HT48R05A-1/HT48C05
HT48R06A-1/HT48C06/HT48R08A-1
Cost-Effective I/O Type 8-Bit MCU

Note that the HT48R05A-1/HT48R08A-1/HT48C05/HT48C06 devices, although mentioned in this datasheet, have already been phased out and are presently no longer available.

Features

- Operating voltage:
  f_{SYS}=4MHz: 2.2V~5.5V
  f_{SYS}=8MHz: 3.3V~5.5V
- 13 bidirectional I/O lines
- An interrupt input shared with an I/O line
- 8-bit programmable timer/event counter with overflow interrupt and 8-stage prescaler
- On-chip crystal and RC oscillator
- Watchdog Timer
- Program memory ROM:
  512×14 for HT48R05A-1/HT48C05
  1024×14 for HT48R06A-1/HT48C06
  2048×14 for HT48R08A-1
- Data memory RAM
  32×8 for HT48R05A-1/HT48C05
  64×8 for HT48R06A-1/HT48C06
  96×8 for HT48R08A-1
- Buzzer driving pair and PFD supported
- HALT function and wake-up feature reduce power consumption
- Up to 0.5μs instruction cycle with 8MHz system clock at V_{DD}=5V
- All instructions in one or two machine cycles
- 14-bit table read instruction
- Two-level subroutine nesting
- Bit manipulation instruction
- Powerful instructions:
  62 for HT48R05A-1/HT48C05
  63 for HT48R06A-1/HT48C06 and HT48R08A-1
- Low voltage reset function
- 16-pin SSOP/NSOP package
  18-pin DIP/SOP package

General Description

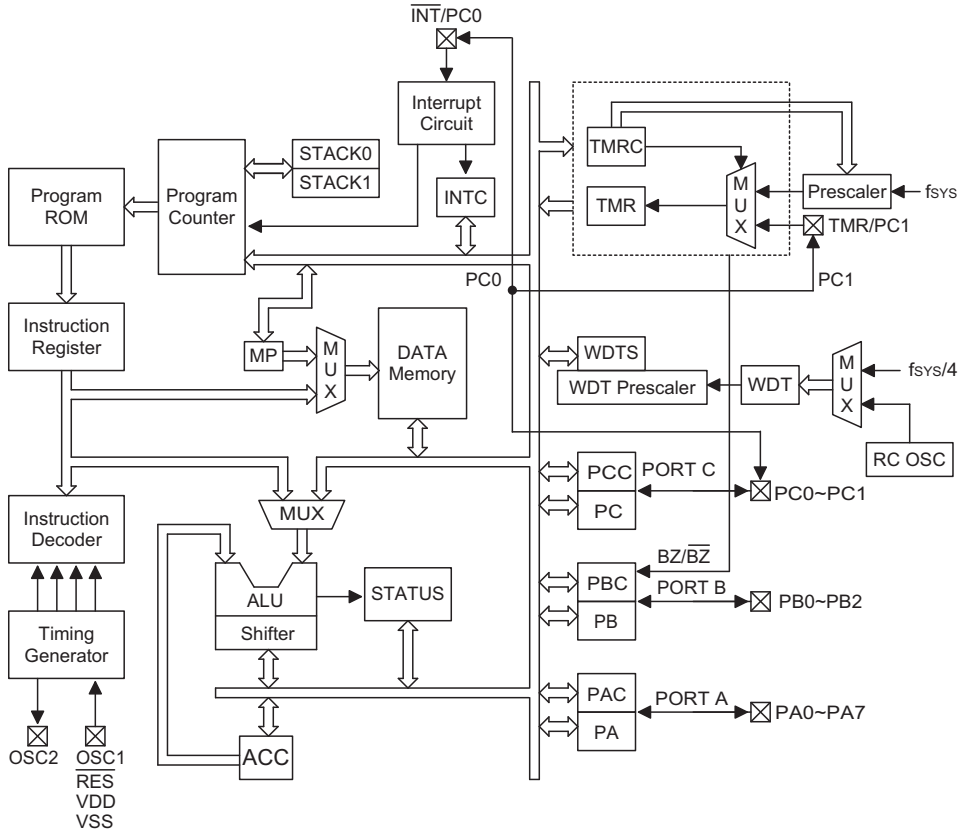
The HT48R05A-1/HT48C05, HT48R06A-1/HT48C06 and HT48R08A-1 are 8-bit high performance, RISC architecture microcontroller devices specifically designed for cost-effective multiple I/O control product applications. The mask version HT48C05 and HT48C06 are fully pin and functionally compatible with the OTP version HT48R05A-1 and HT48R06A-1 devices.

The advantages of low power consumption, I/O flexibility, timer functions, oscillator options, HALT and wake-up functions, Watchdog Timer, buzzer driver, as well as low cost, enhance the versatility of these devices to suit a wide range of application possibilities such as industrial control, consumer products, subsystem controllers, etc.

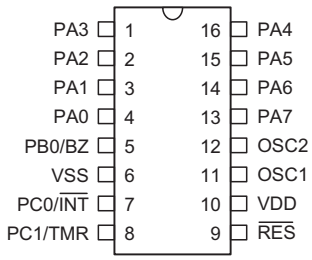
Selection Table

Part No.	VDD	Program Memory	Data Memory	I/O	Timer	Int.	PFD	Stack	Package Types
HT48R05A-1 HT48C05	2.2V~5.5V	0.5K×14	32×8	13	8-bit×1	2	√	2	16SSOP/NSOP, 18DIP/SOP
HT48R06A-1	2.2V~5.5V	1K×14	64×8	13	8-bit×1	2	√	2	16SSOP/NSOP, 18SOP
HT48C06	2.2V~5.5V	1K×14	64×8	13	8-bit×1	2	√	2	16SSOP/NSOP, 18DIP/SOP
HT48R08A-1	2.2V~5.5V	2K×14	96×8	13	8-bit×1	2	√	2	16SSOP/NSOP, 18DIP/SOP

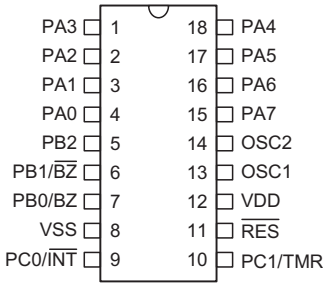
Block Diagram



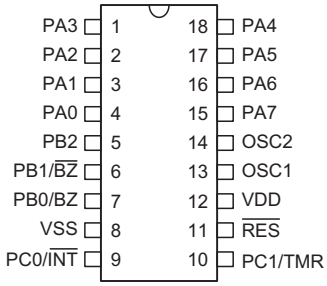
Pin Assignment



HT48R05A-1/HT48C05
HT48R06A-1/HT48C06
HT48R08A-1
16 SSOP-A/NSOP-A



HT48R05A-1/HT48C05
HT48C06
HT48R08A-1
18 DIP-A/SOP-A



HT48R06A-1
18 SOP-A

Pin Description

Pin Name	I/O	Options	Description
PA0~PA7	I/O	Pull-high* Wake-up	Bidirectional 8-bit input/output port. Each bit can be configured as wake-up input by options. Software instructions determine the CMOS output or Schmitt trigger input with a pull-high resistor (determined by pull-high options).
PB0/BZ PB1/$\overline{BZ}$ PB2	I/O	Pull-high* I/O or BZ/$\overline{BZ}$	Bidirectional 3-bit input/output port. Software instructions determine the CMOS output or Schmitt trigger input with a pull-high resistor (determined by pull-high options). The PB0 and PB1 are pin-shared with the BZ and $\overline{BZ}$, respectively. Once the PB0 and PB1 are selected as buzzer driving outputs, the output signals come from an internal PFD generator (shared with a timer/event counter).
VSS	—	—	Negative power supply, ground
PC0/$\overline{INT}$ PC1/TMR	I/O	Pull-high*	Bidirectional I/O lines. Software instructions determine the CMOS output or Schmitt trigger input with a pull-high resistor (determined by pull-high options). The external interrupt and timer input are pin-shared with the PC0 and PC1, respectively. The external interrupt input is activated on a high to low transition.
$\overline{RES}$	I	—	Schmitt trigger reset input. Active low
VDD	—	—	Positive power supply
OSC1 OSC2	I O	Crystal or RC	OSC1, OSC2 are connected to an RC network or Crystal (determined by options) for the internal system clock. In the case of RC operation, OSC2 is the output terminal for 1/4 system clock.

