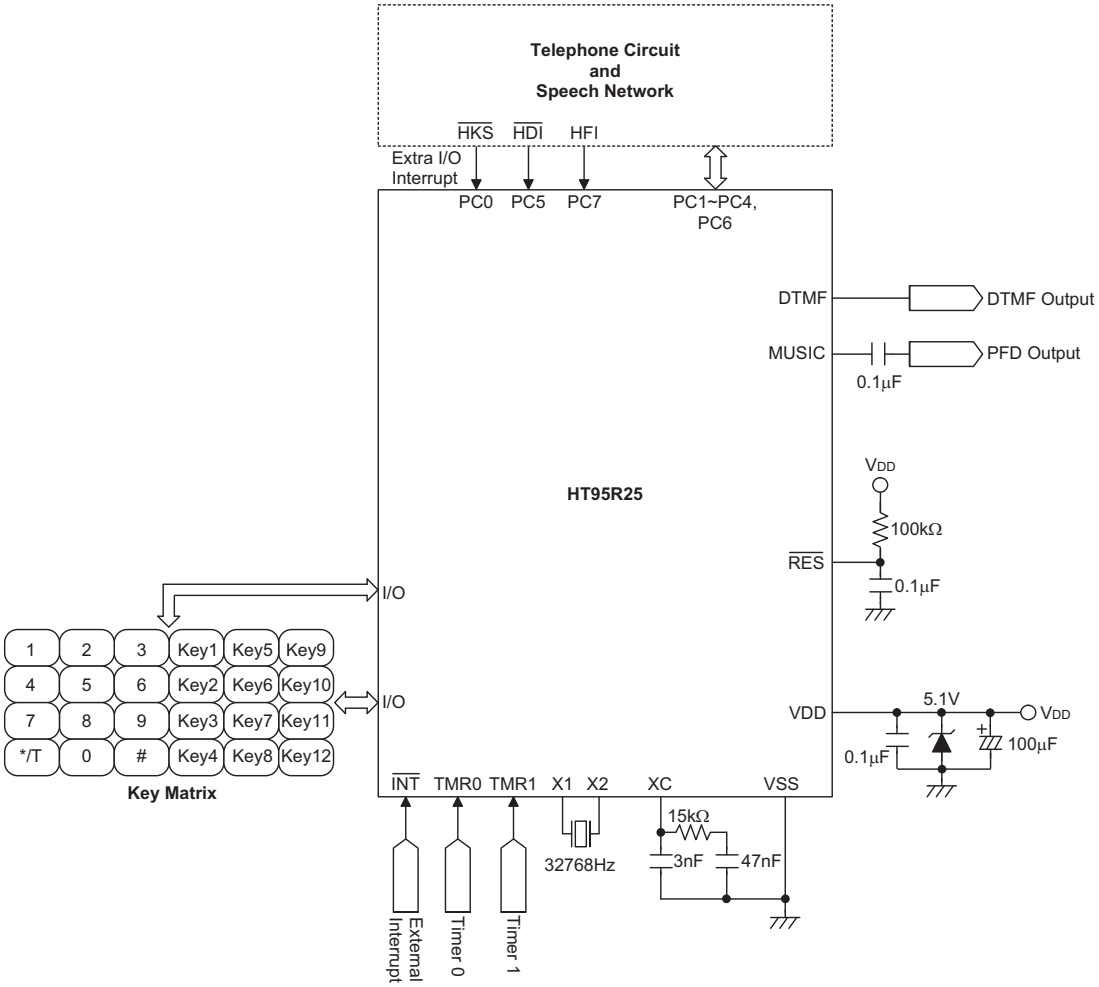
Configuration Options

Configuration options refer to certain options within the MCU that are programmed into the MTP Program Memory device during the programming process. During the development process, these options are selected using the HT-IDE software development tools. As these options are programmed into the device using the hardware programming tools, once they are selected they cannot be changed later by the application software.

All options must be defined for proper system function, the details of which are shown in the table.

