## Technical Document

- Tools Information
- FAQs
- Application Note

## Features

- Operating voltage: 2.4V~5.2V
- Up to 1μs (0.5μs) instruction cycle with 4MHz (8MHz) system clock
- System clock: 4MHz~8MHz (2.4V)
- Crystal or RC oscillator for system clock
- 12 I/O pins
- 2K×15 program ROM
- 80×8 RAM
- Two 8-bit programmable timer counter with 8-stage prescaler and one time base counter
- Watchdog Timer
- 4-level subroutine nesting
- HALT function and wake-up feature reduce power consumption
- PWM circuit direct drive speaker or output by transistor
- 28-pin SOP package

## Applications

- Intelligent educational leisure products
- Alert and warning systems
- Sound effect generators

## General Description

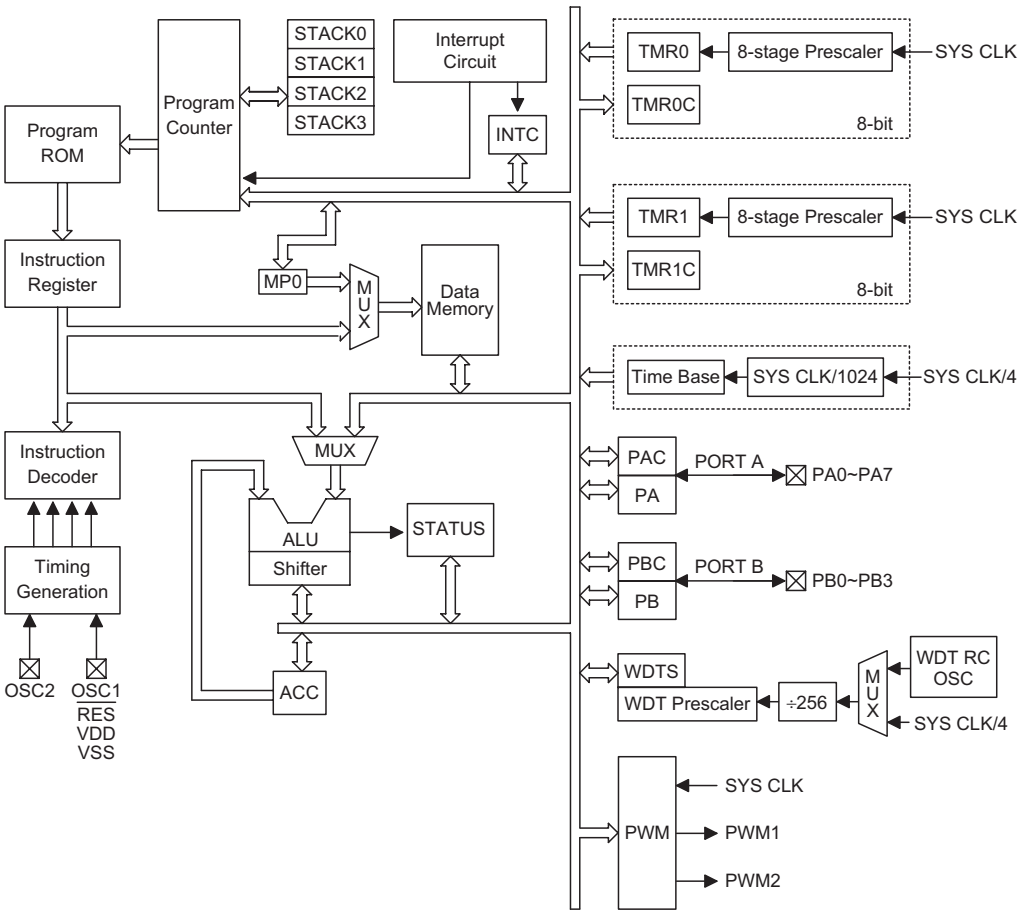
The HT83XXX is 8-bit high performance microcontroller with voice synthesizer and tone generator. The HT83XXX is designed for applications on multiple I/Os with sound effects, such as voice and melody. It can provide various sampling rates and beats, tone levels, tempos for speech synthesizer and melody generator.

The HT83XXX is excellent for versatile voice and sound effect product applications. The efficient MCU instructions allow users to program the powerful custom applications. The system frequency of HT83XXX can be up to 8MHz under 2.4V and include a HALT function to reduce power consumption.

