Features

- Operating voltage:
  f_{SYS}= 6MHz: 1.8V~3.3V
- Internal 6MHz RC oscillator for f_{SYS}
- Power down and wake-up functions to reduce power consumption
- Two bit to define microcontroller system clock (f_{SYS}/1, f_{SYS}/2, f_{SYS}/4)
- All instructions executed in one or two machine cycles
- Table read instructions
- 63 powerful instructions
- 6-level subroutine nesting
- Bit manipulation instruction
- Program Memory: 4K×15
- Data Memory: 128×8~160×8
- Watchdog Timer function
- Up to 40 bidirectional I/O lines with pull-high options
- All I/O pins have falling and rising edge wake-up function
- Single 16-bit internal timer with overflow interrupt and timer input
- SPI interface shared with I/O lines
- Low voltage reset function (LVR) for DC_DC output controlled by configuration option
- Built-in DC/DC to provide stable 2.8V, 3.0V, 3.3V with error ±5% selected by configuration options
- Low voltage detector (LVD) with levels 1.8V/2.0V/2.2V/2.5V/2.8V ±5% for battery input (BAT_IN) selected by application program
- Wide range of available package types
- Optional Peripheral -- EEPROM Memory with 128×8 capacity

General Description

The device is an 8-bit high performance, RISC architecture microcontroller devices specifically designed for multiple I/O, mouse/keyboard appliances and SPI control product applications. The advantages of low power consumption, I/O flexibility, Timer functions, Watchdog timer, Power Down, wake-up functions together with the optional peripherals such as EEPROM Memory and RF transceiver provide the devices with versatility for industrial control, consumer products, subsystem controllers, RF module control, etc.

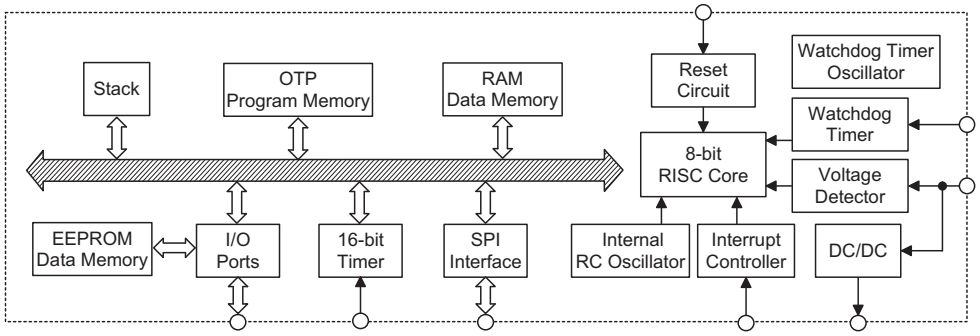
Selection Table

Part No.	Program Memory	Data Memory	Data EEPROM	I/O	Timer	DC/DC Converter	SPI Interface	RF Transceiver	Built-in OSC	Stack	Package
HT82M75R	4K×15	128×8	—	24	16-bit×1	√	√	—	√	6	20/28SSOP 32QFN
HT82K75R	4K×15	160×8	—	40	16-bit×1	√	√	—	√	6	48SSOP
HT82M75RE (1)	4K×15	128×8	128×8	22	16-bit×1	√	√	—	√	6	32QFN
HT82K75RE (1)	4K×15	160×8	128×8	38	16-bit×1	√	√	—	√	6	48SSOP

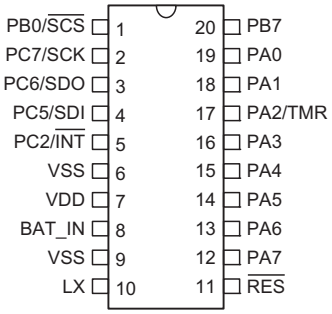
Note: (1) There is an additional peripheral known as the Data EEPROM with capacity of 128 bytes in HT82M75RE and HT82K75RE devices. All information related to the Data EEPROM will be described in the following EEPROM Data Memory section.

(2) As devices exist in more than one package format, the table reflects the situation for the package with the most pins.

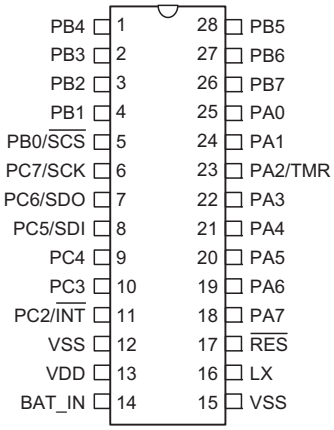
Block Diagram



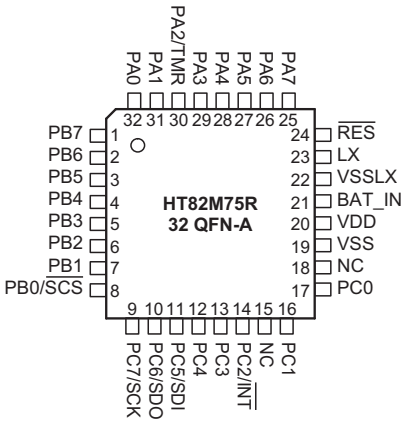
Pin Assignment

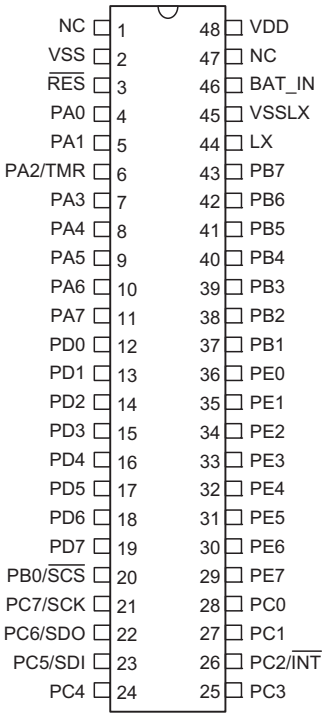


HT82M75R
20 SSOP-A



HT82M75R
28 SSOP-A





HT82K75R
48 SSOP-A

Pin Description

The following table only includes the pins which are directly related to the MCU. The pin descriptions of the additional peripheral functions are located at the corresponding section of the datasheet along with the relevant peripheral function functional description.

Pin Name	I/O	Options	Description
PA0~PA1 PA2/TMR PA3~PA7	I/O	Pull-high Wake-up	Bidirectional 8-bit input/output port. Each pin can be configured as a wake-up input (both falling and rising edge) by a configuration option. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if the pins have pull-high resistors. PA2 is shared with the external timer input pin TMR.
PB0/$\overline{SCS}$ PB1~PB7	I/O	Pull-high or Wake-up CMOS/NMOS	Bidirectional 8-bit input/output port. Each pin can be configured as a wake-up input (both falling and rising edge) by a configuration option. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if the pins have pull-high resistors. Also a configuration option determines if the PB0 is a CMOS output type or NMOS output type. PB0 is shared with $\overline{SCS}$ of the SPI interface.
PC0~PC1 PC2/$\overline{INT}$ PC3~PC4 PC5/SDI PC6/SDO PC7/SCK	I/O	Pull-high or Wake-up CMOS/NMOS	Bidirectional 8-bit input/output port. Each pin can be configured as a wake-up input (both falling and rising edge) by a configuration option. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if the pins have pull-high resistors. Also a configuration option determines if the PC pins are CMOS output type or NMOS output type. The $\overline{INT}$ is shared with PC2, PC5~PC7 are shared with the SPI interface.
PD0~PD7 (HT82K75R only)	I/O	Pull-high or Wake-up	Bidirectional 8-bit input/output port. Each nibble can be configured as wake-up inputs (both falling and rising edge) by a configuration option. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if the pins have pull-high resistors.
PE0~PE7 (HT82K75R only)	I/O	Pull-high or Wake-up	Bidirectional 8-bit input/output port. Each nibble can be configured as wake-up inputs (both falling and rising edge) by a configuration option. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if the pins have pull-high resistors.
VSS	—	—	Negative power supply, ground
$\overline{RES}$	I	—	Schmitt Trigger reset input. Active low
VDD	—	—	Positive power supply
BAT_IN	I	—	Battery input
LX	I	—	DC/DC LX switch
VSSLX	I	—	DC/DC ground

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				
V_{BAT}	BAT_IN Operating Voltage	—	—	1.8	2.2	3.3	V
I_{DD}	Operating Current (Crystal OSC)	3V	No load, f_{SYS}= 6MHz	—	3	6	mA
I_{STB}	Standby Current	—	No load, system HALT WDT disable, LVR disable	—	—	20	μA
V_{IL1}	Input Low Voltage for I/O (Schmitt Trigger)	—	—	0	—	$0.3V_{DD}$	V
V_{IH1}	Input High Voltage for I/O (Schmitt Trigger)	—	—	$0.7V_{DD}$	—	V_{DD}	V
V_{IL2}	Input Low Voltage ($\overline{RES}$)	—	—	0	—	$0.3V_{DD}$	V
V_{IH2}	Input High Voltage ($\overline{RES}$)	—	—	$0.9V_{DD}$	—	V_{DD}	V
I_{OL1}	Other I/O Pins Sink Current	3V	$V_{OL}=0.1V_{DD}$	4	—	—	mA
I_{OH1}	Other I/O Pins Source Current	3V	$V_{OH}=0.9V_{DD}$	–2.5	–4.5	—	mA
R_{PH1}	Other Pins Internal Pull-high Resistance	3V	—	10	30	50	kΩ

A.C. Characteristics

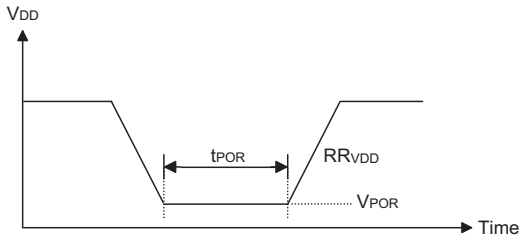
Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
f_{SYS}	System Clock	3V	—	5.7	6	6.3	MHz
t_{RCSYS}	Watchdog OSC Period	3V	—	—	71	—	μs
t_{WDT1}	Watchdog Time-out Period with 6-stage Prescaler	3V	WDTS=1	—	4.57	—	ms
t_{RES}	External Reset Low Pulse Width	—	—	1	—	—	ms
t_{SST}	System Start-up Timer	—	—	—	512	—	$1/f_{SYS}$
t_{LVR}	Low Voltage Width to Reset	—	—	0.25	1	2	ms
$t_{Wake-up}$	MCU Wake-up Timer	—	—	—	—	1	ms
$t_{configure}$	Watchdog Time-out Period	—	—	—	1024	—	t_{RCSYS}

Power-On Reset Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
I_{POR}	Operating current	1.8V~3.3V	—	—	1.0	—	μA
RR_{VDD}	V_{DD} Rise Rate to Ensure Power-on Reset	—	Without 0.1μF between V_{DD} and V_{SS}	0.05	—	—	V/ms
V_{POR}	Maximum V_{DD} Start Voltage to Ensure Power-on Reset	—	Without 0.1μF between V_{DD} and V_{SS} ,Ta=25°C	0.9	—	1.5	V
t_{POR}	Power-on Reset Low Pulse Width	—	Without 0.1μF between V_{DD} and V_{SS}	2	—	—	μs
			With 0.1μF between V_{DD} and V_{SS}	10	—	—	μs



System Architecture

A key factor in the high-performance features of the Holtek range of microcontrollers is attributed to the internal system architecture. The 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.

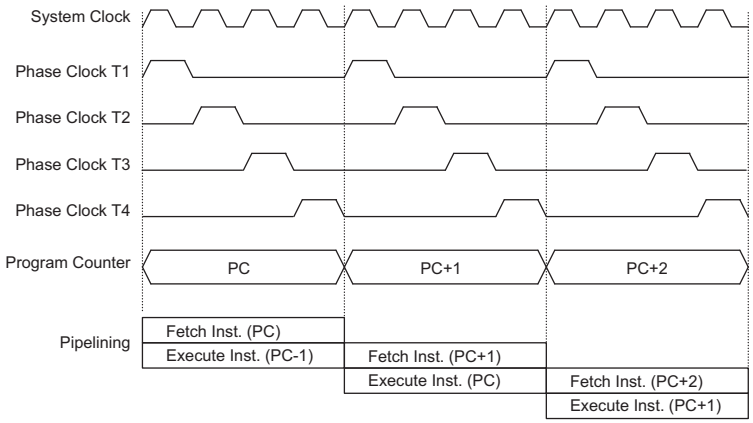
Clocking and Pipelining

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

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

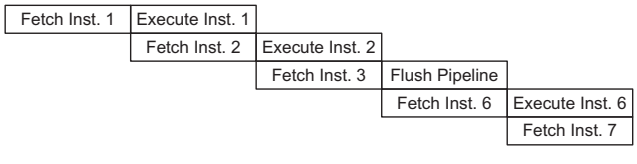
Program Counter

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. It must be noted that only the lower 8 bits, known as the Program Counter Low Register, are directly addressable by user.



System Clocking and Pipelining

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



Instruction Fetching

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.

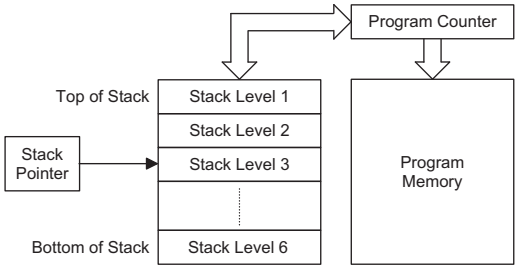
The lower byte of the Program Counter, known as the Program Counter Low register or PCL, is available for program control and is a readable and writeable register. By transferring data directly into this register, a short program jump can be executed directly, however, as only this low byte is available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

The lower byte of the Program Counter is fully accessible under program control. Manipulating the PCL might cause program branching, so an extra cycle is needed to pre-fetch. Further information on the PCL register can be found in the Special Function Register section.

Stack

This is a special part of the memory which is used to save the contents of the Program Counter only. The stack has 6 levels and is neither part of the data nor part of the program space, and is neither readable nor writeable. The activated level is indexed by the Stack Pointer, SP, and is neither readable nor writeable. At a subroutine call or interrupt acknowledge signal, the contents of the Program Counter are pushed onto the stack. At the end of a subroutine or an interrupt routine, signaled by a return instruction, RET or RETI, the Program Counter is restored to its previous value from the stack. After a device reset, the Stack Pointer will point to the top of the stack.

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



Mode	Program Counter Bits											
	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
SPI Interrupt	0	0	0	0	0	0	0	0	0	1	0	0
Timer/Event Counter Overflow	0	0	0	0	0	0	0	0	1	0	0	0
External interrupt	0	0	0	0	0	0	0	0	1	1	0	0
Skip	Program Counter + 2											
Loading PCL	PC11	PC10	PC9	PC8	@7	@6	@5	@4	@3	@2	@1	@0
Jump, Call Branch	#11	#10	#9	#8	#7	#6	#5	#4	#3	#2	#1	#0
Return from Subroutine	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0

Program Counter

Note: PC11~PC8: Current Program Counter bits
#11~#0: Instruction code address bits
@7~@0: PCL bits
S11~S0: Stack register bits

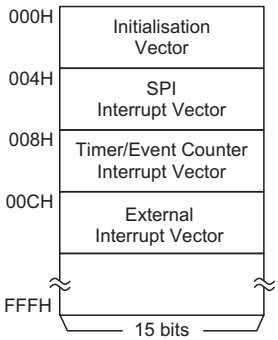
Arithmetic and Logic Unit – ALU

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. The device is supplied with One-Time Programmable, OTP, memory where users can program their application code into the device. By using the appropriate programming tools, OTP devices offer users the flexibility to freely develop their applications which may be useful during debug or for products requiring frequent upgrades or program changes. OTP devices are also applicable for use in applications that require low or medium volume production runs.