* All pull-high resistors are controlled by an option bit.

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 ..... –60°C to 150°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
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

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
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	
V_{IL1}	Input Low Voltage for I/O Ports, TMR and $\overline{INT}$	—	—	0	—	$0.3V_{DD}$	V
V_{IH1}	Input High Voltage for I/O Ports, TMR and $\overline{INT}$	—	—	$0.7V_{DD}$	—	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	—	
I_{OH}	I/O Port Source Current	3V	$V_{OH}=0.9V_{DD}$	−2	−4	—	mA
		5V		−5	−10	—	
R_{PH}	Pull-high Resistance	3V	—	20	60	100	kΩ
		5V		10	30	50	

A.C. Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
f_{SYS}	System Clock (Crystal OSC, RC OSC)	—	2.2V~5.5V	400	—	4000	kHz
		—	3.3V~5.5V	400	—	8000	kHz
f_{TIMER}	Timer I/P Frequency (TMR)	—	2.2V~5.5V	0	—	4000	kHz
		—	3.3V~5.5V	0	—	8000	kHz
t_{WDTOSC}	Watchdog Oscillator Period	3V	—	45	90	180	μs
		5V		32	65	130	
t_{WDT1}	Watchdog Time-out Period (RC)	3V	Without WDT prescaler	11	23	46	ms
		5V		8	17	33	
t_{WDT2}	Watchdog Time-out Period (System Clock)	—	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_{INT}	Interrupt Pulse Width	—	—	1	—	—	μs
t_{LVR}	Low Voltage Width to Reset	—	—	0.25	1	2	ms

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

Functional Description

Execution Flow

The system clock for the microcontroller is derived from either a crystal or an RC oscillator. The system clock is internally divided into four non-overlapping clocks. One instruction cycle consists of four system clock cycles.

Instruction fetching and execution are pipelined in such a way that a fetch takes an instruction cycle while decoding and execution takes the next instruction cycle. However, the pipelining scheme causes each instruction to effectively execute in a cycle. If an instruction changes the program counter, two cycles are required to complete the instruction.

Program Counter – PC

The program counter (PC) controls the sequence in which the instructions stored in program ROM are executed and its contents specify full range of program memory.

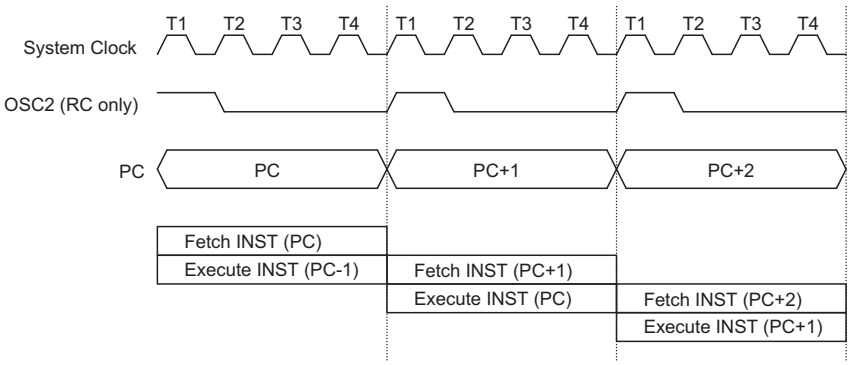
After accessing a program memory word to fetch an instruction code, the contents of the program counter are incremented by one. The program counter then points to the memory word containing the next instruction code.

When executing a jump instruction, conditional skip execution, loading PCL register, subroutine call, initial reset, internal interrupt, external interrupt or return from subroutine, the PC manipulates the program transfer by loading the address corresponding to each instruction.

The conditional skip is activated by instructions. Once the condition is met, the next instruction, fetched during the current instruction execution, is discarded and a dummy cycle replaces it to get the proper instruction. Otherwise proceed with the next instruction.

The lower byte of the program counter (PCL) is a readable and writable register (06H). Moving data into the PCL performs a short jump. The destination will be within 256 locations.

When a control transfer takes place, an additional dummy cycle is required.



Execution Flow

Mode	Program Counter										
	***10**	***9**	***8**	***7**	***6**	***5**	***4**	***3**	***2**	***1**	***0**
Initial Reset	0	0	0	0	0	0	0	0	0	0	0
External Interrupt	0	0	0	0	0	0	0	0	1	0	0
Timer/Event Counter Overflow	0	0	0	0	0	0	0	1	0	0	0
Skip	Program Counter+2										
Loading PCL	*10	*9	*8	@7	@6	@5	@4	@3	@2	@1	@0
Jump, Call Branch	#10	#9	#8	#7	#6	#5	#4	#3	#2	#1	#0
Return from Subroutine	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0

Program Counter

Note: *10~*0: Program Counter bits
S10~S0: Stack register bits
#10~#0: Instruction code bits
@7~@0: PCL bits

For HT48R05A-1/HT48C05, the Program Counter is 9 bits wide, i.e. from *8~*0

For HT48R06A-1/HT48C06, the Program Counter is 10 bits wide, i.e. from *9~*0

For HT48R08A-1, the Program Counter is 11 bits wide, i.e. from *10~*0

Program Memory – ROM

The program memory is used to store the program instructions which are to be executed. It also contains data, table, and interrupt entries, and is organized into 512×14 bits (HT48R05A-1/HT48C05), 1024×14 bits (HT48R06A-1/HT48C06) or 2048×14 bits (HT48R08A-1), addressed by the program counter and table pointer.

Certain locations in the program memory are reserved for special usage:

- Location 000H

  This area is reserved for program initialization. After chip reset, the program always begins execution at location 000H.

- Location 004H

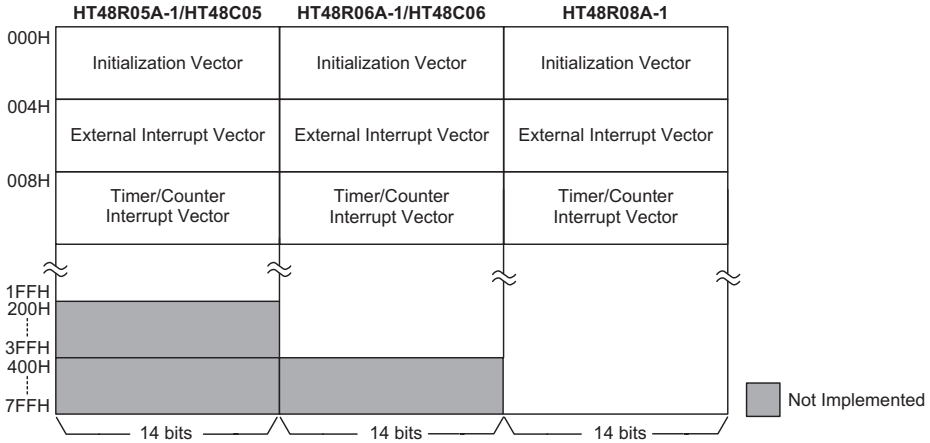
  This area is reserved for the external interrupt service program. If the $\overline{INT}$ input pin is activated, the interrupt is enabled and the stack is not full, the program begins execution at location 004H.

