# HT48RA0-3/HT48CA0-3
# Remote Type 8-Bit MCU

## Technical Document

- Tools Information
- FAQs
- Application Note
  - HA0016E  Writing and Reading to the HT24 EEPROM with the HT48 MCU Series
  - HA0018E  Controlling the HT1621 LCD Controller with the HT48 MCU Series
  - HA0041E  Using the HT48CA0 to Generate the HT6221 Output Signals
  - HA0075E  MCU Reset and Oscillator Circuits Application Note
  - HA0076E  HT48RAx/HT48CAx Software Application Note
  - HA0082E  HT48xA0-1 and HT48xA0-2 Power-on Reset Timing

## Features

- Operating voltage: $f_{SYS}$=4MHz($\pm$3%) at 2.0V~3.6V, Temperature = 0°C ~ +50°C
- 10 bidirectional I/O lines
- 6 Schmitt trigger input lines (PB7 without Pull-high resistor)
- One programmable carrier output - using 9-bit timer
- On-chip RC oscillator - 4MHz $\pm$3% when $V_{DD}$=2.0V~3.6V; Temperature = 0°C ~ +50°C
- Watchdog Timer
- 1K$\times$14 program memory
- 32$\times$8 data RAM
- Power-down and wake-up features reduce power consumption
- 62 powerful instructions
- Up to 1$\mu$s instruction cycle with 4MHz system clock
- All instructions executed in 1 or 2 machine cycles
- 14-bit table read instructions
- One-level subroutine nesting
- Bit manipulation instructions
- Low voltage reset function
- 20-pin SOP/SSOP package

## General Description

The HT48RA0-3/HT48CA0-3 are 8-bit high performance, RISC architecture microcontroller devices specifically designed for multiple I/O control product applications. The mask version HT48CA0-3 is fully pin and functionally compatible with the OTP version HT48RA0-3 device.

The advantages of low power consumption, I/O flexibility, timer functions, watchdog timer, HALT and wake-up functions, as well as low cost, enhance the versatility of this device to suit a wide range of application possibilities such as industrial control, consumer products, and particularly suitable for use in products such as infrared remote controllers and various subsystem controllers.

**Block Diagram**
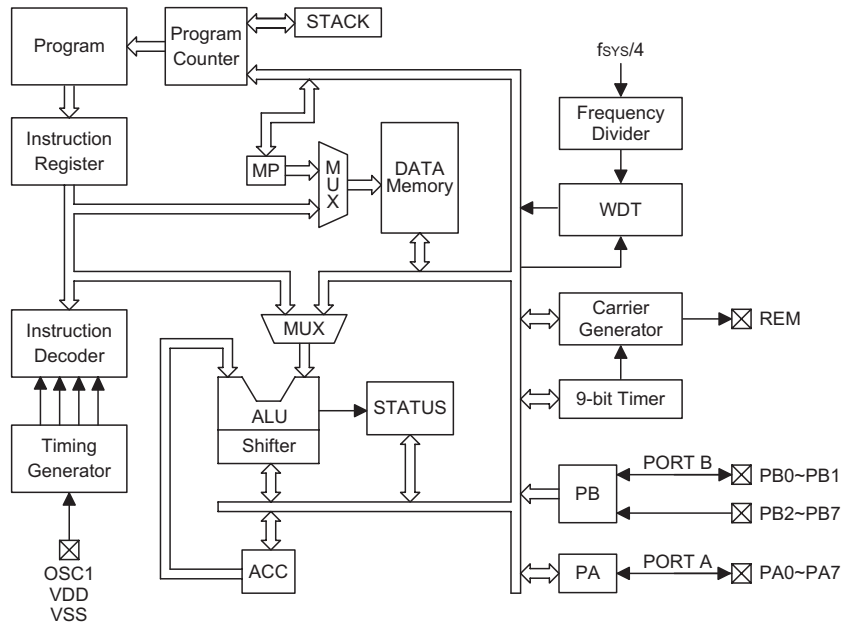


**Pin Assignment**
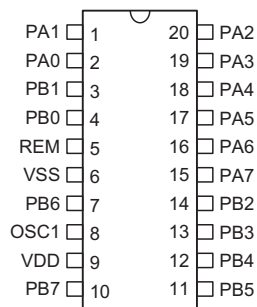


**HT48RA0-3/HT48CA0-3**
**— 20 SOP-A/SSOP-A**

## Pin Description

| Pin Name | I/O | Configuration Option | Description |
|---|---|---|---|
| PA0~PA7 | I/O | — | Bidirectional 8-bit input/output port with pull-high resistors. Software instructions determine if the pin is an NMOS output or Schmitt Trigger input. |
| PB0, PB1 | I/O | Wake-up | Bidirectional 2-bit input/output lines with pull-high resistors. Each individual bit can be configured as a wake-up input by a configuration option. Software instructions determine if the pin is an NMOS output or Schmitt Trigger input. |
| PB2~PB6 | I | Wake-up | 5-bit Schmitt Trigger input lines with pull-high resistors. Each individual bit can be configured as a wake-up input by a configuration option. |
| PB7 | I | Wake-up | 1-bit Schmitt trigger input lines without pull-high resistor. This bit can be configured as a wake-up input by a configuration option. |
| REM | O | — | Carrier output pin. |
| OSC1 | I | — | OSC1 is connected to an external resistor for the internal system clock. |
| VDD | — | — | Positive power supply |
| VSS | — | — | Negative power supply, ground |

## Absolute Maximum Ratings

Supply Voltage ...........................$V_{SS}$−0.3V to $V_{SS}$+4.0V

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

$I_{OL}$ Total ............................................150mA

Total Power Dissipation ....................................500mW

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

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

$I_{OH}$ Total............................................−100mA

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

## D.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $V_{DD}$ | Operating Voltage | — | — | 2.0 | — | 3.6 | V |
| $I_{DD}$ | Operating Current | 3V | No load, $f_{SYS}$=4MHz | — | 0.7 | 1.5 | mA |
| $I_{STB}$ | Standby Current | 3V | No load, system HALT | — | — | 1 | μA |
| $V_{IL}$ | Input Low Voltage for I/O Ports | 3V | — | 0 | — | $0.2V_{DD}$ | V |
| $V_{IH}$ | Input High Voltage for I/O Ports | 3V | — | $0.8V_{DD}$ | — | $V_{DD}$ | V |
| $V_{LVR}$ | Low Voltage Reset Voltage | — | — | 1.8 | 1.9 | 2.0 | V |
| $I_{OL}$ | I/O Ports Sink Current | 3V | $V_{OL}$=$0.1V_{DD}$ | 4 | 8 | — | mA |
| $I_{OH}$ | REM Output Source Current | 3V | $V_{OH}$=$0.9V_{DD}$ | −5 | −7 | — | mA |
| $R_{PH}$ | Pull-high Resistance | 3V | — | 100 | 150 | 200 | kΩ |
| $V_{POR}$ | $V_{DD}$ Start Voltage to ensure Power-on Reset | — | — | — | — | 100 | mV |
| $R_{POR}$ | $V_{DD}$ Rise Rate to ensure Power-on Reset | — | — | 0.035 | — | — | V/ms |

## A.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|--------|-----------|-----------------|---|------|------|------|------|
| | | $V_{DD}$ | Conditions | | | | |
| $f_{SYS}$ | System Clock | 3V | $f_{SYS}$=4MHz(±3%) , Temp. = 0°C ~ +50°C | 3880 | 4000 | 4120 | kHz |
| $t_{SST}$ | System Start-up Timer Period | — | Power-up, reset or wake-up from HALT | — | 1024 | — | $t_{SYS}$ |
| $t_{LVR}$ | Low Voltage Width to Reset | — | — | 0.25 | 1 | 2 | ms |
| $t_{POR}$ | Power-on Reset Low Pulse Width | — | — | 1 | — | — | µs |

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

## Functional Description

### Execution Flow

The HT48RA0-3/HT48CA0-3 system clock is an RC type clock which requires the connection of an external resistor for its operation. It 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 one instruction cycle while decoding and execution takes the next instruction cycle. However, the pipelining scheme causes each instruction to effectively execute within one cycle. If an instruction changes the program counter, two cycles are required to complete the instruction.

### Program Counter – PC

The 10-bit program counter (PC) controls the sequence in which the instructions stored in program memory are executed and its contents specify a maximum of 1024 addresses.