Name	Options
Wake-up PA	Port A wake-up selection. Define the activity of wake-up function. All port A have the capability to wake-up the chip from a HALT. This wake-up function is selected per bit.
Pull-high PA Pull-high PB Pull-high PC0~PC1 Pull-high PC4~PC7 Pull-high PD Pull-high PE Pull-high PF Pull-high PG3~PG0	Pull-high option. This option determines whether the pull-high resistance is viable or not. Port A pull-high option is selected per bit. Port B pull-high option is selected per bit. Port C pull-high option is selected per bit. Port D pull-high option is selected per nibble. Port E pull-high option is selected per nibble. Port F pull-high option is selected per nibble. Port G pull-high option is selected per nibble.
CLRWDT	This option defines how to clear the WDT by instruction. One clear instruction: The ″CLR WDT″ can clear the WDT. Two clear instructions: Only when both of the ″CLR WDT1″ and ″CLR WDT2″ have been executed, then WDT can be cleared.
WDT	Watchdog enable/disable
WDT Clock Source	WDT clock source selection RC → Select the WDT OSC to be the WDT source. T1 → Select the instruction clock to be the WDT source. 32kHz → Select the external 32768Hz to be the WDT source.

Application Circuits



Instruction Set

Introduction

Central to the successful operation of any microcontroller is its instruction set, which is a set of program instruction codes that directs the microcontroller to perform certain operations. In the case of Holtek microcontrollers, a comprehensive and flexible set of over 60 instructions is provided to enable programmers to implement their application with the minimum of programming overheads.

For easier understanding of the various instruction codes, they have been subdivided into several functional groupings.

Instruction Timing

Most instructions are implemented within one instruction cycle. The exceptions to this are branch, call, or table read instructions where two instruction cycles are required. One instruction cycle is equal to 4 system clock cycles, therefore in the case of an 8MHz system oscillator, most instructions would be implemented within 0.5μs and branch or call instructions would be implemented within 1μs. Although instructions which require one more cycle to implement are generally limited to the JMP, CALL, RET, RETI and table read instructions, it is important to realize that any other instructions which involve manipulation of the Program Counter Low register or PCL will also take one more cycle to implement. As instructions which change the contents of the PCL will imply a direct jump to that new address, one more cycle will be required. Examples of such instructions would be ″CLR PCL″ or ″MOV PCL, A″. For the case of skip instructions, it must be noted that if the result of the comparison involves a skip operation then this will also take one more cycle, if no skip is involved then only one cycle is required.

Moving and Transferring Data

The transfer of data within the microcontroller program is one of the most frequently used operations. Making use of three kinds of MOV instructions, data can be transferred from registers to the Accumulator and vice-versa as well as being able to move specific immediate data directly into the Accumulator. One of the most important data transfer applications is to receive data from the input ports and transfer data to the output ports.

Arithmetic Operations

The ability to perform certain arithmetic operations and data manipulation is a necessary feature of most microcontroller applications. Within the Holtek microcontroller instruction set are a range of add and subtract instruction mnemonics to enable the necessary arithmetic to be carried out. Care must be taken to ensure correct handling of carry and borrow data when results exceed 255 for addition and less than 0 for subtraction. The increment and decrement instructions INC, INCA, DEC and DECA provide a simple means of increasing or decreasing by a value of one of the values in the destination specified.

Logical and Rotate Operations

The standard logical operations such as AND, OR, XOR and CPL all have their own instruction within the Holtek microcontroller instruction set. As with the case of most instructions involving data manipulation, data must pass through the Accumulator which may involve additional programming steps. In all logical data operations, the zero flag may be set if the result of the operation is zero. Another form of logical data manipulation comes from the rotate instructions such as RR, RL, RRC and RLC which provide a simple means of rotating one bit right or left. Different rotate instructions exist depending on program requirements. Rotate instructions are useful for serial port programming applications where data can be rotated from an internal register into the Carry bit from where it can be examined and the necessary serial bit set high or low. Another application where rotate data operations are used is to implement multiplication and division calculations.