Program Memory Structure

Structure

The Program Memory has a capacity of 4K×15 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 separate table pointer registers.

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 serial interface. When 8-bits of data have been received or transmitted success-fully from serial interface. The program will jump to this location and begin execution if the interrupt is enable and the stack is not full.
- Location 008H
  This vector is used by the timer/event counter. If a counter overflow occurs, the program will jump to this location and begin execution if the timer interrupt is enabled and the stack is not full.
- Location 00CH
  This vector is used by the external interrupt. If the $\overline{INT}$ external input pin on the device receives a high to low transition, the program will jump to this location and begin execution, if the interrupt is enabled and the stack is not full.
- Table location
  Any location in the program memory can be used as look-up tables. There are three method to read the ROM data by two table read instructions: "TABRDC" and "TABRDL", transfer the contents of the lower-order byte to the specified data memory, and the higher-order byte to TBLH (08H).

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

Table Location

Note: PC11~PC8: Current program counter bits when TBHP is disabled
TBHP register bit3~bit0 when TBHP is enabled
@7~@0: Table Pointer TBLP bits

- The three methods are shown as follows: The instructions ″TABRDC [m]″ (the current page, one page=256words), where the table location is defined by TBLP (07H) in the current page. And the configuration option TBHP is disabled (default).
- The instructions ″TABRDC [m]″, where the table location is defined by registers TBLP (07H) and TBHP (01FH). And the configuration option TBHP is enabled.
- The instructions ″TABRDL [m]″, where the table location is defined by register TBLP (07H) in the last page (F00H~FFFH).

Only the destination of the lower-order byte in the table is well-defined, the other bits of the table word are transferred to the lower portion of TBLH, and the remaining 1-bit words are read as ″0″. The Table Higher-order byte register (TBLH) is read only. The table pointer (TBLP, TBHP) is a read/write register (07H, 1FH), which indicates the table location. Before accessing the table, the location must be placed in the TBLP and TBHP (If the configuration option TBHP is disabled, the value in TBHP has no effect). The TBLH is read only and cannot be restored. If the main routine and the ISR (Interrupt Service Routine) both employ the table read instruction, the contents of the TBLH in the
main routine are likely to be changed by the table read instruction used in the ISR. Errors can occur. In other words, using the table read instruction in the main routine and the ISR simultaneously should be avoided.
However, if the table read instruction has to be applied in both the main routine and the ISR, the interrupt should be disabled prior to the table read instruction. It will not be enabled until the TBLH has been backed up. All table related instructions require two cycles to complete the operation. These areas may function as normal program memory depending on the requirements.

Once TBHP is enabled, the instruction ″TABRDC [m]″ reads the ROM data as defined by TBLP and TBHP value. Otherwise, the configuration option TBHP is disabled, the instruction ″TABRDC [m]″ reads the ROM data as defined by TBLP and the current program counter bits.

Table Program Example

The following example shows how the table pointer and table data is defined and retrieved from the microcontroller. This example uses raw table data located in the last page which is stored there using the ORG statement. The value at this ORG statement is ″F00H″ which refers to the start address of the last page within the 4K Program Memory of device. The table pointer is setup here to have an initial value of ″06H″. This will ensure that the first data read from the data table will be at the Program Memory address ″F06H″ 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.

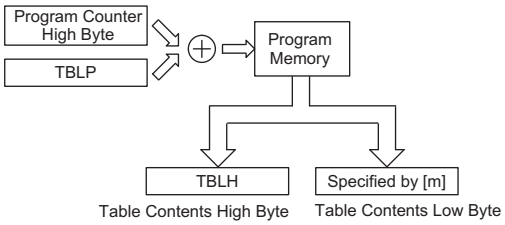


Table Read – TBLP only

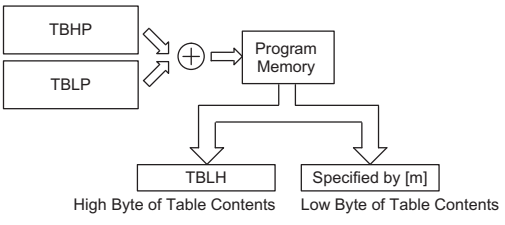


Table Read – TBLP/TBHP

```
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 tempregl
                            ; data at prog. memory address "F06H" 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 "F05H" transferred to
                            ; tempreg2 and TBLH
                            ; in this example the data "1AH" is transferred to
                            ; tempreg1 and data "0FH" to register tempreg2
                            ; the value "00H" will be transferred to the high byte
                            ; register TBLH
            :
            :

org         F00h            ; sets initial address of last page

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

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 the table read instructions. If using the table read instructions, the Interrupt Service Routines may change the value of 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.

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 Data Memory is reserved for general purpose use. All locations within this area are read and write accessible under program control.

Structure

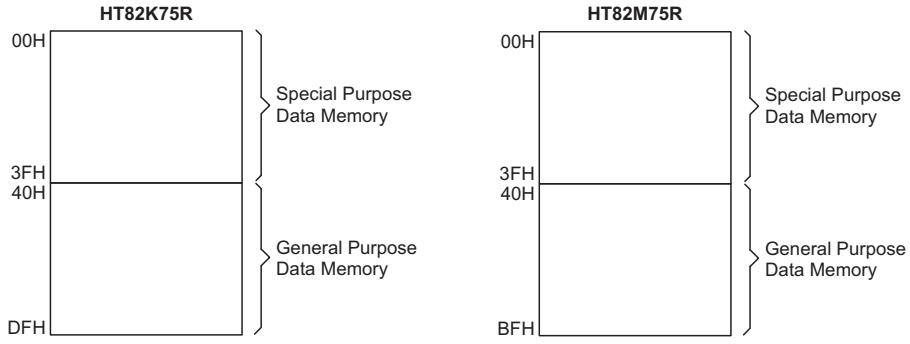
The two sections of Data Memory, the Special Purpose and General Purpose Data Memory are located at consecutive locations. All are implemented in RAM and are 8-bit wide. The start address of the Data Memory for all devices is the address ″00H″. Registers which are common to all microcontrollers, such as ACC, PCL, etc., have the same Data Memory address.

General Purpose Data Memory

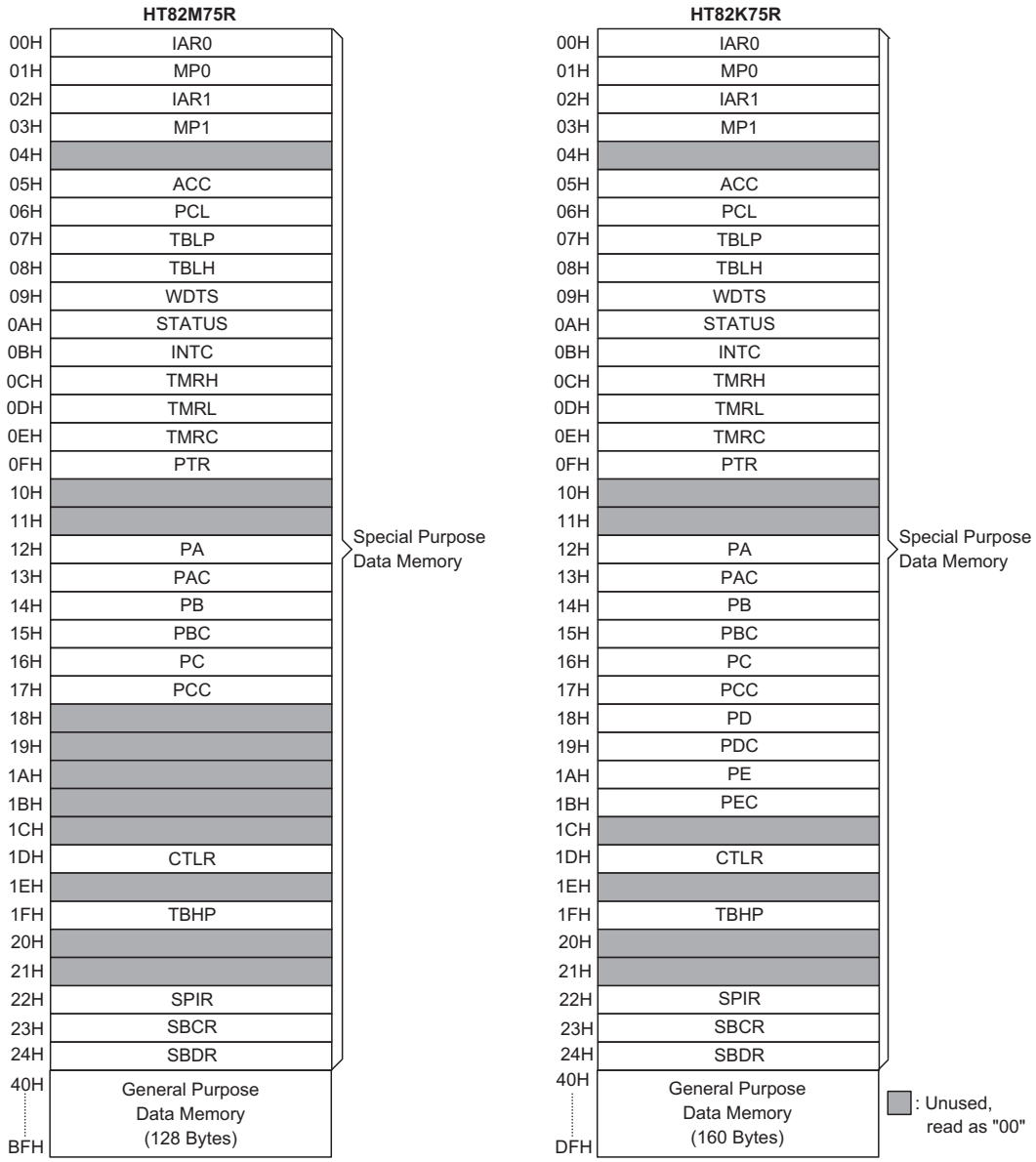
All microcontroller programs require an area of read/write memory where temporary data can be stored and retrieved for use later. It is this area of RAM memory that is known as General Purpose Data Memory. This area of Data Memory is fully accessible by the user program for both read and write operations. 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.

Special Purpose Data Memory

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



Data Memory Structure



Special Purpose Data Memory Structure

Special Function Registers

To ensure successful operation of the microcontroller, certain internal registers are implemented in the Data Memory area. These registers ensure correct operation of internal functions such as timers, interrupts, etc., as well as external functions such as I/O data control. The location of these registers within the 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 and 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 all of the data banks if the Data Memory is divided into 2 or more banks. As 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 can only access data in Bank 0 while MP1 can access all data banks if the Data Memory is divided into 2 or more banks.

```
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 mp,a           ; setup memory pointer with first RAM address

loop:
            clr IAR0               ; clear the data at address defined by MP0
            inc mp0            ; increment memory pointer
            sdz block          ; check if last memory location has been cleared
            jmp loop

