# HT48CA3
# 8-Bit Remote Type Mask MCU

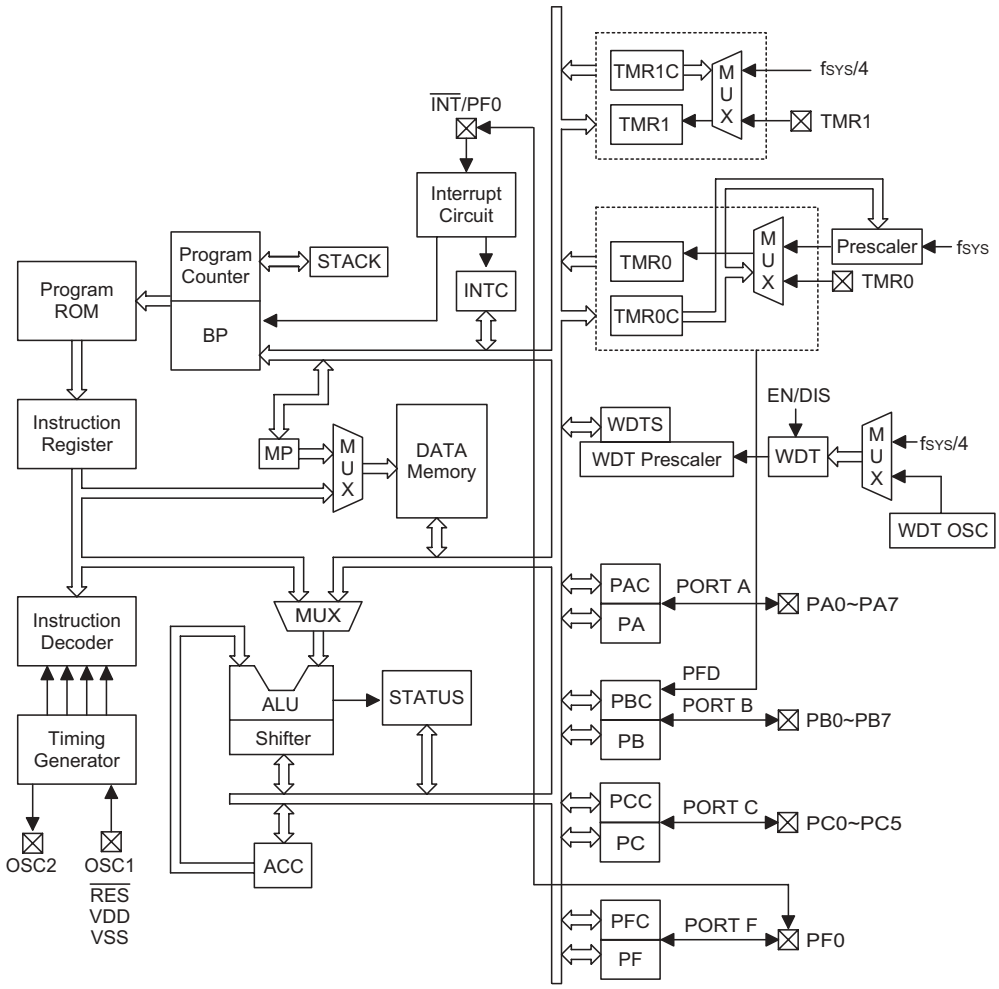
## Features

- Operating voltage: 2.2V~3.6V
- 23 bidirectional I/O lines (max.)
- 1 interrupt input shared with an I/O line
- 8-bit programmable timer/event counter with overflow interrupt and 8-stage prescaler (TMR0)
- 16-bit programmable timer/event counter and overflow interrupts (TMR1)
- On-chip crystal and RC oscillator
- Watchdog Timer
- 24K×16 program memory ROM (8K×16 bits×3 banks)
- 224×8 data memory RAM
- PFD supported
- HALT function and wake-up feature reduce power consumption
- 8-level subroutine nesting
- Up to 1μs instruction cycle with 4MHz system clock at $V_{DD}$=3V
- Bit manipulation instruction
- 16-bit table read instruction
- 63 powerful instructions
- All instructions in one or two machine cycles
- 28-pin SKDIP/SOP package

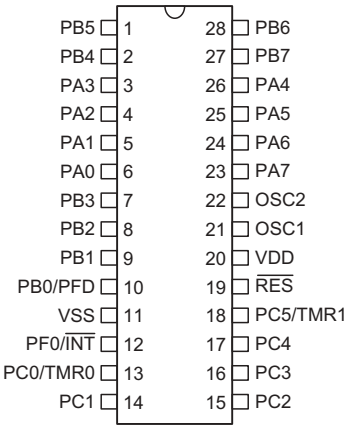
## General Description

This device is an 8-bit high performance RISC-like MCU designed for multiple I/O product applications. The device is particularly suitable for use in products such as universal remote controller (URC). A HALT feature is included to reduce power consumption. The data ROM can be used to store codes of remote control.

## Block Diagram



## Pin Assignment




**HT48CA3**
**– 28 SKDIP-A/SOP-A**

## Pin Description

| Pin Name | I/O | ROM Code Option | Description |
|---|---|---|---|
| $\overline{RES}$ | I | — | Schmitt trigger reset input, active low. |
| PA0~PA7 | I/O | Wake-up*<br>Pull-high*** | Bidirectional 8-bit input/output port. Each bit can be configured as a wake-up input by a mask option. Software instructions determine the CMOS output or Schmitt trigger input with/without pull-high resistor. The pull-high resistor of each input/output line is also optional. |
| PB0/PFD<br>PB1~PB7 | I/O | Pull-high**<br>PB0 or PFD | Bidirectional 8-bit input/output port. Software instructions determine the CMOS output or Schmitt trigger input with/without pull-high resistor. The pull-high resistor of each input/output line is also optional. The output mode of PB0 can be used as an internal PFD signal output and it can be used as a various frequency carrier signal. |
| VSS | — | — | Negative power supply, ground |
| PC0/TMR0<br>PC1~PC4<br>PC5/TMR1 | I/O | Pull-high* | Bidirectional 6-bit input/output port. Software instructions determine the CMOS output or Schmitt trigger input with/without pull-high resistor. The pull-high resistor of each input/output line is also optional. PC0 and PC5 are pin shared with TMR0 and TMR1 function pins. |
| PF0/$\overline{INT}$ | I/O | Pull-high* | Bidirectional 1-bit input/output port. Software instructions determine the CMOS output or Schmitt trigger input with/without pull-high resistor. The pull-high resistor of this input/output line is also optional. PF0 is pin shared with the $\overline{INT}$ function pin. |
| VDD | — | — | Positive power supply |
| OSC1<br>OSC2 | I<br>O | Crystal or RC | OSC1, OSC2 are connected to an RC network or Crystal (determined by hardware option) for the internal system clock. In the case of RC operation, OSC2 is the output terminal for 1/4 system clock. |

Note: *: Bit option
**: Nibble option
***: Byte option

## Absolute Maximum Ratings

Supply Voltage ..... $V_{SS}$–0.3V to $V_{SS}$+5.5V

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

Storage Temperature ..... –50°C to 125°C

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

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

## D.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $V_{DD}$ | Operating Voltage | — | — | 2.2 | — | 3.6 | V |
| $I_{DD}$ | Operating Current | 3V | No load, $f_{SYS}$=4MHz | — | 3 | 5 | mA |
| $I_{STB1}$ | Standby Current (WDT Enabled) | 3V | No load, system HALT | — | 5 | 10 | μA |
| $I_{STB2}$ | Standby Current (WDT Disabled) | 3V | No load, system HALT | — | 0.1 | 1 | μA |
| $V_{IL1}$ | Input Low Voltage for I/O Ports | — | — | 0 | — | $0.2V_{DD}$ | V |
| $V_{IH1}$ | Input High Voltage for I/O Ports | — | — | $0.8V_{DD}$ | — | $V_{DD}$ | V |
| $V_{IL2}$ | Input Low Voltage ($\overline{RES}$ Ports) | — | — | 0 | — | $0.4V_{DD}$ | V |
| $V_{IH2}$ | Input High Voltage ($\overline{RES}$ Ports) | — | — | $0.9V_{DD}$ | — | $V_{DD}$ | V |
| $I_{OL}$ | I/O Port Sink Current | 3V | $V_{OL}$=$0.1V_{DD}$ | 5 | 10 | — | mA |
| $I_{OH1}$ | I/O Port Source Current | 3V | $V_{OH}$=$0.9V_{DD}$ | –2 | –5 | — | mA |
| $I_{OH2}$ | I/O Port Source Current | 3V | $V_{OH}$=$0.8V_{DD}$ | –4 | –8 | — | mA |
| $R_{PH}$ | Pull-high Resistance | 3V | — | 40 | 60 | 80 | kΩ |