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 or return from subroutine, the PC manipulates the program transfer by loading the address corresponding to each instruction.
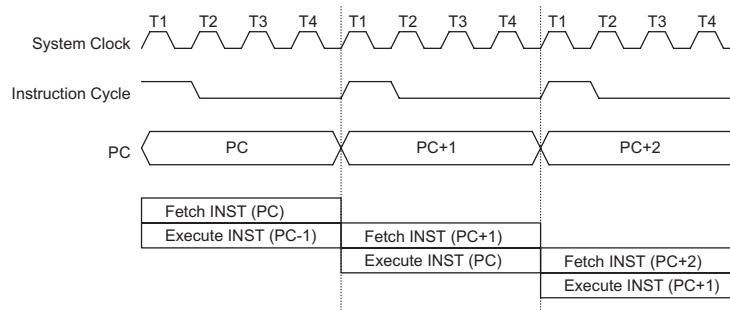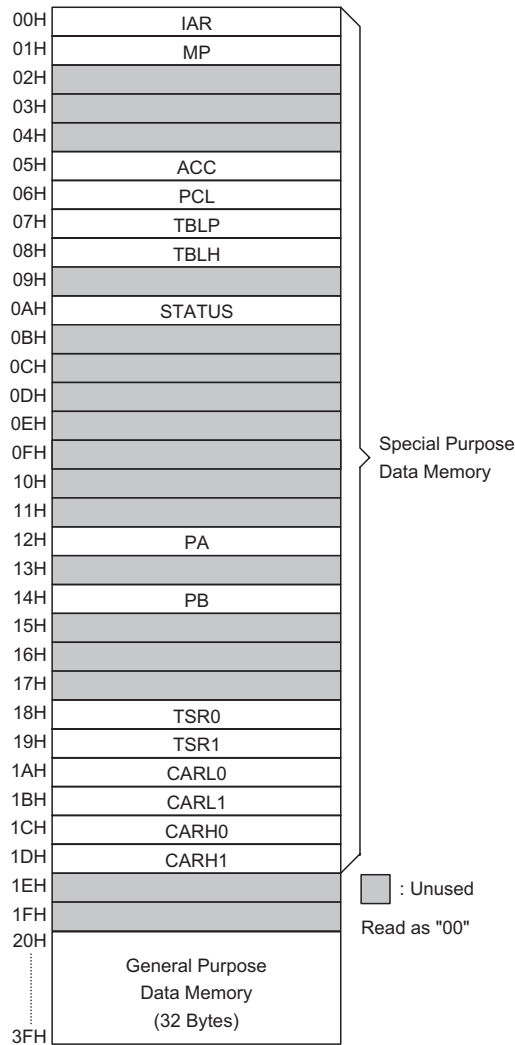


**Execution Flow**

| Mode | Program Counter | | | | | | | | | |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| | *9 | *8 | *7 | *6 | *5 | *4 | *3 | *2 | *1 | *0 |
| Initial reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Skip | Program Counter + 2 | | | | | | | | | |
| Loading PCL | *9 | *8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| Jump, call branch | #9 | #8 | #7 | #6 | #5 | #4 | #3 | #2 | #1 | #0 |
| Return from subroutine | S9 | S8 | S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 |

**Program Counter**

Note:   *9~*0: Program counter bits          S9~S0: Stack register bits

      #9~#0: Instruction code bits          @7~@0: PCL bits

The conditional skip is activated by instruction. 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 writeable 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.

**Program Memory − ROM**

The program memory is used to store the program instructions which are to be executed. It also contains data and table and is organized into 1024×14 bits, 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 the initialization program. After a device reset, the program always begins execution at location 000H.

• Table location
  Any location in the Program Memory space can be used as a look-up table. The instructions TABRDC [m] (the current page, one page=256 words) and TABRDL [m] (the last page) transfer the contents of the lower-order byte to the specified data memory register, and the higher-order byte to TBLH (08H). Only the destination of the lower-order byte in the ta-

ble is well-defined, the other bits of the table word are transferred to the lower portion of TBLH, 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), where P indicates the table location. Before accessing the table, the location must be placed in TBLP. The TBLH is read only and cannot be restored. All table related instructions need 2 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 used to save the contents of the program counter only. The stack is organized into one level 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 and is neither readable nor writeable. At a subroutine call the contents of the program counter are pushed onto the stack. At the end of a subroutine signaled by a return instruction, RET, 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 ″CALL″ is subsequently executed, stack overflow occurs and the first entry will be lost and only the most recent return address is stored.

**Data Memory − RAM**

The data memory is divided into two functional groups: special function registers and general purpose data memory (32×8). Most are read/write, but some are read only.

The remaining space before the 20H is reserved for future expanded usage and reading these locations will return the result 00H. The general purpose data memory, addressed from 20H to 3FH, is used for data and control information under instruction command. All 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 the SET [m].i and CLR [m].i instructions, respectively. They are also indirectly accessible through memory pointer register (MP;01H).



**Program Memory**

| Instruction(s) | Table Location | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *9 | *8 | *7 | *6 | *5 | *4 | *3 | *2 | *1 | *0 |
| TABRDC [m] | P9 | P8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| TABRDL [m] | 1 | 1 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |

**Table Location**

Note:  *9~*0: Table location bits          @7~@0: Table pointer bits
       P9~P8: Current program counter bits

| 00H | IAR |
|---|---|
| 01H | MP |
| 02H | |
| 03H | |
| 04H | |
| 05H | ACC |
| 06H | PCL |
| 07H | TBLP |
| 08H | TBLH |
| 09H | |
| 0AH | STATUS |
| 0BH | |
| 0CH | |
| 0DH | |
| 0EH | |
| 0FH | |
| 10H | |
| 11H | |
| 12H | PA |
| 13H | |
| 14H | PB |
| 15H | |
| 16H | |
| 17H | |
| 18H | TSR0 |
| 19H | TSR1 |
| 1AH | CARL0 |
| 1BH | CARL1 |
| 1CH | CARH0 |
| 1DH | CARH1 |
| 1EH | |
| 1FH | |
| 20H | |
| | General Purpose Data Memory (32 Bytes) |
| 3FH | |

Special Purpose Data Memory

▢ : Unused

Read as "00"

**RAM Mapping**

### Indirect Addressing Register

Location 00H is an indirect addressing register that is not physically implemented. Any read/write operation to [00H] accesses the 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. Bit 7 of MP is undefined and reading will return the result "1". Any writing operation to MP will only transfer the lower 7-bits of data to MP.

### Accumulator

The accumulator closely relates to ALU operations. It is also mapped to location 05H of the data memory and is capable of carrying out immediate data operations. Data movement between two data memory locations has to pass through the accumulator.

### Arithmetic and Logic Unit − ALU

This circuit performs 8-bit arithmetic and logic operation. 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 contents of the status register.

### Status Register − STATUS

This 8-bit status 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.

| 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 when either a system power-up or executing the CLR WDT instruction. PDF is set by executing the HALT instruction. |
| 5 | TO | TO is cleared by a 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**

With the exception of the TO and PDF flags, the other status register bits can be altered by instructions like most other register. Any data written into the status register will not change the TO or PDF flags. In addition it should be noted that operations related to the status register may give different results from those intended. The TO and PDF flags can only be changed by the Watchdog Timer overflow, device power-up, clearing the Watchdog Timer and executing the HALT instruction.

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

In addition, on executing a subroutine call, the status register will not be automatically pushed onto the stack. 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.

### Oscillator Configuration

Only an external RC oscillator type is supported for the HT48RA0-3/HT48CA0-3.
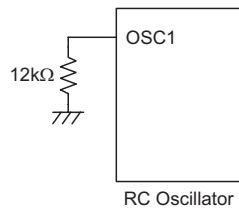


RC Oscillator

**System Oscillator**

An external resistor between OSC1 and VSS in needed whose resistance must be 12kΩ for a 4MHz frequency.

The RC oscillator provides ± 3% accuracy, the conditions are:

• $V_{DD}$= 2.0V ~ 3.6V

• Temperature = 0°C ~ +50°C

• $f_{SYS}$= 4MHz

### Watchdog Timer – WDT

The WDT clock source is implemented by the instruction clock which is the system clock divided by 4. The clock source is processed by a frequency divider and a prescaler to provide various time out periods.

WDT time out period = $\dfrac{\text{Clock Source}}{2^n}$

Where n= 8~11 selected by a configuration option.