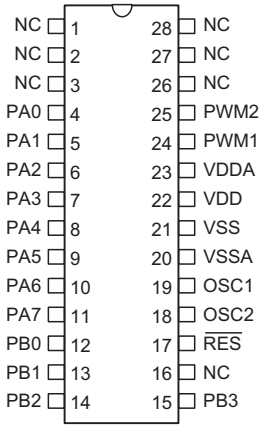
## Selection Table

| Body | HT83004 | HT83007 | HT83010 | HT83020 | HT83038 | HT83050 | HT83074 |
|---|---|---|---|---|---|---|---|
| Voice ROM Size | 64K-bit | 128K-bit | 192K-bit | 384K-bit | 768K-bit | 1024K-bit | 1536K-bit |
| Voice Length | 3 sec | 6 sec | 9 sec | 18 sec | 36 sec | 48 sec | 72 sec |

## Block Diagram



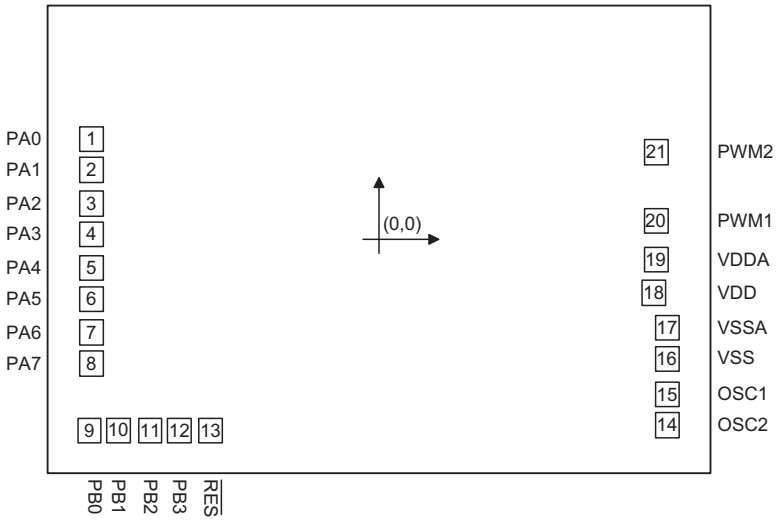

## Pin Assignment




**HT83004/HT83007/HT83010/HT83020/HT83038/HT83050/HT83074**
**—28 SOP-A**

## Pad Assignment

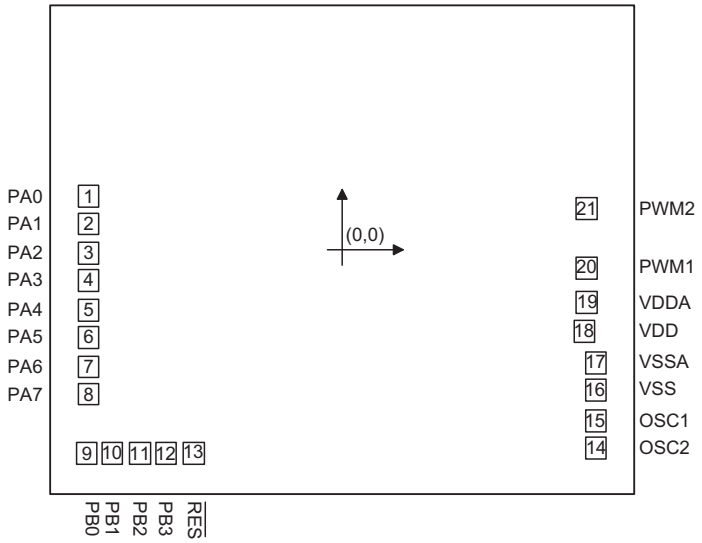
**HT83004/HT83007/HT83010**




Chip size: 2280×1475 $(\mu m)^2$

* The IC substrate should be connected to VSS in the PCB layout artwork.