- Location 008H

  This area is reserved for the timer/event counter interrupt service program. If a timer interrupt results from a timer/event counter overflow, and if the interrupt is enabled and the stack is not full, the program begins execution at location 008H.

- Table location

  Any location in the program memory can be used as look-up tables. The instructions "TABRDC [m]" (the current page, 1 page=256 words) and "TABRDL [m]" (the last page; However this statement is not valid for the HT48R05A-1/HT48C05 devices) transfer the contents of the lower-order byte to the specified data memory, and the higher-order byte to TBLH (08H). Only the destination of the lower-order byte in the table is well-defined, the other bits of the table word are transferred to the lower portion of TBLH, and the remaining 2 bits are read as "0". The Table Higher-order byte register (TBLH) is read only. The table pointer (TBLP) is a read/write register (07H), which indicates the table location. Before accessing the table, the location must be placed in TBLP. The TBLH is read only and cannot be restored. If the main routine and the ISR (Interrupt Service Routine) both employ the table read instruction, the contents of the TBLH in the main routine are likely to be changed by the table read instruction used in the ISR. Errors can occur. In other words, using the table read instruction in the main routine and the ISR simultaneously should be avoided. However, if the table read instruction has to be applied in both the main routine and the ISR, the interrupt is supposed to be disabled prior to the table read instruction. It will not be enabled until the TBLH has



Program Memory

Instruction	Table Location										
	*10	*9	*8	*7	*6	*5	*4	*3	*2	*1	*0
TABRDC [m]	P10	P9	P8	@7	@6	@5	@4	@3	@2	@1	@0

Table Location

Note: *10~*0: Table location bits P10~P8: Current program counter bits

@7~@0: Table pointer bits

For HT48R05A-1/HT48C05, the table address location is 9 bits, i.e. from *8~*0

For HT48R06A-1/HT48C06, the table address location is 10 bits, i.e. from *9~*0

For HT48R08A-1, the table address location is 11 bits, i.e. from *10~*0

been backed up. All table related instructions require two cycles to complete the operation. These areas may function as normal program memory depending upon the requirements.

Stack Register – STACK

This is a special part of the memory which is used to save the contents of the Program Counter only. The stack is organized into 2 levels and is neither part of the data nor part of the program space, and is neither readable nor writable. The activated level is indexed by the stack pointer (SP) and is neither readable nor writeable. At a subroutine call or interrupt acknowledgment, 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 chip reset, the SP will point to the top of the stack.

If the stack is full and a non-masked interrupt takes place, the interrupt request flag will be recorded but the acknowledgment 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. In a similar case, if the stack is full and a ″CALL″ is subsequently executed, stack overflow occurs and the first entry will be lost (only the most recent 2 return addresses are stored).

Data Memory – RAM

The data memory is designed with 49×8 bits (HT48R05A-1/HT48C05), 81×8 bits (HT48R06A-1/HT48C06) or 113×8 bits (HT48R08A-1). The data memory is divided into two functional groups: special function registers and general purpose data memory 32×8 (HT48R05A-1/HT48C05), 64×8 (HT48R06A-1/HT48C06) or 96×8 (HT48R08A-1). Most are read/write, but some are read only.

The special function registers include the indirect addressing register (00H), timer/event counter (TMR;0DH), timer/event counter control register (TMRC;0EH), program counter lower-order byte register (PCL;06H), memory pointer register (MP;01H), accumulator (ACC;05H), table pointer (TBLP;07H), table higher-order byte register (TBLH;08H), status register (STATUS;0AH), interrupt control register (INTC;0BH), Watchdog Timer option setting register (WDTS;09H), I/O registers (PA;12H, PB;14H, PC;16H) and I/O control registers (PAC;13H, PBC;15H, PCC;17H). The remaining space before the 60H (HT48R05A-1/HT48C05), 40H (HT48R06A-1/HT48C06) or 20H (HT48R08A-1) is reserved for future expanded usage and reading these locations will get ″00H″. The general purpose data memory, addressed from 60H to 7FH (HT48R05A-1/HT48C05), 40H to 7FH (HT48R06A-1/HT48C06) or 20H to 7FH (HT48R08A-1), is used for data and control information under instruction commands.

All of the data memory areas can handle arithmetic, logic, increment, decrement and rotate operations directly. Except for some dedicated bits, each bit in the data memory can be set and reset by ″SET [m].i″ and ″CLR [m].i″. They are also indirectly accessible through memory pointer register (MP;01H).

Indirect Addressing Register

Location 00H is an indirect addressing register that is not physically implemented. Any read/write operation of [00H] accesses data memory pointed to by MP (01H). Reading location 00H itself indirectly will return the result 00H. Writing indirectly results in no operation.

The memory pointer register MP (01H) is a 7-bit register. The bit 7 of MP is undefined and reading will return the result ″1″. Any writing operation to MP will only transfer the lower 7-bit data to MP.

Accumulator

The accumulator is closely related to ALU operations. It is also mapped to location 05H of the data memory and can carry out immediate data operations. The data movement between two data memory locations must pass through the accumulator.

Arithmetic and Logic Unit – ALU

This circuit performs 8-bit arithmetic and logic operations. The ALU provides the following functions:

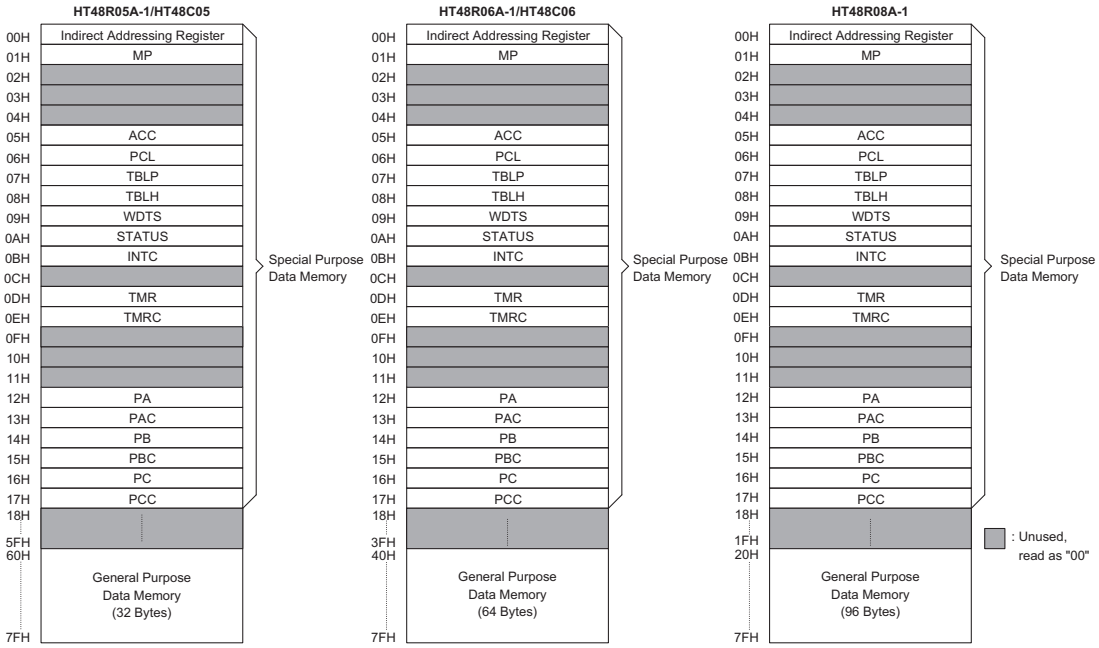
- Arithmetic operations (ADD, ADC, SUB, SBC, DAA)
- Logic operations (AND, OR, XOR, CPL)
- Rotation (RL, RR, RLC, RRC)
- Increment and Decrement (INC, DEC)
- Branch decision (SZ, SNZ, SIZ, SDZ ....)