Branches and Control Transfer

Program branching takes the form of either jumps to specified locations using the JMP instruction or to a subroutine using the CALL instruction. They differ in the sense that in the case of a subroutine call, the program must return to the instruction immediately when the subroutine has been carried out. This is done by placing a return instruction RET in the subroutine which will cause the program to jump back to the address right after the CALL instruction. In the case of a JMP instruction, the program simply jumps to the desired location. There is no requirement to jump back to the original jumping off point as in the case of the CALL instruction. One special and extremely useful set of branch instructions are the conditional branches. Here a decision is first made regarding the condition of a certain data memory or individual bits. Depending upon the conditions, the program will continue with the next instruction or skip over it and jump to the following instruction. These instructions are the key to decision making and branching within the program perhaps determined by the condition of certain input switches or by the condition of internal data bits.

Bit Operations

The ability to provide single bit operations on Data Memory is an extremely flexible feature of all Holtek microcontrollers. This feature is especially useful for output port bit programming where individual bits or port pins can be directly set high or low using either the "SET [m].i" or "CLR [m].i" instructions respectively. The feature removes the need for programmers to first read the 8-bit output port, manipulate the input data to ensure that other bits are not changed and then output the port with the correct new data. This read-modify-write process is taken care of automatically when these bit operation instructions are used.

Table Read Operations

Data storage is normally implemented by using registers. However, when working with large amounts of fixed data, the volume involved often makes it inconvenient to store the fixed data in the Data Memory. To overcome this problem, Holtek microcontrollers allow an area of Program Memory to be setup as a table where data can be directly stored. A set of easy to use instructions provides the means by which this fixed data can be referenced and retrieved from the Program Memory.

Other Operations

In addition to the above functional instructions, a range of other instructions also exist such as the "HALT" instruction for Power-down operations and instructions to control the operation of the Watchdog Timer for reliable program operations under extreme electric or electromagnetic environments. For their relevant operations, refer to the functional related sections.

Instruction Set Summary

The 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	1Note	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	1Note	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	1Note	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	1Note	Z, C, AC, OV
DAA [m]	Decimal adjust ACC for Addition with result in Data Memory	1Note	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	1Note	Z
ORM A,[m]	Logical OR ACC to Data Memory	1Note	Z
XORM A,[m]	Logical XOR ACC to Data Memory	1Note	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	1Note	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	1Note	Z
DECA [m]	Decrement Data Memory with result in ACC	1	Z
DEC [m]	Decrement Data Memory	1Note	Z

Mnemonic	Description	Cycles	Flag Affected
Rotate			
RRA [m]	Rotate Data Memory right with result in ACC	1	None
RR [m]	Rotate Data Memory right	1Note	None
RRCA [m]	Rotate Data Memory right through Carry with result in ACC	1	C
RRC [m]	Rotate Data Memory right through Carry	1Note	C
RLA [m]	Rotate Data Memory left with result in ACC	1	None
RL [m]	Rotate Data Memory left	1Note	None
RLCA [m]	Rotate Data Memory left through Carry with result in ACC	1	C
RLC [m]	Rotate Data Memory left through Carry	1Note	C
Data Move			
MOV A,[m]	Move Data Memory to ACC	1	None
MOV [m],A	Move ACC to Data Memory	1Note	None
MOV A,x	Move immediate data to ACC	1	None
Bit Operation			
CLR [m].i	Clear bit of Data Memory	1Note	None
SET [m].i	Set bit of Data Memory	1Note	None
Branch			
JMP addr	Jump unconditionally	2	None
SZ [m]	Skip if Data Memory is zero	1Note	None
SZA [m]	Skip if Data Memory is zero with data movement to ACC	1note	None
SZ [m].i	Skip if bit i of Data Memory is zero	1Note	None
SNZ [m].i	Skip if bit i of Data Memory is not zero	1Note	None
SIZ [m]	Skip if increment Data Memory is zero	1Note	None
SDZ [m]	Skip if decrement Data Memory is zero	1Note	None
SIZA [m]	Skip if increment Data Memory is zero with result in ACC	1Note	None
SDZA [m]	Skip if decrement Data Memory is zero with result in ACC	1Note	None
CALL addr	Subroutine call	2	None
RET	Return from subroutine	2	None
RET A,x	Return from subroutine and load immediate data to ACC	2	None
RETI	Return from interrupt	2	None
Table Read			
TABRDC [m]	Read table (current page) to TBLH and Data Memory	2Note	None
TABRDL [m]	Read table (last page) to TBLH and Data Memory	2Note	None
Miscellaneous			
NOP	No operation	1	None
CLR [m]	Clear Data Memory	1Note	None
SET [m]	Set Data Memory	1Note	None
CLR WDT	Clear Watchdog Timer	1	TO, PDF
CLR WDT1	Pre-clear Watchdog Timer	1	TO, PDF
CLR WDT2	Pre-clear Watchdog Timer	1	TO, PDF
SWAP [m]	Swap nibbles of Data Memory	1Note	None
SWAPA [m]	Swap nibbles of Data Memory with result in ACC	1	None
HALT	Enter power down mode	1	TO, PDF