The WDT timer is designed to prevent a software malfunction or sequence jumping to an unknown location with unpredictable results. The Watchdog Timer can be disabled by configuration option. If the Watchdog Timer is disabled, all the executions related to the WDT result in no operation and the WDT will lose its protection purpose. In this situation the logic can only be restarted by external logic.

A WDT overflow under normal operation will initialise a ″chip reset″ and set the status bit ″TO″. To clear the contents of the WDT prescaler, two methods are adopted, software instructions or a HALT instruction. There are two types of software instructions. One type is the single instruction ″CLR WDT″, the other type comprises two instructions, ″CLR WDT1″ and ″CLR WDT2″. Of these two types of instructions, only one can be active depending on the configuration option – ″CLR WDT times selection option″. If the ″CLR WDT″ is selected (i.e.. CLR WDT times equal one), any execution of the CLR WDT instruction will clear the WDT. In case ″CLR WDT1″ and ″CLR WDT2″ are chosen (i.e.. CLR WDT times equal two), these two instructions must be executed to clear the WDT; otherwise, the WDT may reset the chip due to a time-out.



**Watchdog Timer**
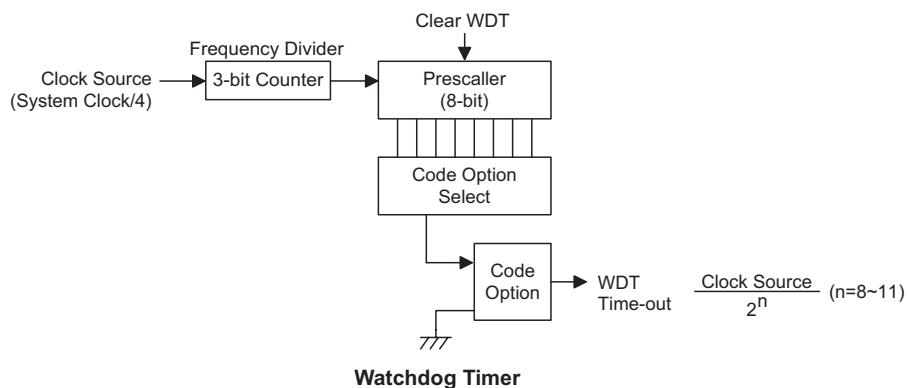
**Power Down Operation − HALT**

The Power-down mode is initialised by the HALT instruction and results in the following:

• The system oscillator turns off and the WDT stops.

• The contents of the on-chip Data Memory and registers remain unchanged.

• WDT prescaler is cleared.

• All I/O ports maintain their original status.

• The PDF flag is set and the TO flag is cleared.

The system can quit the HALT mode by means of an external falling edge signal on port B. By examining the TO and PDF flags, the reason for chip reset can be determined. The PDF flag is cleared when the system powers up or when a CLR WDT instruction is executed and is set when the HALT instruction is executed. The TO flag is set if the WDT time-out occurs during normal operation.

The port B wake-up can be considered as a continuation of normal execution. Each bit in port B can be independently selected to wake up the device by the code option. Awakening from an I/O port stimulus, the program will resume execution of the next instruction.

Once a wake-up event(s) occurs, it takes 1024 $t_{SYS}$ (system clock periods) to resume normal operation. In other words, a dummy cycle period will be inserted after the wake-up.

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

**Reset**

There are three ways in which a reset can occur:

• Power On reset

• Low Voltage reset

• WDT time-out reset during normal operation
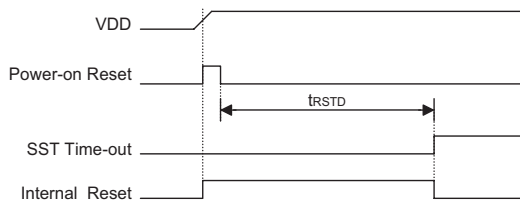
Some registers remain unchanged during reset conditions. Most registers are reset to the "initial condition" when the reset conditions are met. By examining the PDF and TO flags, the program can distinguish between different chip resets.

| TO | PDF | RESET Conditions |
|----|-----|------------------|
| 0 | 0 | Power-on reset during power-up |
| u | u | LVR reset during normal operation |
| 1 | u | WDT time-out during normal operation |

Note: "u" means unchanged.

To guarantee that the system oscillator has started and stabilized, the SST (System Start-up Timer) provides an extra-delay of 1024 system clock pulses when the system powers up or when the system awakes from a HALT state.

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

The functional unit chip reset status is shown below.

| | |
|---|---|
| Program Counter | 000H |
| WDT Prescaler | Clear |
| Input/Output ports | Input mode |
| Stack Pointer | Points to the top of the stack |
| Carrier output | Low level |
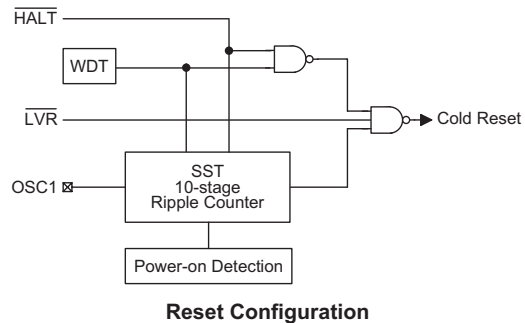


**Reset Configuration**



**Reset Timing Chart**

The chip reset status of the registers is summarised in the following table:

| Register | Power On | Low Voltage Reset | WDT Time-out (Normal Operation) |
|---|---|---|---|
| Program Counter | 000H | 000H | 000H |
| MP | -xxx xxxx | -uuu uuuu | -uuu uuuu |
| ACC | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| TBLP | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| TBLH | --xx xxxx | --uu uuuu | --uu uuuu |
| STATUS | --00 xxxx | --uu uuuu | --1u uuuu |
| PA | 1111 1111 | 1111 1111 | 1111 1111 |
| PB | 1111 1111 | 1111 1111 | 1111 1111 |
| TSR0 | 0000 0000 | 0000 0000 | 0000 0000 |
| TSR1 | 1000 0000 | 1000 0000 | 1000 0000 |
| CARL0 | 0000 0000 | 0000 0000 | 0000 0000 |
| CARL1 | 0000 0000 | 0000 0000 | 0000 0000 |
| CARH0 | 0000 0000 | 0000 0000 | 0000 0000 |
| CARH1 | 0000 0010 | 0000 0010 | 0000 0010 |

Note:  ″u″  means unchanged

  ″x″  means unknown

  ″-″ stands for unimplemented

**Input/Output Ports**

There are an 8-bit bidirectional input/output port, a 6-bit input with 2-bit I/O port in the HT48RA0-3/HT48CA0-3, labeled PA and PB which are mapped to [12H], [14H] of the Data Memory, respectively. Each bit of PA can be selected as NMOS output or Schmitt trigger input with pull-high resistor by a software instruction. PB0~PB1 have the same structure as PA, while PB2~PB6 can only be used for input operation - Schmitt trigger with pull-high resistors. PB7 is used for input operation - Schmitt trigger but without pull-high resistor.

When PA and PB are used for input operation, these ports are non-latched, that is, the inputs should be ready at the T2 rising edge of the instruction ″MOV A, [m]″ (m=12H or 14H). For PA and PB0~PB1 output operation, all data is latched and remains unchanged until the output latch is rewritten.

When PA and PB0~PB1 is used for input operation, it should be noted that before reading data from the pads, a ″1″ should be written to the related bits to disable the NMOS device. That is, the instruction ″SET [m].i″ (i=0~7 for PA, i=0~1 for PB) is executed first to disable related NMOS device, and then ″MOV A, [m]″ to get stable data.

After chip reset, PA and PB remain at a high level input line. Each bit of PA and PB0~PB1 output latches can be set or cleared by the ″SET [m].i″ and ″CLR [m].i″ (m=12H or 14H) instructions respectively.

Some instructions first input data and then follow the output operations. For example, ″SET [m].i″, ″CLR [m]″, ″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 to the accumulator.