The ALU not only saves the results of a data operation but also changes the status register.

Status Register – STATUS

This 8-bit register (0AH) 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). It also records the status information and controls the operation sequence.

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 from those intended. The TO flag can be affected only by system power-up, a WDT time-out or executing the ″CLR WDT″ or ″HALT″ instruction. The PDF flag can be affected only by executing the ″HALT″ or ″CLR WDT″ instruction or a system power-up.



RAM Mapping

Bit No.	Label	Function
0	C	C is set if the 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.
1	AC	AC is set if the 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.
2	Z	Z is set if the result of an arithmetic or logic operation is zero; otherwise Z is cleared.
3	OV	OV is set if the 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.
4	PDF	PDF is cleared by system power-up or executing the ″CLR WDT″ instruction. PDF is set by executing the ″HALT″ instruction.
5	TO	TO is cleared by system power-up or executing the ″CLR WDT″ or ″HALT″ instruction. TO is set by a WDT time-out.
6~7	—	Unused bit, read as ″0″

Status (0AH) Register

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

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

Interrupt

The device provides an external interrupt and internal timer/event counter interrupts. The Interrupt Control Register (INTC;0BH) contains the interrupt control bits to set the enable or disable and the interrupt request flags.

Once an interrupt subroutine is serviced, all the other interrupts will be blocked (by clearing the EMI bit). This scheme may prevent any further interrupt nesting. Other interrupt requests may happen during this interval but only the interrupt request flag is recorded. If a certain interrupt requires servicing within the service routine, the EMI bit and the corresponding bit of INTC may be set 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 SP is decremented. If immediate service is desired, the stack must be prevented from becoming full.

All these kinds of interrupts have a wake-up capability. As an interrupt is serviced, a control transfer occurs by

Bit No.	Label	Function
0	EMI	Controls the master (global) interrupt (1= enabled; 0= disabled)
1	EEI	Controls the external interrupt (1= enabled; 0= disabled)
2	ETI	Controls the timer/event counter interrupt (1= enabled; 0= disabled)
3, 6~7	—	Unused bit, read as ″0″
4	EIF	External interrupt request flag (1= active; 0= inactive)
5	TF	Internal timer/event counter request flag (1= active; 0= inactive)

INTC (0BH) Register

pushing the program counter onto the stack, followed by a branch to a subroutine at specified location in the program memory. Only the program counter is pushed onto the stack. If the contents of the register or status register (STATUS) are altered by the interrupt service program which corrupts the desired control sequence, the contents should be saved in advance.

External interrupts are triggered by a high to low transition of $\overline{INT}$ and the related interrupt request flag (EIF; bit 4 of INTC) will be set. When the interrupt is enabled, the stack is not full and the external interrupt is active, a subroutine call to location 04H will occur. The interrupt request flag (EIF) and EMI bits will be cleared to disable other interrupts.

The internal timer/event counter interrupt is initialized by setting the timer/event counter interrupt request flag (TF; bit 5 of INTC), caused by a timer overflow. When the interrupt is enabled, the stack is not full and the TF bit is set, a subroutine call to location 08H will occur. The related interrupt request flag (TF) will be reset and the EMI bit cleared to disable further interrupts.

During the execution of an interrupt subroutine, other interrupt acknowledgments are held until the ″RETI″ instruction is executed or the EMI bit and the related interrupt control bit are set to 1 (of course, if the stack is not full). To return from the interrupt subroutine, ″RET″ or ″RETI″ may be invoked. RETI will set the EMI bit to enable an interrupt service, but RET will not.

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 the case of simultaneous requests the following table shows the priority that is applied. These can be masked by resetting the EMI bit.

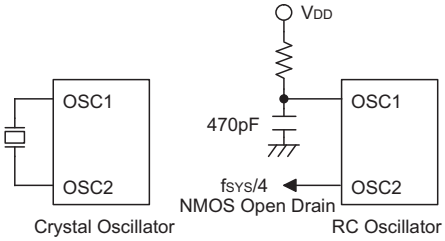
No.	Interrupt Source	Priority	Vector
a	External Interrupt	1	04H
b	Timer/Event Counter Overflow	2	08H

The timer/event counter interrupt request flag (TF), external interrupt request flag (EIF), enable timer/event counter bit (ETI), enable external interrupt bit (EEI) and enable master interrupt bit (EMI) constitute an interrupt control register (INTC) which is located at 0BH in the data memory. EMI, EEI, ETI are used to control the enabling/disabling of interrupts. These bits prevent the requested interrupt from being serviced. Once the interrupt request flags (TF, EIF) are set, they will remain in the INTC register until the interrupts are serviced or cleared by a software instruction.

It is recommended that a program does not use the ″CALL subroutine″ 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 enabling the interrupt is not well controlled, the original control sequence will be damaged once the ″CALL″ operates in the interrupt subroutine.

Oscillator Configuration

There are two oscillator circuits in the microcontroller.



System Oscillator

Both are designed for system clocks, namely the RC oscillator and the Crystal oscillator, which are determined by the options. No matter what oscillator type is selected, the signal provides the system clock. The HALT mode stops the system oscillator and ignores an external signal to conserve power.

If an RC oscillator is used, an external resistor between OSC1 and VDD is required and the resistance must range from 24kΩ to 1MΩ. The system clock, divided by 4, is available on OSC2, which can be used to synchronize external logic. The RC oscillator provides the most cost effective solution. However, the frequency of oscillation may vary with VDD, temperatures and the chip itself due to process variations. It is, therefore, not suitable for timing sensitive operations where an accurate oscillator frequency is desired.

If the Crystal oscillator is used, a crystal across OSC1 and OSC2 is needed to provide the feedback and phase shift required for the oscillator, and no other external components are required. Instead of a crystal, a resonator can also be connected between OSC1 and OSC2 to get a frequency reference, but two external capacitors in OSC1 and OSC2 are required (If the oscillating frequency is less than 1MHz).

The WDT oscillator is a free running on-chip RC oscillator, and no external components are required. Even if the system enters the power down mode, the system clock is stopped, but the WDT oscillator still works with a period of approximately 65μs at 5V. The WDT oscillator can be disabled by options to conserve power.

Watchdog Timer – WDT

The clock source of WDT is implemented by a dedicated RC oscillator (WDT oscillator) or instruction clock (system clock divided by 4), decided by options. This timer is designed to prevent a software malfunction or sequence from jumping to an unknown location with unpredictable results. The Watchdog Timer can be disabled by an option. If the Watchdog Timer is disabled, all the executions related to the WDT result in no operation.

Once the internal WDT oscillator (RC oscillator with a period of 65μs at 5V normally) is selected, it is first divided by 256 (8-stage) to get the nominal time-out period of approximately 17ms at 5V. This time-out period may vary with temperatures, VDD and process variations. By invoking the WDT prescaler, longer time-out periods can be realized. Writing data to WS2, WS1, WS0 (bit 2,1,0 of the WDTS) can give different time-out periods. If WS2, WS1 and WS0 are all equal to "1", the division ratio is up to 1:128, and the maximum time-out period is 2.1s at 5V seconds. If the WDT oscillator is disabled, the WDT clock may still come from the instruction clock and operate in the same manner except that in the HALT state the WDT may stop counting and lose its protecting purpose. In this situation the logic can only be restarted by external logic. The high nibble and bit 3 of the WDTS are reserved for user's defined flags, which can be used to indicate some specified status.

If the device operates in a noisy environment, using the on-chip RC oscillator (WDT OSC) is strongly recommended, since the HALT will stop the system clock.

