HT48F50E

I/O Flash Type MCU with EEPROM

Technical Document

- Tools Information
- FAQs
- Application Note
  - HA0075E MCU Reset and Oscillator Circuits Application Note

Features

- Operating voltage:
  f_{SYS}=4MHz: 2.2V~5.5V
  f_{SYS}=8MHz: 3.3V~5.5V
  f_{SYS}=12MHz: 4.5V~5.5V
- Multi-programmable Flash Type Program Memory: 4096×15
- EEPROM Data Memory:256×8
- RAM Data Memory:160×8
- Max. 33 bidirectional I/O with Pull-high Options
- External Interrupt Input shared with an I/O line
- An 8-bit programmable Timer/Event Counter with overflow interrupt and 8-stage perscaler
- A 16-bit programmable Timer/Event Counter with overflow interrupt
- Two Timer External Inputs
- Crystal and RC System Oscillator
- Watchdog Timer Function
- PFD/Buzzer Driver Outputs
- Power Down and Wake-up Feature for Power Saving Operation
- Up to 0.5μs instruction cycle with 8MHz system clock at V_{DD}=5V
- 6-level subroutine nesting
- Bit Manipulation Instructions
- Table Read Function
- 63 Powerful Instructions
- All Instructions executed in 1 or 2 Machine Cycles
- Low Voltage Reset Function
- Programming Interface
- Full Suite of Supported Hardware and Software Tools Available
- 28-pin SKDIP/SOP/SSOP, 48-pin SSOP package

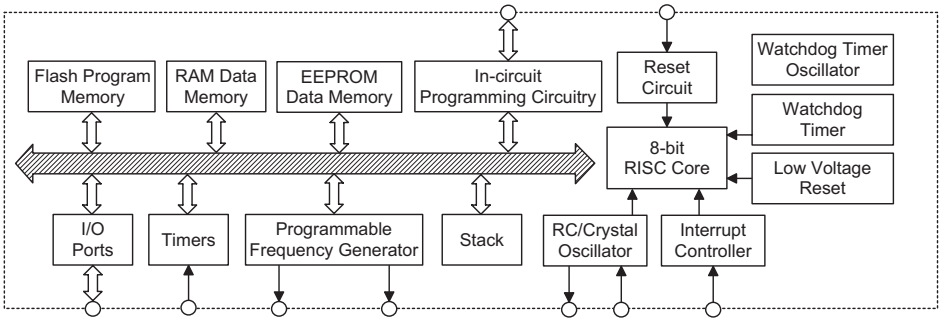
General Description

The HT48F50E is an 8-bit high-performance, RISC architecture microcontroller devices specifically designed for multiple I/O control product applications. Device flexibility is enhanced with their internal special features such as power-down and wake-up functions, oscillator options, buzzer driver, etc. These features combine to ensure applications require a minimum of external components and therefore reduce overall product costs.

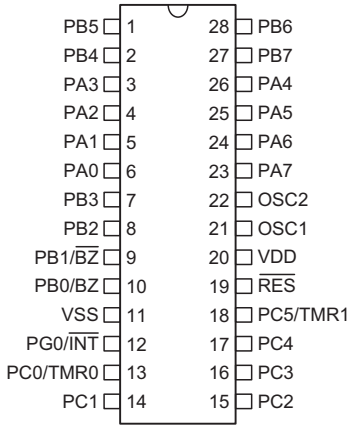
Having the advantages of low-power consumption, high-performance, I/O flexibility as well as low-cost, these devices have the versatility to suit a wide range of application possibilities such as industrial control, consumer products, subsystem controllers, etc.

The device utilises a Flash type Program Memory, and therefore have multi-programmable capabilities offering the advantages of easy and efficient program updates. The non-volatile internal EEPROM also offers the capability of storing information such as product part numbers, calibration data and other specific product information. etc. The device is fully supported by the Holtek range of fully functional development and programming tools, providing a means for fast and efficient product development cycles.

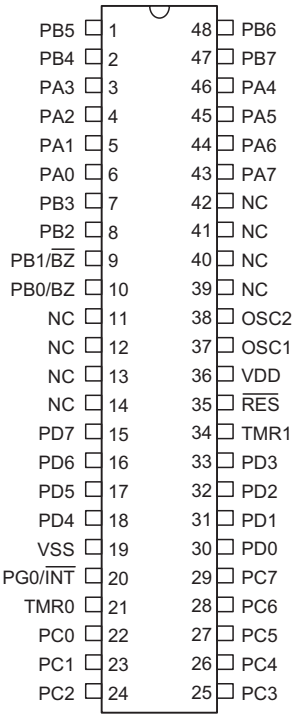
Block Diagram



Pin Assignment



HT48F50E
28 SKDIP-A/SOP-A/SSOP-A



HT48F50E
48 SSOP-A

Pin Description

Pin Name	I/O	Configuration Option	Description
PA0~PA7	I/O	Pull-high* Wake-up CMOS/Schmitt Trigger Input	Bidirectional 8-bit input/output port. Each pin can be configured as a wake-up input by configuration option. Software instructions determine if the pin is a CMOS output or input. Configuration options determine if all pins on this port have pull-high resistors and if the inputs are Schmitt Trigger or non-Schmitt Trigger.
PB0/$\overline{BZ}$ PB1/$\overline{BZ}$ PB2~PB7	I/O	Pull-high* I/O or BZ/$\overline{BZ}$	Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option determines if all pins on this port have pull-high resistors. Pins PB0 and PB1 are pin-shared with BZ and $\overline{BZ}$, respectively.
PC0~PC7	I/O	Pull-high*	Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. A configuration option determines if all pins on this port have pull-high resistors.
PD0~PD7	I/O	Pull-high*	Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. A configuration option determines if all pins on this port have pull-high resistors.
PG0/$\overline{INT}$	I/O	Pull-high*	Bidirectional 1-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. A configuration option determines if the pin has a pull-high resistor. PG0 is pin-shared with external interrupt pin $\overline{INT}$.
TMR0	I	—	Timer/Event Counter 0 external Schmitt trigger input pin (without pull-high resistor).
TMR1	I	—	Timer/Event Counter 1 external Schmitt trigger input pin (without pull-high resistor).
OSC1 OSC2	I O	Crystal or RC	OSC1, OSC2 are connected to an external RC network or external crystal, determined by configuration option, for the internal system clock. If the RC system clock option is selected, pin OSC2 can be used to measure the system clock at 1/4 frequency.
$\overline{RES}$	I	—	Schmitt Trigger reset input. Active low.
VDD	—	—	Positive power supply
VSS	—	—	Negative power supply, ground

Note: 1. Each pin on PA can be programmed through a configuration option to have a wake-up function.
2. Individual pins cannot be selected to have pull-high resistors. If the pull-high configuration is chosen for a particular port, then all input pins on this port will be connected to pull-high resistors.
3. Configuration options determine if the pins on PA are CMOS inputs or Schmitt trigger inputs.
4. Pins PC5~PC7 and PD0~7 only exist on the 48-pin package. On the 28-pin package, these pins are not available.
5. Unbounded pins should be setup as outputs or as inputs with pull-high resistors to conserve power.

Absolute Maximum Ratings

Supply Voltage ..... V_{SS}–0.3V to V_{SS}+6.0V
Input Voltage ..... V_{SS}–0.3V to V_{DD}+0.3V
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_{DD}	Operating Voltage	—	f_{SYS}=4MHz	2.2	—	5.5	V
		—	f_{SYS}=8MHz	3.3	—	5.5	V
		—	f_{SYS}=12MHz	4.5	—	5.5	V
I_{DD1}	Operating Current (Crystal OSC)	3V	No load, f_{SYS}=4MHz	—	0.6	1.5	mA
		5V		—	2	4	mA
I_{DD2}	Operating Current (RC OSC)	3V	No load, f_{SYS}=4MHz	—	0.8	1.5	mA
		5V		—	2.5	4	mA
I_{DD3}	Operating Current (Crystal OSC, RC OSC)	5V	No load, f_{SYS}=8MHz	—	4	8	mA
I_{STB1}	Standby Current (WDT Enabled)	3V	No load, system HALT	—	—	5	μA
		5V		—	—	10	μA
I_{STB2}	Standby Current (WDT Disabled)	3V	No load, system HALT	—	—	1	μA
		5V		—	—	2	μA
V_{IL1}	Input Low Voltage for I/O Ports	—	—	0	—	$0.3V_{DD}$	V
V_{IH1}	Input High Voltage for I/O Ports	—	—	$0.7V_{DD}$	—	V_{DD}	V
V_{IL2}	Input Low Voltage ($\overline{RES}$)	—	—	0	—	$0.4V_{DD}$	V
V_{IH2}	Input High Voltage ($\overline{RES}$)	—	—	$0.9V_{DD}$	—	V_{DD}	V
V_{LVR}	Low Voltage Reset	—	LVR enabled	2.7	3.0	3.3	V
I_{OL}	I/O Port Sink Current	3V	V_{OL}=$0.1V_{DD}$	4	8	—	mA
		5V		10	20	—	mA
I_{OH}	I/O Port Source Current	3V	V_{OH}=$0.9V_{DD}$	–2	–4	—	mA
		5V		–5	–10	—	mA
R_{PH}	Pull-high Resistance	3V	—	20	60	100	kΩ
		5V	—	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 (RC OSC, Crystal OSC)	—	2.2V~5.5V	400	—	4000	kHz
		—	3.3V~5.5V	400	—	8000	kHz
		—	4.5V~5.5V	400	—	12000	kHz
f_{TIMER}	Timer I/P Frequency (TMR)	—	2.2V~5.5V	0	—	4000	kHz
		—	3.3V~5.5V	0	—	8000	kHz
		—	4.5V~5.5V	0	—	12000	kHz
t_{WDTOSC}	Watchdog Oscillator Period	3V	—	45	90	180	μs
		5V	—	32	65	130	μs

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
t_{WDT1}	Watchdog Time-out Period (WDT Internal Clock Source)	3V	Without WDT prescaler	11	23	46	ms
		5V		8	17	33	ms
t_{WDT2}	Watchdog Time-out Period (Instruction Clock Source)	—	Without WDT prescaler	—	1024	—	*t_{SYS}
t_{RES}	External Reset Low Pulse Width	—	—	1	—	—	μs
t_{SST}	System Start-up Timer Period	—	Wake-up from HALT	—	1024	—	*t_{SYS}
t_{LVR}	Low Voltage Reset Time	—	—	1	—	2	ms
t_{INT}	Interrupt Pulse Width	—	—	1	—	—	μs

Note: *t_{SYS}=1/f_{SYS}

EEPROM – A.C. Characteristics

Ta=25°C

Symbol	Parameter	V_{CC}=5V±10%		V_{CC}=2.2V±10%		Unit
		Min.	Max.	Min.	Max.	
f_{SK}	Clock Frequency	0	2	0	1	MHz
t_{SKH}	SK High Time	250	—	500	—	ns
t_{SKL}	SK Low Time	250	—	500	—	ns
t_{CSS}	CS Setup Time	50	—	100	—	ns
t_{CSH}	CS Hold Time	0	—	0	—	ns
t_{CDS}	CS Deselect Time	250	—	250	—	ns
t_{DIS}	DI Setup Time	100	—	200	—	ns
t_{DIH}	DI Hold Time	100	—	200	—	ns
t_{PD1}	DO Delay to ″1″	—	250	—	500	ns
t_{PD0}	DO Delay to ″0″	—	250	—	500	ns
t_{SV}	Status Valid Time	—	250	—	250	ns
t_{PR}	Write Cycle Time	—	5	—	5	ms

System Architecture

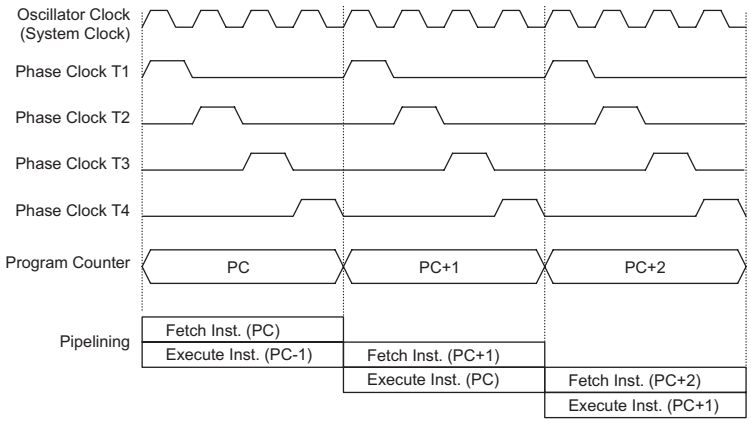
A key factor in the high-performance features of the Holtek range of microcontrollers is attributed to the internal system architecture. The range of devices take advantage of the usual features found within RISC microcontrollers providing increased speed of operation and enhanced performance. The pipelining scheme is implemented in such a way that instruction fetching and instruction execution are overlapped, hence instructions are effectively executed in one cycle, with the exception of branch or call instructions. An 8-bit wide ALU is used in practically all operations of the instruction set. It carries out arithmetic operations, logic operations, rotation, increment, decrement, branch decisions, etc. The internal data path is simplified by moving data through the Accumulator and the ALU. Certain internal registers are implemented in the Data Memory and can be directly or indirectly addressed. The simple addressing methods of these registers along with additional architectural features ensure that a minimum of external components is required to provide a functional I/O control system with maximum reliability and flexibility.

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.

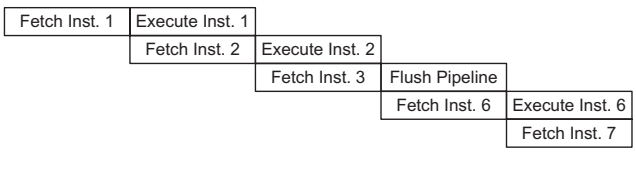
When the RC oscillator is used, OSC2 is freed for use as a T1 phase clock synchronizing pin. This T1 phase clock has a frequency of $f_{SYS}/4$ with a 1:3 high/low duty cycle.

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



System Clocking and Pipelining

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



Instruction Fetching

Program Counter

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

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

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

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

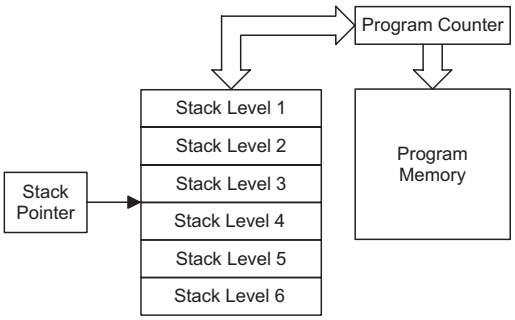
This is a special part of the memory which is used to save the contents of the Program Counter only. The stack is organized into 6 levels and is neither part of the data nor part of the program space, and can neither be read from nor written to. The activated level is indexed by the Stack Pointer, SP, which can also neither be read from nor written to. At a subroutine call or interrupt acknowledge signal, the contents of the Program Counter are pushed onto the stack. At the end of a subroutine or an interrupt routine, signaled by a return instruction, RET or 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
External Interrupt	0	0	0	0	0	0	0	0	0	1	0	0
Timer/Event Counter 0 Overflow	0	0	0	0	0	0	0	0	1	0	0	0
Timer/Event Counter 1 Overflow	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
@7~@0: PCL bits
#11~#0: Instruction code address 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

Flash Program Memory

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

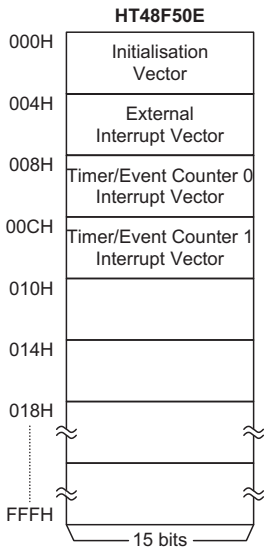
Organization

The Program Memory has a capacity of 4K by 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 a separate table pointer register.