## A.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $f_{SYS}$ | System Clock | 3V | — | 400 | — | 4000 | kHz |
| $f_{TIMER}$ | Timer I/P Frequency (TMR0/TMR1) | 3V | 50% duty | 0 | — | 4000 | kHz |
| $t_{WDTOSC}$ | Watchdog Oscillator | 3V | — | 45 | 90 | 180 | μs |
| $t_{WDT1}$ | Watchdog Time-out Period (WDT OSC) | 3V | Without WDT prescaler | 11.5 | 23 | 46 | ms |
| $t_{WDT2}$ | Watchdog Time-out Period ($f_{SYS}$/4) | 3V | Without WDT prescaler | — | 1024 | — | $t_{SYS}$ |
| $t_{\overline{RES}}$ | External Reset Low Pulse Width | — | — | 1 | — | — | μs |
| $t_{SST}$ | System Start-up Timer Period | — | Power-up, reset or wake-up from HALT | — | 1024 | — | $t_{SYS}$ |
| $t_{\overline{INT}}$ | Interrupt Pulse Width | — | — | 1 | — | — | μs |
| $t_{ACC}$ | Data ROM Access Time | — | — | 1 | — | — | μs |

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

## Functional Description

### Execution flow

The system clock for the MCU 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 the program ROM are executed and its contents specify a full range of program memory.

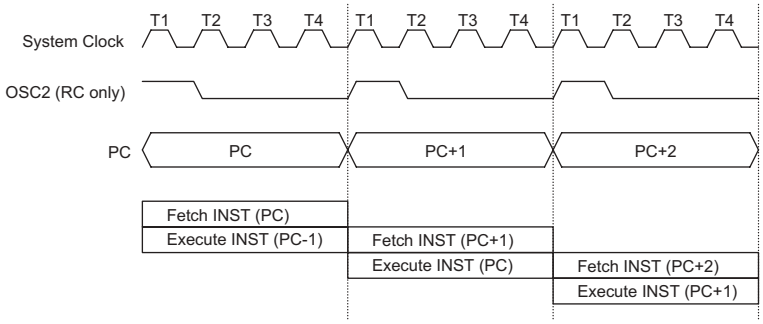
After accessing a program memory word to fetch an instruction code, the contents of the program counter are incremented by one. The program counter then points to the memory word containing the next instruction code.

When executing a jump instruction, conditional skip execution, loading register, subroutine call or return from subroutine, initial reset, internal interrupt, external interrupt or return from interrupts, 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 to the next instruction.

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

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




Execution flow

| Mode | Program Counter | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | *14~*8 | *7 | *6 | *5 | *4 | *3 | *2 | *1 | *0 |
| Initial Reset | 0000000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| External Interrupt | 0000000 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| Timer/Event Counter 0 Overflow | 0000000 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| Timer/Event Counter 1 Overflow | 0000000 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| Skip | *14~*13, (*12~*0+2): (within current bank) | | | | | | | | |
| Loading PCL | *14~*8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| Jump, Call Branch | BP (1~0), #12~#8 | #7 | #6 | #5 | #4 | #3 | #2 | #1 | #0 |
| Return (RET, RETI) | S14~S8 | S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 |

Program Counter

Note: *14~*0: Program counter bits
S14~S0: Stack register bits
#14~#0: Instruction code bits
@7~@0: PCL bits
1 bank: 8K words

**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 8192×16 bits×3 banks, 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

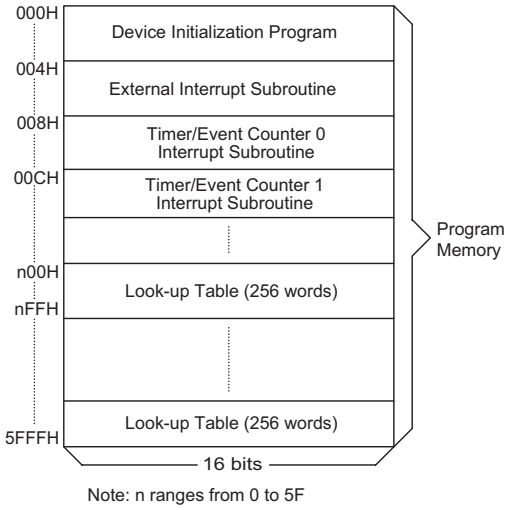
  This area is reserved for the Timer/Event Counter 0 interrupt service program. If a timer interrupt results from a Timer/Event Counter 0 overflow, and if the interrupt is enabled and the stack is not full, the program begins execution at location 008H .

- Location 00CH

  This location is reserved for the Timer/Event Counter 1 interrupt service program. If a timer interrupt results from a Timer/Event Counter 1 overflow, and the interrupt is enabled and the stack is not full, the program begins execution at location 00CH.

- Table location

  Any location in the program memory can be used as look-up tables. The instructions "TABRDC [m]" (page specified by TBHP) and "TABRDL [m]" (the last page) transfer the contents of the lower-order byte to the specified data memory, and the higher-order byte to TBLH (08H). The higher-order byte table pointer TBHP (1FH) and lower-order byte table pointer TBLP (07H) are read/write registers, which indicate the table locations. Before accessing the table, the location has to be placed in TBHP and 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 TBLH in the main routine are likely to be changed by the table read instruction used in the ISR. Errors are thus brought about. Given this, 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 main routine and the ISR, the interrupt(s) is supposed to be disabled prior to the table read instruction. It (They) will not be enabled until the TBLH in the main routine has been backup. All table related instructions require 2 cycles to complete the operation.




Note: n ranges from 0 to 5F

Program memory

**Stack register – STACK**

This is a special part of the memory which is used to save the contents of the program counter (PC) only. The stack is organized into 8 levels and is neither part of the data nor part of the program space, and is neither readable nor writeable. The activated level is indexed by the stack pointer (SP) and is neither readable nor writeable. At a subroutine call or interrupt acknowledge signal, the contents of the program counter are pushed onto the stack. At the end of a subroutine or an interrupt routine, signaled by a return instruction (RET or RETI), the program counter is restored to its previous value from the stack. After a 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 acknowledge signal will be inhibited. When the stack

| Instruction | Table Location | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | *14~*8 | *7 | *6 | *5 | *4 | *3 | *2 | *1 | *0 |
| TABRDC [m] | TBHP | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| TABRDL [m] | 1011111 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |

Table location

Note: *14~*0: Table location bits @7~@0: Table pointer bits

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 8 return addresses are stored).

**Data memory – RAM**

The data memory is designed with 250×8 bits. The data memory is divided into two functional groups: special function registers and general purpose data memory (224×8). Most are read/write, but some are read only.

The special function registers include the indirect addressing registers (R0;00H, R1;02H) bank pointer (BP;04H), Timer/Event Counter 0 (TMR0;0DH), Timer/Event Counter 0 control register (TMR0C;0EH), Timer/Event Counter 1 higher order byte register (TMR1H;0FH), Timer/Event Counter 1 lower order byte register (TMR1L;10H), Timer/Event Counter 1 control register (TMR1C;11H), program counter lower-order byte register (;06H), memory pointer registers (MP0;01H, MP1;03H), accumulator (;05H), table pointer (TBLP;07H, TBHP; 1FH), 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, PF;1CH, and I/O control registers (PAC;13H, PBC;15H, PCC;17H, PFC;1DH). The remaining space before the 20H is reserved for future expanded usage and reading these locations will get "00H". The general purpose data memory, addressed from 20H to FFH, 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 registers (MP0 or MP1).