continue:
```

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

Accumulator – ACC

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

Program Counter Low Register – PCL

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

Look-up Table Registers – TBLP, TBLH, TBHP

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. Once TBHP is enabled, the instruction ″TABRDC [m]″ reads the ROM data as defined by TBLP and TBHP value.

Otherwise, the configuration option TBHP is disabled, the instruction ″TABRDC [m]″ reads the ROM data as defined by TBLP and the current program counter bits.

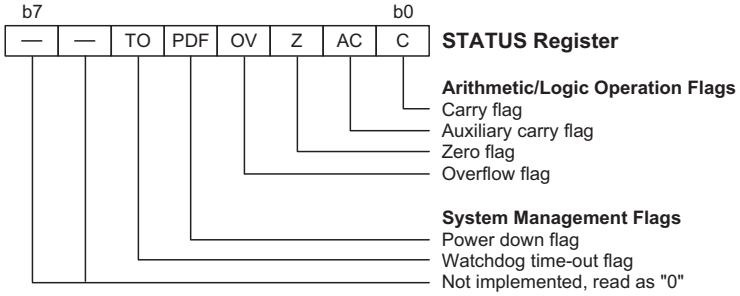
Status Register – STATUS

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

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

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

- **C** is set if an operation results in a carry during an addition operation or if a borrow does not take place during a subtraction operation; otherwise C is cleared. C is also affected by a rotate through carry instruction.
- **AC** is set if an operation results in a carry out of the low nibbles in addition, or no borrow from the high nibble into the low nibble in subtraction; otherwise AC is cleared.
- **Z** is set if the result of an arithmetic or logical operation is zero; otherwise Z is cleared.
- **OV** is set if an operation results in a carry into the highest-order bit but not a carry out of the highest-order bit, or vice versa; otherwise OV is cleared.
- **PDF** is cleared by a system power-up or executing the ″CLR WDT″ instruction. PDF is set by executing the ″HALT″ instruction.
- **TO** is cleared by a system power-up or executing the ″CLR WDT″ or ″HALT″ instruction. TO is set by a WDT time-out.



Status Register

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 interrupt routine can change the status register, precautions must be taken to correctly save it.

Interrupt Control Registers – INTC

The microcontroller provides an internal timer/event counter overflow interrupt. By setting various bits within this register using standard bit manipulation instructions, the enable/disable function of each interrupt 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 – TMRH, TMRL, TMRC

All devices possess a single internal 16-bit count-up timer. An associated register pair known as TMRL/TMRH is the location where the timer 16-bit value is located. This register can also be preloaded with fixed data to allow different time intervals to be setup. An associated control register, known as TMRC, contains the setup information for this timer, which determines in what mode the timer is to be used as well as containing the timer on/off control function.

Watchdog Timer Register – WDTS

The Watchdog function in 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.

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 correspondingly designated registers known as PA, PB, etc. 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 known as PAC, PBC, etc., also mapped to specific addresses with the Data Memory.

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 option for all I/O pins, the user is provided with an I/O structure to meet the needs of a wide range of application possibilities.

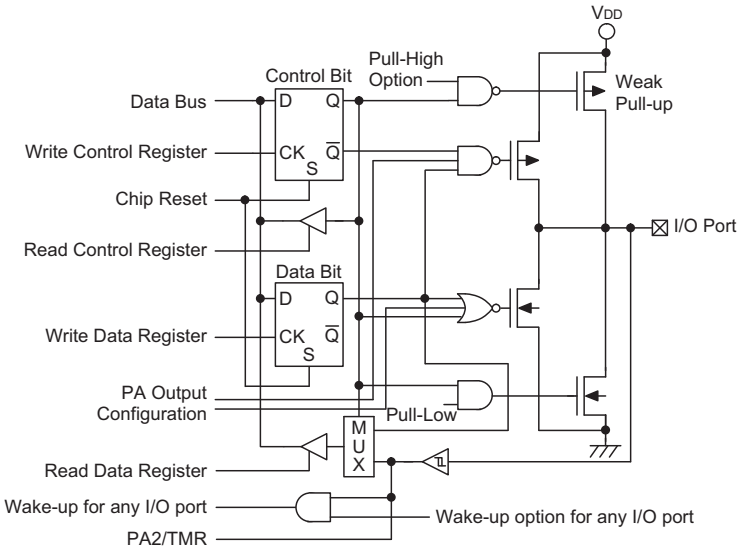
The microcontroller provides 24 or 40 bit bidirectional input/output lines labeled with port names known as PA, PB, etc. These I/O ports are mapped to the Data Memory with addresses as shown in the Special Purpose Data Memory table. All of these I/O lines can be used for input and output operations and one line as an input only. 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.

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, I/O pins, when configured as an input have the capability of being connected to an internal pull-high resistor. The pull-high resistors are selectable via configuration options and are implemented using weak PMOS transistors. The individual pull-high resistor is selected to be connected to each pin on Port A by a configuration option. A configuration option can determine if the pull-high resistors are connected to the lower significant four pins or higher significant four pins on each I/O port except Port A.

Port Pin Wake-up

If the HALT instruction is executed, the device will enter the Power Down Mode, where the system clock will stop resulting in power being conserved, 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 pins from high to low. After a HALT instruction forces the microcontroller into entering the Power Down Mode, the processor will remain in a low-power state until the logic condition of the selected wake-up pin on the port pin changes from high to low. This function is especially suitable for applications that can be woken up via external switches. All of the I/O pins can be configured to have the capability to wake-up the device by high to low and low to high edges using different configuring ways. It means once the I/O pin is configured to have the



Input/Output Ports

wake-up capability, the device can be woken up by any I/O transition. For more details, refer to the Configuration Option Section later.

I/O Port Control Registers

Each I/O port has its own control register known as PAC, PBC, etc., to control the input/output configuration. With this control register, each CMOS/NMOS output or input with or without pull-high resistor structures can be reconfigured dynamically under software control. Each of the I/O ports is directly mapped to a bit in its associated port control register. PC and PB can be set CMOS or NMOS output for option.

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/NMOS 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.

Pin-shared Functions

The flexibility of the microcontroller range is greatly enhanced by the use of pins that have more than one function. Limited numbers of pins can force serious design constraints on designers but by supplying pins with multi-functions, many of these difficulties can be overcome. For some pins, the chosen function of the multi-function I/O pins is set by configuration options while for others the function is set by application program control.

- External Timer Clock Input
  The external timer pin TMR is pin-shared with the I/O pin PA2. To configure this pin to operate as timer input, the corresponding control bits in the timer control register must be correctly set. For applications that do not require an external timer input, this pin can be used as a normal I/O pin. Note that if used as a normal I/O pin the timer mode control bits in the timer control register must select the timer mode, which has an internal clock source, to prevent the input pin from interfering with the timer operation.

- External Interrupt Input
  The external interrupt pin $\overline{INT}$ is pin-shared with the I/O pin PC2. For applications not requiring an external interrupt input, the pin-shared external interrupt pin can be used as a normal I/O pin, however to do this, the external interrupt enable bits in the INTC register must be disabled.

I/O Pin Structures

The 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

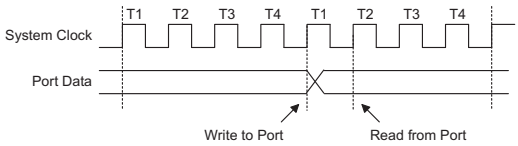
Within the user program, one of the first things to consider is port initialisation. After a reset, all of the data and port control register will be set high. This means that all I/O pins will default to an input state, the level of which depends on the other connected circuitry and whether pull-high options have been selected. If the control registers, known as PAC, PBC, etc., are programmed to setup some pins as outputs, these output pins will have

an initial high output value unless the associated data registers are first programmed. Selecting which pins are inputs and which are outputs can be achieved byte-wide by loading the correct value into the 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.

All I/O have 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 and low to high transition of any of the selected wake-up pins.

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 an internal 16-bit count-up timer which has three operating modes. The timer can be configured to operate as a general timer, external event counter or as a pulse width measurement device.



Read/Write Timing

There are three registers related to the Timer/Event Counter, TMRL, TMRH and TMRC. The TMRL/TMRH register pair are the registers that contains the actual timing value. Writing to this register pair places an initial starting value in the Timer/Event Counter preload register while reading retrieves the contents of the Timer/Event Counter. The TMRC register is a Timer/Event Counter control register, which defines the timer options, and determines how the timer is to be used. The timer clock source can be configured to come from the internal system clock divided by 4 or from an external clock on shared pin PA2/TMR.

Configuring the Timer/Event Counter Input Clock Source

The timer clock source can originate from either the system clock divided by 4 or from an external clock source. The system clock divided by 4 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 shared pin PA2/TMR. Depending upon the condition of the TE bit, each high to low, or low to high transition on the PA2/TMR pin will increment the counter by one.

Timer Registers – TMRH, TMRL

The TMRH and TMRL registers are two 8-bit special function register locations within the special purpose Data Memory where the actual timer value is stored. The value in the timer registers increases by one each time an internal clock pulse is received or an external transition occurs on the PA2/TMR pin. The timer will count from the initial value loaded by the preload register to the full count value of FFFFH at which point the timer overflows and an internal interrupt signal generated. The timer value will then be reset with the initial preload register value and continue counting. For a maximum full range count of 0000H to FFFFH the preload registers must first be cleared to 0000H. It should be noted that after power-on the preload registers will be in an unknown condition. Note that if the Timer/Event Counter is not running and data is written to its preload registers, this data will be immediately written into the actual counter. However, if the counter is enabled and counting, any new data written into the preload registers during this period will remain in the preload registers and will only be written into the actual counter the next time an overflow occurs.

Accessing 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 TMRL, 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 TMRH, is executed. On the other hand, 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.

Timer Control Register – TMRC

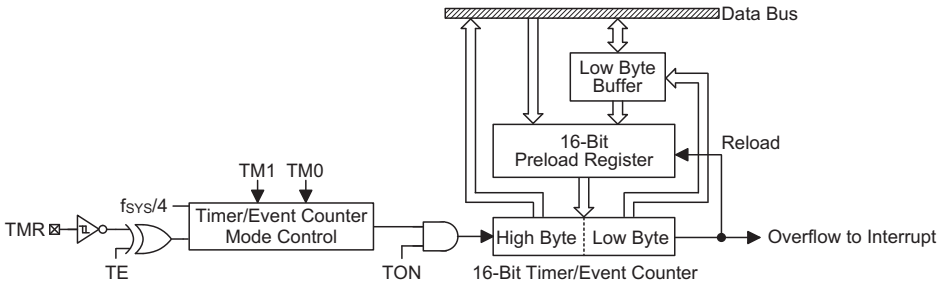
The flexible features of the Holtek microcontroller Timer/Event Counters enable them to operate in three different modes, the options of which are determined by the contents of the Timer Control Register TMRC. Together with the TMRL and TMRH registers, these three

registers control the full operation of the Timer/Event Counter. Before the timer can be used, it is essential that the TMRC register is fully programmed with the right data to ensure its correct operation, a process that is normally carried out during program initialisation.

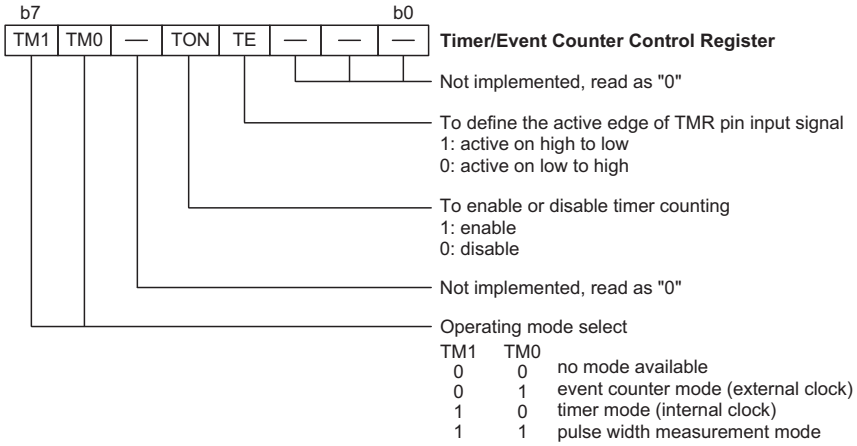
To choose which of the three modes the timer is to operate in, the timer mode, the event counting mode or the pulse width measurement mode, bits TM0 and TM1 must be set to the required logic levels. The timer-on bit TON or bit 4 of the TMRC register provides the basic on/off control of the timer, setting the bit high allows the counter to run, clearing the bit stops the counter. If the timer is in the event count or pulse width measurement mode the active transition edge level type is selected by the logic level of the TE or bit 3 of the TMRC register.

Configuring the Timer Mode

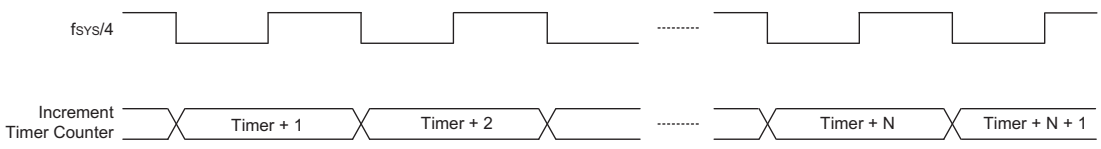
In this mode, the timer can be utilised to measure fixed time intervals, providing an internal interrupt signal each time the counter overflows. To operate in this mode, bits TM1 and TM0 of the TMRC register must be set to 1 and 0 respectively. In this mode, the internal clock is used as the timer clock. The timer-on bit, TON, must be set high to enable the timer to run. Each time an internal clock high to low transition occurs, the timer increments by one. When the timer is full and overflows, the timer will be reset to the value already loaded into the preload register and continue counting. If the timer interrupt is enabled, an interrupt signal will also be generated. The timer interrupt can be disabled by ensuring that the ETI bit in the INTC register is cleared to zero.



16-bit Timer/Event Counter Structure



Timer/Event Counter Control Register



Timer Mode Timing Chart

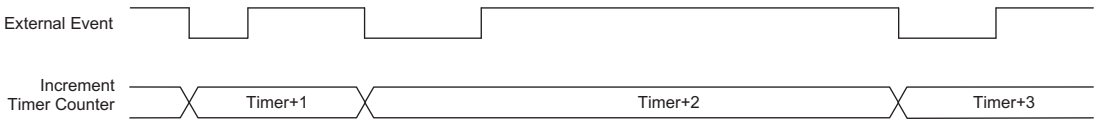
Configuring the Event Counter Mode

In this mode, a number of externally changing logic events, occurring on external pin PA2/TMR, can be recorded by the internal timer. For the timer to operate in the event counting mode, bits TM1 and TM0 of the TMRC register must be set to 0 and 1 respectively. The timer-on bit, TON must be set high to enable the timer to count. With TE low, the counter will increment each time the PA2/TMR pin receives a low to high transition. If the TE bit is high, the counter will increment each time PA2/TMR receives a high to low transition. As in the case of the other two modes, when the counter is full and overflows, the timer will be reset to the value already loaded into the preload register and continue counting. If the timer interrupt is enabled, an interrupt signal will also be generated. The timer interrupt can be disabled by ensuring that the ETI bit in the INTC register is cleared to zero. To ensure that the external pin PA2/TMR is configured to operate as an event counter input pin, two things have to happen. The first is to ensure that the TM0 and TM1 bits place the timer/event counter in the event counting mode, the second is to ensure that the port control register configures the pin as an input. In the Event Counting mode, the Timer/Event Counter will continue to record externally changing logic events on the timer input pin, even if the microcontroller is in the Power Down Mode.

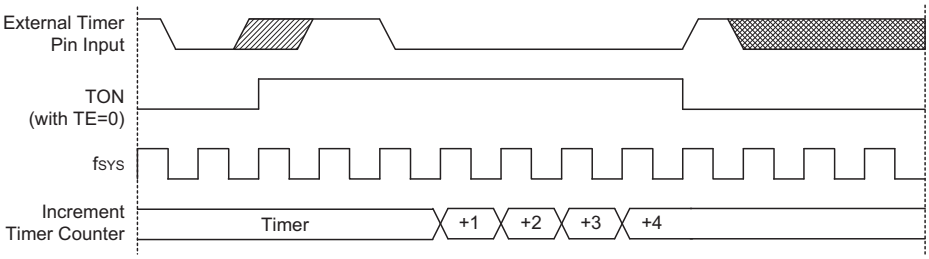
Configuring the Pulse Width Measurement Mode

In this mode, the width of external pulses applied to the pin-shared external pin PA2/TMR can be measured. In the Pulse Width Measurement Mode, the timer clock source is supplied by the internal clock. For the timer to operate in this mode, bits TM0 and TM1 must both be set high. If the TE bit is low, once a high to low transition has been received on the PA2/TMR pin, the timer will start counting until the PA2/TMR pin returns to its original high level. At this point the TON bit will be automatically reset to zero and the timer will stop counting. If the TE bit is high, the timer will begin counting once a low to high transition has been received on the PA2/TMR pin and stop counting when the PA2/TMR pin returns to its original low level. As before, the TON bit will be automatically reset to zero and the timer will stop counting. It is important to note that in the Pulse Width Measurement Mode, the TON 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 TON bit can only be reset to zero under program control. The residual value in the timer, which can now be read by the program, therefore represents the length of the pulse received on pin PA2/TMR. As the TON bit has now been reset any further transitions on the PA2/TMR pin will be ignored. Not until the TON 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 counter is controlled by logical transitions on the PA2/TMR pin and not by the logic level.

As in the case of the other two modes, when the counter is full and overflows, the timer will be reset to the value already loaded into the preload register. If the timer interrupt is enabled, an interrupt signal will also be generated. To ensure that the external pin PA2/TMR is configured to operate as a pulse width measuring input pin, two things have to happen. The first is to ensure that the TM0 and TM1 bits place the timer/event counter in the pulse width measuring mode, the second is to ensure that the port control register configures the pin as an input.



Event Counter Mode Timing Chart



fSYS is sampled at every falling edge of T1.

Pulse Width Measure Mode Timing Chart

I/O Interfacing

The Timer/Event Counter, when configured to run in the event counter or pulse width measurement mode, require the use of the external PA2 pin for correct operation. As this pin is a shared pin it must be configured correctly to ensure it is setup for use as a Timer/Event Counter input and not as a normal I/O pin. This is implemented by ensuring that the mode select bits in the Timer/Event Counter control register, select either the event counter or pulse width measurement mode. Additionally the Port Control Register PAC bit 2 must be set high to ensure that the pin is setup as an input. Any pull-high resistor configuration option on this pin will remain valid even if the pin is used as a Timer/Event Counter input.

Programming Considerations

When configured to run in the timer mode, the internal system clock is used as the timer clock source and is therefore synchronised 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.

When the Timer/Event Counter is read, or if data is written to the preload register, 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 interrupt 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 registers before the timer is switched on; this is because after power-on the initial values of the timer registers are 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. But the timer for internal clock overflow can't wake up the MCU if MCU is in a Power down condition.

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 to be in the timer mode, which uses the internal system clock as the clock source.