Special Vectors

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

- Location 000H
  This vector is reserved for use by the device reset for program initialisation. After a device reset is initiated, the program will jump to this location and begin execution.
- Location 004H
  This vector is used by the external interrupt. If the external interrupt pin on the device goes low, the program will jump to this location and begin execution if the external interrupt is enabled and the stack is not full.
- Location 008H
  This internal vector is used by the Timer/Event Counter 0. If a counter overflow occurs, the program will jump to this location and begin execution if the timer/event counter interrupt is enabled and the stack is not full.
- Location 00CH
  This internal vector is used by the Timer/Event Counter 1. If a counter overflow occurs, the program will jump to this location and begin execution if the timer/event counter interrupt is enabled and the stack is not full.



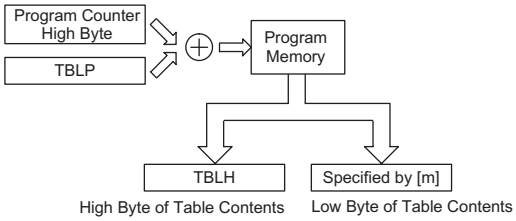
Program Memory Structure

Look-up Table

Any location within the Program Memory can be defined as a look-up table where programmers can store fixed data. To use the look-up table, the table pointer must first be setup by placing the lower order address of the look up data to be retrieved in the table pointer register, TBLP. This register defines the lower 8-bit address of the look-up table.

After setting up the table pointer, the table data can be retrieved from the current Program Memory page or last Program Memory page using the "TABRDC[m]" or "TABRDL [m]" instructions, respectively. When these instructions are executed, the lower order table byte from the Program Memory will be transferred to the user defined Data Memory register [m] as specified in the instruction. The higher order table data byte from the Program Memory will be transferred to the TBLH special register. Any unused bits in this transferred higher order byte will have uncertain values.

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



Look-up Table

Table Program Example

The following example shows how the table pointer and table data is defined and retrieved from the HT48F50E device. This example uses raw table data located in the last page which is stored there using the ORG statement. The value at this ORG statement is "0F00H" which refers to the start address of the last page within the 4K Program Memory of the microcontroller. The table pointer is setup here to have an initial value of "06H". This will ensure that the first data read from the data table will be at the Program Memory address "0F06H" 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.

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

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

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

tabrdl      tempreg1    ; transfers value in table referenced by table pointer
                        ; to tempreg1
                        ; data at prog. memory address "0F06H" 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 "0F05H" transferred to
                        ; tempreg2 and TBLH
                        ; in this example the data "1AH" is transferred to
                        ; tempreg1 and data "0FH" to register tempreg2
      :
      :

org   300h              ; sets initial address of HT48F50E last page

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

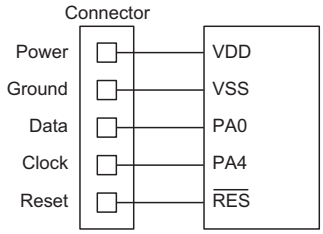
Because the TBLH register is a read-only register and cannot be restored, care should be taken to ensure its protection if both the main routine and Interrupt Service Routine use table read instructions. If using the table read instructions, the Interrupt Service Routines may change the value of the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

In Circuit Programming

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

Pin Name	Function
PA0	Serial data input/output
PA4	Serial clock
$\overline{RES}$	Device reset
VDD	Power supply
VSS	Ground

The Program Memory and EEPROM memory can both be programmed serially in-circuit using a 5-wire interface. Data is downloaded and uploaded serially on a single pin with an additional line for the clock. Two additional lines are required for the power supply and one line for the reset. The technical details regarding the in-circuit programming of the devices are beyond the scope of this document and will be supplied in supplementary literature.



In-circuit Programming Interface

RAM Data Memory

The RAM Data Memory is a volatile area of 8-bit wide RAM internal memory and is the location where temporary information is stored. Divided into two sections, the first of these is an area of RAM where special function registers are located. These registers have fixed locations and are necessary for correct operation of the device. Many of these registers can be read from and written to directly under program control, however, some remain protected from user manipulation. The second area of RAM Data Memory is reserved for general purpose use. All locations within this area are read and write accessible under program control.

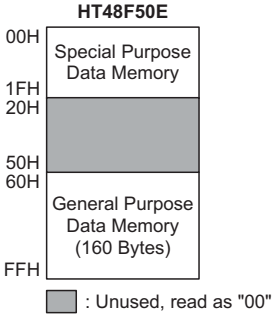
Organization

The RAM Data Memory is subdivided into two banks, known as Bank 0 and Bank 1, all of which are implemented in 8-bit wide RAM. Most of the RAM Data Memory is located in Bank 0 which is also subdivided into two sections, the Special Purpose Data Memory and the General Purpose Data Memory. The start address of the RAM Data Memory for all devices is the address ″00H″, and the last Data Memory address is ″FFH″. Registers which are common to all microcontrollers, such as ACC, PCL, etc., have the same Data Memory address.

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
@7~@0: Table Pointer TBLP bits



Bank 0 RAM Data Memory Structure

Bank 1 of the RAM Data Memory contains only one special function register, known as the EECR register, which is used for EEPROM control and located at address ″40H″ for all devices.

Bank 1 RAM Data Memory Structure

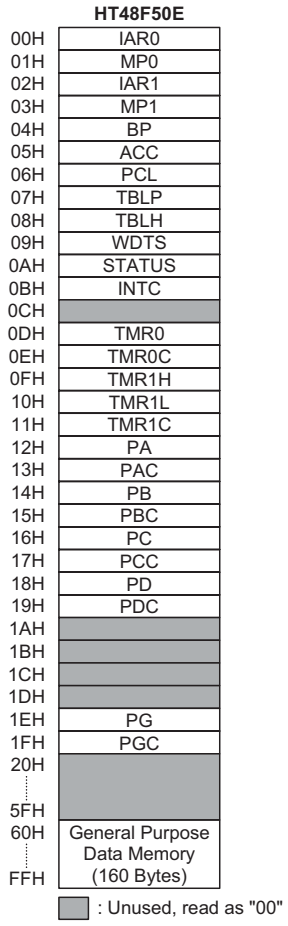
Note: Most of the RAM Data Memory bits can be directly manipulated using the ″SET [m].i″ and ″CLR [m].i″ instructions with the exception of a few dedicated bits. The RAM Data Memory can also be accessed through the Memory Pointer registers MP0 and MP1.

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



Special Purpose Data Memory Structure

Special Function Registers

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

Indirect Addressing Register – IAR0, IAR1

The Indirect Addressing Registers, IAR0 and IAR1, although having their locations in normal RAM register space, do not actually physically exist as normal registers. The method of indirect addressing for RAM data manipulation uses these Indirect Addressing Registers and Memory Pointers, in contrast to direct memory addressing, where the actual memory address is specified. Actions on the IAR0 and IAR1 registers will result in no actual read or write operation to these registers but rather to the memory location specified by their corresponding Memory Pointer, MP0 or MP1. Acting as a pair, IAR0 and MP0 can together only access data from Bank 0, while the IAR1 and MP1 register pair can access data from both Bank 0 and Bank 1. 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 this device, two 8-bit Memory Pointers, known as MP0 and MP1 are provided. These Memory Pointers are physically implemented in the Data Memory and can be manipulated in the same way as normal registers providing a convenient way with which to address and track data. When any operation to the relevant Indirect Addressing Registers is carried out, the actual address that the microcontroller is directed to, is the address specified by the related Memory Pointer. MP0, together with Indirect Addressing Register, IAR0, are used to access data from Bank 0 only, while MP1 and IAR1 are used to access data from both Bank 0 and Bank 1.

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

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

start:
      mov a,04h             ; setup size of block
      mov block,a
      mov a,offset adres1; Accumulator loaded with first RAM address
      mov mp0,a             ; setup memory pointer with first RAM address