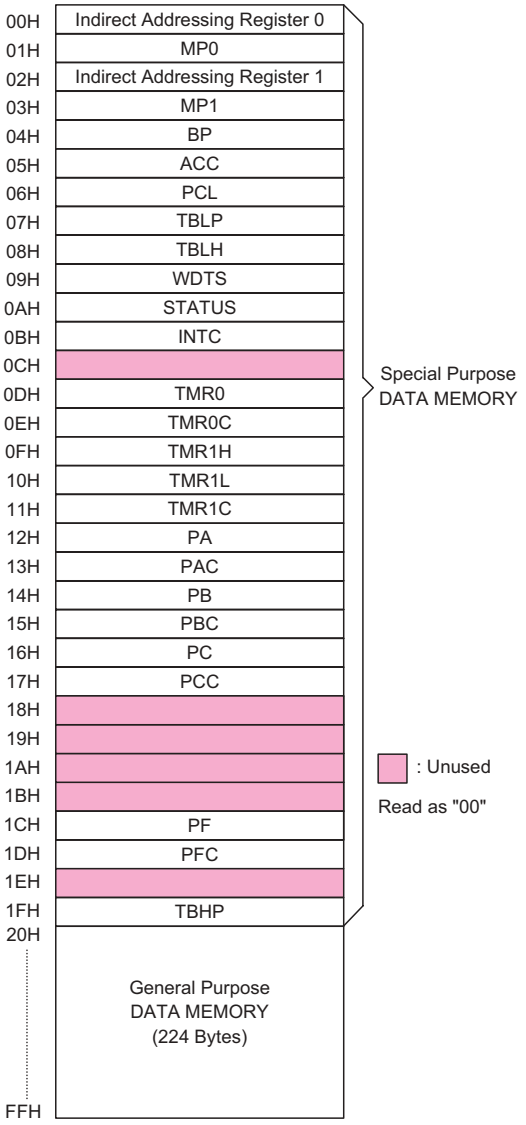
**Indirect addressing register**

Location 00H and 02H are indirect addressing registers that are not physically implemented. Any read/write operation of [00H] ([02H]) will access data memory pointed to by MP0 (MP1). Reading location 00H (02H) itself indirectly will return the result 00H. Writing indirectly results in no operation.

The memory pointer registers (MP0 and MP1) are 8-bit registers.

**Accumulator**

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




RAM mapping

**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)
- Increment and decrement (INC, DEC)
- 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 (PD), and watchdog time-out flag (TO). It also records the status information and controls the operation sequence.

With the exception of the TO and PD 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 PD 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 PD flag can be 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.

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/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 occur 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 the 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 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 the $\overline{INT}$ 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 0 interrupt is initialized by setting the Timer/Event Counter 0 interrupt request flag (T0F; bit 5 of INTC), caused by a timer 0 overflow. When the interrupt is enabled, the stack is not full and the T0F bit is set, a subroutine call to location 08H will occur. The related interrupt request flag (T0F) will be reset and the EMI bit cleared to disable further interrupts.

The internal Timer/Event Counter 1 interrupt is initialized by setting the Timer/Event Counter 1 interrupt request flag (T1F;bit 6 of INTC), caused by a timer 1 overflow. When the interrupt is enabled, the stack is not full and the T1F is set, a subroutine call to location 0CH

| Labels | Bits | Function |
|---|---|---|
| C | 0 | 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. |
| AC | 1 | 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. |
| Z | 2 | Z is set if the result of an arithmetic or logic operation is zero; otherwise Z is cleared. |
| OV | 3 | 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. |
| PD | 4 | PD is cleared by system power-up or executing the ″CLR WDT″ instruction. PD is set by executing the ″HALT″ instruction. |
| TO | 5 | TO is cleared by system power-up or executing the ″CLR WDT″ or ″HALT″ instruction. TO is set by a WDT time-out. |
| — | 6 | Undefined, read as ″0″ |
| — | 7 | Undefined, read as ″0″ |

Status register

will occur. The related interrupt request flag (T1F) will be reset and the EMI bit cleared to disable further interrupts.

During the execution of an interrupt subroutine, other interrupt acknowledge signals are held until the ″RETI″ instruction is executed or the EMI bit and the related interrupt control bit are set to 1 (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.

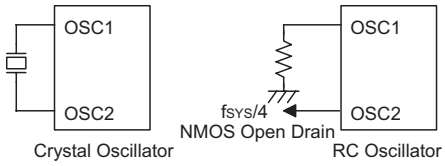
| No. | Interrupt Source | Priority | Vector |
|---|---|---|---|
| a | External Interrupt | 1 | 04H |
| b | Timer/Event Counter 0 Overflow | 2 | 08H |
| c | Timer/Event Counter 1 Overflow | 3 | 0CH |

The Timer/Event Counter 0/1 interrupt request flag (T0F/T1F), external interrupt request flag (EIF), enable Timer/Event Counter 0/1 interrupt bit (ET0I/ET1I), 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, ET0I and ET1I are used to control the enabling/disabling of interrupts. These bits prevent the requested interrupt from being serviced. Once the interrupt request flags (T0F, T1F, 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 2 oscillator circuits in the MCU.




System oscillator

There are 2 oscillator circuits implemented in the micro-controller.

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

If an RC oscillator is used, an external resistor between OSC1 and VSS is required and the resistance should range from 100kΩ to 820kΩ. The system clock, divided by 4, is available on OSC2, which can be used to synchronize external logic. The internal 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.

| Register | Bit No. | Label | Function |
|---|---|---|---|
| INTC (0BH) | 0 | EMI | Controls the master (global) interrupt (1= enabled; 0= disabled) |
| | 1 | EEI | Controls the external interrupt (1= enabled; 0= disabled) |
| | 2 | ET0I | Controls the Timer/Event Counter 0 interrupt (1= enabled; 0= disabled) |
| | 3 | ET1I | Controls the Timer/Event Counter 1 interrupt (1= enabled; 0= disabled) |
| | 4 | EIF | External interrupt request flag (1= active; 0= inactive) |
| | 5 | T0F | Internal Timer/Event Counter 0 request flag (1= active; 0= inactive) |
| | 6 | T1F | Internal Timer/Event Counter 1 request flag (1= active; 0= inactive) |
| | 7 | — | Unused bit, read as ″0″ |

INTC register

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 demanded. Instead of a crystal, the resonator can also be connected between OSC1 and OSC2 to get a frequency reference, but two external capacitors in OSC1 and OSC2 are required.

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 90μs. The WDT oscillator can be disabled by ROM code option to conserve power.

**Watchdog Timer – WDT**

The WDT clock source is implemented by a dedicated RC oscillator (WDT oscillator), instruction clock (system clock divided by 4), determines the ROM code option. 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 ROM code 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 90μs/3V normally) is selected, it is first divided by 256 (8-stage) to get the nominal time-out period of 23ms/3V. 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 2.9s/3V seconds. If the WDT oscillator is disabled, the WDT clock may still come from the instruction clock and operates 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 |

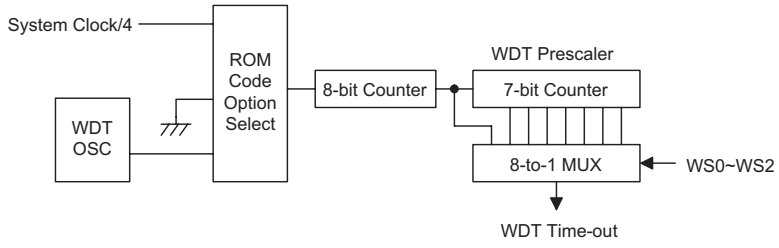
WDTS register

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 PC 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 $\overline{RES}$), 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 ROM code 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 remains 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 PD 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 PD flags are examined, the reason for chip reset can be determined. The PD 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 PC and SP; the others remain in 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 mask option. Awakening from an I/O port stimulus, the program will resume execution of the next instruction. If it awakens from an interrupt, two sequence may occur. 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 a wake-up. If the wake-up results from an interrupt acknowledge signal, the actual interrupt subroutine execution will be delayed by one or more cycles. If the wake-up results in the next instruction execution, this will be executed immediately after the dummy period 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

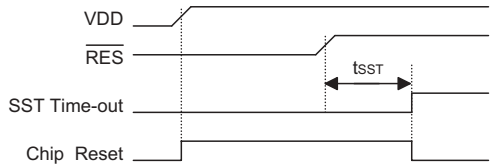
The WDT time-out during HALT is different from other chip reset conditions, since it can perform a "warm reset" that resets only the PC and SP, leaving the other circuits in their original state. Some registers remain unchanged during other reset conditions. Most registers are reset to the "initial condition"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

| TO | PD | 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 |

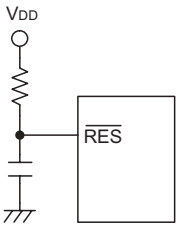
Note: "u" stands for "unchanged"

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.