Each line of PB has a wake-up capability selectable via a configuration option.
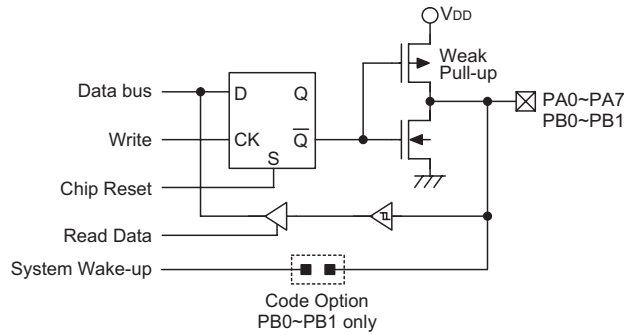
**PA, PB0~PB1 Input/Output Lines**

www.DataSheet4U.com
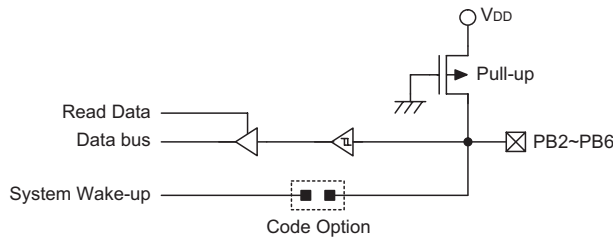
**PB2~PB6 Input Lines**
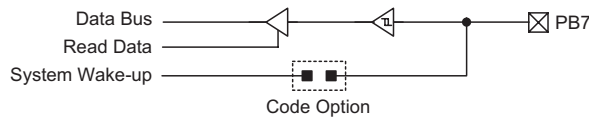
**PB7 Input Line**

## Timer

The timer is an internal unit for creating a remote control transmission pattern. As shown, it consists of a 9-bit down counter (t8 to t0), a flag (t9) permitting the 1-bit timer output, and a zero detector.

| No. | Label | Function |
|-----|-------|----------|
| 0~7 | t0~t7 | Down counter |

**TSR0 (18H) Register**

| No. | Label | Function |
|-----|-------|----------|
| 0 | t8 | Down counter |
| 1 | t9 | Timer enable, initial value is ″0″. |
| 2~6 | — | Unused bit, read as ″0″. |
| 7 | TOEF | Timer operation end flag, initial value is ″1″. |

**TSR1 (19H) Register**

## Timer Operation

The timer starts counting down when a value other than ″0″ is set for the down counter with a timer manipulation instruction. The timer manipulation instructions for making the timer start operation are shown below:

MOV A,XXH    ; XX = 00H ~ FFH
MOV TSR0,A
MOV A,XXH    ; XX ≤ 01H, t8
MOV TSR1,A
SET TSR1.1    ; The timer is started by set t9=1

Addition notes for the 9-bit timer:

• Writing to TSR0 will only put the written data to the TSR0 register (t7~t0) and writing to TSR1 (t8) will transfer the specified data and contents of TSR0 to the Down Counter. TOEF will be cleared after the data transferred from TSR1 and TSR0 to the Down Counter is completed and then wait until TSR1.1 is set by user.

• Setting TSR1.1=1, the timer will start counting. The timer will stop when its count is equal to ″0″ and then TOEF is set equal to ″1″.

- If the TSR1.1 is cleared during the timer counting, the timer will be stopped. Once the TSR1.1 is set (1→0→1), the down counter will reload data from t8~t0, and then the down counter begins counting down with the new load data.

- If TSR1.1 and TOEF are equal to 1 both, the timer can re-start, after new data is written to TSR0, TSR1 (t0~t8) in sequence.

Note: If the contents of the Down counter is 000H, set the t9 to start the timer counting, the timer will only count 1 step. The timer output time=$64/f_{SYS}$. → [ $(0+1) \times 64/f_{SYS}=64/f_{SYS}$ ]

The down counter is decremented $(-1)$ in the cycle of $64/f_{SYS}$. If the value of the down counter becomes "0", the zero detector generates the timer operation end signal to stop the timer operation. At this time, TOEF will be set to "1". The output of the timer operation end signal is continued while the down counter is "0" and the timer is stopped. The following relational expression applies between the timer's output time and the down counter's set value.

Timer output time = (Set value+1) $\times 64/f_{SYS}$

An example is shown below.
```
MOV A,0FFH
MOV TSR0,A
MOV A,01H
MOV TSR1,A
SET TSR1.1
```

In the case above, the timer output time is as follows.

(Set value+1) $\times 64/f_{SYS}$

= $(511+1) \times 16\mu s$

= 8.192ms
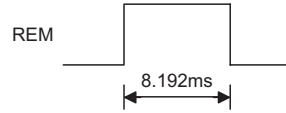


By setting the flag (t9) that enables the timer output to "1", the timer can output its operation status from the REM pin. The REM pin can also output the carrier while the timer is in operation.

Note: The carrier output results if bit 9 of the high-level period setting modulo register (CARH) is cleared ("0").
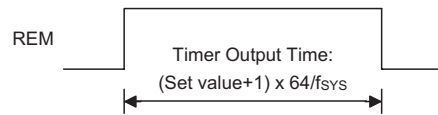


**Timer Output when Carrier is not Output**
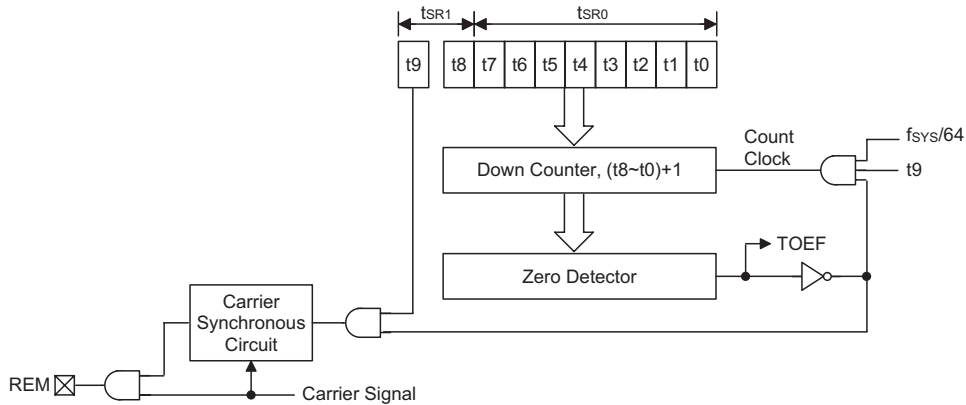


**Timer Configuration**

**Carrier Output**

• Carrier output generator

The carrier generator consists of a 9-bit counter and two modulo registers for setting the high-level and low-level periods - CARH and CARL respectively.

| Register | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|----------|------|------|------|------|------|------|------|------|
| CARL0 | CL.7 | CL. | CL. | CL. | CL.3 | CL.2 | CL.1 | CL.0 |
| CARL1 | — | — | — | — | — | — | Fix "0" | CL.8 |
| CARH0 | CH.7 | CH.6 | CH.5 | CH.4 | CH.3 | CH.2 | CH.1 | CH.0 |
| CARH1 | — | — | — | — | — | — | CH.9 (CARY) | CH.8 |

**CARL0 (1AH) Register, CARL1 (1BH), CARH0 (1CH) Register, CARH1 (1DH), Register**

Note: 1. CARH1.1 (CARY) initial value is "1".

2. CARL1.2 (CARH1.2)~CARL1.7 (CARH1.7) are unused bits, read as "0".

The carrier duty ratio and carrier frequency can be determined by setting the high-level and low-level widths using the respective modulo registers. Each of these widths can be set in a range of 500ns to 64$\mu$s at f$_{SYS}$ = 4MHz.
CARH (CARH1.0, CARH0.7~CARH0.0) and CARL (CARL1.0, CARL0.7~CARL0.0) are read and written using instructions.