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

continue:
```

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

Bank Pointer – BP

The RAM Data Memory is divided into two Banks, known as Bank 0 and Bank 1. With the exception of the EECR register, all of the Special Purpose Registers and General Purpose Registers are contained in Bank 0. Bank 1 contains only one register, which is the EEPROM Control Register, known as EECR. Selecting the required Data Memory area is achieved using the Bank Pointer. If data in Bank 0 is to be accessed, then the BP register must be loaded with the value "00", while if data in Bank 1 is to be accessed, then the BP register must be loaded with the value "01".

Using Memory Pointer MP0 and Indirect Addressing Register IAR0 will always access data from Bank 0, irrespective of the value of the Bank Pointer. The EECR register is located at memory location 40H in Bank 1 and can only be accessed indirectly using memory pointer MP1 and the indirect addressing register, IAR1, after the BP register has first been loaded with the value "01". Data can only be read from or written to the EEPROM via this register.

The Data Memory is initialised to Bank 0 after a reset, except for the WDT time-out reset in the Power Down Mode, in which case, the Data Memory bank remains unaffected. It should be noted that Special Function Data Memory is not affected by the bank selection, which means that the Special Function Registers can be accessed from within either Bank 0 or Bank 1. Directly addressing the Data Memory will always result in Bank 0 being accessed irrespective of the value of the Bank Pointer.

Accumulator – ACC

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

Program Counter Low Register – PCL

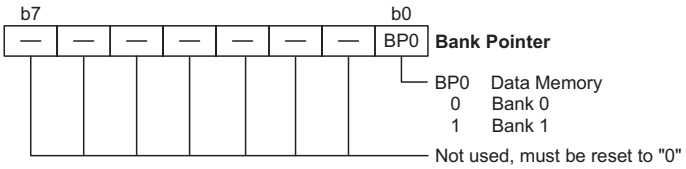
To provide additional program control functions, the low byte of the Program Counter is made accessible to programmers by locating it within the Special Purpose area of the Data Memory. By manipulating this register, direct jumps to other program locations are easily implemented. Loading a value directly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

Look-up Table Registers – TBLP, TBLH

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

Watchdog Timer Register – WDTS

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



Status Register – STATUS

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

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

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

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

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

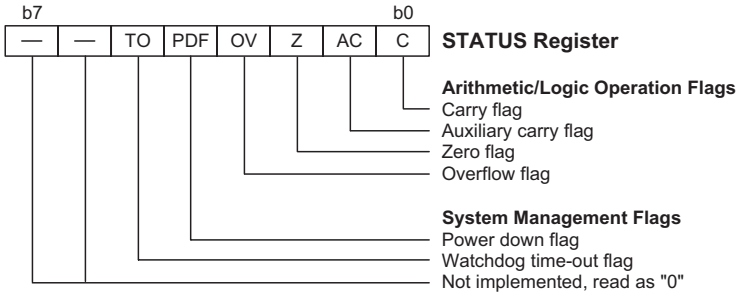
Interrupt Control Register – INTC

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

Note: In situations where other interrupts may require servicing within present interrupt service routines, the EMI bit can be manually set by the program after the present interrupt service routine has been entered.

Timer/Event Counter Registers

This device contains an 8-bit Timer/Event Counter and a 16-bit Timter/Event counter, which have the associated registers known as TMR0, TMR1H and TMR1L, and are the locations where the timers′s 8-bit and 16-bit values are located. The associated control registers, known as TMR0C and TMR1C, contains the setup information for these two timers.



Status Register

Input/Output Ports and Control Registers

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

EEPROM Control Register – EECR

This register is used to control all operations to and from the EEPROM Data Memory. As the EEPROM Data Memory is not mapped like the other memory types, all data to and from the EEPROM must be made through this register. The EECR register is located in Bank 1 of the Data Memory, so before use the Bank Pointer must be setup to a value of "1". The EECR register can only be read and written to indirectly using the MP1 address pointer.

EEPROM Data Memory

One of the special features within all these devices is their internal EEPROM Data Memory. EEPROM, which stands for Electrically Erasable Programmable Read Only Memory, is by its nature a non-volatile form of memory, with data retention even when its power supply is removed. By incorporating this kind of data memory a whole new host of application possibilities are made available to the designer. The availability of EEPROM storage allows information such as product identification numbers, calibration values, specific user data, system setup data or other product information to be stored directly within the product microcontroller.

EEPROM Data Memory Structure

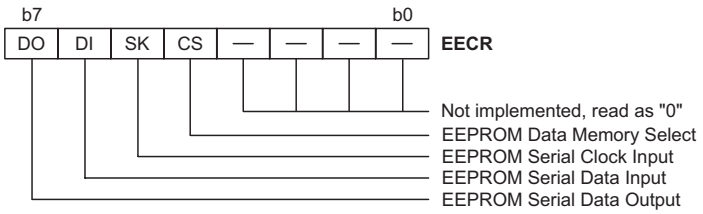
The internal EEPROM Data Memory has a capacity of 256×8 bits. Unlike the Program Memory and RAM Data Memory, the EEPROM Data Memory is not directly mapped and is therefore not directly accessible in the same way as the other types of memory. Instead it has to be accessed indirectly through the EEPROM Control Register.

Accessing the EEPROM Data Memory

The EEPROM Data Memory is accessed using a set of seven instructions. These instructions control all functions of the EEPROM such as read, write, erase, enable etc. The internal EEPROM structure is similar to that of a standard 3-wire EEPROM, for which four pins are used for transfer of instruction, address and data information. These are the Chip Select pin, CS, Serial Clock pin, SK, Data In pin, DI and the Data Out pin, DO. All actions related to the EEPROM must be conducted through the EECR register which is located in Bank 1 of the RAM Data Memory, in which each of these four EEPROM pins is represented by a bit in the EECR register. By manipulating these four bits in the EECR register, in accordance with the accompanying timing diagrams, the microcontroller can communicate with the EEPROM and carry out the required functions, such as reading and writing data.

Bit No.	Label	EEPROM Function
0~3	—	Not implemented bit, read as "0"
4	CS	EEPROM Data Memory select
5	SK	Serial Clock: Used to clock data into and out of the EEPROM
6	DI	Data Input: Instructions, address and data information are written to the EEPROM on this pin
7	DO	Data Output: Data from the EEPROM is readout with this bit. Will be in a high-impedance condition if no data is being read.

EECR Register - Control Bit Functions



EEPROM Control Register

When reading data from the EEPROM, the data will clocked out on the rising edge of SK and appear on DO. The DO pin will normally be in a high-impedance condition unless a READ statement is being executed. When writing to the EEPROM the data must be presented first on DI and then clocked in on the rising edge of SK. After all the instruction, address and data information has been transmitted, CS should be cleared to ″0″ to terminate the instruction transmission. Note that after power on the EEPROM must be initialised as described.

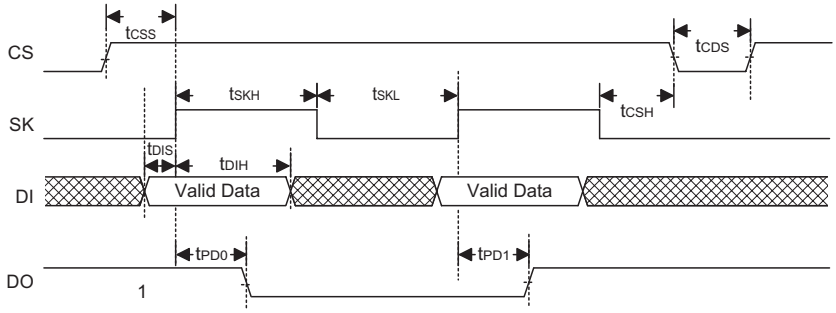
As indirect addressing is the only way to access the EECR register, all read and write operations to this register must take place using the Indirect Addressing Register, IAR1, and the Memory Pointer, MP1. Because the EECR control register is located in Bank 1 of the RAM Data Memory at location 40H, the MP1 Memory Pointer must first be set to the value 40H and the Bank Pointer set to ″1″.

EEPROM Data Memory Instruction Set

Control over the internal EEPROM, to execute functions such as read, write, disable, enable etc., is implemented through instructions of which there are a total of seven. The related instruction is transmitted to the EEPROM via the DI bit, after CS has first been set to ″1″ to enable the EEPROM and a start bit ″1″ has been transmitted. For the READ, WRITE and ERASE instructions, each of the three instructions has its own two bit related instruction code. The 9-bit address should then be transmitted. The address is transmitted in MSB first format.

For the other four instructions, ″EWEN″, ″EWDS″, ″ERAL″ and ″WRAL″, after the start bit has been transmitted a ″00″ instruction code should then follow. The 9-bit address information should then follow. The first two bits of this address is instruction dependant as shown in the table while the remaining bits have don′t care values and can be either high or low.

After any write or erase instruction is issued, the internal write function of the EEPROM will be used to write the data into the device. As this internal write operation uses the EEPROM′s own internal clock, no further instructions will be accepted by the EEPROM until the internal write function has ended. After power on and before any instruction is issued the EEPROM must be properly initialised to ensure proper operation.



Clocking Data In and Out of the EEPROM

Instruction	Function	Start Bit	Instruction Code	Address	Data
READ	Read Out Data Byte(s)	1	10	X, A7~A0	D7~D0
ERASE	Erase Single Data Byte	1	11	X, A7~A0	—
WRITE	Write Single Data Byte	1	01	X, A7~A0	D7~D0
EWEN	Erase/Write Enable	1	00	11 XXXXX XX	—
EWDS	Erase/Write Disable	1	00	00 XXXXX XX	—
ERAL	Erase All	1	00	10 XXXXX XX	—
WRAL	Write All	1	00	01 XXXXX XX	D7~D0

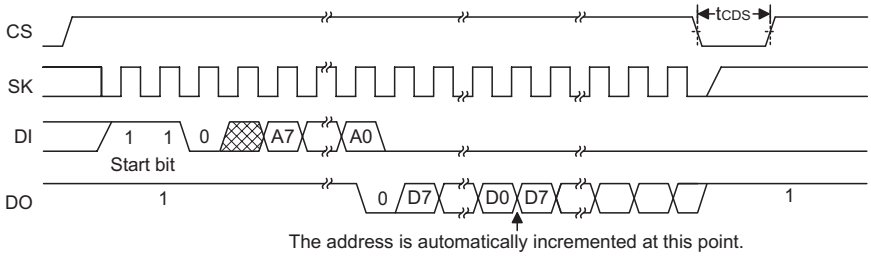
Instruction Set Summary

READ

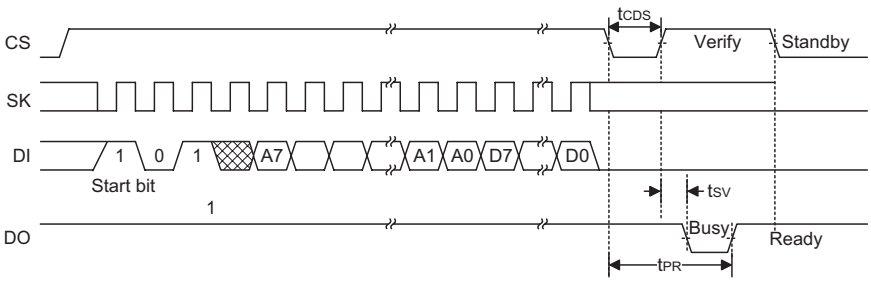
The ″READ″ instruction is used to read out one or more bytes of data from the EEPROM Data Memory. To instigate a ″READ″ instruction, the CS bit should be set high, followed by a high start bit and then the instruction code ″10″, all transmitted via the DI bit. For this device, a dummy bit must be inserted between the last bit of the instruction code and the MSB of the address. The address information should then follow with the MSB being transmitted first. After the last address bit, A0, has been transmitted, the data can be clocked out, bit D7 first, on the rising edge of the SK clock signal and can be read via the DO bit. However, a dummy ″0″ bit will first precede the reading of the first data bit, D7. After the full byte has been read out, the internal address will be automatically incremented allowing the next consecutive data byte to be read out without entering further address data. As long as the CS bit remains high, data bit D7 of the next address will automatically follow data bit D0 of the previous address with no dummy ″0″ being inserted between them. The address will keep incrementing in this way until CS returns to a low value. DO will normally be in a high impedance condition until the ″READ″ instruction is executed. Note that as the ″READ″ instruction is not affected by the condition of the ″EWEN″ or ″EWDS″ instruction, the READ command is always valid and independent of these two instructions.

WRITE

The ″WRITE″ instruction is used to write a single byte of data into the EEPROM. To instigate a WRITE instruction, the CS bit should be set high, followed by a high start bit and then the instruction code ″01″, all transmitted via the DI bit. For this device, a dummy bit must be inserted between the last bit of the instruction code and the MSB of the address. The address information should then follow with the MSB bit being transmitted first. After the last address bit, A0, has been transmitted, the data can be immediately transmitted MSB first. After all the WRITE instruction code, address and data have been transmitted, the data will be written into the EEPROM when the CS bit is cleared to zero. The EEPROM does this by executing an internal write-cycle, which will first erase and then write the previously transmitted data byte into the EEPROM. This process takes place internally using the EEPROM′s own internal clock and does not require any action from the SK clock. No further instructions can be accepted by the EEPROM until this internal write-cycle has finished. To determine when the write cycle has ended, CS should be again brought high and the DO bit polled. If DO is low this indicates that the internal write-cycle is still in progress, however, the DO bit will change to a high value when the internal write-cycle has ended. Before a ″WRITE″ instruction is transmitted an ″EWEN″ instruction must have been transmitted at some point earlier to ensure that the erase/write function of the EEPROM is enabled.



READ Timing



WRITE Timing

EWEN/EWDS

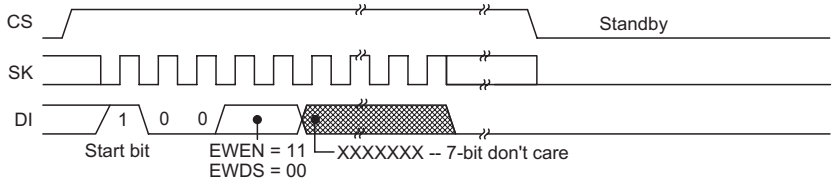
The ″EWEN″ instruction is the Erase/Write Enable instruction and the ″EWDS″ instruction is the Erase/Write Disable instruction. To instigate an ″EWEN″ or ″EWDS″ instruction, the CS bit should first be set high, followed by a high start bit and then the instruction code ″00″. For the ″EWEN″ instruction, a ″11″ should then be transmitted and for the ″EWDS″ instruction a ″00″ should be transmitted. Following on from this, 7-bits of ″don't care″ data should then be transmitted to complete the instruction. If the device is already in the Erase Write Disable mode then no write or erase operations can be executed thus protecting the internal EEPROM data. Before any write or erase instruction is executed an ″EWEN″ instruction must be issued. After the ″EWEN″ instruction is executed, the device will remain in the Erase Write Enable mode until a subsequent ″EWDS″ instruction is issued or until the device is powered down.

ERAL

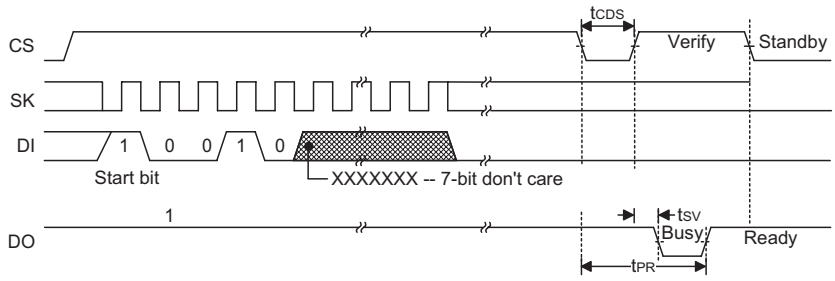
The ″ERAL″ instruction is used to erase the whole contents of the EEPROM memory. After it has been executed all the data in the EEPROM will be set to ″1″. To instigate this instruction, the CS bit should be set high, followed by a high start bit and then the instruction code ″00″. Following on from this, a ″10″ should then be transmitted. This should be followed by 7-bits of ″don't care″ data to complete the instruction. After the ″ERAL″ instruction code has been transmitted, the EEPROM data will be erased when the CS bit is cleared to zero. The EEPROM does this by executing an internal write-cycle. This process takes place internally using the EEPROM's own internal clock and does not require any action from the SK clock. No further instructions can be accepted by the EEPROM until this internal write-cycle has finished. To determine when the write cycle has ended, CS should be again brought high and the DO bit polled. If D0 is low this indicates that the internal write-cycle is still in progress, however the D0 bit will change to a high value when the internal write-cycle has ended. Before an ″ERAL″ instruction is transmitted an ″EWEN″ instruction must have been transmitted at some point earlier to ensure that the erase/write function of the EEPROM is enabled.

WRAL

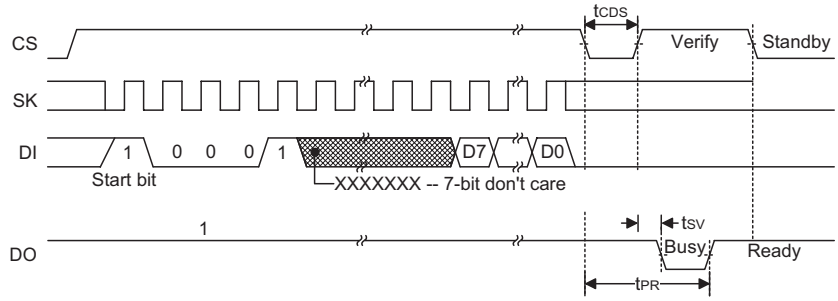
The WRAL instruction is used to write the same data into the entire EEPROM. To instigate this instruction, the CS bit should be set high, followed by a high start bit and then the instruction code ″00″. Following on from this, a ″01″ should then be transmitted. This should be followed by 7-bits of ″don't care″ data. The data information should then follow with the MSB bit being transmitted first. After the instruction code and data have been transmitted, the data will be written into the EEPROM when the CS bit is cleared to zero. The EEPROM does this by executing an internal write-cycle. This process takes place internally using the EEPROM's own internal clock and does not require any action from the SK clock. No further instructions can be accepted by the EEPROM until this internal write-cycle has finished. To determine when the write cycle has ended, CS should be again brought high and the DO bit polled. If D0 is low this indicates that the internal write-cycle is still in progress, however the D0 bit will change to a high value when the internal write-cycle has ended. Before a ″WRAL″ instruction is transmitted an ″EWEN″ instruction must have been transmitted at some point earlier to ensure that the erase/write function of the EEPROM is enabled. The WRAL instruction will automatically erase any previously written data making it unnecessary to first issue an erase instruction.



EWEN/EWDS Timing



ERAL Timing



WRAL Timing

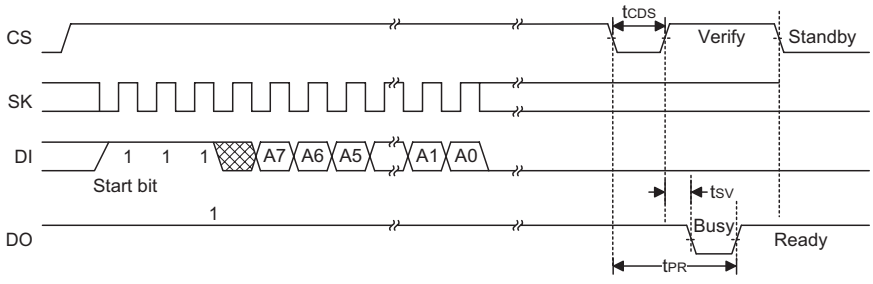
ERASE

The ″ERASE″ instruction is used to erase data at a specified address. The data at the address specified will be set to ″1″. To instigate an ″ERASE″ instruction, the CS bit should be set high, followed by a high start bit and then the instruction code ″11″, all transmitted via the DI bit. For this device, a dummy bit must be inserted between the last bit of the instruction code and the MSB of the address. The address information should then follow with the MSB bit being transmitted first. After all the ″ERASE″ instruction code and address have been transmitted, the data at the specified address will be erased when the CS bit is cleared to zero. The EEPROM does this by executing an internal write cycle which will set all data at the specified address to ″1″. This process takes place internally using the EEPROM′s own internal clock and does not require any action from the SK clock. No further instructions can be accepted by the EEPROM until the write cycle has finished. To determine when the write cycle has ended, the CS should be again brought high and the DO bit polled. If the DO bit is low this indicates that the write-cycle is still in progress, however, the DO bit will change to a high value when the write-cycle has ended. Before an ″ERASE″ instruction is transmitted, an ″EWEN″ instruction must have been transmitted at some point earlier to ensure that the erase/write function of the EEPROM is enabled.

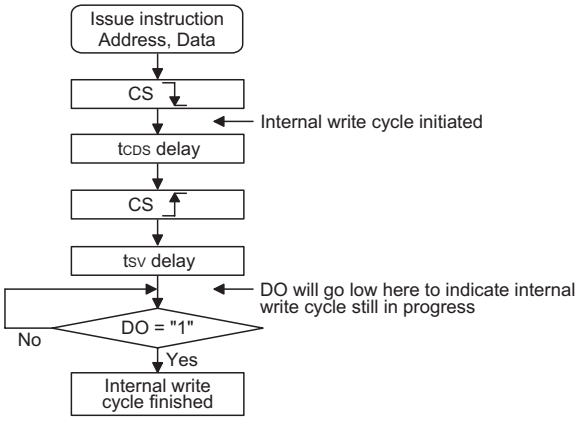
Internal Write Cycle

The write or erase instructions, ″WRITE″, ″ERASE″, ″ERAL″ or ″WRAL″ will all use the EEPROM′s internal write cycle function. As this function is completely internally timed, the SK clock is not required. As the MCU has no control over the timing of this write cycle, it must still have some way of knowing when the internal write cycle has completed. This is because, when the internal write cycle is executing, the EEPROM will not accept any further instructions from the MCU. The MCU must therefore wait until the write cycle has finished before sending any further instructions.

One way for the MCU to know when the write cycle has terminated is to poll the DO bit after the CS bit has issued a low pulse. The low going edge of this CS bit pulse will initiate the internal write cycle, when the bit is returned high the DO bit will go low to indicated that the write cycle is in progress. When the DO bit returns high this indicates that the internal write cycle has ended and that the EEPROM is ready to receive further instructions.



ERASE Timing



Internal Write Cycle Busy Polling

Initialising the EEPROM

After the MCU is powered on and if the EEPROM is to be used, it must be initialised in a specific way before any user instructions are transmitted. This is achieved by first transmitting an EWEN instruction, then by issuing a WRITE instruction to write random data to any single address in the EEPROM. The initialisation procedure can then be terminated by issuing an EWDS instruction, however at this point, if actual user data is to be imminently written to the EEPROM, this last step is optional.

The following is an example program of how this can be implemented:

```
mov   A,01h
mov   BP,A                    ; set to bank 1
mov   A,40h
mov   MP1,A                   ; set MP1 to EECR address
call  EWEN                    ; subroutine to run EWEN instructions
mov   A, 7Fh
mov   EEADDR, A
mov   A, 55h
mov   EEDATA, A
call  WRITE                   ; subroutine to run WRITE instruction
                              ; write 55h data to address 7Fh
call  EWDS                    ; optional subroutine to run EWDS instruction
```

EEPROM Program Examples

The following short programs gives examples of how to send instructions, read and write to the EEPROM. These programs can form a basis of understanding as to how the internal EEPROM memory is to be used to store and retrieve data.

Example 1 – Definitions and Sending Instructions to the EEPROM

```
_CS   EQU    IAR1.4           ; EEPROM lines setup to have a corresponding
_SK   EQU    IAR1.5           ; Bit in the Indirect Addressing Register IAR1
_DI   EQU    IAR1.6           ; EEPROM can only be indirectly addressed using
                              ; MP1
_DO   EQU    IAR1.7
_EECR EQU    40H              ; Setup address of the EEPROM control register
C_Addr_Length      EQU 8      ; Address length – 8-bits
C_Data_Length      EQU 8      ; Data length – always 8-bits
;
DATA   .SECTION at 70h 'DATA'
EE_command         DB  ?      ; Stores the read or write instruction
                              ; information
ADDR        DB     ?          ; Store write data or read data address
WR_Data            DB  ?      ; Store read or write data
COUNT       DB     ?          ; Temporary counter
;
```

```
WriteCommand:                   ; Write instruction code subroutine
      MOV   A,3                 ; Read, write and erase instructions are 3 bits
                                ; long
      MOV   COUNT,A
      SET   _SK                 ; Dummy bit transmission for 256x8 bit EEPROM Memory
                                ; capacity
      CLR   _SK                 ; Not required for 128x8 bit EEPROM Memory capacity devices
WriteCommand_0:
      CLR   _DI                 ; Prepare the transmitted bit
      SZ    EE_command.7        ; Check value of highest instruction code bit
      SET   _DI
      SET   _SK
      CLR   _SK
      CLR   C
      RLC   EE_command          ; Get next bit of instruction code
      SDZ   COUNT               ; Check if last bit has been transmitted
      JMP   WriteCommand_0
      CLR   _DI
      RET
```

Example 2 – Transmitting an Address to the EEPROM

```
WriteAddr:                      ; Write address subroutine
      MOV   A,C_Addr_Length     ; Setup address length
      MOV   COUNT,A
      SET   _SK                 ; Dummy bit transmission for 256x8 bit EEPROM Memory
                                ; capacity
      CLR   _SK                 ; Not required for 128x8 bit EEPROM Memory capacity devices
WriteAddr_0:
      CLR   _DI
      SZ    ADDR.7              ; Check value of address MSB
      SET   _DI
      CLR   C
      RLC   ADDR                ; Get next address bit
      SET   _SK
      CLR   _SK
      SDZ   COUNT               ; Check if address LSB has been written
      JMP   WriteAddr_0
      CLR   _DI
      RET
```

Example 3 – Writing Data to the EEPROM

```
WriteData:
      MOV   A,C_Data_Length     ; Setup data length
      MOV   COUNT,A
WriteData_0:
      CLR   _DI
      SZ    WR_Data.7           ; Check value of data MSB
      SET   _DI
      CLR   C
      RLC   WR_Data             ; Get next address bit
      SET   _SK
      CLR   _SK
      SDZ   COUNT               ; Check if data LSB has been written
      JMP   WriteData_0
      CLR   _CS                 ; CS low edge initiates internal write cycle
      SET   _CS                 ; CS high edge allows DO to be used to indicate
                                ; end of write cycle
      SNZ   _DO                 ; Poll for DO high to indicate end of write
                                ; cycle
      JMP   $-1
      RET
```

Example 4 – Reading Data from the EEPROM