```
org 04h
reti
org 08h             ; Timer/Event Counter interrupt vector
jmp tmrint          ; jump here when Timer overflows
:
org 20h             ; main program
;internal Timer/Event Counter interrupt routine
tmrint:
:
; Timer/Event Counter main program placed here
:
reti
:
:
begin:
;setup Timer registers
mov a,09bh          ; setup Timer low register
mov tmrl,a;         ; load low register first
mov a, 0aah         ; setup timer high register
mov tmrh,a
mov a,080h          ; setup Timer control register
mov tmrc,a          ; timer mode is used
; setup interrupt register
mov a,005h          ; enable master interrupt and timer interrupt
mov intc,a
set tmrc.4          ; start Timer/Event Counter - note mode bits must be previously setup
```

Interrupts

Interrupts are an important part of any microcontroller system. When an internal function such as a Timer/Event Counter overflow, their corresponding interrupt will enforce a temporary suspension of the main program allowing the microcontroller to direct attention to their respective needs. These devices contain several interrupts generated by internal interrupts events and external interrupt.

Interrupt Register

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

Interrupt Operation

A Timer/Event Counter overflow, will generate an interrupt request by setting its corresponding request flag, if its 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 jump to another section of program which is known as the interrupt service routine. Here is located the code to control the appropriate interrupt. The interrupt service routine must be terminated with a RETI 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.

Once an interrupt subroutine is serviced, 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.

Timer/Event Counter Interrupt

For a Timer/Event Counter interrupt to occur, the global interrupt enable bit, EMI, and its corresponding timer interrupt enable bit, ETI, must first be set. An actual Timer/Event Counter interrupt will take place when the Timer/Event Counter request flag, TF, 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, will take place. When the interrupt is serviced, the timer interrupt request flag, TF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

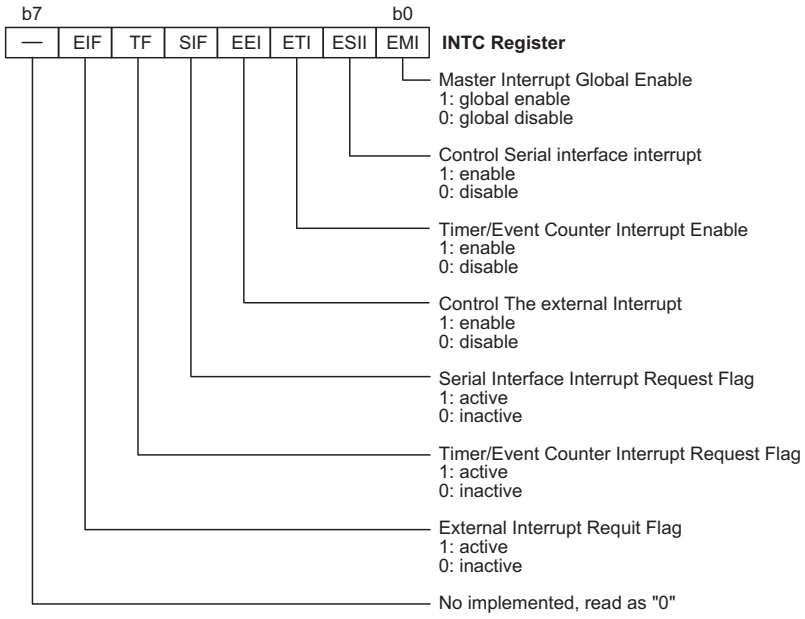
Programming Considerations

By disabling the interrupt enable bit, the requested interrupt can be prevented from being serviced, however, once an interrupt request flag is set, it will remain in this condition in the interrupt control register until the corresponding interrupt is serviced or until the request flag is cleared by a software instruction.

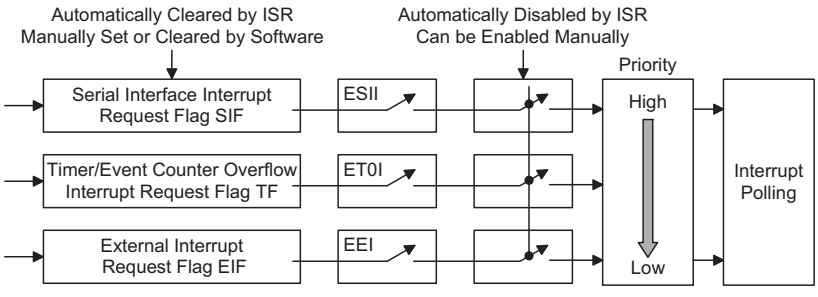
It is recommended that programs do not use the "CALL subroutine" instruction within the interrupt subroutine. Interrupts often occur in an unpredictable manner or need to be serviced immediately in some applications. If only one stack is left and the interrupt is not well controlled, the original control 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 accumulator 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.



Interrupt Control Register



Interrupt Structure

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.

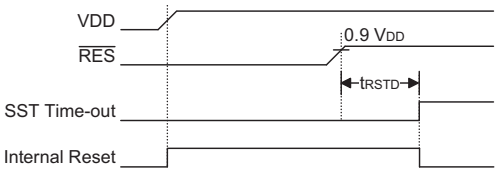
In addition to the power-on reset, situations may arise where it is necessary to forcefully apply a reset condition when the microcontroller is running. One example of this is where after power has been applied and the microcontroller is already running, the $\overline{\text{RES}}$ line is forcefully 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.

Another reset exists in the form of a Low Voltage Reset, LVR, where a full reset, similar to the $\overline{\text{RES}}$ reset is implemented in situations where the power supply voltage falls below a certain threshold.

Reset Functions

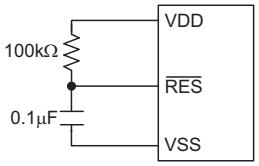
There are five ways in which a microcontroller reset can occur, through events occurring both internally and externally:

- 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.



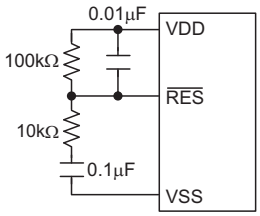
Power-On Reset Timing Chart

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

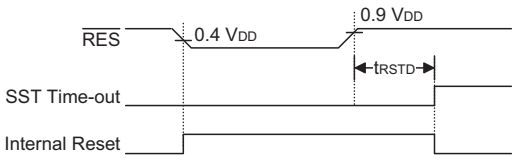
For applications that operate within an environment where more noise is present the Enhanced Reset Circuit shown is recommended.



Enhanced Reset Circuit

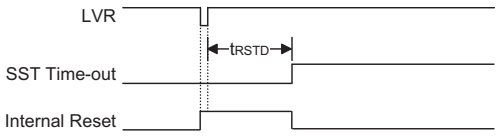
More information regarding external reset circuits is located in Application Note HA0075E on the Holtek website.

- $\overline{\text{RES}}$ Pin Reset
  This type of reset occurs when the microcontroller is already running and the $\overline{\text{RES}}$ 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.



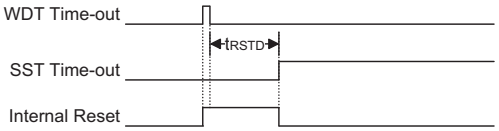
$\overline{\text{RES}}$ Reset Timing Chart

- Low Voltage Reset – LVR
  The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device. The LVR function is selected via a configuration option. If the supply voltage of the device drops to within a range of 0.9V~V_{LVR} such as might occur when changing the battery, the LVR will automatically reset the device internally. For a valid LVR signal, a low supply voltage, i.e., a voltage in the range between 0.9V~V_{LVR} must exist for a time greater than that specified by t_{LVR} in the A.C. characteristics. If the low supply voltage state does not exceed this value, the LVR will ignore the low supply voltage and will not perform a reset function. The actual V_{LVR} value can be selected via configuration options.



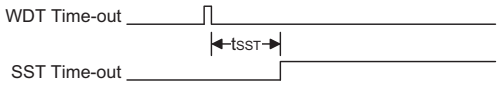
Low Voltage Reset Timing Chart

- 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
0	1	$\overline{RES}$ wake-up HALT
u	u	$\overline{RES}$ or LVR 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 Counter	Timer Counter 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)	WDT time-out (Normal Operation)	$\overline{RES}$ Reset (Normal Operation)	$\overline{RES}$ Reset (HALT)	WDT Time-out (HALT)*
PCL	000H	000H	000H	000H	000H
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	-xxx xxxx	-uuu uuuu	-uuu uuuu	-uuu uuuu	-uuu uuuu
STATUS	--00 xxxx	--1u uuuu	--uu uuuu	--01 uuuu	--11 uuuu
INTC	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
TMRL	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMRH	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMRC	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 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
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
WDTS	---- -111	---- -111	---- -111	---- -111	---- -uuu
MP0	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
MP1	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
CTLR	0100 0x00	0100 0x00	0100 0x00	0100 0x00	uuuu uxuu
PTR	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TBHP	----- 0000	----- 0000	----- 0000	----- 0000	0000 uuuu
SPIR	0000 0000	0000 0000	0000 0000	0000 0000	0000 uuuu
SBCR	0110 0000	0110 0000	0110 0000	0110 0000	uuuu uuuu
SBDR	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu

Note: "**" For the HT82K75R only
"-" not implemented
"u" means "unchanged"
"x" means "unknown"

Oscillator

The clock source for these devices is provided by an integrated oscillator requiring no external components.

This oscillator has one fixed frequencies of 6MHz.

Watchdog Timer Oscillator

The WDT oscillator is a fully self-contained free running on-chip RC oscillator with a typical period of 71μs at 3V requiring no external components. When the device enters the Power Down Mode, the system clock will stop running but the WDT oscillator continues to free-run and to keep the watchdog active. However, to preserve power in certain applications the WDT oscillator can be disabled via a configuration option.

Power Down Mode and Wake-up

Power Down Mode

All of the Holtek microcontrollers have the ability to enter a Power Down 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, the system oscillator is 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.

Entering the Power Down Mode

There is only one way for the device to enter the Power Down Mode and that is to execute the "HALT" instruction in the application program. When this instruction is executed, the following will occur:

- The system oscillator will stop running and the application program will stop at the "HALT" instruction.
- The Data Memory contents and registers will maintain their present condition.
- The WDT will be cleared and resume counting if the WDT function is enabled.
- The I/O ports will maintain their present condition.
- In the status register, the Power Down flag, will be set and the Watchdog time-out flag, TO, will be cleared.

Standby Current Considerations

As the main reason for entering the Power Down Mode is to keep the current consumption of the microcontroller 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/O pins, which are setup as outputs. These should be placed in a condition in which minimum current is drawn or connected only to external circuits that do not draw current, such as other CMOS inputs.

If the configuration option has enabled the Watchdog Timer internal oscillator, then the Watchdog Timer will continue to run when in the Power Down Mode and will thus consume some power.