WS2	WS1	WS0	Division Ratio
0	0	0	1:1
0	0	1	1:2
0	1	0	1:4
0	1	1	1:8
1	0	0	1:16
1	0	1	1:32
1	1	0	1:64
1	1	1	1:128

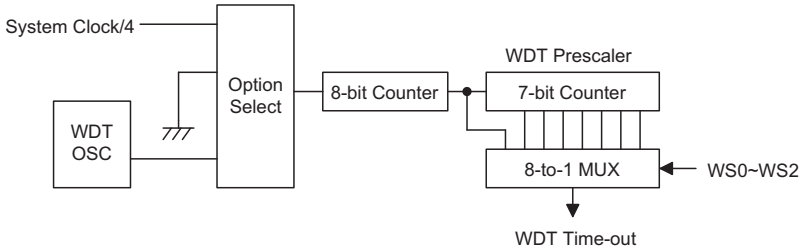
WDTS (09H) Register

The WDT overflow under normal operation will initialize "chip reset" and set the status bit "TO". But in the HALT mode, the overflow will initialize a "warm reset", and only the Program Counter and SP are reset to zero. To clear the contents of WDT (including the WDT prescaler), three methods are adopted; external reset (a low level to RES), software instruction and a "HALT" instruction. The software instruction include "CLR WDT" and the other set – "CLR WDT1" and "CLR WDT2". Of these two types of instruction, only one can be active depending on the option – "CLR WDT times selection option". If the "CLR WDT" is selected (i.e. CLRWDT times equal one), any execution of the "CLR WDT" instruction will clear the WDT. In the case that "CLR WDT1" and "CLR WDT2" are chosen (i.e. CLRWDT times equal two), these two instructions must be executed to clear the WDT; otherwise, the WDT may reset the chip as a result of time-out.

Power Down Operation – HALT

The HALT mode is initialized by the "HALT" instruction and results in the following...

- The system oscillator will be turned off but the WDT oscillator keeps running (if the WDT oscillator is selected).
- The contents of the on chip RAM and registers remain unchanged.
- WDT and WDT prescaler will be cleared and recounted again (if the WDT clock is from the WDT oscillator).
- All of the I/O ports maintain their original status.
- The PDF flag is set and the TO flag is cleared.



Watchdog Timer

The system can leave the HALT mode by means of an external reset, an interrupt, an external falling edge signal on port A or a WDT overflow. An external reset causes a device initialization and the WDT overflow performs a "warm reset". After the TO and PDF flags are examined, the reason for chip reset can be determined. The PDF flag is cleared by system power-up or executing the "CLR WDT" instruction and is set when executing the "HALT" instruction. The TO flag is set if the WDT time-out occurs, and causes a wake-up that only resets the Program Counter and SP; the others keep their original status.

The port A wake-up and interrupt methods can be considered as a continuation of normal execution. Each bit in port A can be independently selected to wake up the device by the options. Awakening from an I/O port stimulus, the program will resume execution of the next instruction. If it is awakening from an interrupt, two sequences may happen. If the related interrupt is disabled or the interrupt is enabled but the stack is full, the program will resume execution at the next instruction. If the interrupt is enabled and the stack is not full, the regular interrupt response takes place. If an interrupt request flag is set to "1" before entering the HALT mode, the wake-up function of the related interrupt will be disabled. Once a wake-up event occurs, it takes 1024 t_{SYS} (system clock period) to resume normal operation. In other words, a dummy period will be inserted after wake-up. If the wake-up results from an interrupt acknowledgment, the actual interrupt subroutine execution will be delayed by one or more cycles. If the wake-up results in the next instruction execution, this will be executed immediately after the dummy period is finished.

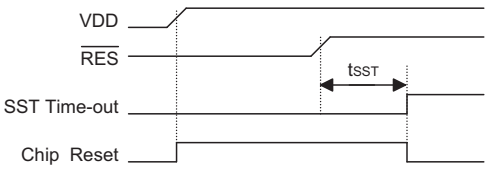
To minimize power consumption, all the I/O pins should be carefully managed before entering the HALT status.

Reset

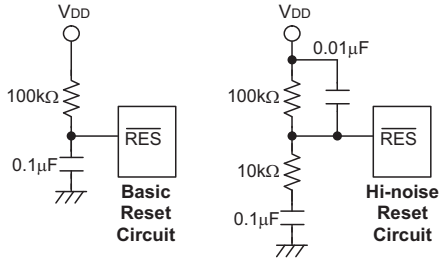
There are three ways in which a reset can occur:

- $\overline{RES}$ reset during normal operation
- $\overline{RES}$ reset during HALT
- WDT time-out reset during normal operation

The WDT time-out during HALT is different from other chip reset conditions, since it can perform a "warm reset" that resets only the Program Counter and SP, leaving the other circuits in their original state. Some registers remain unchanged during other reset conditions. Most registers are reset to the "initial condition" when the reset conditions are met. By examining the PDF and TO flags, the program can distinguish between different "chip resets".

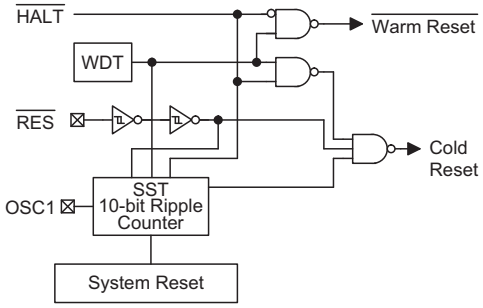


Reset Timing Chart



Reset Circuit

Note: Most applications can use the Basic Reset Circuit as shown, however for applications with extensive noise, it is recommended to use the Hi-noise Reset Circuit.



Reset Configuration

TO	PDF	RESET Conditions
0	0	$\overline{RES}$ reset during power-up
u	u	$\overline{RES}$ reset during normal operation
0	1	$\overline{RES}$ wake-up HALT
1	u	WDT time-out during normal operation
1	1	WDT wake-up HALT

Note: "u" stands for "unchanged"

To guarantee that the system oscillator is started and stabilized, the SST (System Start-up Timer) provides an extra-delay of 1024 system clock pulses when the system reset (power-up, WDT time-out or $\overline{RES}$ reset) or the system awakes from the HALT state.

When a system reset occurs, the SST delay is added during the reset period. Any wake-up from HALT will enable the SST delay.

An extra option load time delay is added during system reset (power-up, WDT time-out at normal mode or $\overline{RES}$ reset).

The functional unit chip reset status are shown below.

Program Counter	000H
Interrupt	Disable
Prescaler	Clear
WDT	Clear. After master reset, WDT begins counting
Timer/Event Counter	Off
Input/Output Ports	Input mode
Stack Pointer	Points to the top of the stack

The states of the registers is summarized in the table.

Register	Reset (Power-on)	WDT time-out (Normal Operation)	$\overline{RES}$ Reset (Normal Operation)	$\overline{RES}$ Reset (HALT)	WDT Time-out (HALT)*
Program Counter	000H	000H	000H	000H	000H
MP	-xxx xxxx	-uuu uuuu	-uuu uuuu	-uuu uuuu	-uuu uuuu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	--xx xxxx	--uu uuuu	--uu uuuu	--uu uuuu	--uu uuuu
WDTS	0000 0111	0000 0111	0000 0111	0000 0111	uuuu uuuu
STATUS	xxxx	--1u uuuu	--uu uuuu	--01 uuuu	--11 uuuu
INTC	--00 -000	--00 -000	--00 -000	--00 -000	--uu -uuu
TMR	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMRC	00-0 1000	00-0 1000	00-0 1000	00-0 1000	uu-u uuuu
PA	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PB	---- -111	---- -111	---- -111	---- -111	---- -uuu
PBC	---- -111	---- -111	---- -111	---- -111	---- -uuu
PC	---- --11	---- --11	---- --11	---- --11	---- --uu
PCC	---- --11	---- --11	---- --11	---- --11	---- --uu

Note: "*" means "warm reset"
"u" means "unchanged"
"x" means "unknown"

Timer/Event Counter

A timer/event counter (TMR) is implemented in the microcontroller. The timer/event counter contains an 8-bit programmable count-up counter and the clock may come from an external source or the system clock.