Example:

```
    MOV    A,XXH        ; XXH = 00H~FFH
    MOV    CARL0,A
    MOV    A,XXH        ; XXH ≤ 01H, CL.8 (CARL1.0)
    MOV    CARL1,A
    MOV    A,XXH        ; XXH = 00H~FFH
    MOV    CARH0,A
    MOV    A,XXH        ; XXH ≥ 02H, CH.8 (CARH1.0)
    MOV    CARH1,A
    CLR    CARH1.1      ; The carrier is started by clearing CARY(CARH1.1)="0"
```
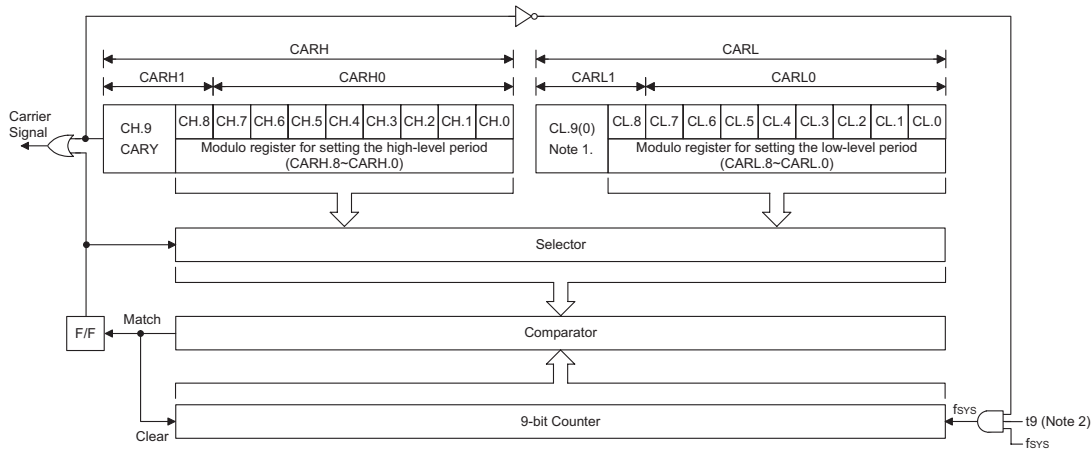


**Configuration of Remote Controller Carrier Generator**

Note: 1. Bit 9 of the modulo register for setting the low-level period (CARL) is fixed to "0".

2. t9: Flag that enables timer output (timer block, see Timer Configuration)

The values of CARH and CARL can be calculated from the following expressions.

CARL (CARL1.0, CARL0.7~CARL0.0) = $(f_{SYS} \times (1-D) \times T) - 1$

CARH (CARH1.0, CARH0.7~CARH0.0) = $(f_{SYS} \times D \times T) - 1$

D: Carrier duty ratio (0 < D < 1)
$f_{SYS}$: Input clock (Mhz)
T: Carrier cycle (μs)
Ensure to input values in the range of 001H to 1FFH to CARL and CARH.

Example:

$f_{SYS}$ = 4MHz, fc = 38.1kHz, T = 1/ fc = 26.25μs, duty = 1/3

CARL = (4M × (1−1/3) × 26.25μs) − 1 = 69 = 45H

CARH = (4M × 1/3 × 26.25μs) − 1 = 34 = 22H

```
MOV   A,045H
MOV   CARL0,A
MOV   A,022H
MOV   CARH0,A
CLR   CARH1.1       ; The carrier is started by clearing CARY(CARH1.1) = "0"
```

• Carrier output control

The remote controller carrier can be output from the REM pin by clearing (0) bit 9 (CARY) of the modulo register for setting the high-level period (CARH).

When performing a carrier output, be sure to set the timer operation after setting the CARH (CARH1.0, CARH0.7~CARH0.0) and CARL (CARL1.0, CARL0.7~CARL0.0) values.

Note that a malfunction may occur if the values of CARH (CARH1.0, CARH0.7~CARH0.0) and CARL (CARL1.0, CARL0.7~CARL0.0) are changed while the carrier is being output from the REM pin.

Executing the timer manipulation instruction starts the carrier output from the low level.
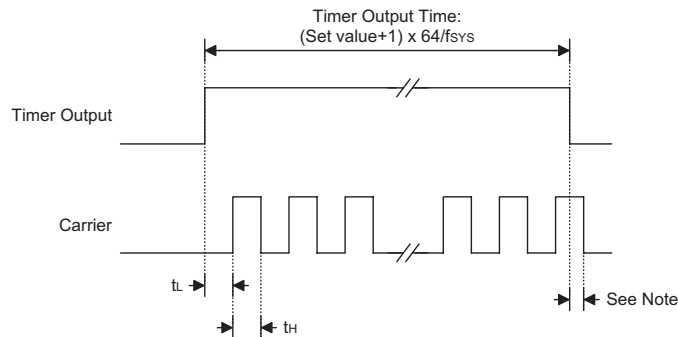


**Timer Output when Carrier Is an Output**

Note: When the carrier signal is active and during the time when the signal is high, if the timer output should go low, the carrier signal will first complete its high level period before going low.

The output from the REM pin is as follows, in accordance with the values set to bit 9 (CARY) of CARH and the timer output enable flag (t9), and the value of the timer block's 9-bit down counter (t0 to t8).

| CARH1.1 (CARY) | Timer Output Enable Flag (t9: TSR1.1) | 9-bit Down Counter (TSR0.0~TSR0.7, TSR1.0) | REM Pin |
|---|---|---|---|
| 0 | 0 | 0 | Low-level output |
| 0 | 0 | Other than 0 | Low-level output |
| 0 | 1 | 0 | 64/fsys (with carrier output) |
| 0 | 1 | Other than 0 | Carrier output (Note) |
| 1 | 0 | — | Low-level output |
| 1 | 1 | — | High-level output |

Note: Input values in the range of 001H to 1FFH to CARH (CARH1.0, CARH0.7~CARH0.0) and CARL (CARL1.0, CARL0.7~CARL0.0).

Caution: CARH (CARH1.0, CARH0.7~CARH0.0) and CARL (CARL1.0, CARL0.7~CARL0.0) must be set while the REM pin is low level (t9 = 0 or t0 to t8 = 0).

| CARH (CARH1.0, CARH0.7~CARH0.0) | CARL (CARL1.0, CARL0.7~CARL0.0) | $t_H$ (μs) | $t_L$ (μs) | t (μs) | $f_C$ (kHz) | Duty |
|---|---|---|---|---|---|---|
| 01H | 01H | 0.5 | 0.5 | 1.0 | 1000 | 1/2 |
| 03H | 05H | 1.0 | 1.5 | 2.5 | 400 | 2/5 |
| 09H | 09H | 2.5 | 2.5 | 5.0 | 200 | 1/2 |
| 13H | 13H | 5.0 | 5.0 | 10.0 | 100 | 1/2 |
| 20H | 20H | 8.25 | 8.25 | 16.5 | 60.6 | 1/2 |
| 21H | 41H | 8.25 | 16.75 | 25 | 40 | 1/3 |
| 22H | 44H | 8.75 | 17.25 | 26.0 | 38.5 | 1/3 |
| 22H | 45H | 8.75 | 17.5 | 26.25 | 38.10 | 1/3 |
| 22H | 46H | 8.8 | 17.6 | 26.4 | 37.9 | 1/3 |
| 23H | 48H | 9.0 | 18.25 | 27.25 | 36.7 | 1/3 |
| 24H | 49H | 9.26 | 18.52 | 27.78 | 36.0 | 1/3 |
| 34H | 6AH | 13.33 | 26.66 | 40.0 | 25 | 1/3 |
| 3BH | 3BH | 15.0 | 15.0 | 30.0 | 33.3 | 1/2 |
| 63H | 63H | 25.0 | 25.0 | 50.0 | 20 | 1/2 |
| 7FH | 7FH | 32.0 | 32.0 | 64.0 | 15.6 | 1/2 |

**Carrier Frequency Setting ($f_{SYS}$=4MHz)**
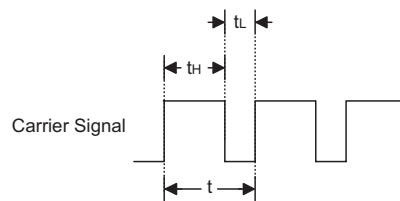
## Low Voltage Reset − LVR

The microcontroller provides a 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 when 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 this state for a time in excess of 1ms. If the low voltage state does not exceed 1ms, the LVR will ignore it and will not perform a reset function.

www.DataSheet4U.com

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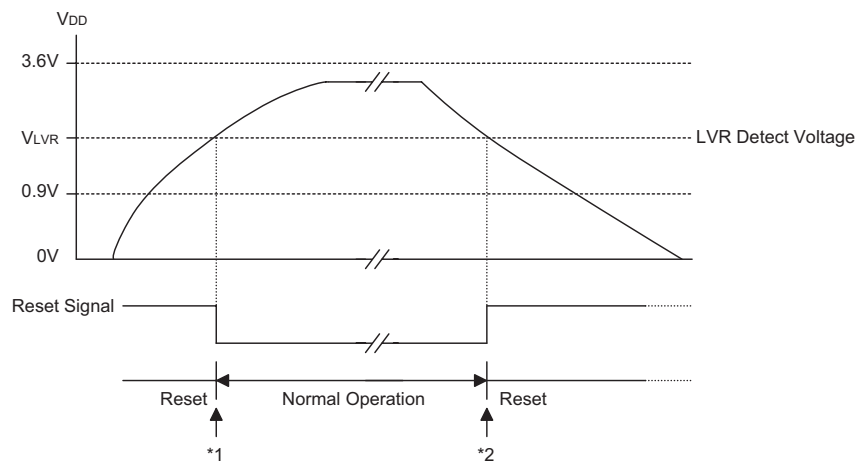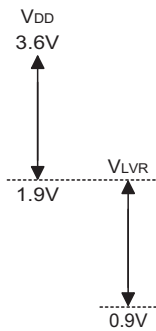