Note: 1. For skip instructions, if the result of the comparison involves a skip then two cycles are required, if no skip takes place only one cycle is required.
2. Any instruction which changes the contents of the PCL will also require 2 cycles for execution.
3. For the ″CLR WDT1″ and ″CLR WDT2″ instructions the TO and PDF flags may be affected by the execution status. The TO and PDF flags are cleared after both ″CLR WDT1″ and ″CLR WDT2″ instructions are consecutively executed. Otherwise the TO and PDF flags remain unchanged.

Instruction Definition

ADC A,[m] Add Data Memory to ACC with Carry

Description The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the Accumulator.

Operation ACC ← ACC + [m] + C

Affected flag(s) OV, Z, AC, C

ADCM A,[m] Add ACC to Data Memory with Carry

Description The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the specified Data Memory.

Operation [m] ← ACC + [m] + C

Affected flag(s) OV, Z, AC, C

ADD A,[m] Add Data Memory to ACC

Description The contents of the specified Data Memory and the Accumulator are added. The result is stored in the Accumulator.

Operation ACC ← ACC + [m]

Affected flag(s) OV, Z, AC, C

ADD A,x Add immediate data to ACC

Description The contents of the Accumulator and the specified immediate data are added. The result is stored in the Accumulator.

Operation ACC ← ACC + x

Affected flag(s) OV, Z, AC, C

ADDM A,[m] Add ACC to Data Memory

Description The contents of the specified Data Memory and the Accumulator are added. The result is stored in the specified Data Memory.

Operation [m] ← ACC + [m]

Affected flag(s) OV, Z, AC, C

AND A,[m] Logical AND Data Memory to ACC

Description Data in the Accumulator and the specified Data Memory perform a bitwise logical AND operation. The result is stored in the Accumulator.

Operation ACC ← ACC ″AND″ [m]

Affected flag(s) Z

AND A,x Logical AND immediate data to ACC

Description Data in the Accumulator and the specified immediate data perform a bitwise logical AND operation. The result is stored in the Accumulator.

Operation ACC ← ACC ″AND″ x

Affected flag(s) Z

ANDM A,[m] Logical AND ACC to Data Memory

Description Data in the specified Data Memory and the Accumulator perform a bitwise logical AND operation. The result is stored in the Data Memory.