Wake-up

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

- An external reset
- An external falling or rising edge on any of the I/O pins
- A system interrupt
- A WDT overflow (if the contents of the PTR are zeros)
- A PTR overflow occurs (if the contents of the PTR are not equal to zeros)

If the system is woken up by an external reset, the device will experience a full system reset, however, if the device is woken up by a WDT overflow, a Watchdog Timer reset will be initiated. Although both of these wake-up methods will initiate a reset operation, the actual source of the wake-up can be determined by examining the TO and PDF flags. The PDF flag is cleared by a system power-up or executing the clear Watchdog Timer instructions and is set when executing the "HALT" instruction. The TO flag is set if a WDT time-out occurs, and causes a wake-up that only resets the Program Counter and Stack Pointer, the other flags remain in their original status. Note that the WDT time-out will not occur if the contents of the Period Timer Register (PTR) are not equal to zeros.

Each pin on Port A or any nibble on other ports can be setup via configuration options to permit a negative or positive transition on the pin to wake-up the system. When a port pin wake-up occurs, the program will resume execution at the instruction following the "HALT" instruction.

If the system is woken up by an interrupt, then two possible situations may occur. The first is where the interrupt is disabled or the interrupt is enabled but the stack is full, in which case the program will resume execution at the instruction following the "HALT" instruction. In this situation, the interrupt will not be immediately serviced, but will rather be serviced later when the related interrupt is

finally enabled or when a stack level becomes free. The other situation is where the related interrupt is enabled and the stack is not full, in which case the regular interrupt response takes place. If an interrupt request flag is set to ″1″ before entering the Power Down Mode, the wake-up function of the related interrupt will be disabled.

No matter what the source of the wake-up event is, once a wake-up situation occurs, a time period equal to 512 system clock periods will be required before normal system operation resumes. However, if the wake-up has originated due to an interrupt, the actual interrupt subroutine execution will be delayed by additional one or more cycles. If the wake-up results in the execution of the next instruction following the ″HALT″ instruction, this will be executed immediately after the 512 system clock period delay has ended.

Watchdog Timer

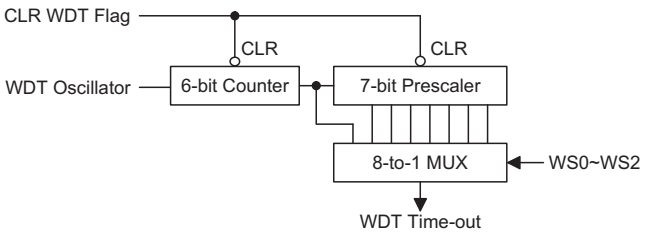
The Watchdog Timer is provided to prevent program malfunctions or sequences from jumping to unknown locations, due to certain uncontrollable external events such as electrical noise. It operates by providing a device reset when the WDT counter overflows. The WDT clock is supplied by its own internal dedicated internal WDT oscillator. Note that if the WDT configuration option has been disabled, then any instruction relating to its operation will result in no operation.

The WDT function is selected by a configuration option. There is also an internal register associated with the WDT named WDTS to disable the Watchdog Timer function and select various WDT time-out periods in the device. The clock source of the WDT comes from the internal WDT oscillator and its clock period may vary with VDD, temperature and process variation. The WDT clock is further divided by an internal 6-stage counter followed by a 7-stage prescaler to obtain longer WDT time-out period selected by the WDT prescaler rate selection bits, WS2~WS0, in the associated WDT register known as WDTS.

There is only one instruction to clear the Watchdog Timer known as ″CLR WDT″. As the instruction ″CLR WDT″ is executed, all contents of the 6-stage counter and 7-stage prescaler will be clear. It makes the WDT time-out period more accurate relatively.

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, the TO bit in the status register will be set and only the Program Counter and Stack Pointer will be reset. Three methods can be adopted to clear the contents of the WDT. 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.

Although the WDT overflow is a source to wake up the MCU from the Power Down Mode, there are some limitations on the conditions at which the WDT overflow occurs. If the WDT function is enabled and the PTR contents are equal to zeros, the WDT overflow will occur to wake up the MCU from the Power Down Mode. If the PTR contents are not equal to zeros, the WDT overflow will not occur in Power Down Mode even if the WDT function has been enabled.



Watchdog Timer

Bit No.	Func. Name	R/W	Description
0	CNT_WK	R	0: MCU wakeup not by period counter 1: MCU wakeup by period counter overflow (Read only)
1	DC_Ctrl	R/W	This bit is used to decide whether the DC block is in operation 0: enable DC_DC output (default) 1: disable DC_DC output
2	2.2 Low Battery	R	Flag for 2.2V battery low signal coming from DC/DC block (error ±5%) 0: battery voltage > 2.2 or 2.0V 1: battery voltage ≤ 2.2 or 2.0 V
3 4	Clock_div	R/W	00: CLK/1=6MHz (default) 01: CLK/2=3MHz 10: CLK/4=1.5MHz 11: CLK/4=1.5MHz
5 6 7	LVD_Set	R/W	LVD detect voltage select 000: 1.8V 001: 2.0V 010: 2.2V (default) 011: 2.5V 100: 2.8V

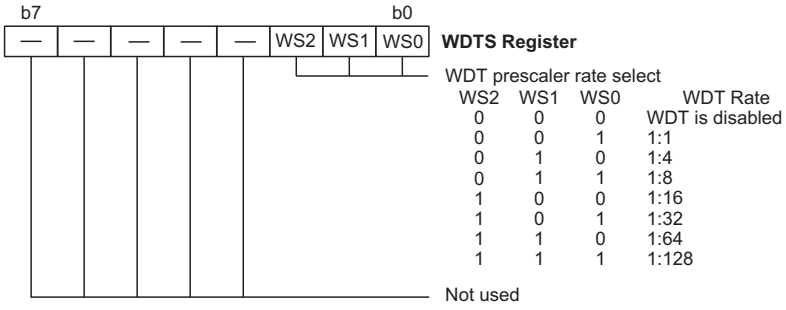
Control Register CTLR

Period Timer Register – PTR

This register is used to define the period of the timer which always counts in the Power Down mode. Once the timer is reached, the MCU will be woken-up by Period Timer Register overflow. Once the MCU is woken-up by the period timer, the CNT_WK bit of the wake-up Register is set to ″1″.

Bit No.	Function Name	R/W	Description
0~7	Period Timer	R/W	The Period Timer is the time interval generator with one second as a unit. If the bits [7:0] are equal to 00H, the MCU will be woken up by one of the wake-up source mentioned in Wake-up Section except the PTR overflow event. If the bits [7:0] are not equal to 00H, the MCU will be woken up from the Power Down mode by the following events except WDT overflow event: • I/O Port wake-up • $\overline{\text{INT}}$ wake-up • Reset • The Period Timer is reached to the values specified by the PTR.

Period Timer Register – PTR

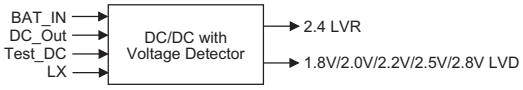


Watchdog Timer Register

DC-to-DC Converter (DC/DC) Section

This circuit is used to generate a stable 2.8V or 3.0V or 3.3V (error±5%) power voltage for whole IC and output to the IRPT. The clock of DC/DC is 140kHz. Also it can detect the battery voltage. If the battery voltage drops to 2.2V or 2.0V determined by a configuration option (error±5%), the DC/DC circuit will output a Low Voltage Detect signal LVD (2.2V/2.0V Low battery flag stored in associated flag bit of the Control Register CTLR.2) to MCU. Also there is a low voltage reset (LVR) circuit to check the DC/DC output voltage. When the DC/DC output voltage drops to 2.4V, the MCU will be reset. The LVR function is controlled by a configuration together with a software control bit named DC_ctrl in the Control Register CTLR. To enable the LVR function, the configuration option of LVR function has to be enabled and the control bit DC_ctrl must be set to 0 to enable the DC/DC circuit. If the configuration option is selected to disable the LVR function or the DC_ctrl bit is set to 1 to disable the DC/DC circuit, then the LVR function will be disabled. If the LVR function is enabled by appropriate setting of the configuration option and software control bit as mentioned above, then the LVR still works even if the MCU enters into the Power Down Mode. It is recommended that the LVR function is enabled when the MCU is in the Power Down Mode.

When the DC/DC output voltage drops to 2.2V, the DC/DC can still work properly and is capable of outputting driving current with 100mA typically.



As the voltage of the Battery-in pin drops to 1.8V, the DC/DC still has the capability of outputting current with 40mA at least.

The DC/DC output signal 1.8V/2.0V/2.2V/2.5V/2.8V LVD is connected to the associated flag in Control Register (i.e. bit 2 in CTLR).

Test_DC is the internal test pin of the DC_DC.

SPI Serial Interface

The device includes one SPI Serial Interfaces. The SPI interface is a full duplex serial data link, originally designed by Motorola, which allows multiple devices connected to the same SPI bus to communicate with each other. The devices communicate using a master/slave technique where only the single master device can initiate a data transfer. A simple four line signal bus is used for all communication.

SPI Interface Communication

Four lines are used for each function. These are, SDI - Serial Data Input, SDO - Serial Data Output, SCK - Serial Clock and $\overline{SCS}$ - Slave Select. Note that the condition of the Slave Select line is conditioned by the CSEN bit in the SBCR control register. If the CSEN bit is high then the $\overline{SCS}$ line is active while if the bit is low then the $\overline{SCS}$ line will be I/O mode. The accompanying timing diagram depicts the basic timing protocol of the SPI bus.

SPI Registers