Using external clock input allows the user to count external events, measure time internals or pulse widths, or generate an accurate time base. While using the internal clock allows the user to generate an accurate time base.

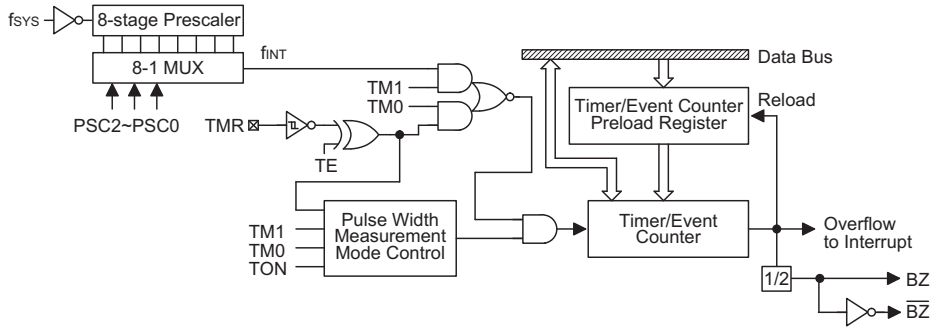
The timer/event counter can generate PFD signal by using external or internal clock and PFD frequency is determine by the equation $f_{INT}/[2\times(256-N)]$.

There are 2 registers related to the timer/event counter; TMR ([0DH]), TMRC ([0EH]). Two physical registers are mapped to TMR location; writing TMR makes the starting value be placed in the timer/event counter preload register and reading TMR retrieves the contents of the timer/event counter. The TMRC is a timer/event counter control register, which defines some options.

The TM0, TM1 bits define the operating mode. The event count mode is used to count external events, which means the clock source comes from an external (TMR) pin. The timer mode functions as a normal timer with the clock source coming from the f_{INT} clock. The pulse width measurement mode can be used to count the high or low level duration of the external signal (TMR). The counting is based on the f_{INT} clock.

In the event count or timer mode, once the timer/event counter starts counting, it will count from the current contents in the timer/event counter to FFH. Once overflow occurs, the counter is reloaded from the timer/event counter preload register and generates the interrupt request flag (TF; bit 5 of INTC) at the same time.

In the pulse width measurement mode with the TON and TE bits equal to one, once the TMR has received a transient from low to high (or high to low if the TE bits is "0") it will start counting until the TMR returns to the original level and resets the TON. The measured result will remain in the timer/event counter even if the activated transient occurs again. In other words, only one cycle measurement can be done. Until setting the TON, the cycle measurement will function again as long as it receives further transient pulse. Note that, in this operating mode, the timer/event counter starts counting not according to the logic level but according to the transient edges. In the case of counter overflows, the counter is reloaded from the timer/event counter preload register and issues the interrupt request just like the other two modes. To enable the counting operation, the timer ON bit (TON; bit 4 of TMRC) should be set to 1. In the pulse width measurement mode, the TON will be cleared automatically after the measurement cycle is completed. But in the other two modes the TON can only be reset by instructions. The overflow of the timer/event counter is one of the wake-up sources. No matter what the operation mode is, writing a 0 to ETI can disable the interrupt service.



Timer/Event Counter

Bit No.	Label	Function
0~2	PSC0~PSC2	To define the prescaler stages, PSC2, PSC1, PSC0= 000: $f_{INT}=f_{SYS}/2$ 001: $f_{INT}=f_{SYS}/4$ 010: $f_{INT}=f_{SYS}/8$ 011: $f_{INT}=f_{SYS}/16$ 100: $f_{INT}=f_{SYS}/32$ 101: $f_{INT}=f_{SYS}/64$ 110: $f_{INT}=f_{SYS}/128$ 111: $f_{INT}=f_{SYS}/256$
3	TE	To define the TMR active edge of the timer/event counter (0=active on low to high; 1=active on high to low)
4	TON	To enable or disable timer counting (0=disabled; 1=enabled)
5	—	Unused bit, read as "0"
6 7	TM0 TM1	To define the operating mode 01=Event count mode (external clock) 10=Timer mode (internal clock) 11=Pulse width measurement mode 00=Unused

TMRC (0EH) Register

In the case of timer/event counter OFF condition, writing data to the timer/event counter preload register will also reload that data to the timer/event counter. But if the timer/event counter is turned on, data written to it will only be kept in the timer/event counter preload register. The timer/event counter will still operate until overflow occurs. When the timer/event counter (reading TMR) is read, the clock will be blocked to avoid errors. As clock blocking may results in a counting error, this must be taken into consideration by the programmer.

The bit0~2 of the TMRC can be used to define the pre-scaling stages of the internal clock sources of the timer/event counter. The definitions are as shown. The overflow signal of the timer/event counter can be used to generate PFD signals for buzzer driving.

Input/Output Ports

There are 13 bidirectional input/output lines in the microcontroller, labeled from PA to PC, which are mapped to the data memory of [12H], [14H] and [16H] respectively. All of these I/O ports can be used for input and output operations. For input operation, these ports are non-latching, that is, the inputs must be ready at the T2 rising edge of instruction "MOV A,[m]" (m=12H, 14H or 16H). For output operation, all the data is latched and remains unchanged until the output latch is rewritten.

Each I/O line has its own control register (PAC, PBC, PCC) to control the input/output configuration. With this control register, CMOS output or Schmitt trigger input with or without pull-high resistor structures can be re-configured dynamically (i.e. on-the-fly) under software control. To function as an input, the corresponding latch of the control register must write "1". The input source also depends on the control register. If the control register bit is "1", the input will read the pad state. If the control register bit is "0", the contents of the latches will move to the internal bus. The latter is possible in the "read-modify-write" instruction.

For output function, CMOS is the only configuration. These control registers are mapped to locations 13H, 15H and 17H.

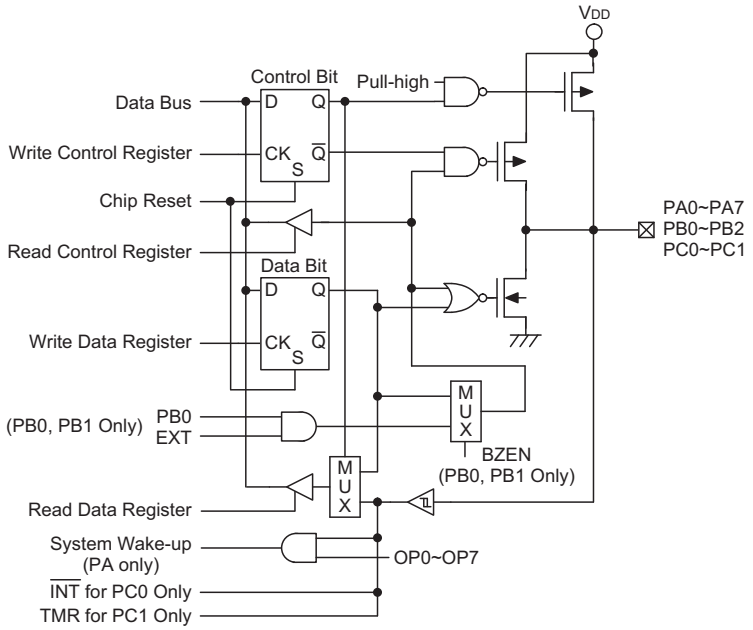
After a chip reset, these input/output lines remain at high levels or floating state (dependent on pull-high options). Each bit of these input/output latches can be set or cleared by "SET [m].i" and "CLR [m].i" (m=12H, 14H or 16H) instructions.