Operation [m] ← ACC ″AND″ [m]

Affected flag(s) Z

CALL addr	Subroutine call
Description	Unconditionally calls a subroutine at the specified address. The Program Counter then increments by 1 to obtain the address of the next instruction which is then pushed onto the stack. The specified address is then loaded and the program continues execution from this new address. As this instruction requires an additional operation, it is a two cycle instruction.
Operation	Stack ← Program Counter + 1 Program Counter ← addr
Affected flag(s)	None
CLR [m]	Clear Data Memory
Description	Each bit of the specified Data Memory is cleared to 0.
Operation	[m] ← 00H
Affected flag(s)	None
CLR [m].i	Clear bit of Data Memory
Description	Bit i of the specified Data Memory is cleared to 0.
Operation	[m].i ← 0
Affected flag(s)	None
CLR WDT	Clear Watchdog Timer
Description	The TO, PDF flags and the WDT are all cleared.
Operation	WDT cleared TO ← 0 PDF ← 0
Affected flag(s)	TO, PDF
CLR WDT1	Pre-clear Watchdog Timer
Description	The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT2 and must be executed alternately with CLR WDT2 to have effect. Repetitively executing this instruction without alternately executing CLR WDT2 will have no effect.
Operation	WDT cleared TO ← 0 PDF ← 0
Affected flag(s)	TO, PDF
CLR WDT2	Pre-clear Watchdog Timer
Description	The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT1 and must be executed alternately with CLR WDT1 to have effect. Repetitively executing this instruction without alternately executing CLR WDT1 will have no effect.
Operation	WDT cleared TO ← 0 PDF ← 0
Affected flag(s)	TO, PDF

CPL [m] Complement Data Memory

Description Each bit of the specified Data Memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice versa.

Operation $[m] \leftarrow \overline{[m]}$

Affected flag(s) Z

CPLA [m] Complement Data Memory with result in ACC

Description Each bit of the specified Data Memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice versa. The complemented result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation $ACC \leftarrow \overline{[m]}$

Affected flag(s) Z

DAA [m] Decimal-Adjust ACC for addition with result in Data Memory

Description Convert the contents of the Accumulator value to a BCD (Binary Coded Decimal) value resulting from the previous addition of two BCD variables. If the low nibble is greater than 9 or if AC flag is set, then a value of 6 will be added to the low nibble. Otherwise the low nibble remains unchanged. If the high nibble is greater than 9 or if the C flag is set, then a value of 6 will be added to the high nibble. Essentially, the decimal conversion is performed by adding 00H, 06H, 60H or 66H depending on the Accumulator and flag conditions. Only the C flag may be affected by this instruction which indicates that if the original BCD sum is greater than 100, it allows multiple precision decimal addition.

Operation $[m] \leftarrow ACC + 00H$ or
$[m] \leftarrow ACC + 06H$ or
$[m] \leftarrow ACC + 60H$ or
$[m] \leftarrow ACC + 66H$

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] ← [m] + 1
Affected flag(s)	Z
INCA [m]	Increment Data Memory with result in ACC
Description	Data in the specified Data Memory is incremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.
Operation	ACC ← [m] + 1
Affected flag(s)	Z
JMP addr	Jump unconditionally
Description	The contents of the Program Counter are replaced with the specified address. Program execution then continues from this new address. As this requires the insertion of a dummy instruction while the new address is loaded, it is a two cycle instruction.
Operation	Program Counter ← addr
Affected flag(s)	None
MOV A,[m]	Move Data Memory to ACC
Description	The contents of the specified Data Memory are copied to the Accumulator.
Operation	ACC ← [m]
Affected flag(s)	None
MOV A,x	Move immediate data to ACC
Description	The immediate data specified is loaded into the Accumulator.
Operation	ACC ← x
Affected flag(s)	None
MOV [m],A	Move ACC to Data Memory
Description	The contents of the Accumulator are copied to the specified Data Memory.
Operation	[m] ← ACC
Affected flag(s)	None
NOP	No operation
Description	No operation is performed. Execution continues with the next instruction.
Operation	No operation
Affected flag(s)	None
OR A,[m]	Logical OR Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical OR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″OR″ [m]
Affected flag(s)	Z