**Low Voltage Reset**

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

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

## Configuration Options

The following table shows eight kinds of configuration options for the HT48RA0-3/HT48CA0-3. All the configuration options must be defined to ensure proper system functioning.

| No. | Code Option |
|---|---|
| 1 | WDT time-out period selection<br>Time-out period= $\dfrac{2^n}{\text{Clock Source}}$, where n=8~11. |
| 2 | WDT enable/disable selection. This option is to decide whether the WDT timer is enabled or disabled. |
| 3 | CLR WDT times selection. This option defines how to clear the WDT by instruction. ″One time″ means that the CLR WDT instruction can clear the WDT. ″Two times″ means only if both of the CLR WDT1 and CLR WDT2 instructions have been executed, the WDT can be cleared. |
| 4 | Wake-up selection. This option defines the wake-up activity function. External input pins (PB only) all have the capability to wake-up the device. |
| 5 | LVR function: enable or disable |

## Application Circuits



**HT48RA0-3/HT48CA0-3**

**OSC Circuit**

## Example



**HT48RA0-3/HT48CA0-3**
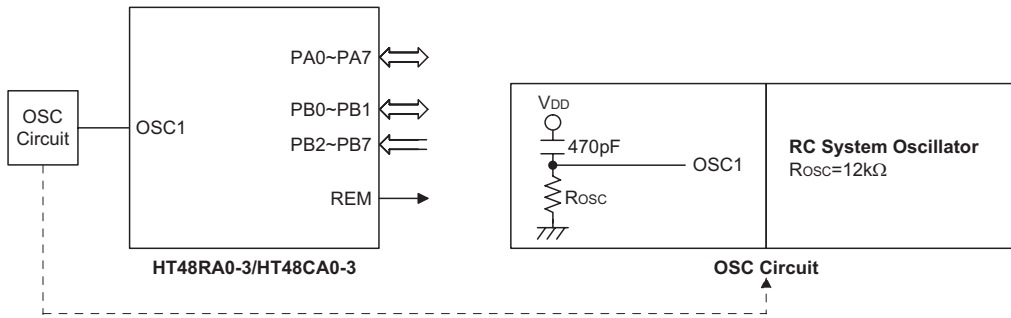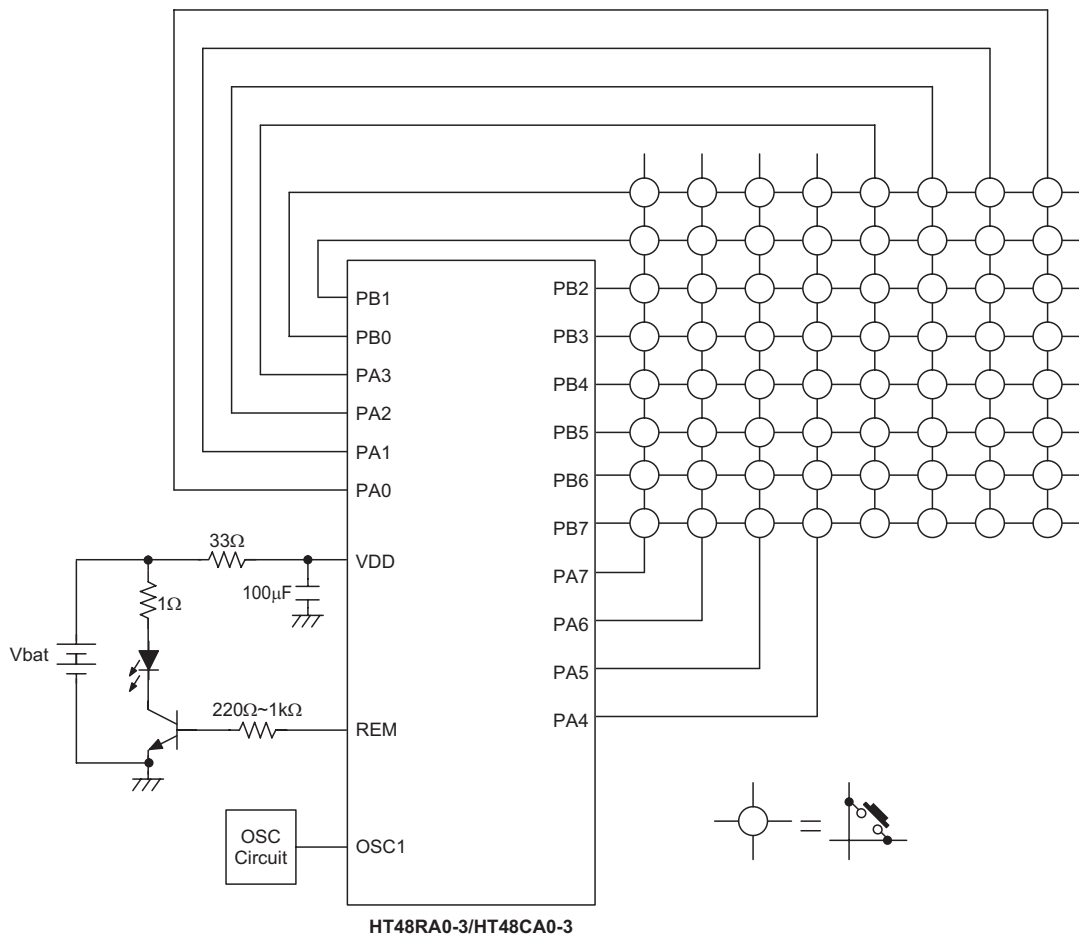
## Instruction Set

### 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 | 1[Note] | 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 | 1[Note] | 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 | 1[Note] | 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 | 1[Note] | Z, C, AC, OV |
| DAA [m] | Decimal adjust ACC for Addition with result in Data Memory | 1[Note] | 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 | 1[Note] | Z |
| ORM A,[m] | Logical OR ACC to Data Memory | 1[Note] | Z |
| XORM A,[m] | Logical XOR ACC to Data Memory | 1[Note] | 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 | 1[Note] | 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[Note] | Z |
| DECA [m] | Decrement Data Memory with result in ACC | 1 | Z |
| DEC [m] | Decrement Data Memory | 1[Note] | 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 | 1$^{Note}$ | None |
| RRCA [m] | Rotate Data Memory right through Carry with result in ACC | 1 | C |
| RRC [m] | Rotate Data Memory right through Carry | 1$^{Note}$ | C |
| RLA [m] | Rotate Data Memory left with result in ACC | 1 | None |
| RL [m] | Rotate Data Memory left | 1$^{Note}$ | None |
| RLCA [m] | Rotate Data Memory left through Carry with result in ACC | 1 | C |
| RLC [m] | Rotate Data Memory left through Carry | 1$^{Note}$ | C |
| **Data Move** | | | |
| MOV A,[m] | Move Data Memory to ACC | 1 | None |
| MOV [m],A | Move ACC to Data Memory | 1$^{Note}$ | None |
| MOV A,x | Move immediate data to ACC | 1 | None |
| **Bit Operation** | | | |
| CLR [m].i | Clear bit of Data Memory | 1$^{Note}$ | None |
| SET [m].i | Set bit of Data Memory | 1$^{Note}$ | None |
| **Branch** | | | |
| JMP addr | Jump unconditionally | 2 | None |
| SZ [m] | Skip if Data Memory is zero | 1$^{Note}$ | None |
| SZA [m] | Skip if Data Memory is zero with data movement to ACC | 1$^{note}$ | None |
| SZ [m].i | Skip if bit i of Data Memory is zero | 1$^{Note}$ | None |
| SNZ [m].i | Skip if bit i of Data Memory is not zero | 1$^{Note}$ | None |
| SIZ [m] | Skip if increment Data Memory is zero | 1$^{Note}$ | None |
| SDZ [m] | Skip if decrement Data Memory is zero | 1$^{Note}$ | None |
| SIZA [m] | Skip if increment Data Memory is zero with result in ACC | 1$^{Note}$ | None |
| SDZA [m] | Skip if decrement Data Memory is zero with result in ACC | 1$^{Note}$ | 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 | 2$^{Note}$ | None |
| TABRDL [m] | Read table (last page) to TBLH and Data Memory | 2$^{Note}$ | None |
| **Miscellaneous** | | | |
| NOP | No operation | 1 | None |
| CLR [m] | Clear Data Memory | 1$^{Note}$ | None |
| SET [m] | Set Data Memory | 1$^{Note}$ | 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 | 1$^{Note}$ | None |
| SWAPA [m] | Swap nibbles of Data Memory with result in ACC | 1 | None |
| HALT | Enter power down mode | 1 | TO, PDF |

Note: 1. For skip instructions, if the result of the comparison involves a skip then two cycles are required, if no skip takes place only one cycle is required.