Some instructions first input data and then follow the output operations. For example, "SET [m].i", "CLR [m].i", "CPL [m]", "CPLA [m]" read the entire port states into the CPU, execute the defined operations (bit-operation), and then write the results back to the latches or the accumulator.

Each line of port A has the capability of waking-up the device. The highest 6-bit of port C and 5 bits of port B are not physically implemented; on reading them a "0" is returned whereas writing then results in a no-operation. See Application note.

There is a pull-high option available for all I/O lines. Once the pull-high option is selected, all I/O lines have pull-high resistors. Otherwise, the pull-high resistors are absent. It should be noted that a non-pull-high I/O line operating in input mode will cause a floating state.

The PB0 and PB1 are pin-shared with BZ and $\overline{BZ}$ signal, respectively. If the BZ/$\overline{BZ}$ option is selected, the output



Input/Output Ports

signal in output mode of PB0/PB1 will be the PFD signal generated by timer/event counter overflow signal. The input mode always remaining its original functions. Once the BZ/$\overline{BZ}$ option is selected, the buzzer output signals are controlled by PB0 data register only. The I/O functions of PB0/PB1 are shown below.

PB0 I/O	I	I	I	I	O	O	O	O	O	O
PB1 I/O	I	O	O	O	I	I	I	O	O	O
PB0/PB1 Mode	x	C	B	B	C	B	B	C	B	B
PB0 Data	x	x	0	1	D	0	1	D_0	0	1
PB1 Data	x	D	x	x	x	x	x	D_1	x	x
PB0 Pad Status	I	I	I	I	D	0	B	D_0	0	B
PB1 Pad Status	I	D	0	B	I	I	I	D_1	0	B

Note: I: input; O: output; D, D_0, D_1: data;
B: buzzer option, BZ or $\overline{BZ}$; x: don't care
C: CMOS output

The PC0 and PC1 are pin-shared with $\overline{INT}$, TMR and pins respectively.

It is recommended that unused or not bonded out I/O lines should be set as output pins by software instruction to avoid consuming power under input floating state.

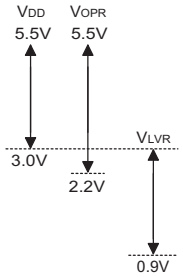
Low Voltage Reset – LVR

The microcontroller provides low voltage reset circuit in order to monitor the supply voltage of the device. If the supply voltage of the device is within the range 0.9V~V_{LVR}, such as changing a battery, the LVR will automatically reset the device internally.

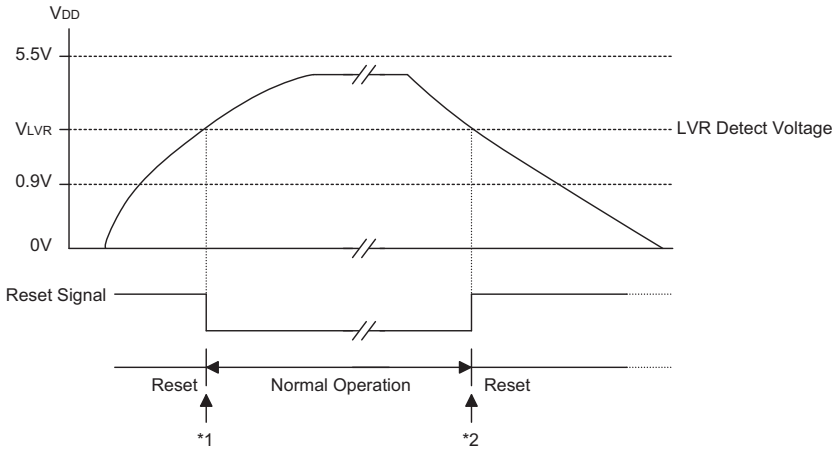
The LVR includes the following specifications:

- The low voltage (0.9V~V_{LVR}) has to remain in their original state to exceed 1ms. If the low voltage state does not exceed 1ms, the LVR will ignore it and do not perform a reset function.
- The LVR uses the "OR" function with the external $\overline{RES}$ signal to perform chip reset.

The relationship between V_{DD} and V_{LVR} is shown below.



Note: V_{OPR} is the voltage range for proper chip operation at 4MHz system clock.



Low Voltage Reset

Note: *1: To make sure that the system oscillator has stabilized, the SST provides an extra delay of 1024 system clock pulses before entering the normal operation.

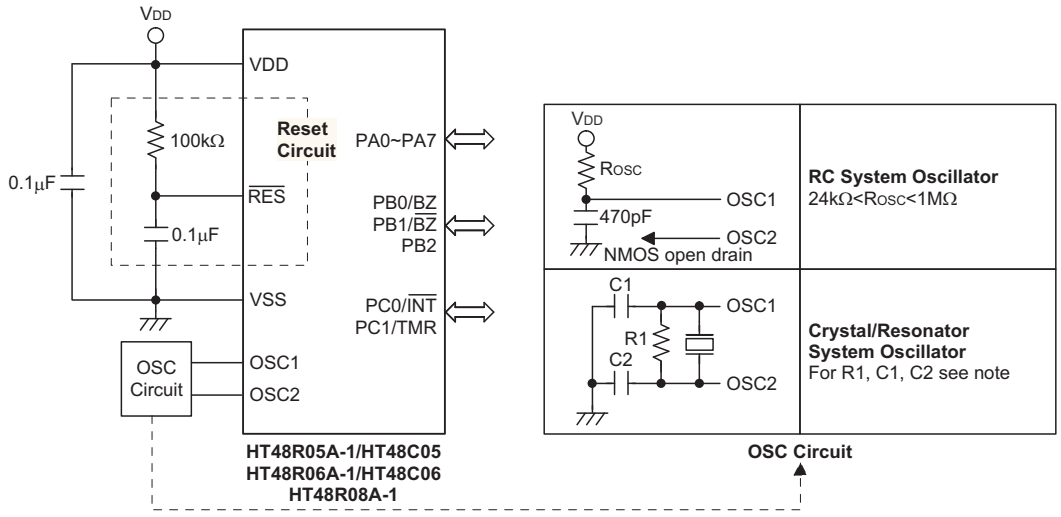
*2: Since low voltage has to be maintained in its original state and exceed 1ms, therefore 1ms delay enters the reset mode.

Options

The following table shows all kinds of options in the microcontroller. All of the options must be defined to ensure proper system functioning.

Items	Options
1	WDT clock source: WDTOSC or $f_{SYS}/4$
2	WDT function: enable or disable
3	LVR function: enable or disable
4	CLRWDT instruction(s): one or two clear WDT instruction(s)
5	System oscillator: RC or crystal
6	Pull-high resistors (PA~PC): none or pull-high
7	BZ function: enable or disable
8	PA0~PA7 wake-up: enable or disable

Application Circuits



Note: 1. Crystal/resonator system oscillators
For crystal oscillators, C1 and C2 are only required for some crystal frequencies to ensure oscillation. For resonator applications C1 and C2 are normally required for oscillation to occur. For most applications it is not necessary to add R1. However if the LVR function is disabled, and if it is required to stop the oscillator when VDD falls below its operating range, it is recommended that R1 is added. The values of C1 and C2 should be selected in consultation with the crystal/resonator manufacturer specifications.

2. Reset circuit
The reset circuit resistance and capacitance values should be chosen to ensure that VDD is stable and remains within its operating voltage range before the $\overline{RES}$ pin reaches a high level. Ensure that the length of the wiring connected to the $\overline{RES}$ pin is kept as short as possible, to avoid noise interference.

3. For applications where noise may interfere with the reset circuit and for details on the oscillator external components, refer to Application Note HA0075E for more information.