OR A,x	Logical OR immediate data to ACC
Description	Data in the Accumulator and the specified immediate data perform a bitwise logical OR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″OR″ x
Affected flag(s)	Z
ORM A,[m]	Logical OR ACC to Data Memory
Description	Data in the specified Data Memory and the Accumulator perform a bitwise logical OR operation. The result is stored in the Data Memory.
Operation	[m] ← ACC ″OR″ [m]
Affected flag(s)	Z
RET	Return from subroutine
Description	The Program Counter is restored from the stack. Program execution continues at the restored address.
Operation	Program Counter ← Stack
Affected flag(s)	None
RET A,x	Return from subroutine and load immediate data to ACC
Description	The Program Counter is restored from the stack and the Accumulator loaded with the specified immediate data. Program execution continues at the restored address.
Operation	Program Counter ← Stack ACC ← x
Affected flag(s)	None
RETI	Return from interrupt
Description	The Program Counter is restored from the stack and the interrupts are re-enabled by setting the EMI bit. EMI is the master interrupt global enable bit. If an interrupt was pending when the RETI instruction is executed, the pending Interrupt routine will be processed before returning to the main program.
Operation	Program Counter ← Stack EMI ← 1
Affected flag(s)	None
RL [m]	Rotate Data Memory left
Description	The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0.
Operation	[m].(i+1) ← [m].i; (i = 0~6) [m].0 ← [m].7
Affected flag(s)	None
RLA [m]	Rotate Data Memory left with result in ACC
Description	The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.
Operation	ACC.(i+1) ← [m].i; (i = 0~6) ACC.0 ← [m].7
Affected flag(s)	None

RLC [m] Rotate Data Memory left through Carry

Description The contents of the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into bit 0.

Operation
[m].(i+1) ← [m].i; (i = 0~6)
[m].0 ← C
C ← [m].7

Affected flag(s) C

RLCA [m] Rotate Data Memory left through Carry with result in ACC

Description Data in the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into the bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

Operation
ACC.(i+1) ← [m].i; (i = 0~6)
ACC.0 ← C
C ← [m].7

Affected flag(s) C

RR [m] Rotate Data Memory right

Description The contents of the specified Data Memory are rotated right by 1 bit with bit 0 rotated into bit 7.

Operation
[m].i ← [m].(i+1); (i = 0~6)
[m].7 ← [m].0

Affected flag(s) None

RRA [m] Rotate Data Memory right with result in ACC

Description Data in the specified Data Memory and the carry flag are 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 $[m] \leftarrow [m] - 1$
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 $ACC \leftarrow [m] - 1$
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 $[m] \leftarrow FFH$

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 $[m].i \leftarrow 1$

Affected flag(s) None

SIZ [m] Skip if increment Data Memory is 0

Description The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation [m] ← [m] + 1
Skip if [m] = 0

Affected flag(s) None

SIZA [m] Skip if increment Data Memory is zero with result in ACC

Description The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation ACC ← [m] + 1
Skip if ACC = 0

Affected flag(s) None

SNZ [m].i Skip if bit i of Data Memory is not 0

Description If bit i of the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction.

Operation Skip if [m].i $\neq$ 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 ← ACC – [m]

Affected flag(s) OV, Z, AC, C

SUBM A,[m] Subtract Data Memory from ACC with result in Data Memory

Description The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation [m] ← ACC – [m]

Affected flag(s) OV, Z, AC, C

SUB A,x Subtract immediate data from ACC

Description The immediate data specified by the code is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation ACC ← ACC – x

Affected flag(s) OV, Z, AC, C

SWAP [m] Swap nibbles of Data Memory

Description The low-order and high-order nibbles of the specified Data Memory are interchanged.

Operation [m].3~[m].0 ↔ [m].7 ~ [m].4

Affected flag(s) None

SWAPA [m] Swap nibbles of Data Memory with result in ACC

Description The low-order and high-order nibbles of the specified Data Memory are interchanged. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.