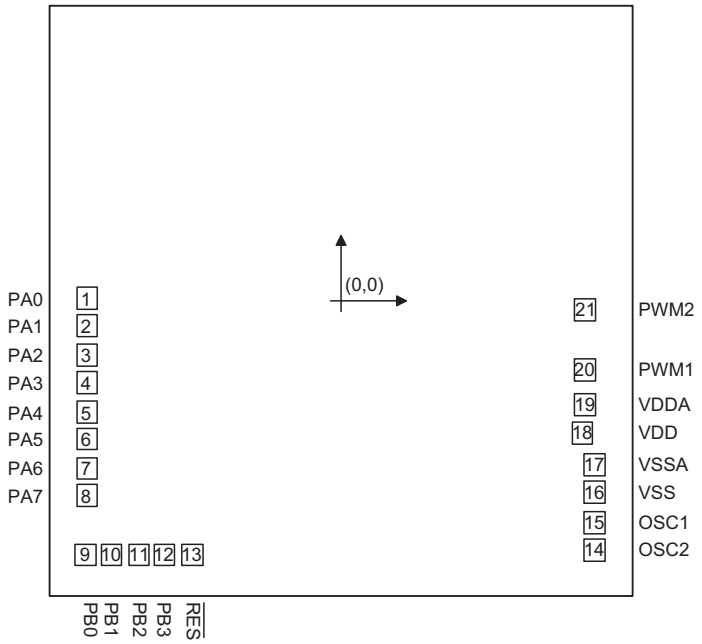
**HT83020/HT83038**




Chip size: 2180×1720 $(\mu m)^2$

* The IC substrate should be connected to in the PCB layout artwork.

**HT83050/HT83074**




Chip size: 2180×2075 $(\mu m)^2$

* The IC substrate should be connected to in the PCB layout artwork.

## Pad Coordinates

**HT83004/HT83007/HT83010**

| Pad No. | X | Y | Pad No. | X | Y |
|---|---|---|---|---|---|
| 1 | –940.400 | 307.150 | 12 | –654.200 | –587.900 |
| 2 | –940.400 | 212.150 | 13 | –551.200 | –587.900 |
| 3 | –940.400 | 109.150 | 14 | 940.400 | –571.200 |
| 4 | –940.400 | 14.150 | 15 | 940.400 | –476.200 |
| 5 | –940.400 | –88.850 | 16 | 940.600 | –368.500 |
| 6 | –940.400 | –183.850 | 17 | 940.600 | –273.000 |
| 7 | –940.400 | –286.850 | 18 | 896.250 | –165.350 |
| 8 | –940.400 | –381.850 | 19 | 904.900 | –63.250 |
| 9 | –947.200 | –587.900 | 20 | 904.900 | 56.300 |
| 10 | –852.200 | –587.900 | 21 | 904.900 | 266.800 |
| 11 | –749.200 | –587.900 | | | |

**HT83020/HT83038**

| Pad No. | X | Y | Pad No. | X | Y |
|---|---|---|---|---|---|
| 1 | –940.400 | 184.650 | 12 | –654.200 | –710.400 |
| 2 | –940.400 | 89.650 | 13 | –551.200 | –710.400 |
| 3 | –940.400 | –13.350 | 14 | 940.400 | –693.700 |
| 4 | –940.400 | –108.350 | 15 | 940.400 | –598.700 |
| 5 | –940.400 | –211.350 | 16 | 940.600 | –491.000 |
| 6 | –940.400 | –306.350 | 17 | 940.600 | –395.500 |
| 7 | –940.400 | –409.350 | 18 | 896.250 | –285.750 |
| 8 | –940.400 | –504.350 | 19 | 904.900 | –185.750 |
| 9 | –947.200 | –710.400 | 20 | 904.900 | –66.200 |
| 10 | –852.200 | –710.400 | 21 | 904.900 | 144.300 |
| 11 | –749.200 | –710.400 | | | |

**HT83050/HT83074**

| Pad No. | X | Y | Pad No. | X | Y |
|---|---|---|---|---|---|
| 1 | –940.400 | 7.150 | 12 | –654.200 | –887.900 |
| 2 | –940.400 | –87.850 | 13 | –551.200 | –887.900 |
| 3 | –940.400 | –190.850 | 14 | 940.400 | –871.200 |
| 4 | –940.400 | –285.850 | 15 | 940.400 | –776.200 |
| 5 | –940.400 | –388.850 | 16 | 940.600 | –668.500 |
| 6 | –940.400 | –483.850 | 17 | 940.600 | –573.000 |
| 7 | –940.400 | –586.850 | 18 | 896.250 | –463.250 |
| 8 | –940.400 | –681.850 | 19 | 904.900 | –363.250 |
| 9 | –947.200 | –887.900 | 20 | 904.900 | –243.700 |
| 10 | –852.200 | –887.900 | 21 | 904.900 | –33.200 |
| 11 | –749.200 | –887.900 | | | |