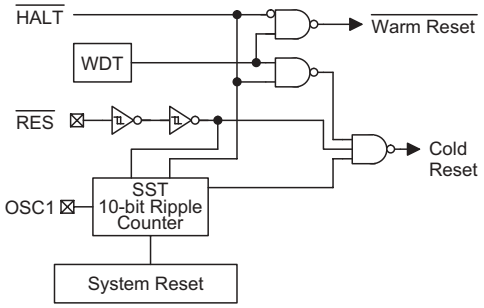
When a system reset occurs, the SST delay is added during the reset period. Any wake-up from HALT will enable the SST delay.




Reset timing chart




Reset circuit




Reset configuration

The functional unit chip reset status are shown below.

| | |
|---|---|
| PC | 000H |
| Interrupt | Disable |
| Prescaler | Clear |
| WDT | Clear. After master reset, WDT begins counting |
| Timer/Event Counter | Off |
| Input/output Ports | Input mode |
| SP | 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)* |
|---|---|---|---|---|---|
| MP0 | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| MP1 | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| BP | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| ACC | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| Program Counter | 0000H | 0000H | 0000H | 0000H | 0000H |
| 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 | --00 xxxx | --1u uuuu | --uu uuuu | --01 uuuu | --11 uuuu |
| INTC | --00 -000 | --00 -000 | --00 -000 | --00 -000 | --uu -uuu |
| TMR0 | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TMR0C | 00-0 1000 | 00-0 1000 | 00-0 1000 | 00-0 1000 | uu-u uuuu |
| TMR1H | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TMR1L | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TMR1C | 00-0 1--- | 00-0 1--- | 00-0 1--- | 00-0 1--- | uu-u u--- |
| PA | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PAC | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PB | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PBC | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PC | --11 1111 | --11 1111 | --11 1111 | --11 1111 | --uu uuuu |
| PCC | --11 1111 | --11 1111 | --11 1111 | --11 1111 | --uu uuuu |
| PF | ---- ---1 | ---- ---1 | ---- ---1 | ---- ---1 | ---- ---u |
| PFC | ---- ---1 | ---- ---1 | ---- ---1 | ---- ---1 | ---- ---u |
| TBHP | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |

Note: "*" stands for "warm reset"
"u" stands for "unchanged"
"x" stands for "unknown"

**Timer/Event Counter**

Two timer/event counters are implemented in the device. The Timer/Event Counter 0 contains an 8-bit programmable count-up counter and the clock may come from an external source or the system clock. The Timer/Event Counter 1 contains an 16-bit programmable count-up counter and the clock may come from an external source or the system clock divided by 4.

Of the two timer/event counters, 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.

Only the Timer/Event Counter 0 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 Timer/Event Counter 0; TMR0(0DH), TMR0C(0EH). In Timer/Event Counter 0 counting mode (TON=1), writing TMR0 will only put the written data to preload register (8 bits). The Timer/Event Counter 0 preload register is changed by each writing TMR0 operations. Reading TMR0 will also latch the TMR0 to the destination. The TMR0C is the Timer/Event Counter 0 control register, which defines the operating mode, counting enable or disable and active edge.

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 (TMR0) 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 (TMR0). The counting is based on the $f_{INT}$ clock.

In the event count or timer mode, once the Timer/Event Counter 0 starts counting, it will count from the current contents in the Timer/Event Counter 0 to FFH. Once overflow occurs, the counter is reloaded from the Timer/Event Counter 0 preload register and generates the corresponding interrupt request flag (T0F; bit 5 of INTC) at the same time.

In pulse width measurement mode with the TON and TE bits are equal to one, once the TMR0 has received a transition from low to high (or high to low if the TE bit is 0) it will start counting until the TMR0 returns to the original level and reset the TON. The measured result will remain in the Timer/Event Counter 0 even if the activated transition 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 transition pulse. Note that, in this operating mode, the Timer/Event Counter 0 starts counting not according to the logic level but according to the transition edges. In the case of counter overflows, the counter 0 is reloaded from the Timer/Event Counter 0 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 TMR0C) should be set to 1. In the pulse width measurement mode, the TON will be cleared automatically after the measurement cycle is complete. But in the other two modes the TON can only be reset by instructions. The overflow of the Timer/Event Counter 0 is one of the wake-up sources. No matter what the operation mode is, writing a 0 to ET0I can disabled the corresponding interrupt service.

In the case of Timer/Event Counter 0 Off condition, writing data to the Timer/Event Counter 0 preload register will also load the data to Timer/Event Counter 0. But if the Timer/Event Counter 0 is turned On, data written to the Timer/Event Counter 0 will only be kept in the Timer/Event Counter 0 preload register. The Timer/Event Counter 0 will still operate until the overflow occurs (a Timer/Event Counter 0 reloading will occur at the same time).

When the Timer/Event Counter 0 (reading TMR0) is read, the clock will be blocked to avoid errors. As this may results in a counting error, this must be taken into consideration by the programmer.

The bit 0~2 of the TMR0C can be used to define the pre-scaling stages of the internal clock sources of Timer/Event Counter 0. The definitions are as shown.

| Label (TMR0C) | Bits | Function |
|---|---|---|
| PSC0~ PSC2 | 0~2 | To define the prescaler stages, PSC2, PSC1, PSC0=<br>000: $f_{INT}=f_{SYS}/2$<br>001: $f_{INT}=f_{SYS}/4$<br>010: $f_{INT}=f_{SYS}/8$<br>011: $f_{INT}=f_{SYS}/16$<br>100: $f_{INT}=f_{SYS}/32$<br>101: $f_{INT}=f_{SYS}/64$<br>110: $f_{INT}=f_{SYS}/128$<br>111: $f_{INT}=f_{SYS}/256$ |
| TE | 3 | To define the TMR0 active edge of Timer/Event Counter 0<br>(0=active on low to high;<br>1=active on high to low) |
| TON | 4 | To enable/disable timer 0 counting<br>(0=disabled; 1=enabled) |
| — | 5 | Unused bit, read as ″0″ |
| TM0<br>TM1 | 6<br>7 | To define the operating mode<br>01=Event count mode (external clock)<br>10=Timer mode (internal clock)<br>11=Pulse width measurement mode<br>00=Unused |

TMR0C register

There are 3 registers related to Timer/Event Counter 1; TMR1H(0FH), TMR1L(10H), TMR1C(11H). Writing TMR1L will only put the written data to an internal lower-order byte buffer (8 bits) and writing TMR1H will transfer the specified data and the contents of the lower-order byte buffer to TMR1H and TMR1L preload registers, respectively. The Timer/Event Counter 1 preload register is changed by each writing TMR1H operations. Reading TMR1H will latch the contents of TMR1H and TMR1L counters to the destination and the lower-order byte buffer, respectively. Reading the TMR1L will read the contents of the lower-order byte buffer. The TMR1C is the Timer/Event Counter 1 control register, which defines the operating mode, counting enable or disable and active edge.

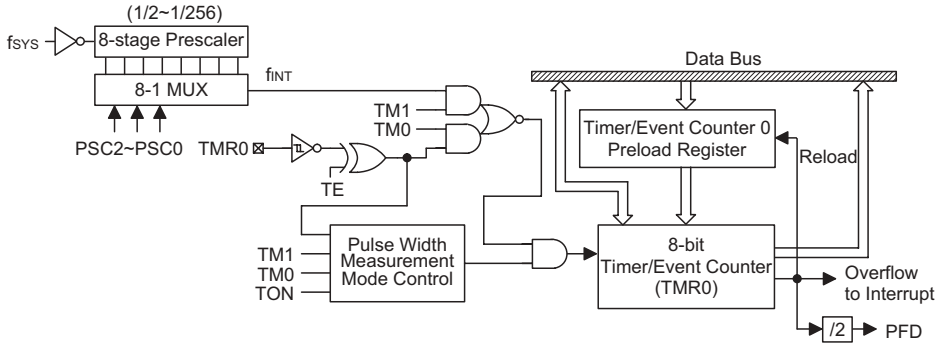
The TM0, TM1 bits define the operating mode. The event count mode is used to count external events, which means the clock source comes from an external (TMR1) pin. The timer mode functions as a normal timer with the clock source coming from the instruction clock. The pulse width measurement mode can be used to count the high or low level duration of the external signal (TMR1). The counting is based on the instruction clock.