```
ReadData:
      MOV   A,C_Data_Length    ; Setup data length
      MOV   COUNT,A
      CLR   WR_Data
ReadData_0:
      CLR   C
      RLC   WR_Data
      SET   _SK
      SZ    _DO                ; check value of data MSB
      SET   WR_Data.0
      CLR   _SK
      SDZ   COUNT              ; check if LSB has been received
      JMP   ReadData_0
      MOV   A,WR_Data
      RET
```

Input/Output Ports

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

This device provides 33 bidirectional input/output lines labeled with port names PA, PB, PC, PD and PG. These I/O ports are mapped to the RAM Data Memory with specific addresses as shown in the Special Purpose Data Memory table. All of these I/O ports can be used for input and output operations. For input operation, these ports are non-latching, which means the inputs must be ready at the T2 rising edge of instruction "MOV A,[m]", where m denotes the port address. For output operation, all the data is latched and remains unchanged until the output latch is rewritten.

Pull-high Resistors

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

Port A Wake-up

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

I/O Port Control Registers

Each I/O port has its own control register PAC, PBC, PCC, PDC and PGC, to control the input/output configuration. With this control register, each CMOS output or input with or without pull-high resistor structures can be reconfigured dynamically under software control. Each pin of the I/O ports is directly mapped to a bit in its associated port control register. For the I/O pin to function as an input, the corresponding bit of the control register must be written as a "1". This will then allow the logic state of the input pin to be directly read by instructions. When the corresponding bit of the control register is written as a "0", the I/O pin will be setup as a CMOS output. If the pin is currently setup as an output, instructions can still be used to read the output register. However, it should be noted that the program will in fact only read the status of the output data latch and not the actual logic status of the output pin. Note that there is an additional configuration option for Port A that can select whether the inputs on this port are Schmitt Trigger types or non-Schmitt Trigger types. Inputs for the other ports are all Schmitt Trigger type.

Pin-shared Functions

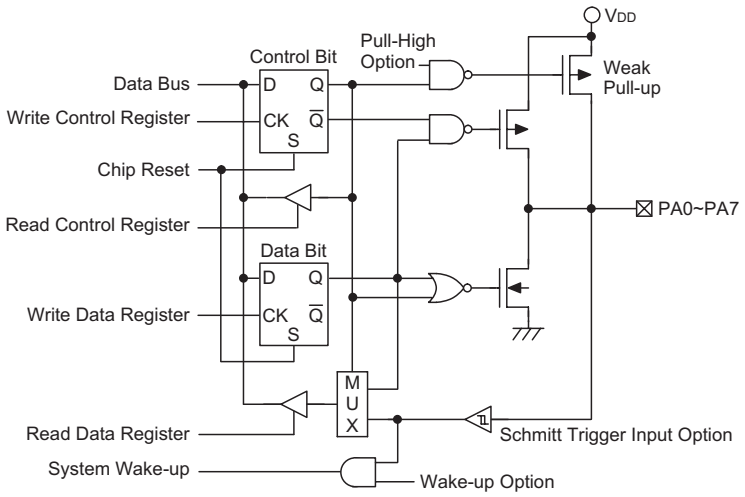
The flexibility of the microcontroller range is greatly enhanced by the use of pins that have more than one function. Limited numbers of pins can force serious design constraints on designers but by supplying pins with multi-functions, many of these difficulties can be overcome. For some pins, the chosen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

- Buzzer
  The buzzer pins BZ and $\overline{BZ}$ are pin-shared with I/O pins PB0 and PB1. The buzzer function is selected via a configuration option and remains fixed after the device is programmed. Note that the corresponding bits of the port control register, PBC, must setup the pins as outputs to enable the buzzer outputs. If the PBC port control register has setup the pins as inputs, then the pins will function as normal logic inputs with the usual pull-high options, even if the buzzer configuration option has been selected.

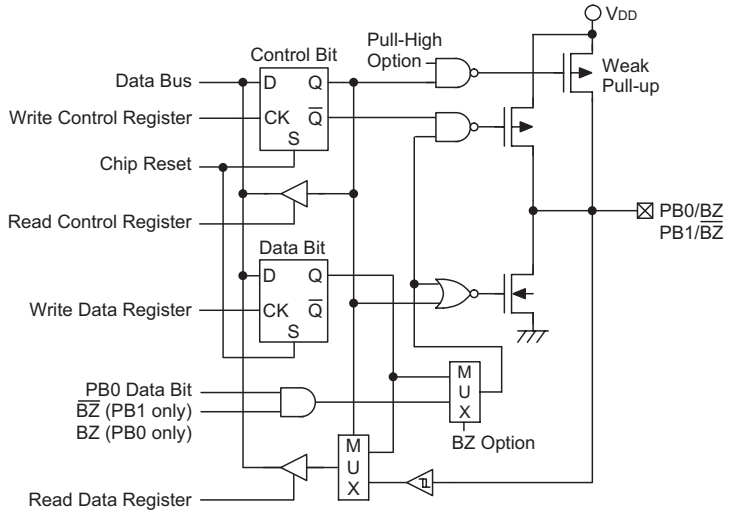
- External Interrupt Input
  The external interrupt pin $\overline{INT}$ is pin-shared with the I/O pin PG0. For the shared function pin to operate as an external interrupt pin and not as a normal I/O pin, the corresponding external interrupt enable bits in the INTC interrupt control register must be correctly set. 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.

- External Timer/Event Counter Input
  This device contains an 8-bit Timer/Event Counter 0 which has an external pin known as TMR0 and a 16-bit Timer/Event Counter 1 which has an external pin known as TMR1. These two external timer pins are individual input pins in 48-pin package but are pin-shared with I/O pin PC0 or PC5 in 28-pin package by internal wire bond. If these shared pins are to be used as Timer/Event Counter inputs, then the Timer/Event Counters must be configured to be in the Event Counter or Pulse Width Measurement Mode. This is achieved by setting the appropriate bits in the relevant Timer/Event Counter Control Registers. These pins must also be setup as inputs by setting the appropriate bits in the Port Control Register. Pull-high resistor option can also be selected via the appropriate port pull-high configuration option. If these shared pins are to be used as normal I/O pins, then the external timer input functions must be disabled, by ensuring that the corresponding Timer/Event Counters are configured to be in the Off Mode or Timer Mode.

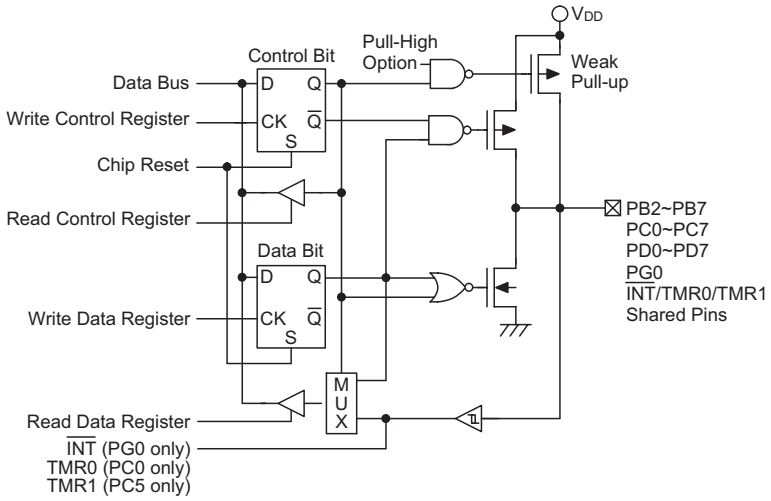
- I/O Pin Structures
  The following diagrams illustrate the I/O pin internal structures. As the exact logical construction of the I/O pin may differ from these drawings, they are supplied as a guide only to assist with the functional understanding of the I/O pins. Note also that the specified pins refer to the largest device package, therefore not all pins specified will exist on all devices.



PA Input/Output Port



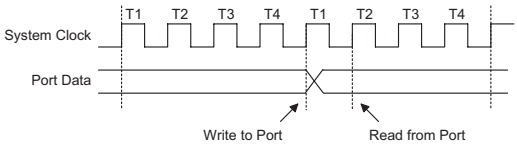
PB0~PB1 Input/Output Port



PB, PC, PD and PG Input/Output Ports

Programming Considerations

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



Read/Write Timing

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

Timer/Event Counters

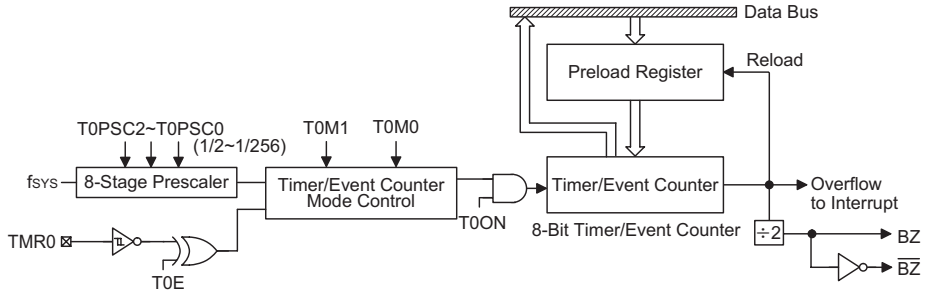
The provision of timers form an important part of any microcontroller, giving the designer a means of carrying out time related functions. This device contains two count-up timers of 8-bit and 16-bit capacities respectively. As each timer has three different operating modes, they can be configured to operate as a general timer, an external event counter or as a pulse width measurement device. The provision of an internal prescaler for the 8-bit Timer/Event Counter to the clock circuitry gives added range to the timer.

There are two types of registers related to the Timer/Event Counters. The first is the register that contains the actual value of the Timer/Event Counter and into which an initial value can be preloaded, and is known as TMR0, TMR1H or TMR1L. Reading from this register retrieves the contents of the Timer/Event Counter. The second type of associated register is the Timer Control Register, which defines the timer options and determines how the Timer/Event Counter is to be used, and has the name TMR0C or TMR1C. This device can have the timer clocks configured to come from the internal clock sources. In addition, the timer clock source can also be configured to come from the external timer pins.

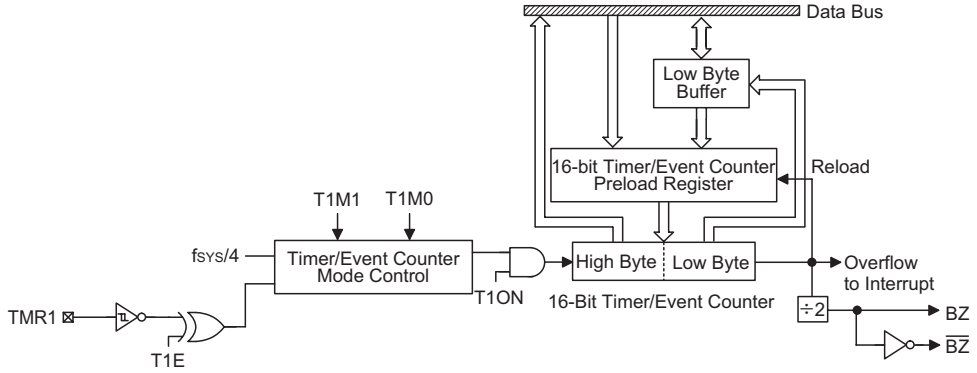
The external clock source is used when the Timer/Event Counter is in the event counting mode, the clock source being provided on the external timer pin. The pin has the name TMR0 or TMR1 and is pin-shared with an I/O pin. Depending upon the condition of the T0E or T1E bit in the Timer Control Register,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

Configuring the Timer/Event Counter Input Clock Source

The Timer/Event Counter's clock can originate from various sources. The system clock source is used when the Timer/Event Counter 0 is in the timer mode or in the pulse width measurement mode. The system clock is divided by a prescaler, the division ratio of which is conditioned by the Timer Control Register named TMR0C bits T0PSC2~T0PSC0. The instruction clock source (system clock source divided by 4) is used when the Timer/Event Counter 1 is in the timer mode or in the pulse width measurement mode. The external clock source is used when the Timer/Event Counter is in the event counting mode, the clock source being provided on the external timer pin, TMR0 or TMR1. Depending upon the condition of the T0E or T1E bit, each high to low, or low to high transition on the external timer pin will increment the counter by one.



8-bit Timer/Event Counter Structure



16-bit Timer/Event Counter Structure

Timer Registers – TMR0, TMR1L/TMR1H

The timer registers are special function registers located in the Special Purpose RAM Data Memory and are the places where the actual timer values are stored. For 8-bit Timer/Event Counter 0, this register is known as TMR0. For 16-bit Timer/Event Counter 1, the timer registers are known as TMR1L and TMR1H. The value in the timer registers increases by one each time an internal clock pulse is received or an external transition occurs on the external timer pin. The timer will count from the initial value loaded by the preload register to the full count of FFH for the 8-bit timer or FFFFH for the 16-bit timer at which point the timer overflows and an internal interrupt signal is generated. The timer value will then be reset with the initial preload register value and continue counting.

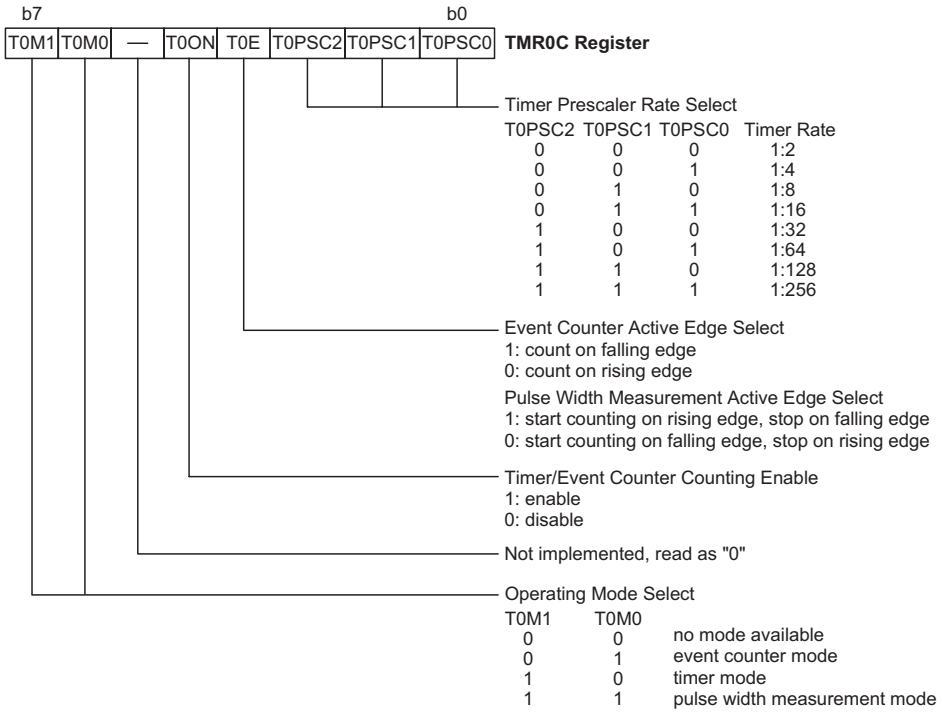
To achieve a maximum full range count of FFH for the 8-bit timer or FFFFH for the 16-bit timer, the preload registers must first be cleared to all zeros. It should be noted that after power-on, the preload register will be in an unknown condition. Note that if the Timer/Event Counter is switched off and data is written to its preload registers, this data will be immediately written into the actual timer registers. However, if the Timer/Event Counter is enabled and counting, any new data written into the preload data registers during this period will remain in the preload registers and will only be written into the timer registers the next time an overflow occurs.

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

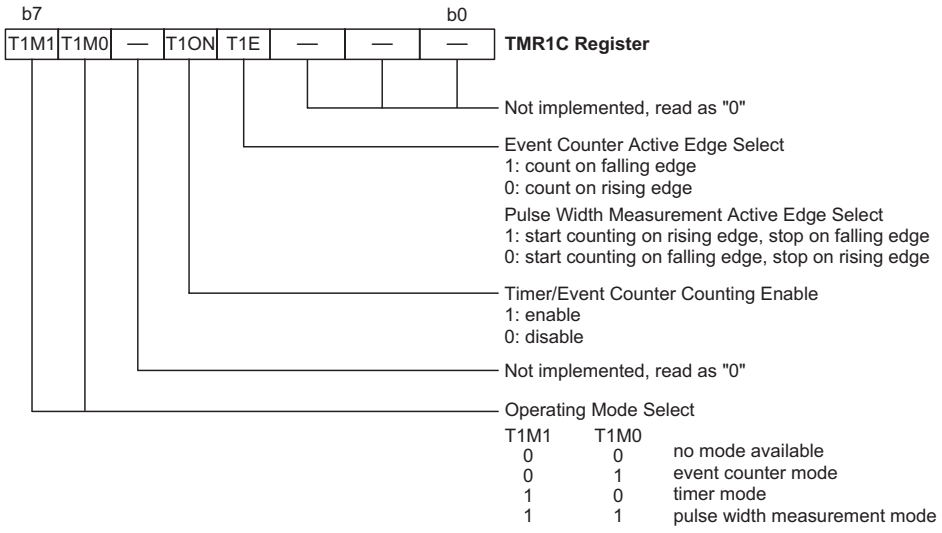
Timer Control Register – TMR0C, TMR1C

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

To choose which of the three modes the Timer/Event Counter is to operate in, either in the timer mode, the event counting mode or the pulse width measurement mode, bits 7 and 6 of the Timer Control Register, which are known as the bit pair T0M1/T0M0 or T1M1/T1M0, must be set to the required logic levels. The Timer/Event Counter on/off bit, which is bit 4 of the Timer Control Register and known as T0ON or T1ON, provides the basic on/off control of the Timer/Event Counter. Setting the bit high allows the Timer/Event Counter to run, clearing the bit stops it running. For 8-bit Timer/Event Counter 0 with prescaler, bits 0~2 of the Timer Control Register determine the division ratio of the input clock prescaler. The prescaler bit settings have no effect if an external clock source is used. If the Timer/Event Counter is in the event count or pulse width measurement mode, the active transition edge level type is selected by the logic level of bit 3 of the Timer Control Register which is known as T0E or T1E.



Timer/Event Counter 0 Control Register



Timer/Event Counter 1 Control Register

Configuring the Timer Mode

In this mode, the Timer/Event Counter can be utilised to measure fixed time intervals, providing an internal interrupt signal each time the Timer/Event Counter overflows. To operate in this mode, the Operating Mode Select bit pair, T0M1/T0M0 or T1M1/T1M0, in the Timer Control Register must be set to the correct value as shown.

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

In this mode the internal clock, f_{SYS} is used as the internal clock for 8-bit Timer/Event Counter 0 and $f_{SYS}/4$ is used as the internal clock for 16-bit Timer/Event Counter 1. However, the clock source, f_{SYS}, for 8-bit timer is further divided by a prescaler, the value of which is determined by the Prescaler Rate Select bits T0PSC2~T0PSC0, which are bits 2~0 in the Timer Control Register. After the other bits in the Timer Control Register have been setup, the enable bit T0ON or T1ON, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter to run. Each time an internal clock cycle occurs, the Timer/Event Counter increments by one. When it is full and overflows, an interrupt signal is generated and the Timer/Event Counter will reload the value already loaded into the preload register and continue counting. The interrupt can be disabled by ensuring that the Timer/Event Counter Interrupt Enable bit in the Interrupt Control Register, INTC, is reset to zero.

Configuring the Event Counter Mode

In this mode, a number of externally changing logic events, occurring on the external timer pin, can be recorded by the Timer/Event Counter. To operate in this mode, the Operating Mode Select bit pair, T0M1/T0M0 or T1M1/T1M0, in the Timer Control Register must be set to the correct value as shown.

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

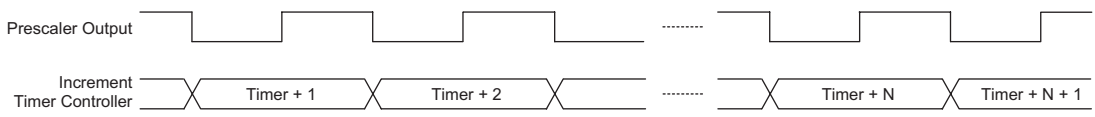
In this mode, the external timer pin, TMR0 or TMR1, is used as the Timer/Event Counter clock source, however it is not divided by the internal prescaler. After the other bits in the Timer Control Register have been setup, the enable bit T0ON or T1ON, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter to run. If the Active Edge Select bit T0E or T1E, which is bit 3 of the Timer Control Register, is low, the Timer/Event Counter will increment each time the external timer pin receives a low to high transition. If the Active Edge Select bit is high, the counter will increment each time the external timer pin receives a high to low transition. When it is full and overflows, an interrupt signal is generated and the Timer/Event Counter will reload the value already loaded into the preload register and continue counting. The interrupt can be disabled by ensuring that the Timer/Event Counter Interrupt Enable bit in the Interrupt Control Register, INTC, is reset to zero.

As the external timer pin is shared with an I/O pin, to ensure that the pin is configured to operate as an event counter input pin, two things have to happen. The first is to ensure that the Operating Mode Select bits in the Timer Control Register 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. It should be noted that in the event counting mode, even if the microcontroller is in the Power Down Mode, the Timer/Event Counter will continue to record externally changing logic events on the timer input pin. As a result when the timer overflows it will generate a timer interrupt and corresponding wake-up source.

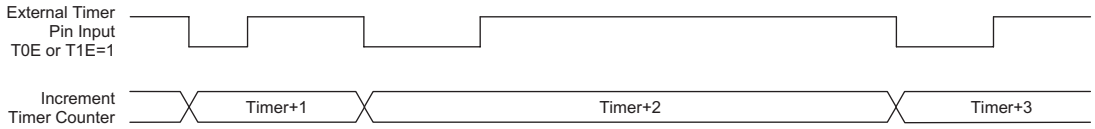
Configuring the Pulse Width Measurement Mode

In this mode, the Timer/Event Counter can be utilised to measure the width of external pulses applied to the external timer pin. To operate in this mode, the Operating Mode Select bit pair, T0M1/T0M0 or T1M1/T1M0, in the Timer Control Register must be set to the correct value as shown.

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



Timer Mode Timing Diagram



Event Counter Mode Timing Diagram

In this mode the internal clock, f_{SYS} is used as the internal clock for 8-bit Timer/Event Counter 0 and $f_{SYS}/4$ is used as the internal clock for 16-bit Timer/Event Counter 1. However, the clock source, f_{SYS}, for 8-bit timer is further divided by a prescaler, the value of which is determined by the Prescaler Rate Select bits T0PSC2~T0PSC0, which are bits 2~0 in the Timer Control Register. After the other bits in the Timer Control Register have been setup, the enable bit T0ON or T1ON, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter, however it will not actually start counting until an active edge is received on the external timer pin.

If the Active Edge Select bit T0E or T1E, which is bit 3 of the Timer Control Register, is low, once a high to low transition has been received on the external timer pin, TMR0 or TMR1, the Timer/Event Counter will start counting until the external timer pin returns to its original high level. At this point the enable bit will be automatically reset to zero and the Timer/Event Counter will stop counting. If the Active Edge Select bit is high, the Timer/Event Counter will begin counting once a low to high transition has been received on the external timer pin and stop counting when the external timer pin returns to its original low level. As before, the enable bit will be automatically reset to zero and the Timer/Event Counter will stop counting. It is important to note that in the Pulse Width Measurement Mode, the enable bit is automatically reset to zero when the external control signal on the external timer pin returns to its original level, whereas in the other two modes the enable bit can only be reset to zero under program control.

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

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

As the external timer pin is shared with an I/O pin, to ensure that the pin is configured to operate as a pulse width measurement pin, two things have to happen. The first is to ensure that the Operating Mode Select bits in the Timer Control Register place the Timer/Event Counter in the Pulse Width Measurement Mode, the second is to ensure that the port control register configures the pin as an input.

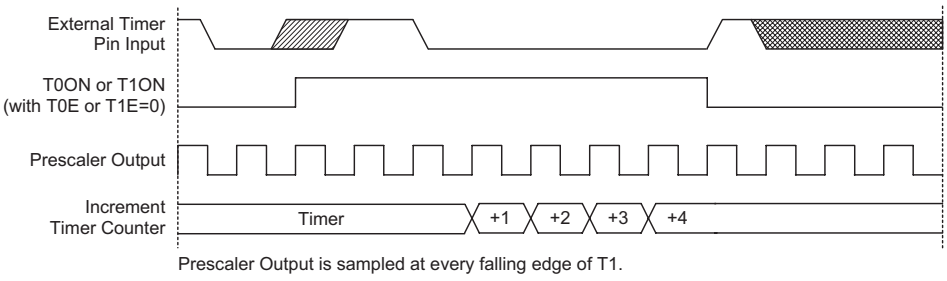
Programmable Frequency Divider (PFD) and Buzzer Application

Operating similar to a programmable frequency divider, the buzzer function within the microcontroller provides a means of producing a variable frequency output suitable for applications, such as piezo-buzzer driving or other interfaces requiring a precise frequency generator.

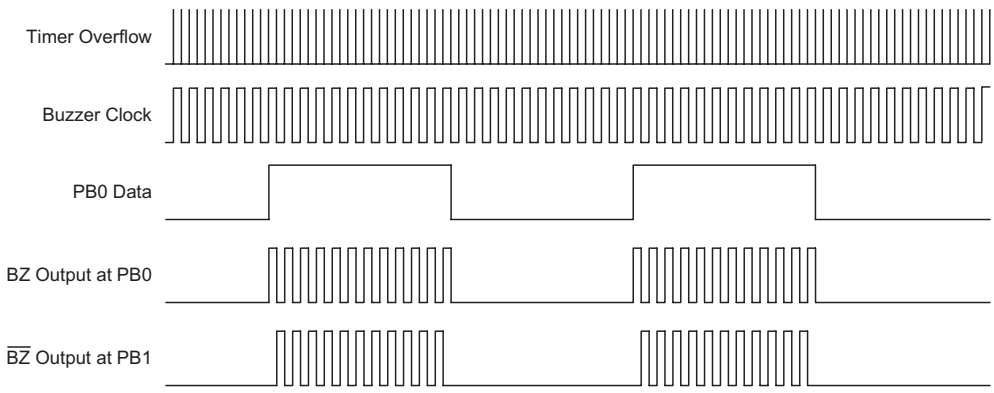
The BZ and $\overline{BZ}$ are a complimentary pair and pin-shared with I/O pins, PB0 and PB1. The function is selected via configuration option, however, if not selected, the pins can operate as normal I/O pins. Note that the $\overline{BZ}$ pin is the inverse of the BZ pin generating a kind of differential output and supplying more power to connected interfaces such as buzzers. Note that the 16-pin NSOP package type only has a single BZ output as pin PB1/$\overline{BZ}$ does not exist on this package.

The clock source for the buzzer circuit can come from the timer 0 or timer 1 overflow signal selected via the configuration option. The output frequency is controlled by loading the required values into the timer prescaler and timer registers to give the required division ratio. The counter will begin to count-up from this preload register value until full, at which point an overflow signal is generated, causing both the BZ and $\overline{BZ}$ outputs to change state. The counter will then be automatically reloaded with the preload register value and continue counting-up.

If the configuration option has selected the buzzer function, then for both buzzer outputs to operate, it is essen-



Pulse Width Measure Mode Timing Diagram



PFD Output Control

tial that the Port B control register PBC bit 0 and PBC bit 1 are setup as outputs. If only one pin is setup as an output, the other pin can still be used as a normal data input pin. However, if both pins are setup as inputs then the buzzer will not function. The buzzer outputs will only be activated if bit PB0 is set to "1". This output data bit is used as the on/off control bit for the buzzer outputs. Note that the BZ and $\overline{BZ}$ outputs will both be low if the PB0 output data bit is cleared to "0". The condition of data bit PB1 has no effect on the overall control of the BZ and $\overline{BZ}$ pins.

Using this method of frequency generation, and if a crystal oscillator is used for the system clock, very precise values of frequency can be generated.

Prescaler

The 8-bit timer in the device possesses a prescaler. Bits 2~0 of the Timer Control Register TMR0C, named T0PSC2~T0PSC0,define the prescaling stages of the internal clock source of the 8-bit Timer/Event Counter.

I/O Interfacing

The Timer/Event Counter, when configured to run in the event counter or pulse width measurement mode, requires the use of an external pin for correct operation. As the external timer pin is pin-shared with an I/O 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. In 28-pin package, the associated Port Control Register bits for the external timer input pins must be set high to ensure that these pins are setup as an inputs. Any pull high configuration for these pins will remain valid even if these pins are used as Timer/Event Counter inputs. In 48-pin package, the external timer input pins are independent pins without pull-high resistors and not pin-shared with I/O pins. There are not any Port Control Register bits that will affect the Timer/Event Counters function.

Programming Considerations

When configured to run in the timer mode, the internal system clock and instruction clock (system clock divided by 4) are used as the internal timer clock sources for 8-bit and 16-bit timers resepctively and are therefore synchronized with the overall operation of the microcontroller. In this mode, when the appropriate timer register is full, the microcontroller will generate an internal interrupt signal directing the program flow to the respective internal interrupt vector. For the pulse width measurement mode, the internal system clock or instruction 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 registers, the clock is inhibited to avoid errors, however as this may result in a counting error, this should be taken into account by the programmer. Care must be taken to ensure that the timers are properly initialised before using them for the first time. The associated timer 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 register before the timer is switched on; this is because after power-on the initial value of the timer register is unknown. After the timer has been initialised the timer can be turned on and off by controlling the enable bit in the

timer control register. Note that setting the timer enable bit high to turn the timer on, should only be executed after the timer mode bits have been properly setup. Setting the timer enable bit high together with a mode bit modification, may lead to improper timer operation if executed as a single timer control register byte write instruction.

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

Timer Program Example

This program example shows how the Timer/Event Counter registers are setup, along with how the interrupts are enabled and managed. Note how the Timer/Event Counter is turned on, by setting bit 4 of the Timer Control Register. The Timer/Event Counter can be turned off in a similar way by clearing the same bit. This example program sets the Timer/Event Counter tobe in the timer mode, which uses the internal system clock as the clock source.