There are three registers for control of the SPI Interface. These are the SBCR register which is the control register and the SBDR which is the data register and SPIR register which is the SPI mode control register. The SBCR register is used to setup the required setup parameters for the SPI bus and also used to store associated operating flags, while the SBDR register is used for data storage.

The SPIR register is used to select SPI mode, clock polarity edge selection and SPI enable or disable selection.

After Power on, the contents of the SBDR register will be in an unknown condition while the SBCR register will default to the condition below:

CKS	M1	M0	SBEN	MLS	CSEN	WCOL	TRF
0	1	1	0	0	0	0	0

Note that data written to the SBDR register will only be written to the TXRX buffer, whereas data read from the SBDR register will actual be read from the register.

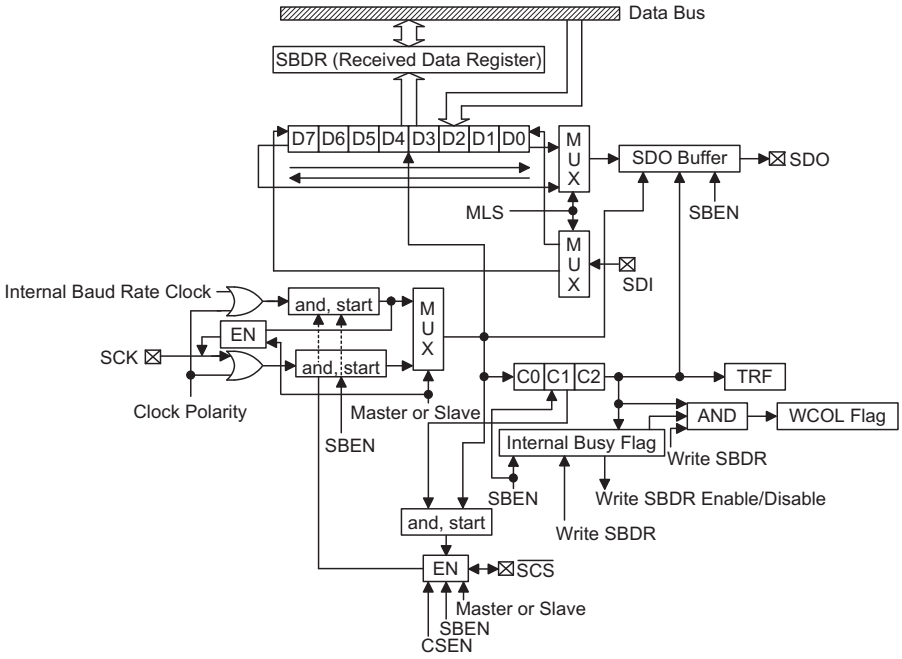
SPI Bus Enable/Disable

To enable the bus, the SBEN bit should be set high, then wait for data to be written to the SBDR (TXRX buffer) register. For the Master Mode, after data has been written to the SBDR (TXRX buffer) register then transmission or reception will start automatically. When all the data has been transferred, the TRF bit should be set. For the Slave Mode, when clock pulses are received on SCK, data in the TXRX buffer will be shifted out or data on SDI will be shifted in.

To Disable the SPI bus SCK, SDI, SDO, $\overline{SCS}$ should be I/O mode.

Bit No.	Label	R/W	Function
0	SPI_CPOL	R/W	0: clock polarity falling (default falling) 1: clock polarity rising
1	SPI_MODE	R/W	0: SPI output the data in the rising edge(polarity=1) or falling edge (polarity=0); SPI read data in the in the falling edge(polarity=1) or rising edge (polarity=0); (default) 1: SPI first output the data immediately after the SPI is enable. And SPI output the data in the falling edge(polarity=1) or rising edge (polarity=0); SPI read data in the in the rising edge(polarity=1) or falling edge (polarity=0)
2	SPI_CSEN	R/W	0: SPI_CSEN disable, $\overline{SCS}$ define as GPIO (default disable) 1: SPI_CSEN Enable , this bit is used to enable/disable software CSEN function
3	SPI_EN	R/W	This bit control the shared PIN ($\overline{SCS}$, SDI, SDO and SCK) is SPI or GPIO mode 0: I/O mode (default) 1: SPI mode
7~4	Reserved bit	R/W	Always 0

SPIR Register



SPI Block Diagram

Note: WCOL: set by SPI cleared by users

CSEN: enable/disable chip selection function pin
master mode: 1/0 = with/without $\overline{SCS}$ output function
Slave mode: 1/0 = with/without $\overline{SCS}$ input control function

SBEN: enable/disable serial bus (0: initialise all status flags)
when SBEN=0, all status flags should be initialised
when SBEN=1, all SPI related function pins should stay at floating state

TRF: 1 = data transmitted or received, 0= data is transmitting or still not received

CPOL: I/O = clock polarity rising/falling edge: For SPIR Register.

If clock polarity set to rising edge (SPI_CPOL=1), serial clock timing follow SCK, otherwise (SPI_CPOL=0) SCK is the serial clock timing.

SPI Operation

All communication is carried out using the 4-line interface for both Master or Slave Mode. The timing diagram shows the basic operation of the bus.

The CSEN bit in the SBCR register controls the $\overline{SCS}$ line of the SPI interface. Setting this bit high, will enable the SPI interface by allowing the $\overline{SCS}$ line to be active, which can then be used to control the SPI inteface. If the CSEN bit is low, the $\overline{SCS}$ line will be in a floating condition and can therefore not be used for control of the SPI interface. The SBEN bit in the SBCR register must also be high which will place the SDI line in a floating condition and the SDO line high. If in the Master Mode the SCK line will be either high or low depending upon the clock polarity control bit in SPIR register. If in the Slave Mode the SCK line will be in a floating condition. If SBEN is low then the bus will be disabled and $\overline{SCS}$, SDI, SDO and SCK will all be I/O mode.

In the Master Mode, the Master will always generate the clock signal. The clock and data transmission will be initiated after data has been written to the SBDR register. In the Slave Mode, the clock signal will be received from an external master device for both data transmission or reception. The following sequences show the order to be followed for data transfer in both Master and Slave Mode:

- Master Mode
  - Step 1. Select the clock source using the CKS bit in the SBCR control register
  - Step 2. Setup the M0 and M1 bits in the SBCR control register to select the Master Mode and the required Baud rate. Values of 00, 01 or 10 can be selected.
  - Step 3. Setup the CSEN bit and setup the MLS bit to choose if the data is MSB or LSB first, this must be same as the Slave device.
  - Step 4. Setup the SBEN bit in the SBCR control register to enable the SPI interface.
  - Step 5. For write operations: write the data to the SBDR register, which will actually place the data into the TXRX buffer. Then use the SCK and $\overline{SCS}$ lines to output the data. Goto to step 6.For read operations: the data transferred in on the SDI line will be stored in the TXRX buffer until all the data has been received at which point it will be latched into the SBDR register.
  - Step 6. Check the WCOL bit, if set high then a collision error has occurred so return to step5. If equal to zero then go to the following step.
  - Step 7. Check the TRF bit or wait for an SPI serial bus interrupt.
  - Step 8. Read data from the SBDR register.
  - Step 9. Clear TRF.
  - Step10. Goto step 5.
- Slave Mode:
  - Step 1. The CKS bit has a don't care value in the slave mode.
  - Step 2. Setup the M0 and M1 bits to 11 to select the Slave Mode. The CKS bit is don't care.
  - Step 3. Setup the CSEN bit and setup the MLS bit to choose if the data is MSB or LSB first, this must be same as the Master device.
  - Step 4. Setup the SBEN bit in the SBCR control register to enable the SPI interface.
  - Step 5. For write operations: write data to the SBDR register, which will actually place the data into the TXRX register, then wait for the master clock and $\overline{SCS}$ signal. After this goto Step 6.
    For read operations: the data transferred in on the SDI line will be stored in the TXRX buffer until all the data has been received at which point it will be latched into the SBDR register.
  - Step 6. Check the WCOL bit, if set high then a collision error has occurred so return to step5. If equal to zero then goto the following step.
  - Step 7. Check the TRF bit or wait for an SPI serial bus interrupt.
  - Step 8. Read data from the SBDR register.
  - Step 9. Clear TRF
  - Step10. step 5

SPI Configuration Options and Status Control

One option is to enable the operation of the WCOL, write collision bit, in the SBCR register. Some control in SPIR register. The SPI_CPOL select the clock polarity of the SCK line . The SPI_MODE select SPI data output mode.

SPI include four pins , can share I/O mode status . The status control combine with four bits for SPIR and SBCR register. Include SPI_CSEN , SPI_EN for SPIR register and CSEN ,SBEN for SBCR register.

SPIR(22H)		SBCR(23H)		I/O Status		Note
SPI_EN	SPI_CSEN	SBEN	CSEN	SPI	$\overline{SCS}$	
0	x	x	x	I/O mode	I/O mode	
1	x	0	x	I/O mode	I/O mode	
1	0	1	x	SPI mode	I/O mode	$\overline{SCS}$ not Floating
1	1	1	0	SPI mode	I/O mode	$\overline{SCS}$ not Floating
1	1	1	1	SPI mode	$\overline{SCS}$ mode	The SPI enable, $\overline{SCS}$, SDI, SDO, SCK the internal Pull-high function is invalid.

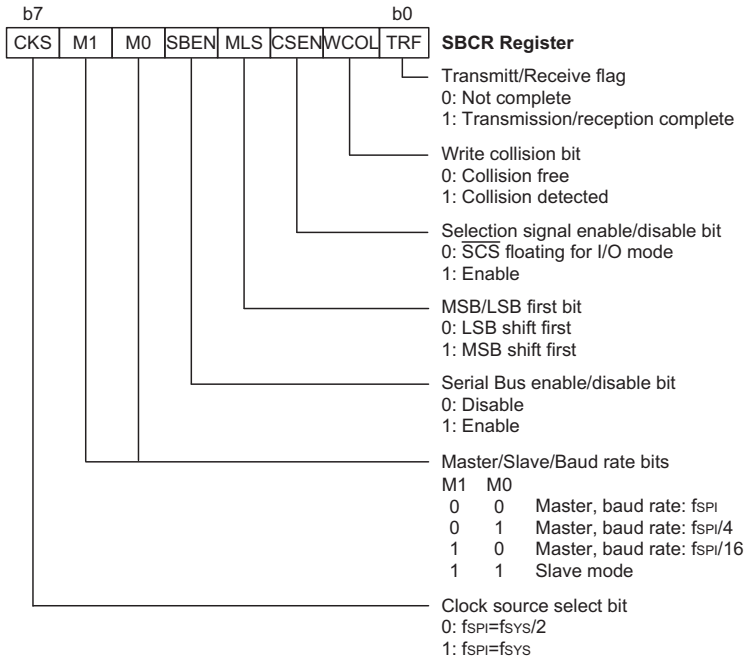
Note: X: don't care

Error Detection

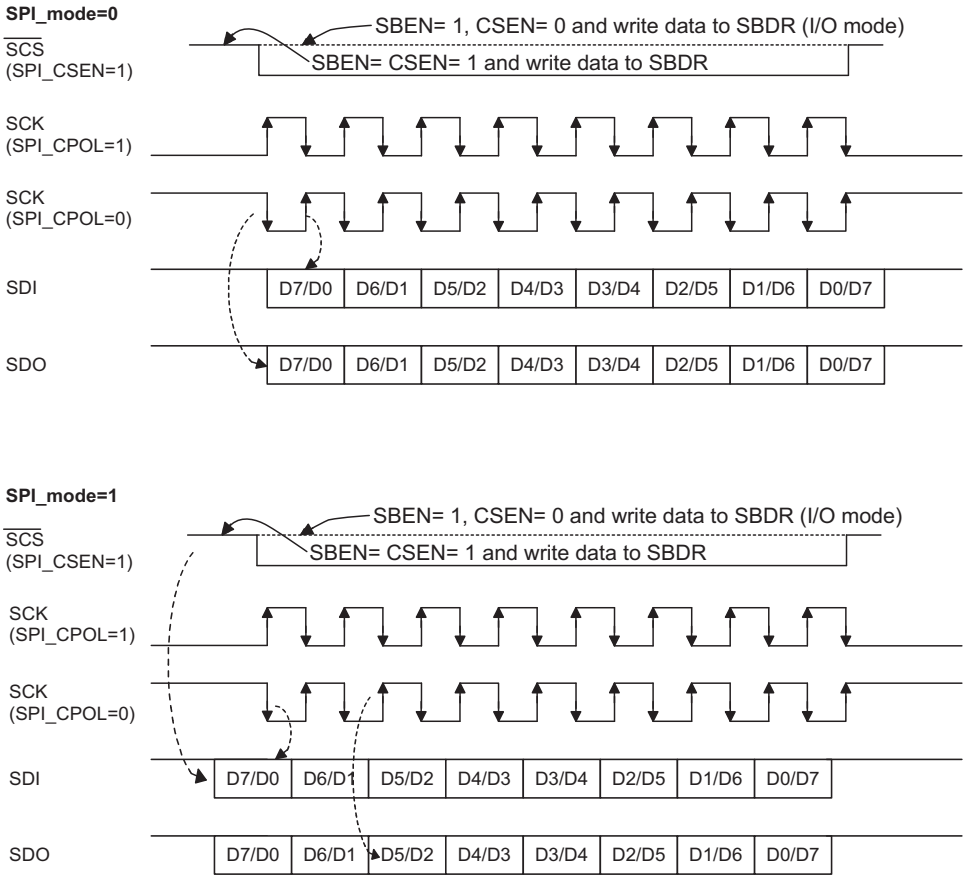
The WCOL bit in the SBCR register is provided to indicate errors during data transfer. The bit is set by the Serial Interface but must be cleared by the application program. This bit indicates a data collision has occurred which happens if a write to the SBDR register takes place during a data transfer operation and will prevent the write operation from continuing. The bit will be set high by the Serial Interface but has to be cleared by the user application program. The overall function of the WCOL bit can be disabled or enabled by a configuration option.

Programming Considerations

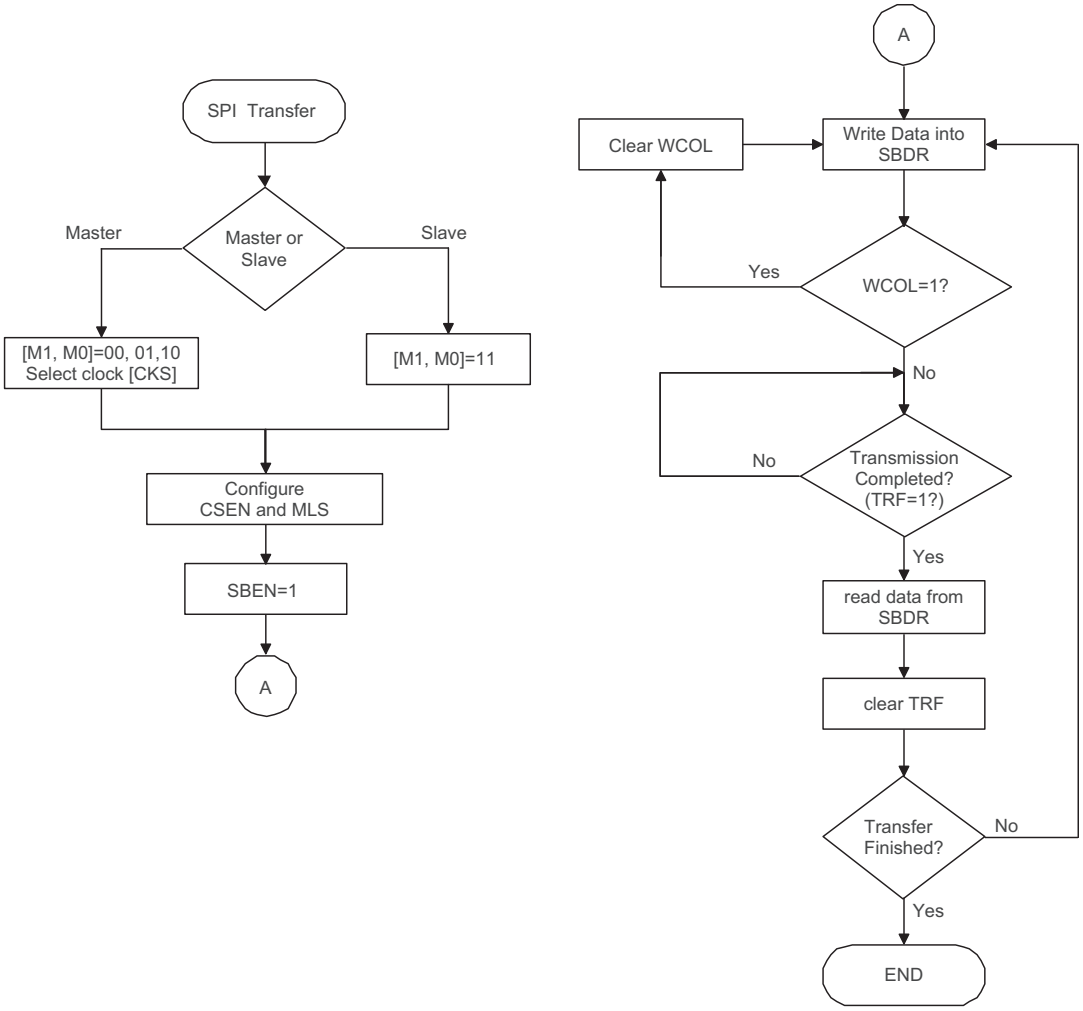
When the device is placed into the Power Down Mode note that data reception and transmission will continue. The TRF bit is used to generate an interrupt when the data has been transferred or received.



SPI Interface Control Register



SPI Bus Timing



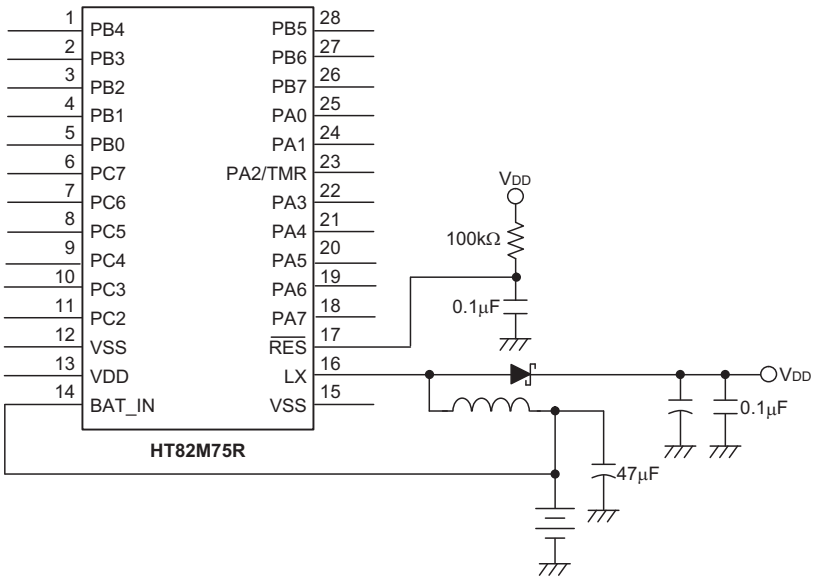
SPI Transfer Control Flowchart

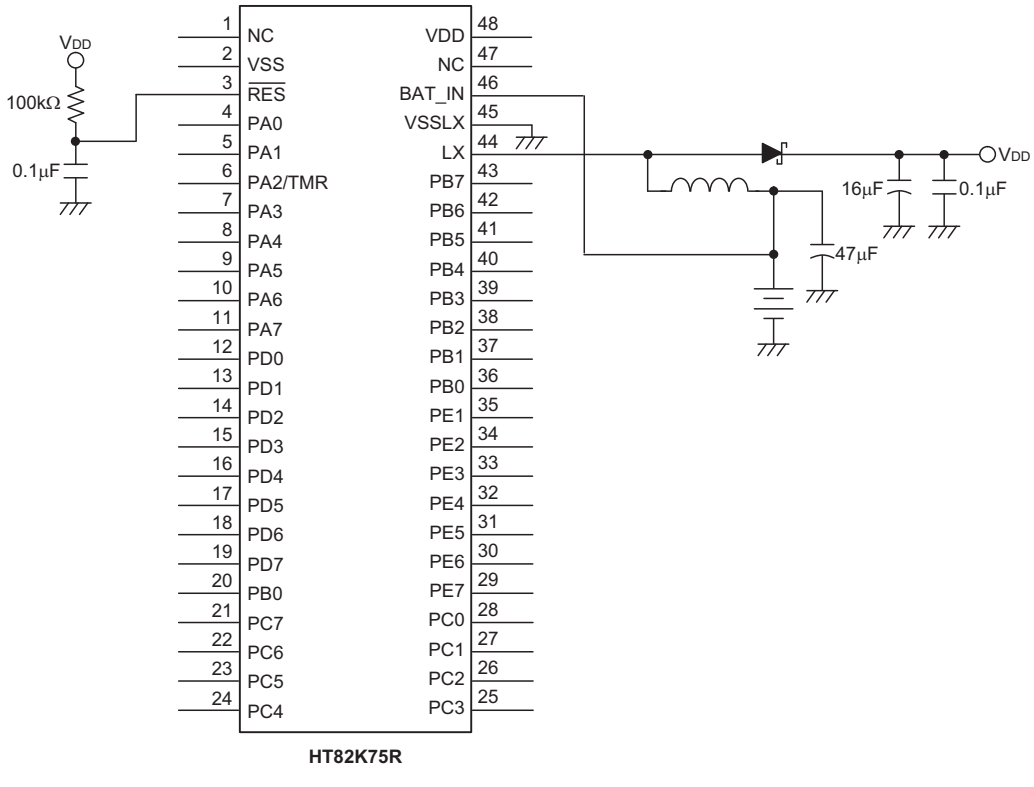
Configuration Options

No.	Options
1	PA0~7 Pull-high by bit: Pull-high or non-pull-high
2	PA wake-up by bit, :Wake-up or non-wake-up
3	PB Pull-high by nibble : Pull-high or non-pull-high
4	PC Pull-high by nibble : Pull-high or non-pull-high
5	SPI_WCOL enable/disable (default)
6	Output slew enable 100ns or 200ns
7	TBHP function (enable /disable)
8	DC_DC output option:2.8V,3.0V,3.3V
9	LVR enable/disable (default disable)
10	WDT clock source : enable, disable for normal mode
11	PB wake-up by bit, Wake-up or non-wake-up