In the event count or timer mode, once the Timer/Event Counter 1 starts counting, it will count from the current contents in the Timer/Event Counter 1 to FFFFH. Once overflow occurs, the counter is reloaded from the Timer/Event Counter 1 preload register and generates the corresponding interrupt request flag (T1F; bit 6 of INTC) at the same time.

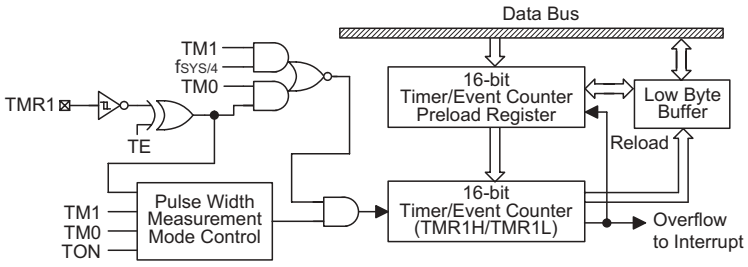
In pulse width measurement mode with the TON and TE bits are equal to one, once the TMR1 has received a transition from low to high (or high to low if the TE bit is 0) it will start counting until the TMR1 returns to the original level and reset the TON. The measured result will remain in the Timer/Event Counter 1 even if the activated transition 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 transition pulse. Note that, in this operating mode, the Timer/Event Counter 1 starts counting not according to the logic level but according to the transition edges. In the case of counter overflows, the counter 1 is reloaded from the Timer/Event Counter 1 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 TMR1C) should be set to 1. In the pulse width measurement mode, the TON will be cleared automatically after the measurement cycle is complete. But in the other two modes the TON can only be reset by instructions. The overflow of the Timer/Event Counter 1 is one of the wake-up sources. No matter what the operation mode is, writing a 0 to ET1I can disabled the corresponding interrupt service.

In the case of Timer/Event Counter 1 OFF condition, writing data to the Timer/Event Counter 1 preload register will also load the data to Timer/Event Counter 1. But if the Timer/Event Counter 1 is turned on, data written to the Timer/Event Counter 1 will only be kept in the




Timer/Event Counter 0




Timer/Event Counter 1

Timer/Event Counter 1 preload register. The Timer/Event Counter 1 will still operate until the overflow occurs (a Timer/Event Counter 1 reloading will occur at the same time).

When the Timer/Event Counter 1 (reading TMR1H) is read, the clock will be blocked to avoid errors. As this may results in a counting error, this must be taken into consideration by the programmer.

The definitions of the TMR1C are as shown.

| Label (TMR1C) | Bits | Function |
|---|---|---|
| — | 0~2 | Unused bit, read as ″0″ |
| TE | 3 | To define the active edge of TMR1 pin input signal<br>(0/1: active on low to high/high to low) |
| TON | 4 | To enable/disable timer 1 counting<br>(0/1: disabled/enabled) |
| — | 5 | Unused bit, read as ″0″ |
| TM0<br>TM1 | 6<br>7 | To define the operating mode<br>01=Event count mode (external clock)<br>10=Timer mode (internal clock)<br>11=Pulse width measurement mode<br>00=Unused |

TMR1C register

**Input/output ports**

There are 23 bi-directional input/output lines in the micro-controller, labeled from PA to PC and PF, which are mapped to the data memory of [12H], [14H], [16H] and [1CH], respectively. All of these I/O ports can be used as 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, 16H or 1CH). 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, PFC) to control the input/output configuration. With this control register, CMOS output or Schmitt trigger input with or without (depends on options) pull-high resistor structures can be reconfigured dynamically (i.e., on-the fly) under software control. To function as an input, the corresponding latch of the control register has to be set as ″1″. The pull-high resistor (if the pull-high resistor is enabled) will be exhibited automatically. The input sources also depends on the control register. If the control register bit is ″1″, the input will read the pad state (″mov″ and read-modify-write instructions ). If the control register bit is 0, the contents of the latches will move to internal data bus (″mov″ and read-modify-write instructions). The input paths (pad state or latches) of read-modify-write instructions are dependent on the control register bits. For output function, CMOS is the only configuration. These control registers are mapped to locations 13H, 15H, 17H and 1DH.

After a chip reset, these input/output lines stay at high levels (pull-high options) or floating state (non-pull-high options). Each bit of these input/output latches can be set or cleared by ″SET [m].i″ (m = 12H, 14H, 16H or 1CH) instructions. Some instructions first input data and then follow the output operations. For example, ″SET [m].i″, ″CLR [m].i″, ″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 2 bits of port C and 7 bits of port F are not physically implemented; on reading them a ″0″ is returned whereas writing then results in a no-operation. Pull-high resistors of each port are decided by a option bit.

The PB0 is pin-shared with PFD signal, respectively. If the PFD option is selected, the output signal in output mode of PB0 will be the PFD signal. The input mode always remain its original functions. The PF0 and PC0 are pin-shared with $\overline{INT}$ and TMR 0. The $\overline{INT}$ signal is directly connected to PF0. The PFD output signal (in output mode) are controlled by the PB0 data register only. The truth table of PB0/PFD is listed below.

The truth table of PB0/PFD is as shown.

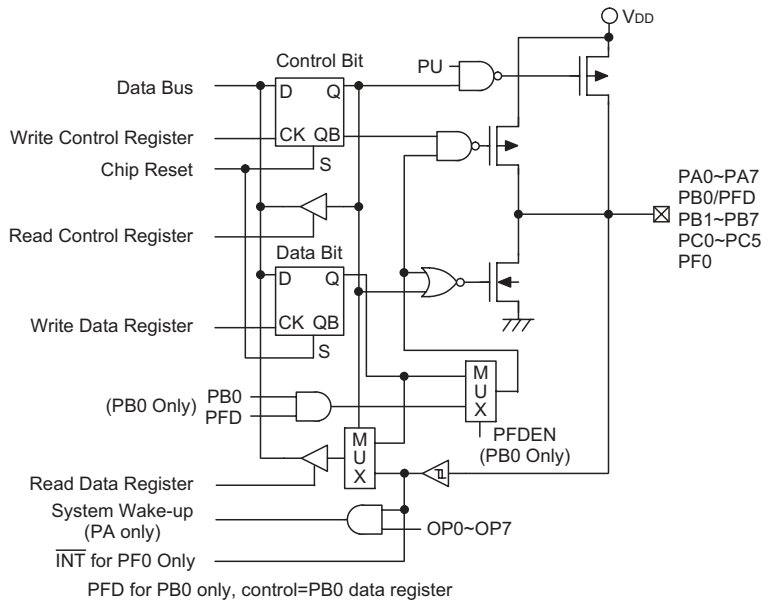
| PBC (15H) Bit0 | I | O | O | O |
|---|---|---|---|---|
| PB0/PFD Option | x | PB0 | PFD | PFD |
| PB0 (14H) Bit0 | x | D | 0 | 1 |
| PB0 Pad Status | I | D | 0 | PFD |

Note: I: Input; O: Output; D: Data

**Bank pointer**

There is a bank pointer used to control the program flow to go to any banks. A bank contains 8K×16 address space. The contents of bank pointer are load into program counter when the JMP or CALL instruction is executed. The program counter is a 15-bit register whose contents are used to specify the executed instruction addresses.

When calling a subroutine or an interrupt event occurring, the contents of the program counter are save into stack registers. If a returning from subroutine occurs, the contents of the program counter will restore from stack registers.