```
org   04h           ; external interrupt vector
reti
org   08h           ; Timer/Event Counter 0 interrupt vector
jmp   tmr0int       ; jump here when Timer/Event Counter 0 overflows
org   0ch           ; Timer/Event Counter 1 interrupt vector
jmp   tmr1int       ; jump here when Timer/Event Counter 1 overflows
      :
org   20h           ; main program
      :
;internal Timer/Event Counter 0 interrupt routine
tmr0int:
      :
; Timer/Event Counter 0 main program placed here
      :
      reti
      :
;internal Timer/Event Counter 1 interrupt routine
tmr1int:
      :
; Timer/Event Counter 1 main program placed here
      :
      reti
      :
begin:
;setup Timer/Event Counter 0 registers
mov   a,09bh        ; setup preload value for Timer/Event Counter 0 – timer counts from
                    ; this value to FFH
mov   tmr0,a;
mov   a,081h        ; setup Timer control register TMR0C
mov   tmr0c,a       ; timer mode and prescaler set to /4

;setup Timer/Event Counter 1 registers
mov   a,09bh        ; setup low byte preload value for Timer/Event Counter 1
mov   tmr1l,a       ; low byte must be setup before high byte
mov   a,0e8h        ; setup high byte preload value for Timer/Event Counter 1
mov   tmr1h,a       ;
mov   a,080h        ; setup Timer control register TMR1C
mov   tmr1c,a       ; Timer/Event Counter 1 has no prescaler and clock source is fSYS/4