12	PC wake-up by bit, Wake-up or non-wake-up
13	PC output type CMOS/NMOS
14	PB0 output type CMOS/NMOS
15	PD pull-high by nibble: pull-high or non-pull-high (*)
16	PE pull-high by nibble: pull-high or non-pull-high (*)
17	PD wake-up by nibble: wake-up or non-wake-up (*)
18	PE wake-up by nibble: wake-up or non-wake-up (*)

Note: For HT82K75R, there are additional configuration options as the asterisk marks shown.

Application Circuits





EEPROM Data Memory

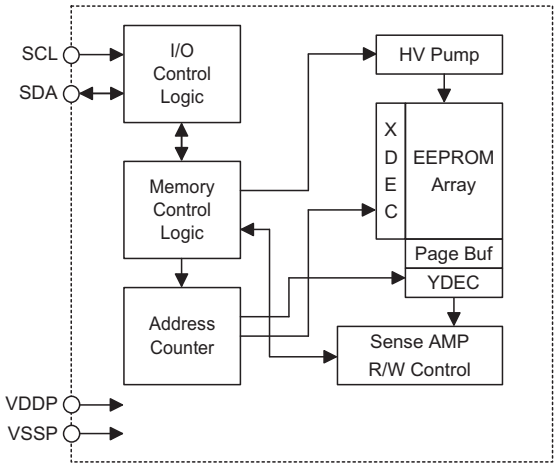
EEPROM Memory Features

- 1K capacity organized into 128×8
- Accessible using two I^2C lines
- Device address: 0×1010000B followed with a read/write operation selection bit
- Device Operations:
  - Byte Write Operation
  - Current Address Read Operation
  - Random Address Read Operation
  - Sequential Address Read Operation

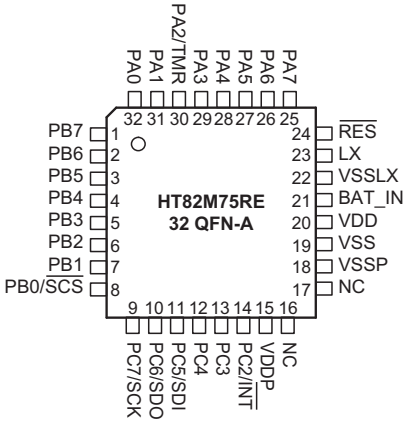
EEPROM Memory Overview

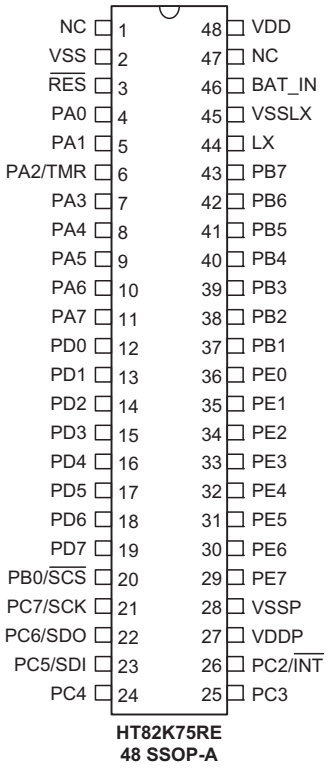
An area of EEPROM, which stands for Electrically Erasable Programmable Read Only Memory, is contained within the device. This type of memory is non-volatile with data retention even after power is removed and is useful for storing information such as product identification numbers, calibration values, user data, system setup data, etc.

Block Diagram



Pin Assignment





EEPROM Memory Pin Description

Pin Name	Type	Description
VDDP	—	External Positive power supply for EEPROM Memory
VSSP	—	External Negative power supply for EEPROM Memory, ground
SDA	I/O	Internal Serial data input/output signal Internal connected with MCU I/O line.
SCL	I	Serial clock input signal Internal connected with MCU I/O line.
NC	—	Implies that the pin is ″Not Connected″ and can therefore not be used.

Note: The pin descriptions for all external pins with the exception of the EEPROM VDDP and VSSP pins are described in the preceding MCU section.

VDDP and VSSP should be externally connected to the MCU power supply named VDD and VSS respectively.

The SDA and SCL lines here are internal connected to the MCU I/O pins PC0 and PC1 respectively for these devices.

Absolute Maximum Ratings

Supply Voltage .....V_{SS}–0.3V to V_{SS}+6.0V
Input Voltage .....V_{SS}–0.3V to V_{CC}+0.3V
Storage Temperature .....–50°C to 125°C
Operating Temperature.....–40°C to 85°C

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=–40°C~85°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{CC}	Conditions				
I_{CC1}*	Operating Current	3V	Read at 100kHz	—	—	1	mA
I_{CC2}*	Operating Current	3V	Write at 100kHz	—	—	3	mA
I_{STB}*	Standby Current	3V	V_{IN}=0 or V_{CC}	—	—	3	μA

Note: "*" The operating current ICC1 and ICC2 listed here are the additional currents consumed when the EEPROM Memory operates in Read Operation and Write Operation respectively. If the EEPROM is operating, the ICC1 or ICC2 should be added to calculate the relevant operating current of the device for different conditions. To calculate the standby current for the whole device, the standby current shown above should also be taken into account.

A.C. Characteristics

Ta=–40°C~85°C

Symbol	Parameter	Remark	Standard Mode*		Unit
			Min.	Max.	
f_{SK}	SCL Clock Frequency	—	—	100	kHz
t_{HIGH}	Clock High Time	—	4000	—	ns
t_{LOW}	Clock Low Time	—	4700	—	ns
t_r	SDA and SCL Rise Time	Note	—	1000	ns
t_f	SDA and SCL Fall Time	Note	—	300	ns
$t_{HD:STA}$	START Condition Hold Time	After this period the first clock pulse is generated	4000	—	ns
$t_{SU:STA}$	START Condition Setup Time	Only relevant for repeated START condition	4000	—	ns
$t_{HD:DAT}$	Data Input Hold Time	—	0	—	ns
$t_{SU:DAT}$	Data Input Setup Time	—	200	—	ns
$t_{SU:STO}$	STOP Condition Setup Time	—	4000	—	ns
t_{AA}	Output Valid from Clock	—	—	3500	ns
t_{BUF}	Bus Free Time	Time in which the bus must be free before a new transmission can start	4700	—	ns
t_{SP}	Input Filter Time Constant (SDA and SCL Pins)	Noise suppression time	—	100	ns
t_{WR}	Write Cycle Time	—	—	5	ms

Note: These parameters are periodically sampled but not 100% tested

* The standard mode means V_{CC}=2.2V to 3.6V

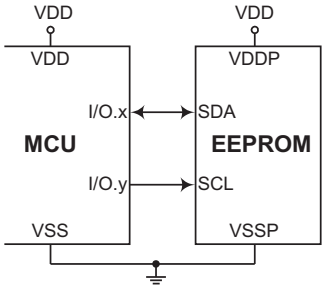
For relative timing, refer to timing diagrams

EEPROM Data Memory Functional Description

The embedded EEPROM Data Memory is an I2C type device and therefore operates using a two wire serial bus. It has a capacity is 1K organized into a structure of 128 8-bit words and contains the information or data important for user.

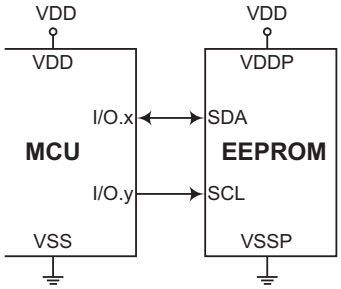
EEPROM Data Memory Internal Connection

In addition to the pins described above there are other MCU to EEPROM Data Memory interconnecting lines that are described in the above EEPROM Pin Description table. Note that the SDA and SCL lines are internal connected to the MCU I/O pins respectively and are not bonded to external pins.



Dual VDD/Single VSS Power Supply

MCU to EEPROM Internal Connection



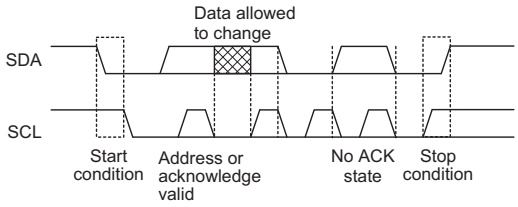
Dual VDD / Dual VSS Power Supply

MCU to EEPROM Internal Connection

Accessing the EEPROM Data Memory

The two I2C lines are the Serial Clock line, SCL, and the Serial Data line SDA. The SDA and SCL pins are internal connected to the host MCU I/O pins. Normal I/O control software instructions are used to control the reading and writing operations on the EEPROM Data Memory.

- Serial data - SDA
  The SDA line is the bidirectional EEPROM serial data line which is controlled by the host MCU I/O pin. The host MCU should configure this I/O pin as input or output dynamically opposite to the data direction of the EEPROM. The SDA line is an internal line and not connected to an output pin.
- Serial clock - SCL
  The SCL line is the EEPROM serial clock input line which is controlled by the host MCU I/O pin. The host MCU should configure this I/O pin connected to the SCL line as output pin. The SCL line is an internal line and not connected to an output pin. The SCL input clocks data into the EEPROM on its positive edge and clocks data out of the EEPROM on its negative edge.
- Clock and data transition
  Data transfer may be initiated only when the bus is not busy. During data transfer, the data line must remain stable whenever the clock line is high. Changes in the data line while the clock line is high will be interpreted as a START or STOP condition.
- Start condition
  A high-to-low transition of SDA with SCL high will be interpreted as a start condition which must precede any other command - refer to the Start and Stop Definition Timing diagram.
- Stop condition
  A low-to-high transition of SDA with SCL high will be interpreted as a stop condition. After a read sequence the stop command will place the EEPROM in a standby power mode - refer to Start and Stop Definition Timing Diagram.
- Acknowledge
  All addresses and data words are serially transmitted to and from the EEPROM in 8-bit words. The EEPROM sends a zero to acknowledge that it has received each word. This happens during the ninth clock cycle.



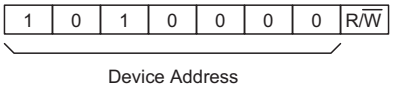
Start and Stop Definition Timing Diagram

Device Addressing

All EEPROM devices require an 8-bit device address word following a start condition to enable the EEPROM for read or write operations. The device address word consist of a mandatory one, zero sequence for the first four most significant bits. Refer to the diagram showing the Device Address. This is common to all the EEPROM devices. The next three bits are all zero bits.

The 8th bit of device address is the read/write operation select bit. A read operation is initiated if this bit is high and a write operation is initiated if this bit is low.

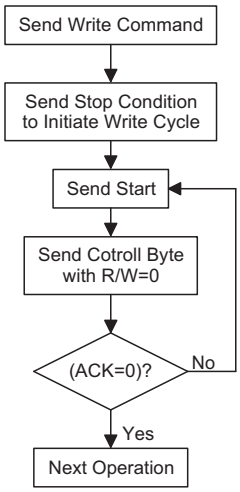
If the comparison of the device address is successful then the EEPROM will output a zero as an ACK bit. If not, the EEPROM will return to a standby state.



Device Operations

- Byte Write
  A write operation requires an 8-bit data word address following the device address word and acknowledgment. Upon receipt of this address, the EEPROM will again respond with a zero and then clock in the first 8-bit data word. After receiving the 8-bit data word, the EEPROM will output a zero and the addressing device must terminate the write sequence with a stop condition. At this time the EEPROM enters an internally-timed write cycle to the non-volatile memory. All inputs are disabled during this write cycle and EEPROM will not respond until the write cycle is completed.

- Acknowledge polling
  To maximize bus throughput, one technique is to allow the master to poll for an acknowledge signal after the start condition and the control byte for a write command have been sent. If the device is still busy implementing its write cycle, then no ACK will be returned. The master can send the next read/write command when the ACK signal has finally been received.



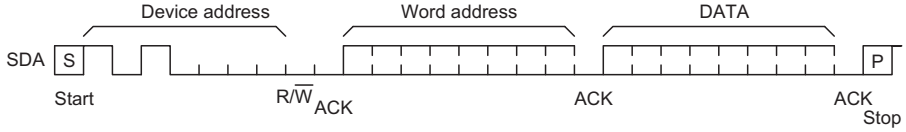
Acknowledge Polling Flow

- Read operations
  The data EEPROM supports three read operations, namely, current address read, random address read and sequential read. During read operation execution, the read/write select bit should be set to 1.

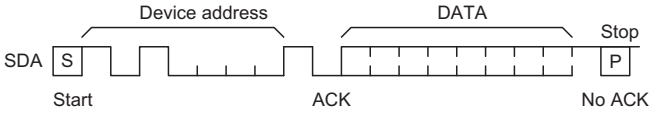
- Current address read