Input/output ports

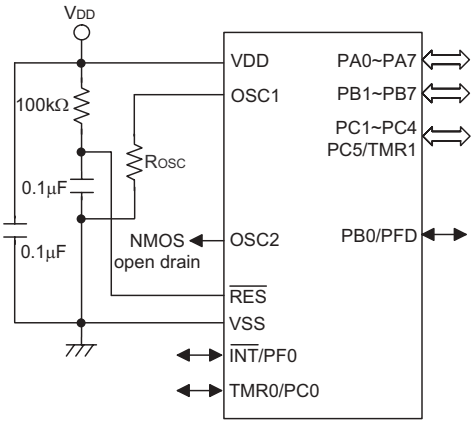
**Options**

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

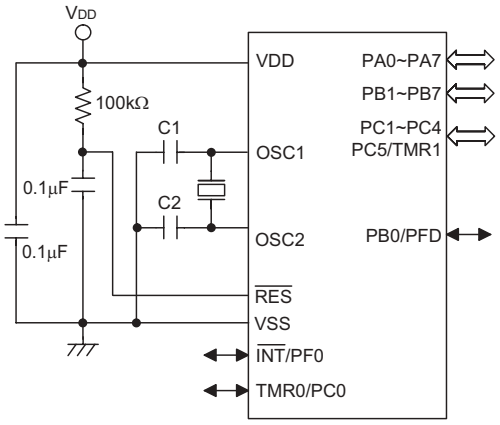
| Function |
|---|
| PA0~PA7 wake-up enable/disable |
| PC pull-high enable/disable |
| PA pull-high enable/disable: Byte option |
| PF pull-high enable/disable |
| PB pull-high (PB0~PB3, PB4~PB7) enable/disable: Nibble option |
| PB0 or PFD |
| CLR WDT instructions |
| System oscillators<br>RC or crystal |
| WDT enable/disable |
| WDT clock source: WDTOSC or system clock/4 (T1D) |

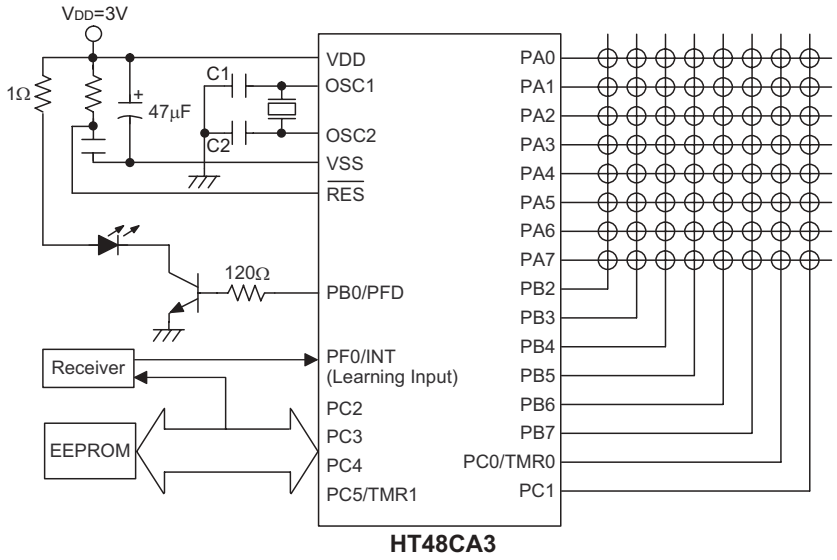
## Application Circuits

**RC oscillator for multiple I/O applications**




**Crystal or ceramic resonator for multiple I/O applications**




Note: If 400kHz<$f_{SYS}$<1MHz, C1=C2=300pF, else C1=C2

## Instruction Set Summary

| Mnemonic | Description | Instruction Cycle | Flag Affected |
|---|---|---|---|
| **Arithmetic** | | | |
| ADD A,[m] | Add data memory to ACC | 1 | Z,C,AC,OV |
| ADDM A,[m] | Add ACC to data memory | 1(1) | 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 register with carry | 1(1) | Z,C,AC,OV |
| SUB A,x | Subtract immediate data from ACC | 1 | Z,C,AC,OV |
| SUB A,[m] | Subtract data memory from ACC | 1 | Z,C,AC,OV |
| SUBM A,[m] | Subtract data memory from ACC with result in data memory | 1(1) | 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 and result in data memory | 1(1) | Z,C,AC,OV |
| DAA [m] | Decimal adjust ACC for addition with result in data memory | 1(1) | C |
| **Logic Operation** | | | |
| AND A,[m] | AND data memory to ACC | 1 | Z |
| OR A,[m] | OR data memory to ACC | 1 | Z |
| XOR A,[m] | Exclusive-OR data memory to ACC | 1 | Z |
| ANDM A,[m] | AND ACC to data memory | 1(1) | Z |
| ORM A,[m] | OR ACC to data memory | 1(1) | Z |
| XORM A,[m] | Exclusive-OR ACC to data memory | 1(1) | Z |
| AND A,x | AND immediate data to ACC | 1 | Z |
| OR A,x | OR immediate data to ACC | 1 | Z |
| XOR A,x | Exclusive-OR immediate data to ACC | 1 | Z |
| CPL [m] | Complement data memory | 1(1) | Z |
| CPLA [m] | Complement data memory with result in ACC | 1 | Z |
| **Increment & Decrement** | | | |
| INCA [m] | Increment data memory with result in ACC | 1 | Z |
| INC [m] | Increment data memory | 1(1) | Z |
| DECA [m] | Decrement data memory with result in ACC | 1 | Z |
| DEC [m] | Decrement data memory | 1(1) | Z |
| **Rotate** | | | |
| RRA [m] | Rotate data memory right with result in ACC | 1 | None |
| RR [m] | Rotate data memory right | 1(1) | None |
| RRCA [m] | Rotate data memory right through carry with result in ACC | 1 | C |
| RRC [m] | Rotate data memory right through carry | 1(1) | C |
| RLA [m] | Rotate data memory left with result in ACC | 1 | None |
| RL [m] | Rotate data memory left | 1(1) | None |
| RLCA [m] | Rotate data memory left through carry with result in ACC | 1 | C |
| RLC [m] | Rotate data memory left through carry | 1(1) | C |
| **Data Move** | | | |
| MOV A,[m] | Move data memory to ACC | 1 | None |
| MOV [m],A | Move ACC to data memory | 1(1) | None |
| MOV A,x | Move immediate data to ACC | 1 | None |
| **Bit Operation** | | | |
| CLR [m].i | Clear bit of data memory | 1(1) | None |
| SET [m].i | Set bit of data memory | 1(1) | None |

| Mnemonic | Description | Instruction Cycle | Flag Affected |
|---|---|---|---|
| **Branch** | | | |
| JMP addr | Jump unconditionally | 2 | None |
| SZ [m] | Skip if data memory is zero | 1(2) | None |
| SZA [m] | Skip if data memory is zero with data movement to ACC | 1(2) | None |
| SZ [m].i | Skip if bit i of data memory is zero | 1(2) | None |
| SNZ [m].i | Skip if bit i of data memory is not zero | 1(2) | None |
| SIZ [m] | Skip if increment data memory is zero | 1(3) | None |
| SDZ [m] | Skip if decrement data memory is zero | 1(3) | None |
| SIZA [m] | Skip if increment data memory is zero with result in ACC | 1(2) | None |
| SDZA [m] | Skip if decrement data memory is zero with result in ACC | 1(2) | 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 ROM code (current page) to data memory and TBLH | 2(1) | None |
| TABRDL [m] | Read ROM code (last page) to data memory and TBLH | 2(1) | None |
| **Miscellaneous** | | | |
| NOP | No operation | 1 | None |
| CLR [m] | Clear data memory | 1(1) | None |
| SET [m] | Set data memory | 1(1) | None |
| CLR WDT | Clear Watchdog Timer | 1 | TO,PD |
| CLR WDT1 | Pre-clear Watchdog Timer | 1 | TO(4),PD(4) |
| CLR WDT2 | Pre-clear Watchdog Timer | 1 | TO(4),PD(4) |
| SWAP [m] | Swap nibbles of data memory | 1(1) | None |
| SWAPA [m] | Swap nibbles of data memory with result in ACC | 1 | None |
| HALT | Enter power down mode | 1 | TO,PD |

Note: x: 8 bits immediate data

m: Data memory address

A: Accumulator

i: 0~7 number of bits

addr: Program memory address

√: Flag is affected

–: Flag is not affected

(1): If a loading to the PCL register occurs, the execution cycle of instructions will be delayed for one more cycle (four system clocks).

(2): If a skipping to the next instruction occurs, the execution cycle of instructions will be delayed for one more cycle (four system clocks). Otherwise the original instruction cycle is unchanged.

(3): (1) and (2)

(4): The flags may be affected by the execution status. If the Watchdog Timer is cleared by executing the CLR WDT1 or CLR WDT2 instruction, the TO is set and the PD is cleared.
Otherwise the TO and PD flags remain unchanged.