; setup interrupt register
mov   a,00dh; enable master interrupt and timer interrupts
mov   intc,a
      :
set   tmr0c.4       ; start Timer/Event Counter 0 – note mode bits must be previously setup
set   tmr1c.4       ; start Timer/Event Counter 1 – note mode bits must be previously setup
```

Interrupts

Interrupts are an important part of any microcontroller system. When an external event or an internal function such as a Timer/Event Counter requires microcontroller attention, their corresponding interrupt will enforce a temporary suspension of the main program allowing the microcontroller to direct attention to their respective needs. The device contains a single external interrupt and two internal timer interrupt functions. The external interrupt is controlled by the action of the external $\overline{INT}$ pin, while the internal interrupts are controlled by the Timer/Event Counters overflow.

Interrupt Register

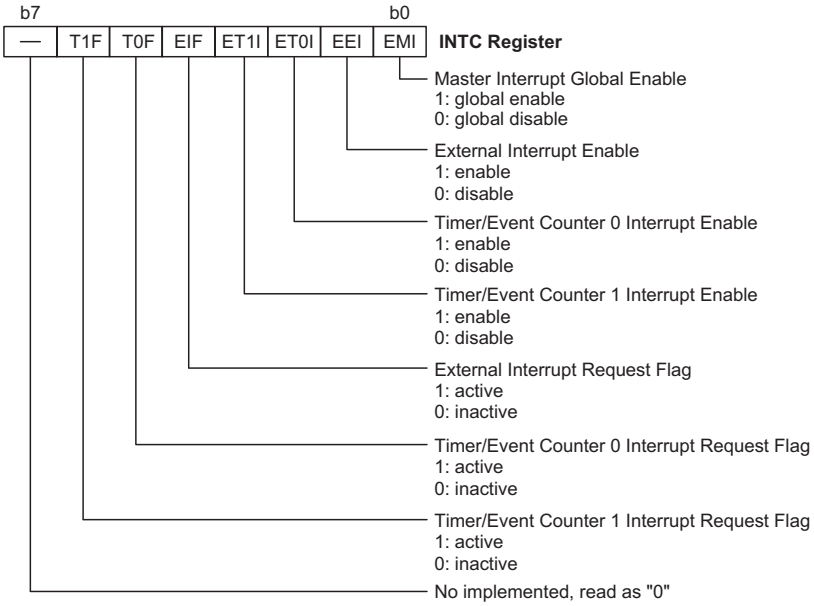
Overall interrupt control, which means interrupt enabling and request flag setting, is controlled by a single INTC register, which is located in the RAM Data Memory. By controlling the appropriate enable bits in this register each individual interrupt can be enabled or disabled. Also when an interrupt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

Interrupt Operation

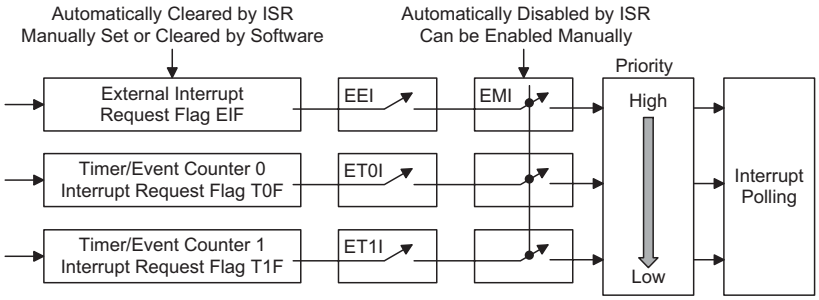
A Timer/Event Counter overflow or the external interrupt line being pulled low will all generate an interrupt request by setting their corresponding request flag, if their appropriate interrupt enable bit is set. When this happens, the Program Counter, which stores the address of the next instruction to be executed, will be transferred onto the stack. The Program Counter will then be loaded with a new address which will be the value of the corresponding interrupt vector. The microcontroller will then fetch its next instruction from this interrupt vector. The instruction at this vector will usually be a JMP statement which will take program execution to another section of program which is known as the interrupt service routine. Here is located the code to control the appropriate interrupt. The interrupt service routine must be terminated with a RETI statement, which retrieves the original Program Counter address from the stack and allows the microcontroller to continue with normal execution at the point where the interrupt occurred.

The various interrupt enable bits, together with their associated request flags, are shown in the following diagram with their order of priority.

Once an interrupt subroutine is serviced, all the other interrupts will be blocked, as the EMI bit will be cleared automatically. This will prevent any further interrupt nesting from occurring. However, if other interrupt requests occur during this interval, although the interrupt will not be immediately serviced, the request flag will still be recorded. If an interrupt requires immediate servicing while the program is already in another interrupt service routine, the EMI bit should be set after entering the routine, to allow interrupt nesting. If the stack is full, the interrupt request will not be acknowledged, even if the related interrupt is enabled, until the Stack Pointer is decremented. If immediate service is desired, the stack must be prevented from becoming full.



Interrupt Control Register



Interrupt Structure

Interrupt Priority

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

Interrupt Source	Priority
External Interrupt	1
Timer/Event Counter 0 Overflow	2
Timer/Event Counter 1 Overflow	3

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

External Interrupt

For an external interrupt to occur, the global interrupt enable bit, EMI, and external interrupt enable bit, EEI, must first be set. An actual external interrupt will take place when the external interrupt request flag, EIF, is set, a situation that will occur when a high to low transition appears on the $\overline{INT}$ line. The external interrupt pin is pin-shared with an I/O pin PG.0 and can only be configured as an external interrupt pin if the corresponding external interrupt enable bit in the INTC register has been set. The pin must also be setup as an input by setting the corresponding PGC.0 bit in the port control register. When the interrupt is enabled, the stack is not full and a high to low transition appears on the external interrupt pin, a subroutine call to the external interrupt vector at location 04H, will take place. When the interrupt is serviced, the external interrupt request flag, EIF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts. Note that any pull-high resistor configuration options on this pin will remain valid even if the pin is used as an external interrupt input.

Timer/Event Counter Interrupts

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

Programming Considerations

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

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

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

Reset and Initialisation

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

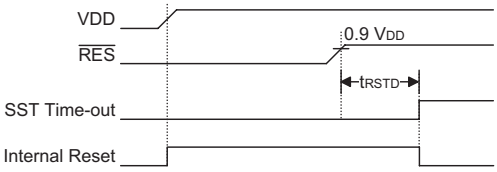
In addition to the power-on reset, situations may arise where it is necessary to forcefully apply a reset condition when the microcontroller is running. One example of this is where after power has been applied and the microcontroller is already running, the $\overline{\text{RES}}$ line is forcefully pulled low. In such a case, known as a normal operation reset, some of the microcontroller registers remain unchanged allowing the microcontroller to proceed with normal operation after the reset line is allowed to return high. Another type of reset is when the Watchdog Timer overflows and resets the microcontroller. All types of reset operations result in different register conditions being setup.

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

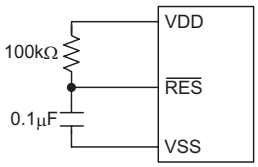
There are five ways in which a microcontroller reset can occur, through events occurring both internally and externally:

- 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.



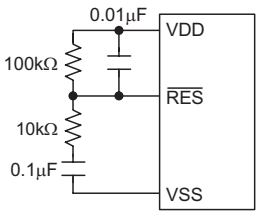
Power-On Reset Timing Chart

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

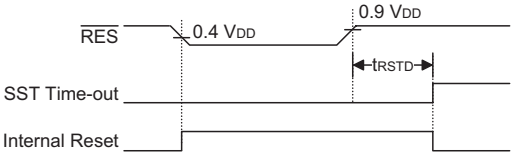
For applications that operate within an environment where more noise is present the Enhanced Reset Circuit shown is recommended.



Enhanced Reset Circuit

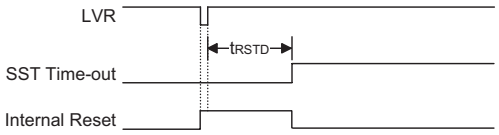
More information regarding external reset circuits is located in Application Note HA0075E on the Holtek website.

- $\overline{\text{RES}}$ 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.



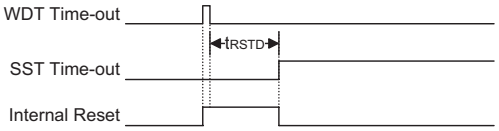
$\overline{\text{RES}}$ Reset Timing Chart

- Low Voltage Reset – LVR
  The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device, which 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. The LVR includes the following specifications: For a valid LVR signal, a low voltage, i.e., a voltage in the range between 0.9V~V_{LVR} must exist for greater than the value t_{LVR} specified in the A.C. characteristics. If the low voltage state does not exceed 1ms, the LVR will ignore it and will not perform a reset function.



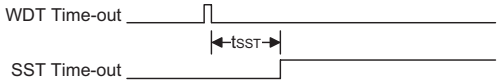
Low Voltage Reset Timing Chart

- 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
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
Prescaler	The Timer Counter Prescaler will be cleared
Input/Output Ports	I/O ports will be setup as inputs
Stack Pointer	Stack Pointer will point to the top of the stack

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

HT48F50E

Register	Reset (Power-on)	$\overline{\text{RES}}$ or LVR Reset	WDT Time-out (Normal Operation)	WDT Time-out (HALT)
MP0	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
MP1	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
BP	0000 0000	0000 0000	0000 0000	uuuu uuuu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCL	0000 0000	0000 0000	0000 0000	0000 0000
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	--xx xxxx	--uu uuuu	--uu uuuu	--uu uuuu
WDTS	0000 0111	0000 0111	0000 0111	uuuu uuuu
STATUS	--00 xxxx	--uu uuuu	--1u uuuu	--11 uuuu
INTC	--00 -000	--00 -000	--00 -000	--uu -uuu
TMR0	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR0C	00-0 1000	00-0 1000	00-0 1000	uu-u uuuu
TMR1H	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR1L	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR1C	00-0 1---	00-0 1---	00-0 1---	uu-u u---
PA	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PB	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PCC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PD	1111 1111	1111 1111	1111 1111	uuuu uuuu
PDC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PG	---- ---1	---- ---1	---- ---1	---- ---u
PGC	---- ---1	---- ---1	---- ---1	---- ---u

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

Oscillator

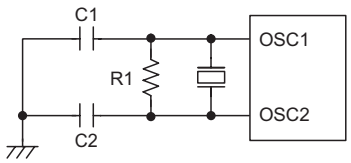
Various oscillator options offer the user a wide range of functions according to their various application requirements. Two types of system clocks can be selected while various clock source options for the Watchdog Timer, are provided for maximum flexibility. All oscillator options are selected through the configuration options.

System Clock Configurations

There are two methods of generating the system clock, using an external crystal/ceramic oscillator or an external RC network. The chosen method is selected through the configuration options.

System Crystal/Ceramic Oscillator

After selecting the correct oscillator configuration option, for most crystal oscillator configurations, the simple connection of a crystal across OSC1 and OSC2 will create the necessary phase shift and feedback for oscillation, without requiring external capacitors. However, for some crystal types and frequencies, to ensure oscillation, it may be necessary to add two small value capacitors, C1 and C2. Using a ceramic resonator will usually require two small value capacitors, C1 and C2, to be connected as shown for oscillation to occur. The values of C1 and C2 should be selected in consultation with the crystal or resonator manufacturer's specification. In most applications, resistor R1 is not required, however for those applications where the LVR function is not used, R1 may be necessary to ensure the oscillator stops running when VDD falls below its operating range.

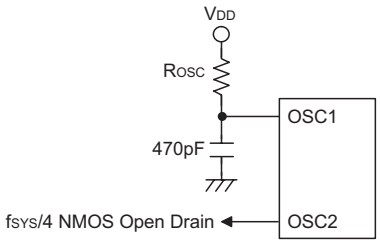


Crystal/Ceramic Oscillator

More information regarding the oscillator is located in Application Note HA0075E on the Holtek website.

System RC Oscillator

After selecting the correct configuration option, using the external system RC oscillator requires that a resistor, with a value between 24kΩ and 1MΩ, is connected between OSC1 and VDD, and a 470pF capacitor is connected to ground. Although this is a cost effective oscillator configuration, the oscillation frequency can vary with VDD, temperature and process variations and is therefore not suitable for applications where timing is critical or where accurate oscillator frequencies are required. For the value of the external resistor R_{OSC} refer to the Appendix section for typical RC Oscillator vs. Temperature and VDD characteristics graphics.



RC Oscillator

Note that it is the only microcontroller internal circuitry together with the external resistor, that determine the frequency of the oscillator. The external capacitor shown on the diagram does not influence the frequency of oscillation. The external capacitor is added to improve oscillator stability, especially if the open-drain OSC2 output is utilised in the application circuit.

Watchdog Timer Oscillator

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

Power Down Mode and Wake-up

Power Down Mode

All of the Holtek microcontrollers have the ability to enter a Power Down Mode, also known as the HALT Mode or Sleep 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 MCU 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 clock source is selected to come from the WDT oscillator. The WDT will stop if its clock source originates from the system clock.
- The I/O ports will maintain their present condition.
- In the status register, the Power Down flag, PDF, 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 MCU to as low a value as possible, perhaps only in the order of several micro-amps, there are other considerations which must also be taken into account by the circuit designer if the power consumption is to be minimized. Special attention must be made to the I/O pins on the device. All high-impedance input pins must be connected to either a fixed high or low level as any floating input pins could create internal oscillations and result in increased current consumption. Care must also be taken with the loads, which are connected to I/Os, which are setup as outputs. These should be placed in a condition in which minimum current is drawn or connected only to external circuits that do not draw current, such as other CMOS inputs. Also note that additional standby current will also be required if the configuration options have enabled the Watchdog Timer internal oscillator.

Wake-up

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 edge on Port A
- A system interrupt
- A WDT overflow

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

Each pin on Port A can be setup via an individual configuration option to permit a negative transition on the pin to wake-up the system. When a Port A pin wake-up occurs, the program will resume execution at the instruction following the "HALT" instruction.

If the system is woken up by an interrupt, then two possible situations may occur. The first is where the related interrupt is disabled or the interrupt is enabled but the stack is full, in which case the program will resume execution at the instruction following the "HALT" instruction. In this situation, the interrupt which woke-up the device will not be immediately serviced, but will rather be serviced later when the related interrupt is finally enabled or when a stack level becomes free. The other situation is where the related interrupt is enabled and the stack is not full, in which case the regular interrupt response takes place. If an interrupt request flag is set 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 1024 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 an 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 1024 system clock period delay has ended.

Watchdog Timer

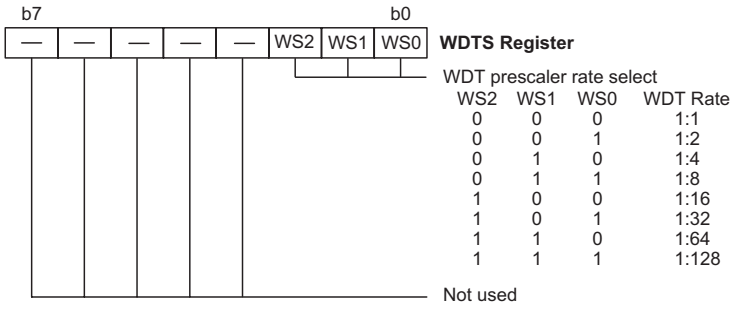
The Watchdog Timer is provided to prevent program malfunctions or sequences from jumping to unknown locations, due to certain uncontrollable external events such as electrical noise. It operates by providing a device reset when the WDT counter overflows. The WDT clock is supplied by one of two sources selected by configuration option: its own self-contained dedicated internal WDT oscillator, or the instruction clock which is the system clock divided by 4. Note that if the WDT configuration option has been disabled, then any instruction relating to its operation will result in no operation.

The internal WDT oscillator has an approximate period of 65μs at a supply voltage of 5V. If selected, it is first divided by 256 via an 8-stage counter to give a nominal period of 17ms. Note that this period can vary with VDD, temperature and process variations. For longer WDT time-out periods the WDT prescaler can be utilized. By writing the required value to bits 0, 1 and 2 of the WDTS register, known as WS0, WS1 and WS2, longer time-out periods can be achieved. With WS0, WS1 and WS2 all equal to 1, the division ratio is 1:128 which gives a maximum time-out period of about 2.1s.

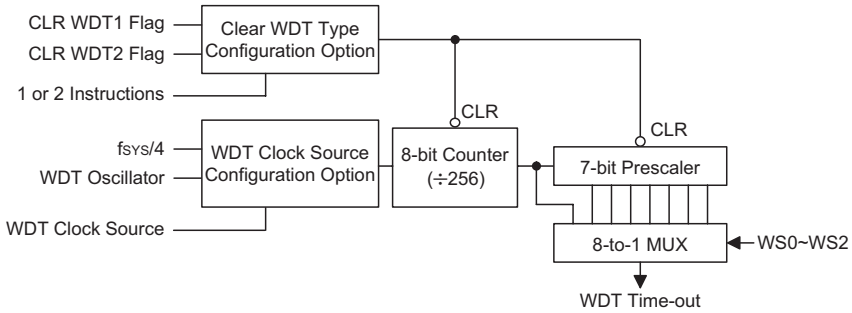
A configuration option can select the instruction clock, which is the system clock divided by 4, as the WDT clock source instead of the internal WDT oscillator. If the instruction clock is used as the clock source, it must be noted that when the system enters the Power Down Mode, as the system clock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

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

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



Watchdog Timer Register



Watchdog Timer

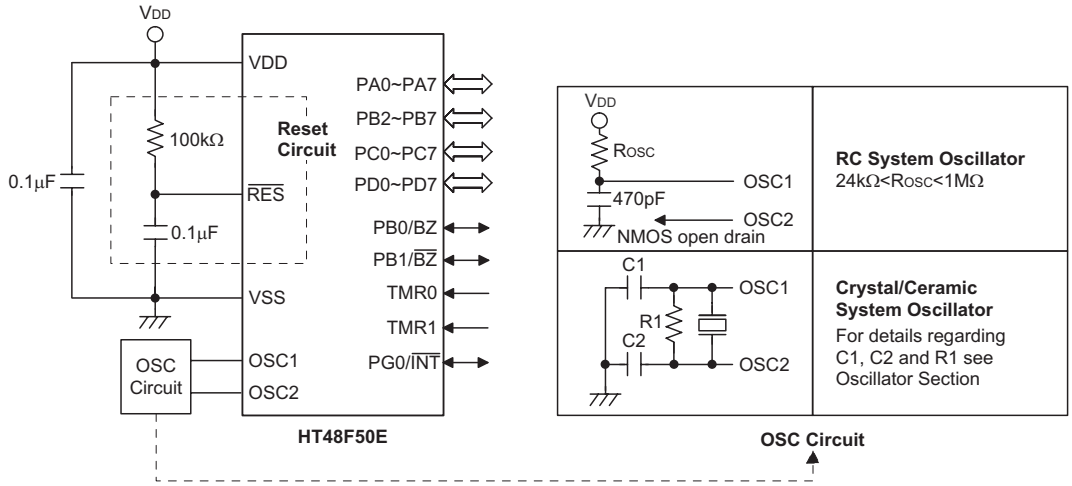
Configuration Options

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

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

No.	Options
1	Watchdog Timer: enable or disable
2	Watchdog Timer clock source: WDT oscillator or $f_{SYS}/4$
3	CLRWDT instructions: 1 or 2 instructions
4	PA0~PA7: wake-up enable or disable (bit option)
5	PA, PB, PC, PD and PG: pull-high enable or disable (by port)
6	PA input type: CMOS or Schmitt Trigger
7	Buzzer function: enable or normal I/O
8	Buzzer clock source: Timer 0 or Timer 1
9	System oscillator: Crystal or RC
10	LVR function: enable or disable

Application Circuits



Instruction Set

Introduction

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

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

Instruction Timing