2. Any instruction which changes the contents of the PCL will also require 2 cycles for execution.

3. For the ″CLR WDT1″ and ″CLR WDT2″ instructions the TO and PDF flags may be affected by the execution status. The TO and PDF flags are cleared after both ″CLR WDT1″ and ″CLR WDT2″ instructions are consecutively executed. Otherwise the TO and PDF flags remain unchanged.

## Instruction Definition

| | |
|---|---|
| **ADC A,[m]** | Add Data Memory to ACC with Carry |
| Description | The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the Accumulator. |
| Operation | ACC ← ACC + [m] + C |
| Affected flag(s) | OV, Z, AC, C |
| **ADCM A,[m]** | Add ACC to Data Memory with Carry |
| Description | The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the specified Data Memory. |
| Operation | [m] ← ACC + [m] + C |
| Affected flag(s) | OV, Z, AC, C |
| **ADD A,[m]** | Add Data Memory to ACC |
| Description | The contents of the specified Data Memory and the Accumulator are added. The result is stored in the Accumulator. |
| Operation | ACC ← ACC + [m] |
| Affected flag(s) | OV, Z, AC, C |
| **ADD A,x** | Add immediate data to ACC |
| Description | The contents of the Accumulator and the specified immediate data are added. The result is stored in the Accumulator. |
| Operation | ACC ← ACC + x |
| Affected flag(s) | OV, Z, AC, C |
| **ADDM A,[m]** | Add ACC to Data Memory |
| Description | The contents of the specified Data Memory and the Accumulator are added. The result is stored in the specified Data Memory. |
| Operation | [m] ← ACC + [m] |
| Affected flag(s) | OV, Z, AC, C |
| **AND A,[m]** | Logical AND Data Memory to ACC |
| Description | Data in the Accumulator and the specified Data Memory perform a bitwise logical AND operation. The result is stored in the Accumulator. |
| Operation | ACC ← ACC ″AND″ [m] |
| Affected flag(s) | Z |
| **AND A,x** | Logical AND immediate data to ACC |
| Description | Data in the Accumulator and the specified immediate data perform a bitwise logical AND operation. The result is stored in the Accumulator. |
| Operation | ACC ← ACC ″AND″ x |
| Affected flag(s) | Z |
| **ANDM A,[m]** | Logical AND ACC to Data Memory |
| Description | Data in the specified Data Memory and the Accumulator perform a bitwise logical AND operation. The result is stored in the Data Memory. |
| Operation | [m] ← ACC ″AND″ [m] |
| Affected flag(s) | Z |

| **CALL addr** | Subroutine call |
|---|---|
| Description | Unconditionally calls a subroutine at the specified address. The Program Counter then increments by 1 to obtain the address of the next instruction which is then pushed onto the stack. The specified address is then loaded and the program continues execution from this new address. As this instruction requires an additional operation, it is a two cycle instruction. |
| Operation | Stack ← Program Counter + 1 |
|  | Program Counter ← addr |
| Affected flag(s) | None |

| **CLR [m]** | Clear Data Memory |
|---|---|
| Description | Each bit of the specified Data Memory is cleared to 0. |
| Operation | [m] ← 00H |
| Affected flag(s) | None |

| **CLR [m].i** | Clear bit of Data Memory |
|---|---|
| Description | Bit i of the specified Data Memory is cleared to 0. |
| Operation | [m].i ← 0 |
| Affected flag(s) | None |

| **CLR WDT** | Clear Watchdog Timer |
|---|---|
| Description | The TO, PDF flags and the WDT are all cleared. |
| Operation | WDT cleared |
|  | TO ← 0 |
|  | PDF ← 0 |
| Affected flag(s) | TO, PDF |

| **CLR WDT1** | Pre-clear Watchdog Timer |
|---|---|
| Description | The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT2 and must be executed alternately with CLR WDT2 to have effect. Repetitively executing this instruction without alternately executing CLR WDT2 will have no effect. |
| Operation | WDT cleared |
|  | TO ← 0 |
|  | PDF ← 0 |
| Affected flag(s) | TO, PDF |

| **CLR WDT2** | Pre-clear Watchdog Timer |
|---|---|
| Description | The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT1 and must be executed alternately with CLR WDT1 to have effect. Repetitively executing this instruction without alternately executing CLR WDT1 will have no effect. |
| Operation | WDT cleared |
|  | TO ← 0 |
|  | PDF ← 0 |
| Affected flag(s) | TO, PDF |

**CPL [m]**                  Complement Data Memory

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

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

Affected flag(s)             Z

**CPLA [m]**                 Complement Data Memory with result in ACC

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

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

Affected flag(s)             Z

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

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

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

Affected flag(s)             C

**DEC [m]**                  Decrement Data Memory

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

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

Affected flag(s)             Z

**DECA [m]**                 Decrement Data Memory with result in ACC

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

Operation                    $ACC \leftarrow [m] - 1$

Affected flag(s)             Z

**HALT**                     Enter power down mode

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

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

Affected flag(s)             TO, PDF

**INC [m]**            Increment Data Memory

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

Operation              [m] ← [m] + 1

Affected flag(s)       Z

**INCA [m]**           Increment Data Memory with result in ACC

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

Operation              ACC ← [m] + 1

Affected flag(s)       Z

**JMP addr**           Jump unconditionally

Description            The contents of the Program Counter are replaced with the specified address. Program execution then continues from this new address. As this requires the insertion of a dummy instruction while the new address is loaded, it is a two cycle instruction.

Operation              Program Counter ← addr

Affected flag(s)       None

**MOV A,[m]**          Move Data Memory to ACC

Description            The contents of the specified Data Memory are copied to the Accumulator.

Operation              ACC ← [m]

Affected flag(s)       None

**MOV A,x**            Move immediate data to ACC

Description            The immediate data specified is loaded into the Accumulator.

Operation              ACC ← x

Affected flag(s)       None

**MOV [m],A**          Move ACC to Data Memory

Description            The contents of the Accumulator are copied to the specified Data Memory.

Operation              [m] ← ACC

Affected flag(s)       None

**NOP**                No operation

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

Operation              No operation

Affected flag(s)       None

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

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

Operation              ACC ← ACC ″OR″ [m]

Affected flag(s)       Z

**OR A,x**                    Logical OR immediate data to ACC

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

Operation                     ACC ← ACC ″OR″ x

Affected flag(s)              Z

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

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

Operation                     [m] ← ACC ″OR″ [m]

Affected flag(s)              Z

**RET**                       Return from subroutine

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

Operation                     Program Counter ← Stack

Affected flag(s)              None

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

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

Operation                     Program Counter ← Stack
                              ACC ← x

Affected flag(s)              None

**RETI**                      Return from interrupt

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

Operation                     Program Counter ← Stack
                              EMI ← 1

Affected flag(s)              None

**RL [m]**                    Rotate Data Memory left

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

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

Affected flag(s)              None

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

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

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

Affected flag(s)              None

| **RLC [m]** | Rotate Data Memory left through Carry |
|---|---|
| Description | The contents of the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into bit 0. |
| Operation | [m].(i+1) ← [m].i; (i = 0~6)<br>[m].0 ← C<br>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)<br>ACC.0 ← C<br>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)<br>[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)<br>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)<br>[m].7 ← C<br>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)<br>ACC.7 ← C<br>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 sub-tracted from the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