## Instruction Definition

**ADC A,[m]** Add data memory and carry to the accumulator

Description The contents of the specified data memory, accumulator and the carry flag are added simultaneously, leaving the result in the accumulator.

Operation ACC ← ACC+[m]+C

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | √ | √ | √ | √ |

**ADCM A,[m]** Add the accumulator and carry to data memory

Description The contents of the specified data memory, accumulator and the carry flag are added simultaneously, leaving the result in the specified data memory.

Operation [m] ← ACC+[m]+C

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | √ | √ | √ | √ |

**ADD A,[m]** Add data memory to the accumulator

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)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | √ | √ | √ | √ |

**ADD A,x** Add immediate data to the accumulator

Description The contents of the accumulator and the specified data are added, leaving the result in the accumulator.

Operation ACC ← ACC+x

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | √ | √ | √ | √ |

**ADDM A,[m]** Add the accumulator to the data memory

Description The contents of the specified data memory and the accumulator are added. The result is stored in the data memory.

Operation [m] ← ACC+[m]

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | √ | √ | √ | √ |

**AND A,[m]** Logical AND accumulator with data memory

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)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**AND A,x** Logical AND immediate data to the accumulator

Description Data in the accumulator and the specified data perform a bitwise logical_AND operation. The result is stored in the accumulator.

Operation ACC ← ACC ″AND″ x

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**ANDM A,[m]** Logical AND data memory with the accumulator

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)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**CALL addr** Subroutine call

Description The instruction unconditionally calls a subroutine located at the indicated address. The program counter increments once to obtain the address of the next instruction, and pushes this onto the stack. The indicated address is then loaded. Program execution continues with the instruction at this address.

Operation Stack ← PC+1
PC ← addr

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**CLR [m]** Clear data memory

Description The contents of the specified data memory are cleared to 0.

Operation [m] ← 00H

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**CLR [m].i** Clear bit of data memory

Description The bit i of the specified data memory is cleared to 0.

Operation [m].i ← 0

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**CLR WDT** Clear Watchdog Timer

Description The WDT and the WDT Prescaler are cleared (re-counting from 0). The power down bit (PD) and time-out bit (TO) are cleared.

Operation WDT and WDT Prescaler ← 00H
PD and TO ← 0

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | 0 | 0 | — | — | — | — |

**CLR WDT1** Preclear Watchdog Timer

Description The TO, PD flags, WDT and the WDT Prescaler has cleared (re-counting from 0), if the other preclear WDT instruction has been executed. Only execution of this instruction without the other preclear instruction just sets the indicated flag which implies this instruction has been executed and the TO and PD flags remain unchanged.

Operation WDT and WDT Prescaler ← 00H*
PD and TO ← 0*

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | 0* | 0* | — | — | — | — |

**CLR WDT2** Preclear Watchdog Timer

Description The TO, PD flags, WDT and the WDT Prescaler are cleared (re-counting from 0), if the other preclear WDT instruction has been executed. Only execution of this instruction without the other preclear instruction, sets the indicated flag which implies this instruction has been executed and the TO and PD flags remain unchanged.

Operation WDT and WDT Prescaler ← 00H*
PD and TO ← 0*

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | 0* | 0* | — | — | — | — |

**CPL [m]** Complement data memory

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

Operation [m] ← $[\overline{m}]$

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**CPLA [m]** Complement data memory and place result in the accumulator

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

Operation ACC ← $[\overline{m}]$

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**DAA [m]** Decimal-Adjust accumulator for addition

Description The accumulator value is adjusted to the BCD (Binary Coded Decimal) code. The accumulator is divided into two nibbles. Each nibble is adjusted to the BCD code and an internal carry (AC1) will be done if the low nibble of the accumulator is greater than 9. The BCD adjustment is done by adding 6 to the original value if the original value is greater than 9 or a carry (AC or C) is set; otherwise the original value remains unchanged. The result is stored in the data memory and only the carry flag (C) may be affected.

Operation If ACC.3~ACC.0 >9 or AC=1
then [m].3~[m].0 ← (ACC.3~ACC.0)+6, AC1=$\overline{AC}$
else [m].3~[m].0) ← (ACC.3~ACC.0), AC1=0
and
If ACC.7~ACC.4+AC1 >9 or C=1
then [m].7~[m].4 ← ACC.7~ACC.4+6+AC1,C=1
else [m].7~[m].4 ← ACC.7~ACC.4+AC1,C=C

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | √ |

**DEC [m]** Decrement data memory

Description Data in the specified data memory is decremented by 1.

Operation [m] ← [m]–1

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**DECA [m]** Decrement data memory and place result in the accumulator

Description Data in the specified data memory is decremented by 1, leaving the result in the accumulator. The contents of the data memory remain unchanged.

Operation ACC ← [m]–1

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**HALT** Enter power down mode

Description This instruction stops program execution and turns off the system clock. The contents of the RAM and registers are retained. The WDT and prescaler are cleared. The power down bit (PD) is set and the WDT time-out bit (TO) is cleared.

Operation PC ← PC+1
PD ← 1
TO ← 0

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | 0 | 1 | — | — | — | — |

**INC [m]** Increment data memory

Description Data in the specified data memory is incremented by 1

Operation [m] ← [m]+1

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**INCA [m]** Increment data memory and place result in the accumulator

Description Data in the specified data memory is incremented by 1, leaving the result in the accumulator. The contents of the data memory remain unchanged.

Operation ACC ← [m]+1

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**JMP addr** Directly jump

Description Bits of the program counter are replaced with the directly-specified address unconditionally, and control is passed to this destination.

Operation PC ← addr

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**MOV A,[m]** Move data memory to the accumulator

Description The contents of the specified data memory are copied to the accumulator.

Operation ACC ← [m]

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**MOV A,x** Move immediate data to the accumulator

Description The 8-bit data specified by the code is loaded into the accumulator.

Operation ACC ← x

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**MOV [m],A** Move the accumulator to data memory

Description The contents of the accumulator are copied to the specified data memory (one of the data memories).

Operation [m] ← ACC

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**NOP** No operation

Description No operation is performed. Execution continues with the next instruction.

Operation PC ← PC+1

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**OR A,[m]** Logical OR accumulator with data memory

Description Data in the accumulator and the specified data memory (one of the data memories) perform a bitwise logical_OR operation. The result is stored in the accumulator.

Operation ACC ← ACC ″OR″ [m]

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**OR A,x** Logical OR immediate data to the accumulator

Description Data in the accumulator and the specified data perform a bitwise logical_OR operation. The result is stored in the accumulator.

Operation ACC ← ACC ″OR″ x

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**ORM A,[m]** Logical OR data memory with the accumulator

Description Data in the data memory (one of the data memories) 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)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**RET** Return from subroutine

Description The program counter is restored from the stack. This is a 2-cycle instruction.

Operation PC ← Stack

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**RET A,x** Return and place immediate data in the accumulator

Description The program counter is restored from the stack and the accumulator loaded with the specified 8-bit immediate data.

Operation PC ← Stack
ACC ← x

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**RETI** Return from interrupt

Description The program counter is restored from the stack, and interrupts are enabled by setting the EMI bit. EMI is the enable master (global) interrupt bit (bit 0; register INTC).

Operation PC ← Stack
EMI ← 1

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**RL [m]** Rotate data memory left

Description The contents of the specified data memory are rotated 1 bit left with bit 7 rotated into bit 0.

Operation [m].(i+1) ← [m].i; [m].i:bit i of the data memory (i=0~6)
[m].0 ← [m].7

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**RLA [m]** Rotate data memory left and place result in the accumulator

Description Data in the specified data memory is rotated 1 bit left with bit 7 rotated into bit 0, leaving the rotated result in the accumulator. The contents of the data memory remain unchanged.

Operation ACC.(i+1) ← [m].i; [m].i:bit i of the data memory (i=0~6)
ACC.0 ← [m].7

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**RLC [m]** Rotate data memory left through carry

Description The contents of the specified data memory and the carry flag are rotated 1 bit left. Bit 7 replaces the carry bit; the original carry flag is rotated into the bit 0 position.

Operation [m].(i+1) ← [m].i; [m].i:bit i of the data memory (i=0~6)
[m].0 ← C
C ← [m].7

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | √ |

**RLCA [m]** Rotate left through carry and place result in the accumulator

Description Data in the specified data memory and the carry flag are rotated 1 bit left. Bit 7 replaces the carry bit and the original carry flag is rotated into bit 0 position. The rotated result is stored in the accumulator but the contents of the data memory remain unchanged.