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

Moving and Transferring Data

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

Arithmetic Operations

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

Logical and Rotate Operations

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

Branches and Control Transfer

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

Bit Operations

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

Table Read Operations

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

Other Operations

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

Instruction Set Summary

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

Table conventions:

x: Bits immediate data

m: Data Memory address

A: Accumulator

i: 0~7 number of bits

addr: Program memory address

Mnemonic	Description	Cycles	Flag Affected
Arithmetic			
ADD A,[m]	Add Data Memory to ACC	1	Z, C, AC, OV
ADDM A,[m]	Add ACC to Data Memory	1Note	Z, C, AC, OV
ADD A,x	Add immediate data to ACC	1	Z, C, AC, OV
ADC A,[m]	Add Data Memory to ACC with Carry	1	Z, C, AC, OV
ADCM A,[m]	Add ACC to Data memory with Carry	1Note	Z, C, AC, OV
SUB A,x	Subtract immediate data from the ACC	1	Z, C, AC, OV
SUB A,[m]	Subtract Data Memory from ACC	1	Z, C, AC, OV
SUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory	1Note	Z, C, AC, OV
SBC A,[m]	Subtract Data Memory from ACC with Carry	1	Z, C, AC, OV
SBCM A,[m]	Subtract Data Memory from ACC with Carry, result in Data Memory	1Note	Z, C, AC, OV
DAA [m]	Decimal adjust ACC for Addition with result in Data Memory	1Note	C
Logic Operation			
AND A,[m]	Logical AND Data Memory to ACC	1	Z
OR A,[m]	Logical OR Data Memory to ACC	1	Z
XOR A,[m]	Logical XOR Data Memory to ACC	1	Z
ANDM A,[m]	Logical AND ACC to Data Memory	1Note	Z
ORM A,[m]	Logical OR ACC to Data Memory	1Note	Z
XORM A,[m]	Logical XOR ACC to Data Memory	1Note	Z
AND A,x	Logical AND immediate Data to ACC	1	Z
OR A,x	Logical OR immediate Data to ACC	1	Z
XOR A,x	Logical XOR immediate Data to ACC	1	Z
CPL [m]	Complement Data Memory	1Note	Z
CPLA [m]	Complement Data Memory with result in ACC	1	Z
Increment & Decrement			
INCA [m]	Increment Data Memory with result in ACC	1	Z
INC [m]	Increment Data Memory	1Note	Z
DECA [m]	Decrement Data Memory with result in ACC	1	Z
DEC [m]	Decrement Data Memory	1Note	Z

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

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

Instruction Definition

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

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

Operation ACC ← ACC + [m] + C

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

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

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

Operation [m] ← ACC + [m] + C

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

ADD A,[m] Add Data Memory to ACC

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

Operation ACC ← ACC + [m]

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

ADD A,x Add immediate data to ACC

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

Operation ACC ← ACC + x

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

ADDM A,[m] Add ACC to Data Memory

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

Operation [m] ← ACC + [m]

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

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

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

Operation ACC ← ACC "AND" [m]

Affected flag(s) Z

AND A,x Logical AND immediate data to ACC

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

Operation ACC ← ACC "AND" x

Affected flag(s) Z

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

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

Operation [m] ← ACC "AND" [m]

Affected flag(s) Z

CALL addr Subroutine call

Description Unconditionally calls a subroutine at the specified address. The Program Counter then increments by 1 to obtain the address of the next instruction which is then pushed onto the stack. The specified address is then loaded and the program continues execution from this new address. As this instruction requires an additional operation, it is a two cycle instruction.

Operation Stack ← Program Counter + 1
Program Counter ← addr

Affected flag(s) None

CLR [m] Clear Data Memory

Description Each bit of the specified Data Memory is cleared to 0.

Operation [m] ← 00H

Affected flag(s) None

CLR [m].i Clear bit of Data Memory

Description Bit i of the specified Data Memory is cleared to 0.

Operation [m].i ← 0

Affected flag(s) None

CLR WDT Clear Watchdog Timer

Description The TO, PDF flags and the WDT are all cleared.

Operation WDT cleared
TO ← 0
PDF ← 0

Affected flag(s) TO, PDF

CLR WDT1 Pre-clear Watchdog Timer

Description The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT2 and must be executed alternately with CLR WDT2 to have effect. Repetitively executing this instruction without alternately executing CLR WDT2 will have no effect.

Operation WDT cleared
TO ← 0
PDF ← 0

Affected flag(s) TO, PDF

CLR WDT2 Pre-clear Watchdog Timer

Description The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT1 and must be executed alternately with CLR WDT1 to have effect. Repetitively executing this instruction without alternately executing CLR WDT1 will have no effect.

Operation WDT cleared
TO ← 0
PDF ← 0

Affected flag(s) TO, PDF

CPL [m] Complement Data Memory

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

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

Affected flag(s) Z

CPLA [m] Complement Data Memory with result in ACC

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

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

Affected flag(s) Z

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

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

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

Affected flag(s) C

DEC [m] Decrement Data Memory

Description Data in the specified Data Memory is decremented by 1.

Operation $[m] \leftarrow [m] - 1$

Affected flag(s) Z

DECA [m] Decrement Data Memory with result in ACC

Description Data in the specified Data Memory is decremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.

Operation $ACC \leftarrow [m] - 1$

Affected flag(s) Z

HALT Enter power down mode

Description This instruction stops the program execution and turns off the system clock. The contents of the Data Memory and registers are retained. The WDT and prescaler are cleared. The power down flag PDF is set and the WDT time-out flag TO is cleared.

Operation $TO \leftarrow 0$
$PDF \leftarrow 1$

Affected flag(s) TO, PDF

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

OR A,x Logical OR immediate data to ACC

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

Operation: ACC ← ACC "OR" x

Affected flag(s): Z

ORM A,[m] Logical OR ACC to Data Memory

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

Operation: [m] ← ACC "OR" [m]

Affected flag(s): Z

RET Return from subroutine

Description: The Program Counter is restored from the stack. Program execution continues at the restored address.

Operation: Program Counter ← Stack

Affected flag(s): None

RET A,x Return from subroutine and load immediate data to ACC

Description: The Program Counter is restored from the stack and the Accumulator loaded with the specified immediate data. Program execution continues at the restored address.

Operation: Program Counter ← Stack
ACC ← x

Affected flag(s): None

RETI Return from interrupt

Description: The Program Counter is restored from the stack and the interrupts are re-enabled by setting the EMI bit. EMI is the master interrupt global enable bit. If an interrupt was pending when the RETI instruction is executed, the pending Interrupt routine will be processed before returning to the main program.

Operation: Program Counter ← Stack
EMI ← 1

Affected flag(s): None

RL [m] Rotate Data Memory left

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

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

Affected flag(s): None

RLA [m] Rotate Data Memory left with result in ACC

Description: The contents of the specified Data Memory are rotated left by 1 bit with bit 7 rotated into bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

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

Affected flag(s): None

RLC [m] Rotate Data Memory left through Carry

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

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

Affected flag(s) C

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

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

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

Affected flag(s) C

RR [m] Rotate Data Memory right

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

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

Affected flag(s) None

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

Description Data in the specified Data Memory and the carry flag are rotated right by 1 bit with bit 0 rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

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

Affected flag(s) None

RRC [m] Rotate Data Memory right through Carry

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

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

Affected flag(s) C

RRCA [m] Rotate Data Memory right through Carry with result in ACC

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

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

Affected flag(s) C

SBC A,[m] Subtract Data Memory from ACC with Carry

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

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

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

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

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

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

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

SDZ [m] Skip if decrement Data Memory is 0

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

Operation $[m] \leftarrow [m] - 1$
Skip if $[m] = 0$

Affected flag(s) None

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

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

Operation $ACC \leftarrow [m] - 1$
Skip if $ACC = 0$

Affected flag(s) None

SET [m] Set Data Memory

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

Operation $[m] \leftarrow FFH$

Affected flag(s) None

SET [m].i Set bit of Data Memory

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

Operation $[m].i \leftarrow 1$

Affected flag(s) None

SIZ [m] Skip if increment Data Memory is 0

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

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

Affected flag(s) None

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

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

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

Affected flag(s) None

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

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

Operation Skip if [m].i $\neq$ 0

Affected flag(s) None

SUB A,[m] Subtract Data Memory from ACC

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

Operation ACC ← ACC – [m]

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

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

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

Operation [m] ← ACC – [m]

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

SUB A,x Subtract immediate data from ACC

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

Operation ACC ← ACC – x

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

SWAP [m] Swap nibbles of Data Memory

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

Operation [m].3~[m].0 ↔ [m].7 ~ [m].4

Affected flag(s) None

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

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

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

Affected flag(s) None

SZ [m] Skip if Data Memory is 0

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

Operation Skip if [m] = 0

Affected flag(s) None

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

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

Operation ACC ← [m]
Skip if [m] = 0

Affected flag(s) None

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

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

Operation Skip if [m].i = 0

Affected flag(s) None

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

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

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

Affected flag(s) None

TABRDL [m] Read table (last page) to TBLH and Data Memory

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

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

Affected flag(s) None

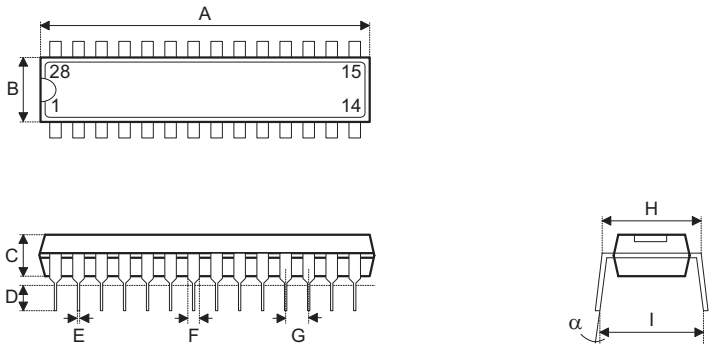
XOR A,[m]	Logical XOR Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical XOR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″XOR″ [m]
Affected flag(s)	Z

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

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

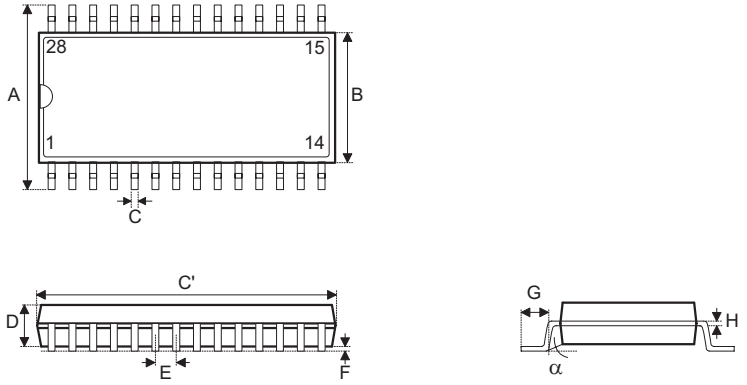
Package Information

28-pin SKDIP (300mil) Outline Dimensions



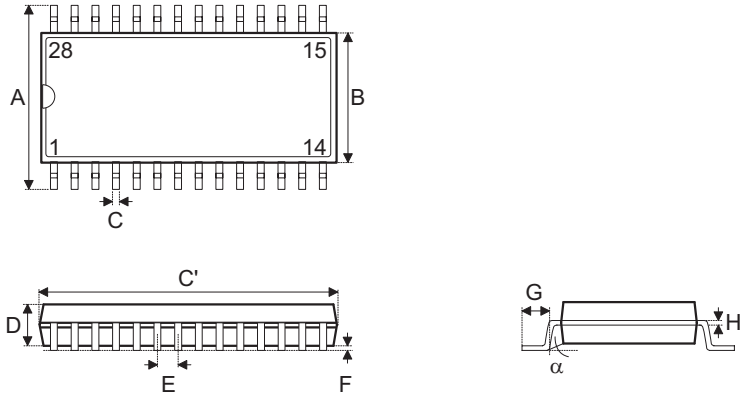
Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	1375	—	1395
B	278	—	298
C	125	—	135
D	125	—	145
E	16	—	20
F	50	—	70
G	—	100	—
H	295	—	315
I	330	—	375
α	0°	—	15°

28-pin SOP (300mil) Outline Dimensions



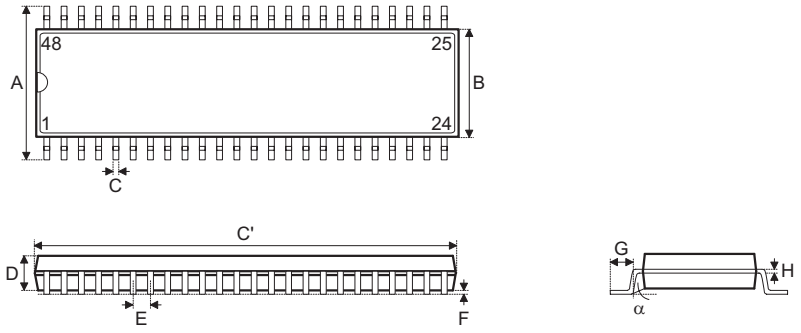
Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	394	—	419
B	290	—	300
C	14	—	20
C′	697	—	713
D	92	—	104
E	—	50	—
F	4	—	—
G	32	—	38
H	4	—	12
α	0°	—	10°

28-pin SSOP (150mil) Outline Dimensions



Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	228	—	244
B	150	—	157
C	8	—	12
C′	386	—	394
D	54	—	60
E	—	25	—
F	4	—	10
G	22	—	28
H	7	—	10
α	0°	—	8°

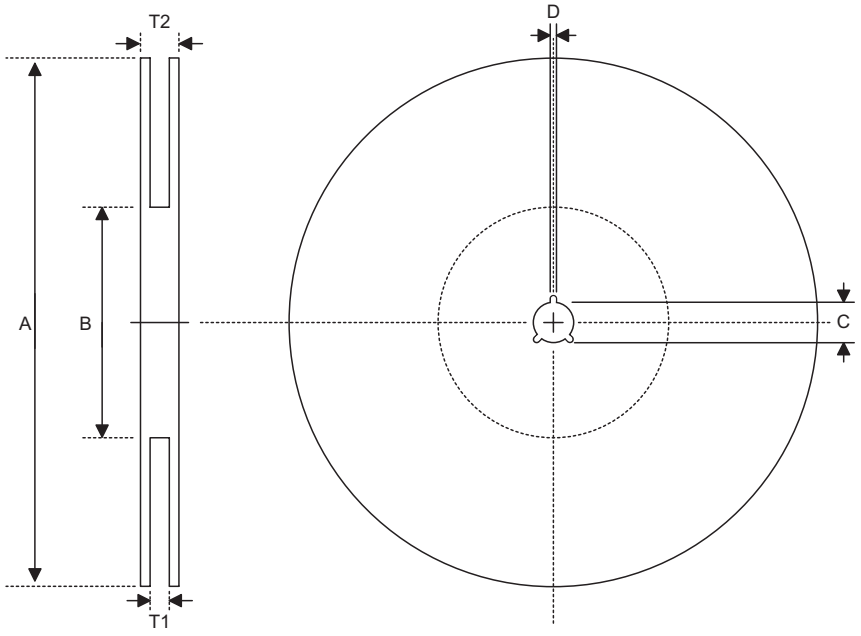
48-pin SSOP (300mil) Outline Dimensions



Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	395	—	420
B	291	—	299
C	8	—	12
C′	613	—	637
D	85	—	99
E	—	25	—
F	4	—	10
G	25	—	35
H	4	—	12
α	0°	—	8°

Product Tape and Reel Specifications

Reel Dimensions



SOP 28W (300mil)

Symbol	Description	Dimensions in mm
A	Reel Outer Diameter	330±1
B	Reel Inner Diameter	62±1.5
C	Spindle Hole Diameter	13+0.5 −0.2
D	Key Slit Width	2±0.5
T1	Space Between Flange	24.8+0.3 −0.2
T2	Reel Thickness	30.2±0.2

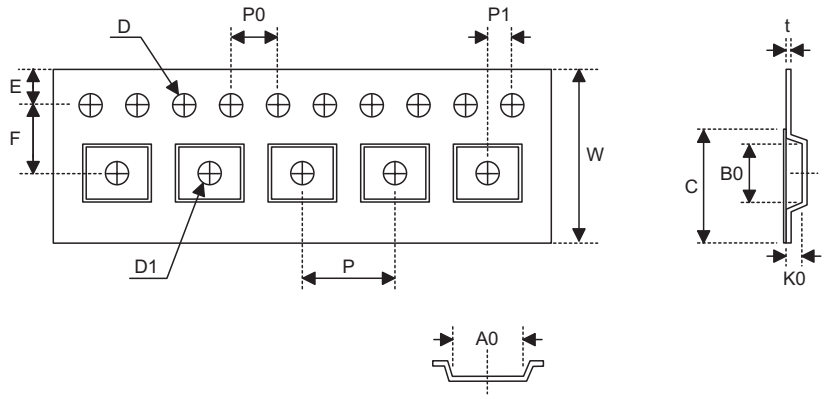
SSOP 28S (150mil)

Symbol	Description	Dimensions in mm
A	Reel Outer Diameter	330±1
B	Reel Inner Diameter	62±1.5
C	Spindle Hole Diameter	13+0.5 −0.2
D	Key Slit Width	2±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±1
B	Reel Inner Diameter	100±0.1
C	Spindle Hole Diameter	13+0.5 −0.2
D	Key Slit Width	2±0.5
T1	Space Between Flange	32.2+0.3 −0.2
T2	Reel Thickness	38.2±0.2

Carrier Tape Dimensions

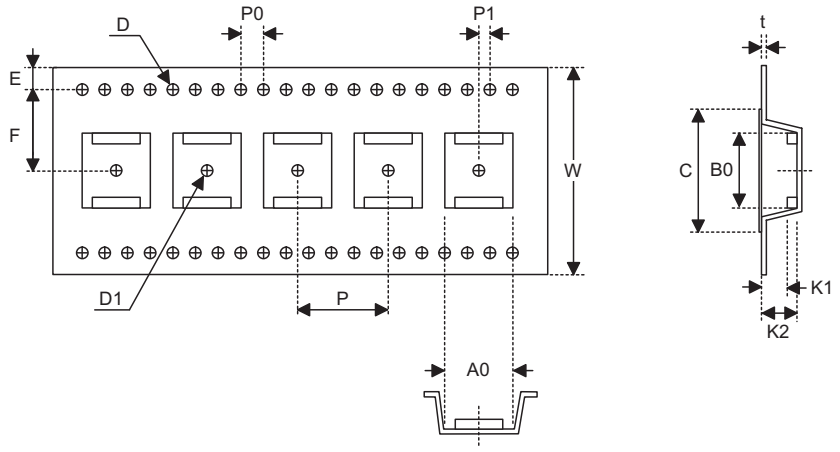


SOP 28W (300mil)

Symbol	Description	Dimensions in mm
W	Carrier Tape Width	24±0.3
P	Cavity Pitch	12±0.1
E	Perforation Position	1.75±0.1
F	Cavity to Perforation (Width Direction)	11.5±0.1
D	Perforation Diameter	1.5+0.1
D1	Cavity Hole Diameter	1.5+0.25
P0	Perforation Pitch	4±0.1
P1	Cavity to Perforation (Length Direction)	2±0.1
A0	Cavity Length	10.85±0.1
B0	Cavity Width	18.34±0.1
K0	Cavity Depth	2.97±0.1
t	Carrier Tape Thickness	0.35±0.01
C	Cover Tape Width	21.3

SSOP 28S (150mil)

Symbol	Description	Dimensions in mm
W	Carrier Tape Width	16±0.3
P	Cavity Pitch	8±0.1
E	Perforation Position	1.75±0.1
F	Cavity to Perforation (Width Direction)	7.5±0.1
D	Perforation Diameter	1.55+0.1
D1	Cavity Hole Diameter	1.5+0.25
P0	Perforation Pitch	4±0.1
P1	Cavity to Perforation (Length Direction)	2±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	0.3±0.05
C	Cover Tape Width	13.3



SSOP 48W

Symbol	Description	Dimensions in mm
W	Carrier Tape Width	32±0.3
P	Cavity Pitch	16±0.1
E	Perforation Position	1.75±0.1
F	Cavity to Perforation (Width Direction)	14.2±0.1
D	Perforation Diameter	2 Min.
D1	Cavity Hole Diameter	1.5+0.25
P0	Perforation Pitch	4±0.1
P1	Cavity to Perforation (Length Direction)	2±0.1
A0	Cavity Length	12±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

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. (Shanghai Sales Office)
7th Floor, Building 2, No.889, Yi Shan Rd., Shanghai, China 200233
Tel: 86-21-6485-5560
Fax: 86-21-6485-0313
http://www.holtek.com.cn

Holtek Semiconductor Inc. (Shenzhen Sales Office)
5/F, Unit A, Productivity Building, Cross of Science M 3rd Road and Gaoxin M 2nd Road, Science Park, Nanshan District, Shenzhen, China 518057
Tel: 86-755-8616-9908, 86-755-8616-9308
Fax: 86-755-8616-9722

Holtek Semiconductor Inc. (Beijing Sales Office)
Suite 1721, Jinyu Tower, A129 West Xuan Wu Men Street, Xicheng District, Beijing, China 100031
Tel: 86-10-6641-0030, 86-10-6641-7751, 86-10-6641-7752
Fax: 86-10-6641-0125

Holtek Semiconductor Inc. (Chengdu Sales Office)
709, Building 3, Champagne Plaza, No.97 Dongda Street, Chengdu, Sichuan, China 610016
Tel: 86-28-6653-6590
Fax: 86-28-6653-6591

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