  The internal data word address counter maintains the last address accessed during the last read or write operation, incremented by one. This address stays valid between operations as long as the EEPROM power is maintained. The address will roll over during a read from the last byte of the last memory page to the first byte of the first page. Once the device address with the read/write select bit set to one is clocked in and acknowledged by the EEPROM, the current address data word is serially clocked out. The microcontroller should respond a No ACK - High - signal and a following stop condition.

- Random read
  A random read requires a dummy byte write sequence to load in the data word address which is then clocked in and acknowledged by the EEPROM. The microcontroller must then generate another start condition. The microcontroller now initiates a current address read by sending a device address with the read/write select bit high. The EEPROM acknowledges the device address and serially clocks out the data word. The microcontroller should respond with a No ACK signal - high - followed by a stop condition.

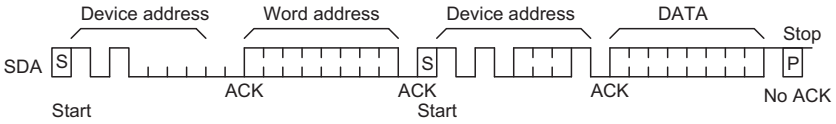
- Sequential read
  Sequential reads are initiated by either a current address read or a random address read. After the microcontroller receives a data word, it responds with an acknowledgment. As long as the EEPROM receives an acknowledgment, it will continue to increment the data word address and serially clock out sequential data words. When the memory address limit is reached, the data word address will roll over and the sequential read continues. The sequential read operation is terminated when the microcontroller responds with a No ACK signal - high - followed by a stop condition.



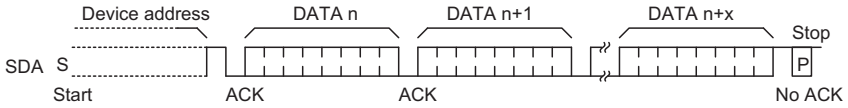
Byte Write Timing



Current Read Timing

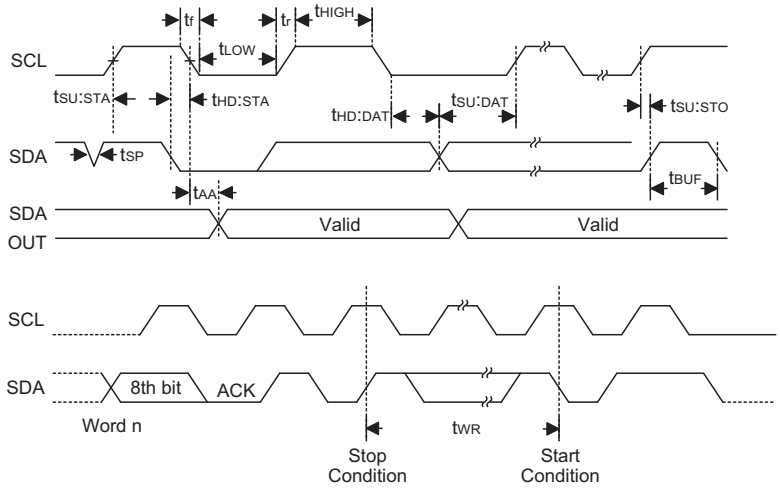


Random Read Timing



Sequential Read Timing

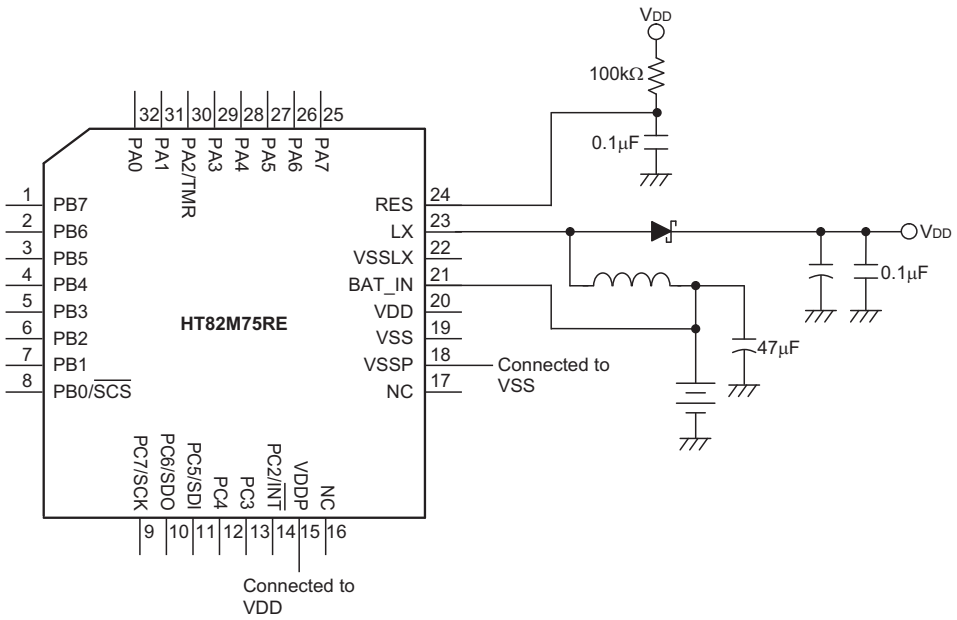
Timing Diagrams



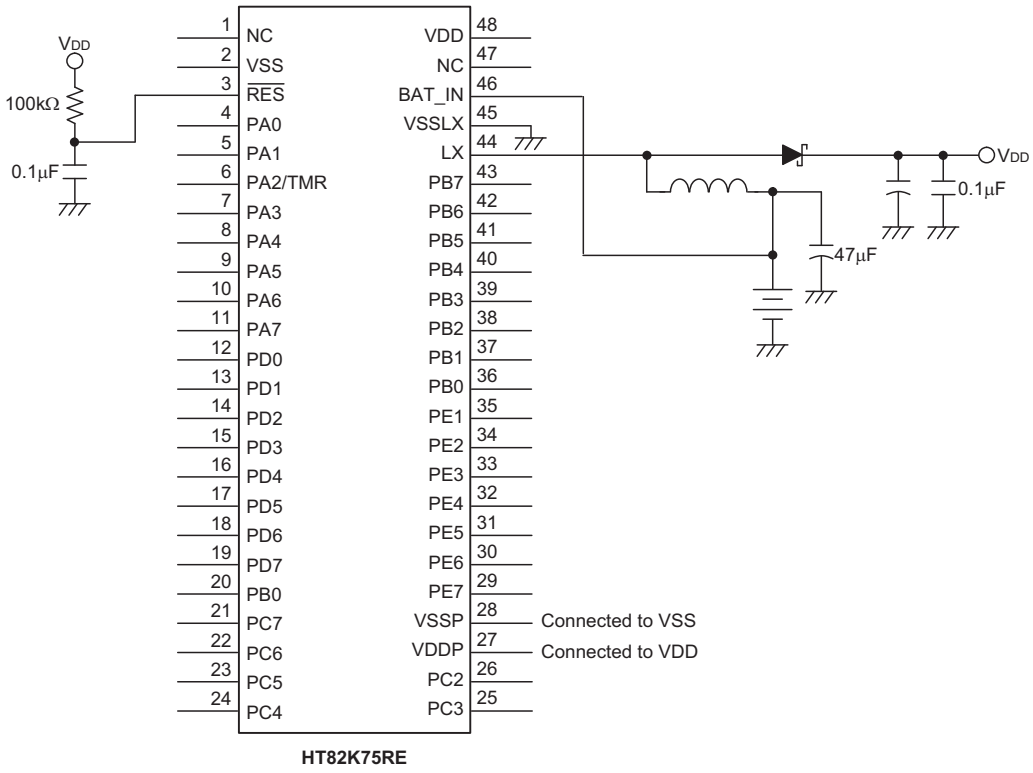
Note: The write cycle time t_{WR} is the time from a valid stop condition of a write sequence to the end of the valid start condition of sequential command.

Application Circuits with EEPROM Data Memory

For 32-pin QFN Application Circuit



For 48-pin SSOP Application Circuit



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](4)	Read ROM code (locate by TBLP and TBHP) to data memory and TBLH	2Note	None
TABRDC [m](5)	Read ROM code (current page) to data memory and TBLH	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.
4. Configuration option ″TBHP option″ is enabled
5. Configuration option ″TBHP option″ is disabled

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 ← ACC – [m] – $\overline{C}$

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

SBCM A,[m] Subtract Data Memory from ACC with Carry and result in Data Memory

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

Operation [m] ← ACC – [m] – $\overline{C}$

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

SDZ [m] Skip if decrement Data Memory is 0

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

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

Affected flag(s) None

SDZA [m] Skip if decrement Data Memory is zero with result in ACC

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

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

Affected flag(s) None

SET [m] Set Data Memory

Description Each bit of the specified Data Memory is set to 1.

Operation [m] ← FFH

Affected flag(s) None

SET [m].i Set bit of Data Memory

Description Bit i of the specified Data Memory is set to 1.

Operation [m].i ← 1

Affected flag(s) None

SIZ [m] Skip if increment Data Memory is 0

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

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

Affected flag(s) None

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

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

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

Affected flag(s) None

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

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

Operation Skip if [m].i $\neq$ 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

TABRDC [m]	Move the ROM code (locate by TBLP and TBHP) to TBLH and data memory (ROM code TBHP is enabled)
Description	The low byte of ROM code addressed by the table pointers (TBLP and TBHP) is moved to the specified data memory and the high byte transferred to TBLH directly.
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

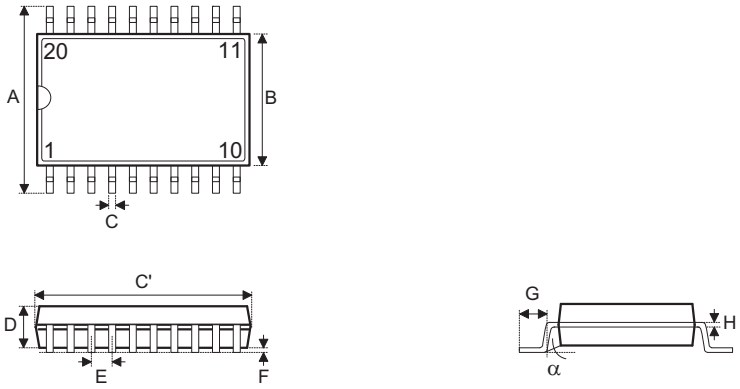
XOR A,[m]	Logical XOR Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical XOR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″XOR″ [m]
Affected flag(s)	Z

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

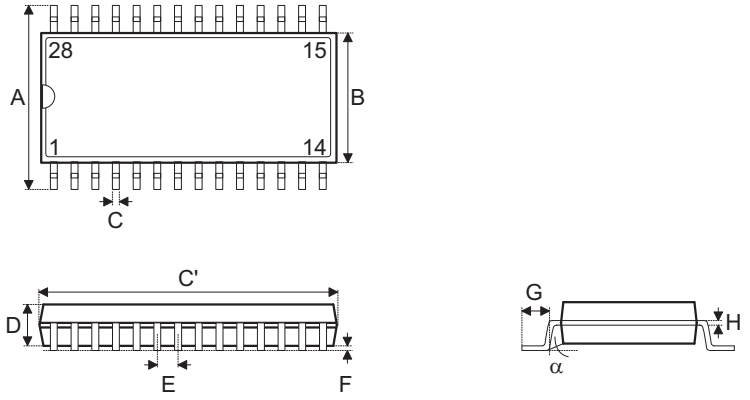
20-pin SSOP (150mil) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.228	—	0.244
B	0.150	—	0.158
C	0.008	—	0.012
C′	0.335	—	0.347
D	0.049	—	0.065
E	—	0.025	—
F	0.004	—	0.010
G	0.015	—	0.050
H	0.007	—	0.010
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	5.79	—	6.20
B	3.81	—	4.01
C	0.20	—	0.30
C′	8.51	—	8.81
D	1.24	—	1.65
E	—	0.64	—
F	0.10	—	0.25
G	0.38	—	1.27
H	0.18	—	0.25
α	0°	—	8°

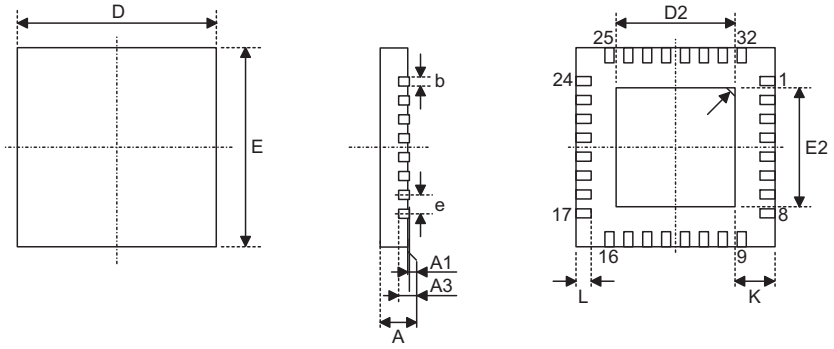
28-pin SSOP (150mil) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.228	—	0.244
B	0.150	—	0.157
C	0.008	—	0.012
C′	0.386	—	0.394
D	0.054	—	0.060
E	—	0.025	—
F	0.004	—	0.010
G	0.022	—	0.028
H	0.007	—	0.010
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	5.79	—	6.20
B	3.81	—	3.99
C	0.20	—	0.30
C′	9.80	—	10.01
D	1.37	—	1.52
E	—	0.64	—
F	0.10	—	0.25
G	0.56	—	0.71
H	0.18	—	0.25
α	0°	—	8°

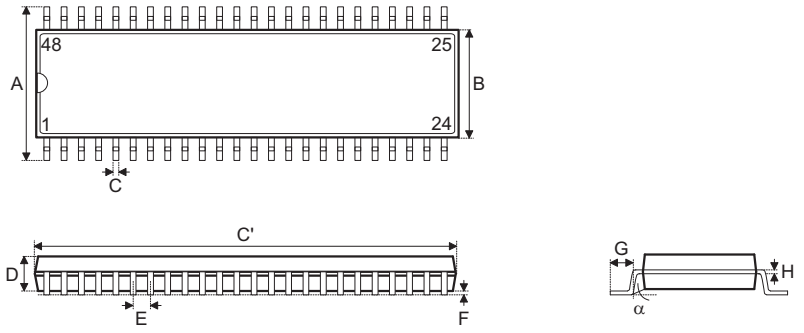
SAW Type 32-pin (5mm×5mm) QFN Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.028	—	0.031
A1	0.000	—	0.002
A3	—	0.008	—
b	0.007	—	0.012
D	—	0.197	—
E	—	0.197	—
e	—	0.020	—
D2	0.049	—	0.128
E2	0.049	—	0.128
L	0.012	—	0.020
K	—	—	—

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	0.70	—	0.80
A1	0.00	—	0.05
A3	—	0.20	—
b	0.18	—	0.30
D	—	5.00	—
E	—	5.00	—
e	—	0.50	—
D2	1.25	—	3.25
E2	1.25	—	3.25
L	0.30	—	0.50
K	—	—	—

48-pin SSOP (300mil) Outline Dimensions

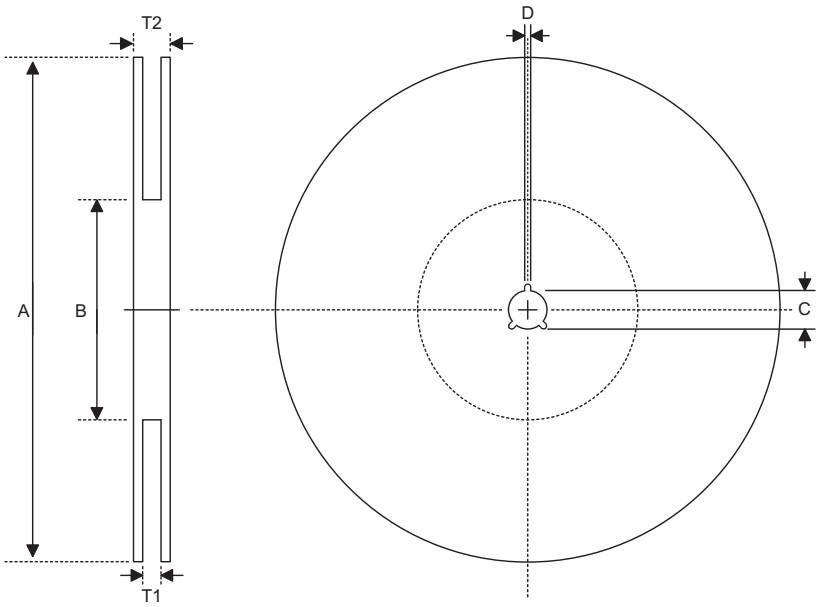


Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.395	—	0.420
B	0.291	—	0.299
C	0.008	—	0.012
C′	0.613	—	0.637
D	0.085	—	0.099
E	—	0.025	—
F	0.004	—	0.010
G	0.025	—	0.035
H	0.004	—	0.012
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	10.03	—	10.67
B	7.39	—	7.59
C	0.20	—	0.30
C′	15.57	—	16.18
D	2.16	—	2.51
E	—	0.64	—
F	0.10	—	0.25
G	0.64	—	0.89
H	0.10	—	0.30
α	0°	—	8°

Product Tape and Reel Specifications

Reel Dimensions



SSOP 20S (150mil)

Symbol	Description	Dimensions in mm
A	Reel Outer Diameter	330.0±1.0
B	Reel Inner Diameter	100.0±1.5
C	Spindle Hole Diameter	13.0 $^{+0.5/-0.2}$
D	Key Slit Width	2.0±0.5
T1	Space Between Flange	16.8 $^{+0.3/-0.2}$
T2	Reel Thickness	22.2±0.2

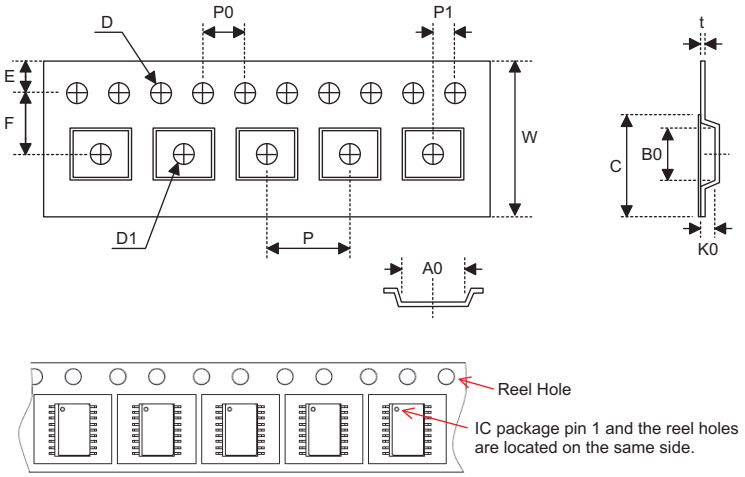
SSOP 28S (150mil)

Symbol	Description	Dimensions in mm
A	Reel Outer Diameter	330.0±1.0
B	Reel Inner Diameter	100.0±1.5
C	Spindle Hole Diameter	13.0 $^{+0.5/-0.2}$
D	Key Slit Width	2.0±0.5
T1	Space Between Flange	16.8 $^{+0.3/-0.2}$
T2	Reel Thickness	22.2±0.2

SSOP 48W

Symbol	Description	Dimensions in mm
A	Reel Outer Diameter	330.0±1.0
B	Reel Inner Diameter	100.0±0.1
C	Spindle Hole Diameter	13.0 $^{+0.5/-0.2}$
D	Key Slit Width	2.0±0.5
T1	Space Between Flange	32.2 $^{+0.3/-0.2}$
T2	Reel Thickness	38.2±0.2

Carrier Tape Dimensions



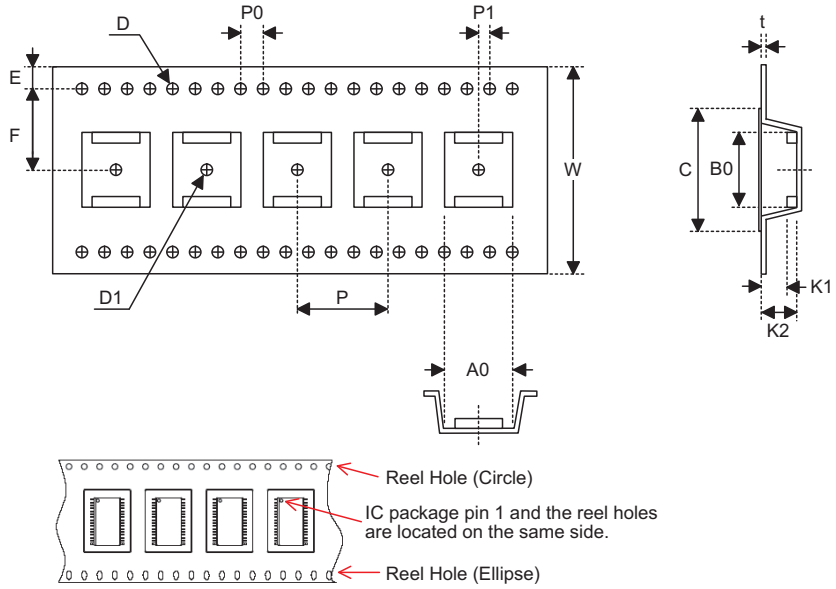
SSOP 20S (150mil)

Symbol	Description	Dimensions in mm
W	Carrier Tape Width	16.0 +0.3/-0.1
P	Cavity Pitch	8.0±0.1
E	Perforation Position	1.75±0.10
F	Cavity to Perforation (Width Direction)	7.5±0.1
D	Perforation Diameter	1.5 +0.1/-0.0
D1	Cavity Hole Diameter	1.50 +0.25/-0.00
P0	Perforation Pitch	4.0±0.1
P1	Cavity to Perforation (Length Direction)	2.0±0.1
A0	Cavity Length	6.5±0.1
B0	Cavity Width	9.0±0.1
K0	Cavity Depth	2.3±0.1
t	Carrier Tape Thickness	0.30±0.05
C	Cover Tape Width	13.3±0.1

SSOP 28S (150mil)

Symbol	Description	Dimensions in mm
W	Carrier Tape Width	16.0±0.3
P	Cavity Pitch	8.0±0.1
E	Perforation Position	1.75±0.1
F	Cavity to Perforation (Width Direction)	7.5±0.1
D	Perforation Diameter	1.55 +0.10/-0.00
D1	Cavity Hole Diameter	1.50 +0.25/-0.00
P0	Perforation Pitch	4.0±0.1
P1	Cavity to Perforation (Length Direction)	2.0±0.1
A0	Cavity Length	6.5±0.1
B0	Cavity Width	10.3±0.1
K0	Cavity Depth	2.1±0.1
t	Carrier Tape Thickness	0.30±0.05
C	Cover Tape Width	13.3±0.1

Carrier Tape Dimensions



SSOP 48W

Symbol	Description	Dimensions in mm
W	Carrier Tape Width	32.0±0.3
P	Cavity Pitch	16.0±0.1
E	Perforation Position	1.75±0.10
F	Cavity to Perforation (Width Direction)	14.2±0.1
D	Perforation Diameter	2 Min.
D1	Cavity Hole Diameter	1.50 $^{+0.25/-0.00}$
P0	Perforation Pitch	4.0±0.1
P1	Cavity to Perforation (Length Direction)	2.0±0.1
A0	Cavity Length	12.0±0.1
B0	Cavity Width	16.2±0.1
K1	Cavity Depth	2.4±0.1
K2	Cavity Depth	3.2±0.1
t	Carrier Tape Thickness	0.35±0.05
C	Cover Tape Width	25.5±0.1

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 Inc. (Shenzhen Sales Office)
5F, Unit A, Productivity Building, No.5 Gaoxin M 2nd Road, Nanshan District, Shenzhen, China 518057
Tel: 86-755-8616-9908, 86-755-8616-9308
Fax: 86-755-8616-9722

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