Operation ACC.3 ~ ACC.0 ← [m].7 ~ [m].4
ACC.7 ~ ACC.4 ← [m].3 ~ [m].0

Affected flag(s) None

SZ [m] Skip if Data Memory is 0

Description If the contents of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation Skip if [m] = 0

Affected flag(s) None

SZA [m] Skip if Data Memory is 0 with data movement to ACC

Description The contents of the specified Data Memory are copied to the Accumulator. If the value is zero, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation ACC ← [m]
Skip if [m] = 0

Affected flag(s) None

SZ [m].i Skip if bit i of Data Memory is 0

Description If bit i of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction.

Operation Skip if [m].i = 0

Affected flag(s) None

TABRDC [m] Read table (current page) to TBLH and Data Memory

Description The low byte of the program code (current page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.

Operation [m] ← program code (low byte)
TBLH ← program code (high byte)

Affected flag(s) None

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

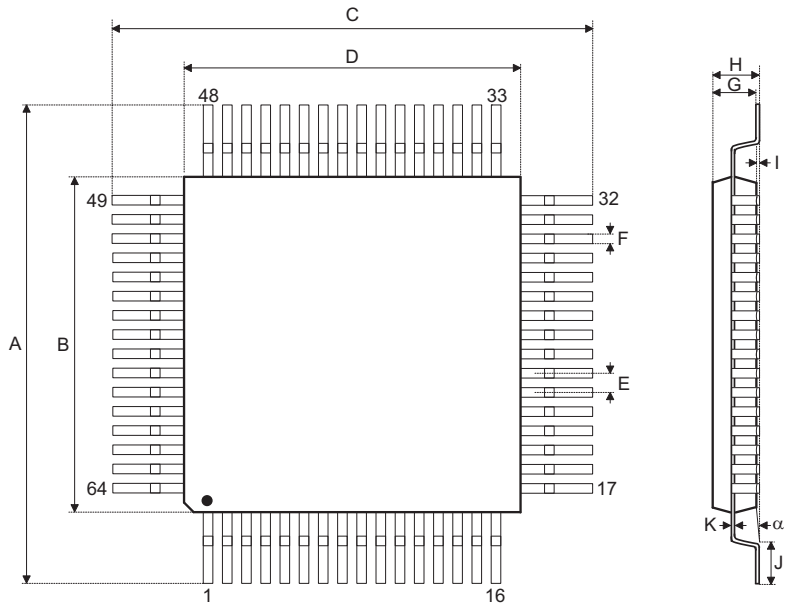
XOR A,[m]	Logical XOR Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical XOR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″XOR″ [m]
Affected flag(s)	Z

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

64-pin LQFP (7mm×7mm) Outline Dimensions



Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	8.9	—	9.1
B	6.9	—	7.1
C	8.9	—	9.1
D	6.9	—	7.1
E	—	0.4	—
F	0.13	—	0.23
G	1.35	—	1.45
H	—	—	1.6
I	0.05	—	0.15
J	0.45	—	0.75
K	0.09	—	0.20
α	0°	—	7°

Holtek Semiconductor Inc. (Headquarters)
No.3, Creation Rd. II, Science Park, Hsinchu, Taiwan
Tel: 886-3-563-1999
Fax: 886-3-563-1189
http://www.holtek.com.tw

Holtek Semiconductor Inc. (Taipei Sales Office)
4F-2, No. 3-2, YuanQu St., Nankang Software Park, Taipei 115, Taiwan
Tel: 886-2-2655-7070
Fax: 886-2-2655-7373
Fax: 886-2-2655-7383 (International sales hotline)

Holtek Semiconductor (China) Inc. (Dongguan Sales Office)
Building No. 10, Xinzhu Court, (No. 1 Headquarters), 4 Cuizhu Road, Songshan Lake, Dongguan, China 523808
Tel: 86-769-2626-1300
Fax: 86-769-2626-1311

Holtek Semiconductor (USA), Inc. (North America Sales Office)
46729 Fremont Blvd., Fremont, CA 94538
Tel: 1-510-252-9880
Fax: 1-510-252-9885
http://www.holtek.com