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 enable master (global) interrupt bit (bit 0; register INTC). 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 ≠ 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

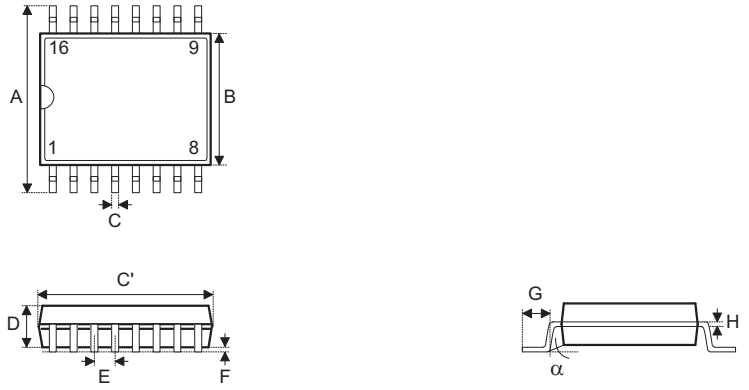
Package Information

Note that the package information provided here is for consultation purposes only. As this information may be updated at regular intervals users are reminded to consult the Holtek website for the latest version of the Package/Carton Information.

Additional supplementary information with regard to packaging is listed below. Click on the relevant section to be transferred to the relevant website page.

- Further Package Information (include Outline Dimensions, Product Tape and Reel Specifications)
- Packing Meterials Information
- Carton Information

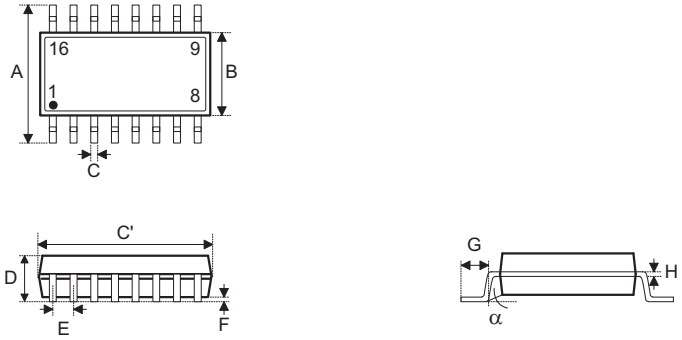
16-pin SSOP (150mil) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	—	0.236 BSC	—
B	—	0.154 BSC	—
C	0.008	—	0.012
C′	—	0.193 BSC	—
D	—	—	0.069
E	—	0.025 BSC	—
F	0.004	—	0.010
G	0.016	—	0.050
H	0.004	—	0.010
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	6.000 BSC	—
B	—	3.900 BSC	—
C	0.20	—	0.30
C′	—	4.900 BSC	—
D	—	—	1.75
E	—	0.635 BSC	—
F	0.10	—	0.25
G	0.41	—	1.27
H	0.10	—	0.25
α	0°	—	8°

16-pin NSOP (150mil) Outline Dimensions



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

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	6.000 BSC	—
B	—	3.900 BSC	—
C	0.31	—	0.51
C′	—	9.900 BSC	—
D	—	—	1.75
E	—	1.270 BSC	—
F	0.10	—	0.25
G	0.40	—	1.27
H	0.10	—	0.25
α	0°	—	8°

18-pin DIP (300mil) Outline Dimensions

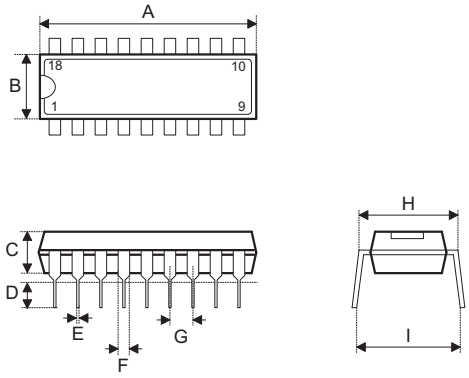


Fig1. Full Lead Packages

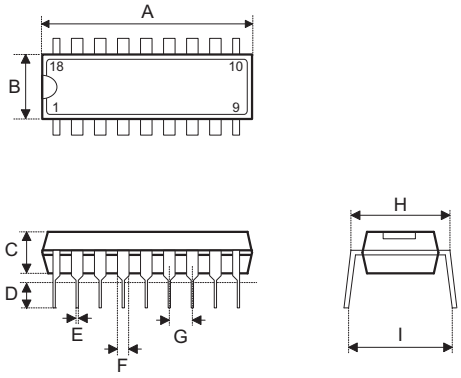


Fig2. 1/2 Lead Packages

• see fig1

Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.880	0.900	0.920
B	0.240	0.250	0.280
C	0.115	0.130	0.195
D	0.115	0.130	0.150
E	0.014	0.018	0.022
F	0.045	0.060	0.070
G	—	0.1 BSC	—
H	0.300	0.310	0.325
I	—	—	0.430

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	22.35	22.86	23.37
B	6.10	6.35	7.11
C	2.92	3.30	4.95
D	2.92	3.30	3.81
E	0.36	0.46	0.56
F	1.14	1.52	1.78
G	—	2.54 BSC	—
H	7.62	7.87	8.26
I	—	—	10.92

• see fig2

Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.845	0.865	0.885
B	0.275	0.285	0.295
C	0.120	0.135	0.150
D	0.110	0.130	0.150
E	0.014	0.018	0.022
F	0.045	0.050	0.060
G	—	0.1 BSC	—
H	0.300	0.310	0.325
I	—	—	0.430

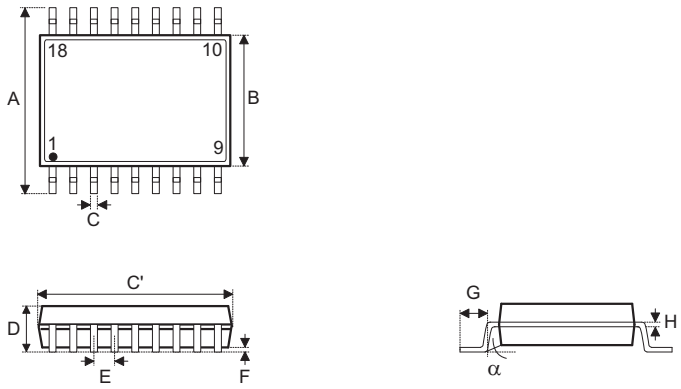
Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	21.46	21.97	22.48
B	6.99	7.24	7.49
C	3.05	3.43	3.81
D	2.79	3.30	3.81
E	0.36	0.46	0.56
F	1.14	1.27	1.52
G	—	2.54 BSC	—
H	7.62	7.87	8.26
I	—	—	10.92

• see fig2

Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	0.845	0.870	0.880
B	0.240	0.250	0.280
C	0.115	0.130	0.195
D	0.115	0.130	0.150
E	0.014	0.018	0.022
F	0.045	0.060	0.070
G	—	0.1 BSC	—
H	0.300	0.310	0.325
I	—	—	0.430

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	21.46	22.10	22.35
B	6.10	6.35	7.11
C	2.92	3.30	4.95
D	2.92	3.30	3.81
E	0.36	0.46	0.56
F	1.14	1.52	1.78
G	—	2.54 BSC	—
H	7.62	7.87	8.26
I	—	—	10.92

18-pin SOP (300mil) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	—	0.406 BSC	—
B	—	0.295 BSC	—
C	0.012	—	0.020
C′	—	0.455 BSC	—
D	—	—	0.104
E	—	0.050 BSC	—
F	0.004	—	0.012
G	0.016	—	0.050
H	0.008	—	0.013
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	10.30 BSC	—
B	—	7.5 BSC	—
C	0.31		0.51
C′	—	11.55 BSC	—
D	—		2.65
E	—	1.27 BSC	—
F	0.10	—	0.30
G	0.40	—	1.27
H	0.20	—	0.33
α	0°	—	8°