Operation ACC.(i+1) ← [m].i; [m].i:bit i of the data memory (i=0~6)
ACC.0 ← C
C ← [m].7

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | √ |

**RR [m]** Rotate data memory right

Description The contents of the specified data memory are rotated 1 bit right with bit 0 rotated to bit 7.

Operation [m].i ← [m].(i+1); [m].i:bit i of the data memory (i=0~6)
[m].7 ← [m].0

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**RRA [m]** Rotate right and place result in the accumulator

Description Data in the specified data memory is rotated 1 bit right with bit 0 rotated into bit 7, leaving the rotated result in the accumulator. The contents of the data memory remain unchanged.

Operation ACC.(i) ← [m].(i+1); [m].i:bit i of the data memory (i=0~6)
ACC.7 ← [m].0

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**RRC [m]** Rotate data memory right through carry

Description The contents of the specified data memory and the carry flag are together rotated 1 bit right. Bit 0 replaces the carry bit; the original carry flag is rotated into the bit 7 position.

Operation [m].i ← [m].(i+1); [m].i:bit i of the data memory (i=0~6)
[m].7 ← C
C ← [m].0

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | √ |

**RRCA [m]** Rotate right through carry and place result in the accumulator

Description Data of the specified data memory and the carry flag are rotated 1 bit right. Bit 0 replaces the carry bit and the original carry flag is rotated into the bit 7 position. The rotated result is stored in the accumulator. The contents of the data memory remain unchanged.

Operation ACC.i ← [m].(i+1); [m].i:bit i of the data memory (i=0~6)
ACC.7 ← C
C ← [m].0

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | √ |

**SBC A,[m]** Subtract data memory and carry from the accumulator

Description The contents of the specified data memory and the complement of the carry flag are subtracted from the accumulator, leaving the result in the accumulator.

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

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | √ | √ | √ | √ |

**SBCM A,[m]** Subtract data memory and carry from the accumulator

Description The contents of the specified data memory and the complement of the carry flag are subtracted from the accumulator, leaving the result in the data memory.

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

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | √ | √ | √ | √ |

**SDZ [m]** Skip if decrement data memory is 0

Description The contents of the specified data memory are decremented by 1. If the result is 0, the next instruction is skipped. If the result is 0, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction (2 cycles). Otherwise proceed with the next instruction (1 cycle).

Operation Skip if ([m]–1)=0, [m] ← ([m]–1)

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**SDZA [m]** Decrement data memory and place result in ACC, skip if 0

Description The contents of the specified data memory are decremented by 1. If the result is 0, the next instruction is skipped. The result is stored in the accumulator but the data memory remains unchanged. If the result is 0, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction (2 cycles). Otherwise proceed with the next instruction (1 cycle).

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

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**SET [m]** Set data memory

Description Each bit of the specified data memory is set to 1.

Operation [m] ← FFH

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**SET [m].i** Set bit of data memory

Description Bit "i" of the specified data memory is set to 1.

Operation [m].i ← 1

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**SIZ [m]** Skip if increment data memory is 0

Description The contents of the specified data memory are incremented by 1. If the result is 0, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction (2 cycles). Otherwise proceed with the next instruction (1 cycle).

Operation Skip if ([m]+1)=0, [m] ← ([m]+1)

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**SIZA [m]** Increment data memory and place result in ACC, skip if 0

Description The contents of the specified data memory are incremented by 1. If the result is 0, the next instruction is skipped and the result is stored in the accumulator. The data memory remains unchanged. If the result is 0, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction (2 cycles). Otherwise proceed with the next instruction (1 cycle).

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

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**SNZ [m].i** Skip if bit "i" of the data memory is not 0

Description If bit "i" of the specified data memory is not 0, the next instruction is skipped. If bit "i" of the data memory is not 0, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction (2 cycles). Otherwise proceed with the next instruction (1 cycle).

Operation Skip if [m].i≠0

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**SUB A,[m]** Subtract data memory from the accumulator

Description The specified data memory is subtracted from the contents of the accumulator, leaving the result in the accumulator.

Operation ACC ← ACC+[$\overline{m}$]+1

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | √ | √ | √ | √ |

**SUBM A,[m]** Subtract data memory from the accumulator

Description The specified data memory is subtracted from the contents of the accumulator, leaving the result in the data memory.

Operation [m] ← ACC+[$\overline{m}$]+1

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | √ | √ | √ | √ |

**SUB A,x** Subtract immediate data from the accumulator

Description The immediate data specified by the code is subtracted from the contents of the accumulator, leaving the result in the accumulator.

Operation ACC ← ACC+$\overline{x}$+1

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | √ | √ | √ | √ |

**SWAP [m]** Swap nibbles within the data memory

Description The low-order and high-order nibbles of the specified data memory (1 of the data memories) are interchanged.

Operation [m].3~[m].0 ↔ [m].7~[m].4

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**SWAPA [m]** Swap data memory and place result in the accumulator

Description The low-order and high-order nibbles of the specified data memory are interchanged, writing the result to 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)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**SZ [m]** Skip if data memory is 0

Description If the contents of the specified data memory are 0, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction (2 cycles). Otherwise proceed with the next instruction (1 cycle).

Operation Skip if [m]=0

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**SZA [m]** Move data memory to ACC, skip if 0

Description The contents of the specified data memory are copied to the accumulator. If the contents is 0, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction (2 cycles). Otherwise proceed with the next instruction (1 cycle).

Operation Skip if [m]=0

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**SZ [m].i** Skip if bit ″i″ of the data memory is 0

Description If bit ″i″ of the specified data memory is 0, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction (2 cycles). Otherwise proceed with the next instruction (1 cycle).

Operation Skip if [m].i=0

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**TABRDC [m]** Move the ROM code (current page) to TBLH and data memory

Description The low byte of ROM code (current page) addressed by the table pointer (TBLP) is moved to the specified data memory and the high byte transferred to TBLH directly.

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

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**TABRDL [m]** Move the ROM code (last page) to TBLH and data memory

Description The low byte of ROM code (last page) addressed by the table pointer (TBLP) is moved to the data memory and the high byte transferred to TBLH directly.

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

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | — | — | — |

**XOR A,[m]** Logical XOR accumulator with data memory

Description Data in the accumulator and the indicated data memory perform a bitwise logical Exclusive_OR operation and the result is stored in the accumulator.

Operation ACC ← ACC ″XOR″ [m]

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**XORM A,[m]** Logical XOR data memory with the accumulator

Description Data in the indicated data memory and the accumulator perform a bitwise logical Exclusive_OR operation. The result is stored in the data memory. The 0 flag is affected.

Operation [m] ← ACC ″XOR″ [m]

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**XOR A,x** Logical XOR immediate data to the accumulator

Description Data in the accumulator and the specified data perform a bitwise logical Exclusive_OR operation. The result is stored in the accumulator. The 0 flag is affected.

Operation ACC ← ACC ″XOR″ x

Affected flag(s)

| TC2 | TC1 | TO | PD | OV | Z | AC | C |
|---|---|---|---|---|---|---|---|
| — | — | — | — | — | √ | — | — |

**Holtek Semiconductor Inc. (Headquarters)**
No.3, Creation Rd. II, Science-based Industrial Park, Hsinchu, Taiwan
Tel: 886-3-563-1999
Fax: 886-3-563-1189

**Holtek Semiconductor Inc. (Sales Office)**
11F, No.576, Sec.7 Chung Hsiao E. Rd., Taipei, Taiwan
Tel: 886-2-2782-9635
Fax: 886-2-2782-9636
Fax: 886-2-2782-7128 (International sales hotline)

**Holtek Semiconductor (Hong Kong) Ltd.**
RM.711, Tower 2, Cheung Sha Wan Plaza, 833 Cheung Sha Wan Rd., Kowloon, Hong Kong
Tel: 852-2-745-8288
Fax: 852-2-742-8657

**Holtek Semiconductor (Shanghai) Inc.**
7th Floor, Building 2, No.889, Yi Shan Rd., Shanghai, China
Tel: 021-6485-5560
Fax: 021-6485-0313

**Holmate Technology Corp.**
48531 Warm Springs Boulevard, Suite 413, Fremont, CA 94539
Tel: 510-252-9880
Fax: 510-252-9885