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

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

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

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

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

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

**SDZ [m]**                  Skip if decrement Data Memory is 0

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

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

Affected flag(s)             None

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

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

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

Affected flag(s)             None

**SET [m]**                  Set Data Memory

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

Operation                    [m] ← FFH

Affected flag(s)             None

**SET [m].i**                Set bit of Data Memory

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

Operation                    [m].i ← 1

Affected flag(s)             None

| **SIZ [m]** | Skip if increment Data Memory is 0 |
|---|---|
| Description | The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | [m] ← [m] + 1<br>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<br>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<br>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]<br>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)<br>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)<br>TBLH ← program code (high byte) |
| Affected flag(s) | None |

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

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

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

## Package Information

**20-pin SOP (300mil) Outline Dimensions**
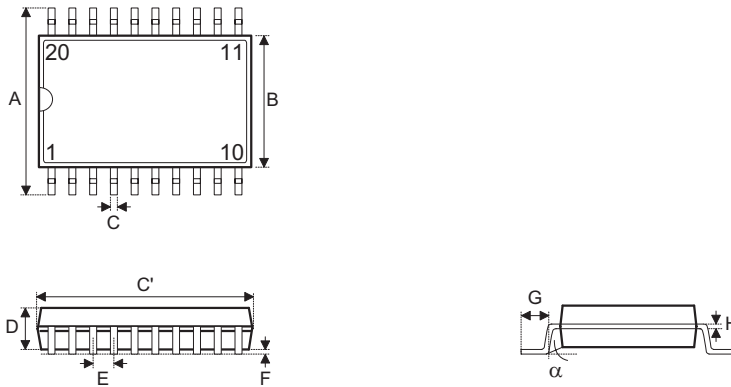


| Symbol | Dimensions in mil | | |
|---|---|---|---|
| | **Min.** | **Nom.** | **Max.** |
| A | 394 | — | 419 |
| B | 290 | — | 300 |
| C | 14 | — | 20 |
| C′ | 490 | — | 510 |
| D | 92 | — | 104 |
| E | — | 50 | — |
| F | 4 | — | — |
| G | 32 | — | 38 |
| H | 4 | — | 12 |
| α | 0° | — | 10° |

**20-pin SSOP (150mil) Outline Dimensions**
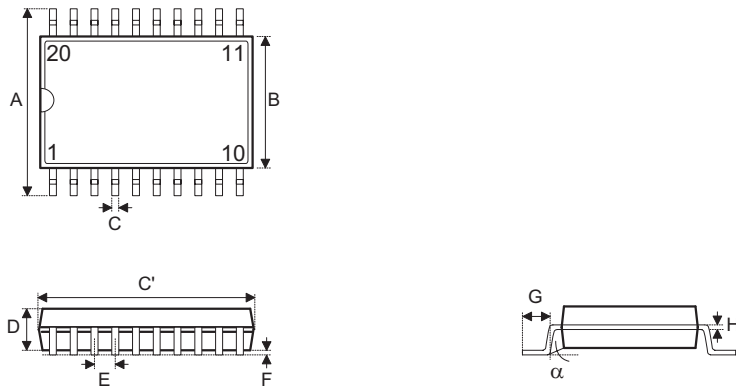


| Symbol | Dimensions in mil | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | 228 | — | 244 |
| B | 150 | — | 158 |
| C | 8 | — | 12 |
| C′ | 335 | — | 347 |
| D | 49 | — | 65 |
| E | — | 25 | — |
| F | 4 | — | 10 |
| G | 15 | — | 50 |
| H | 7 | — | 10 |
| α | 0° | — | 8° |

**Product Tape and Reel Specifications**

**Reel Dimensions**
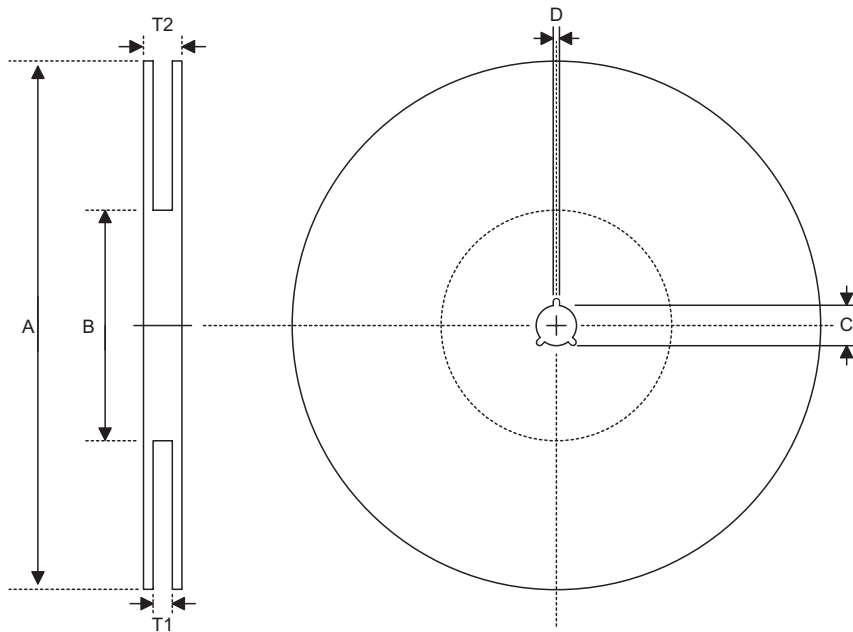


SOP 20W

| Symbol | Description | Dimensions in mm |
|--------|-------------|------------------|
| A | Reel Outer Diameter | 330±1.0 |
| B | Reel Inner Diameter | 62±1.5 |
| C | Spindle Hole Diameter | 13.0+0.5 −0.2 |
| D | Key Slit Width | 2.0±0.5 |
| T1 | Space Between Flange | 24.8+0.3 −0.2 |
| T2 | Reel Thickness | 30.2±0.2 |

SSOP 20S (150mil)

| Symbol | Description | Dimensions in mm |
|--------|-------------|------------------|
| A | Reel Outer Diameter | 330±1.0 |
| B | Reel Inner Diameter | 62±1.5 |
| C | Spindle Hole Diameter | 13.0+0.5 −0.2 |
| D | Key Slit Width | 2.0±0.5 |
| T1 | Space Between Flange | 16.8+0.3 −0.2 |
| T2 | Reel Thickness | 22.2±0.2 |

**Carrier Tape Dimensions**
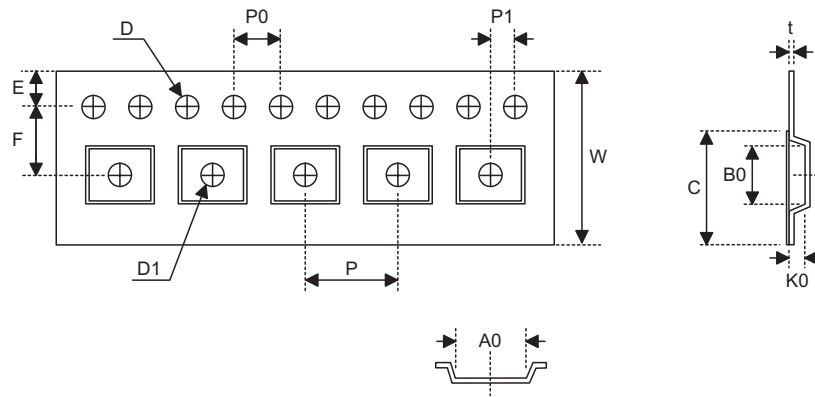


SOP 20W

| Symbol | Description | Dimensions in mm |
|--------|-------------|------------------|
| W | Carrier Tape Width | 24.0+0.3<br>−0.1 |
| P | Cavity Pitch | 12.0±0.1 |
| E | Perforation Position | 1.75±0.1 |
| F | Cavity to Perforation (Width Direction) | 11.5±0.1 |
| D | Perforation Diameter | 1.5+0.1 |
| D1 | Cavity Hole Diameter | 1.5+0.25 |
| P0 | Perforation Pitch | 4.0±0.1 |
| P1 | Cavity to Perforation (Length Direction) | 2.0±0.1 |
| A0 | Cavity Length | 10.8±0.1 |
| B0 | Cavity Width | 13.3±0.1 |
| K0 | Cavity Depth | 3.2±0.1 |
| t | Carrier Tape Thickness | 0.3±0.05 |
| C | Cover Tape Width | 21.3 |

SSOP 20S (150mil)

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