## Pad Description

| Pad Name | I/O | Mask Option | Description |
|---|---|---|---|
| PA0~PA7 | I/O | Wake-up, Pull-high or None | Bidirectional 8-bit I/O port. Each bit can be configured as a wake-up input by mask option. Software instructions determine the output or Schmitt trigger input with or without pull-high resistor (mask option). |
| PB0~PB3 | I/O | Pull-high or None | Bidirectional 4-bit I/O port. Software instructions determine the CMOS output or Schmitt trigger input (pull-high resistor depending on mask option). |
| | — | — | Negative power supply, ground |
| | — | — | Positive power supply |
| | — | — | PWM negative power supply, ground |
| | — | — | PWM positive power supply, ground |
| $\overline{RES}$ | I | — | Schmitt trigger reset input, active low |
| OSC1, OSC2 | — | RC or Crystal | OSC1 and OSC2 are connected to an RC network or crystal (by mask option) for the internal system clock. In the case of RC operation, OSC2 is the output terminal for 1/4 system clock. The system clock may came form the crystal, the two pins cannot be floating. |
| PWM1, PWM2 | O | — | PWM output for driving a external transistor or speaker |

## Absolute Maximum Ratings

Supply Voltage .....$V_{SS}$+2.4V 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 .....–20°C to 70°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

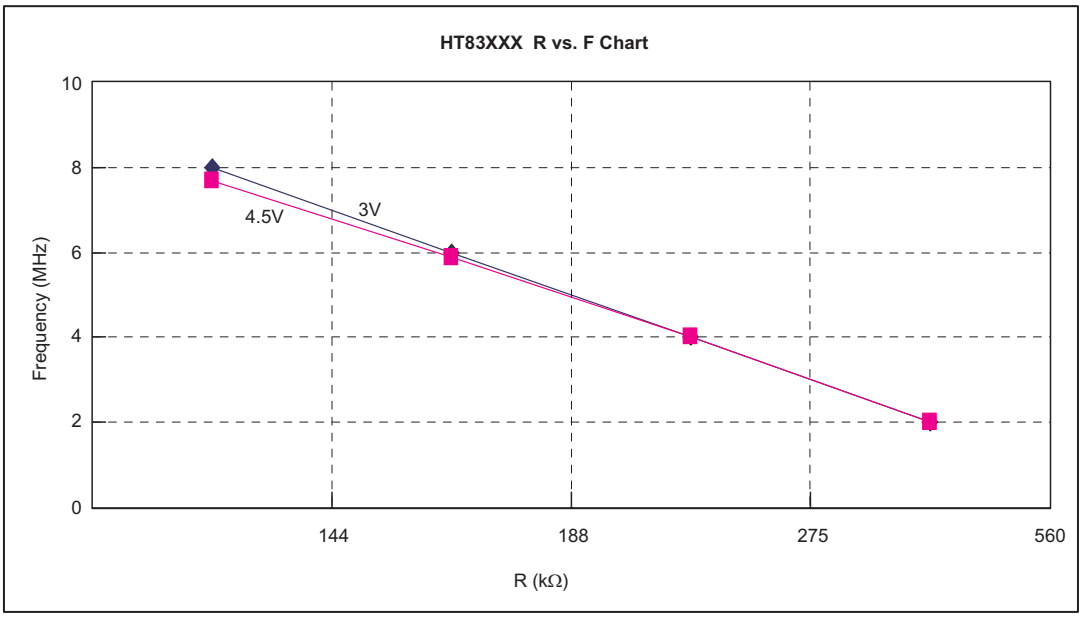
| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $V_{DD}$ | Operating Voltage | — | $f_{SYS}$=4MHz/8MHz | 2.4 | — | 5.2 | V |
| $I_{STB1}$ | Standby Current (Watchdog Off) | 3V | No load, system HALT | — | — | 1 | μA |
| | | 5V | | — | — | 2 | μA |
| $I_{STB2}$ | Standby Current (Watchdog On) | 3V | No load, system HALT | — | — | 2 | μA |
| | | 5V | | — | — | 4 | μA |
| $I_{DD}$ | Operating Current | 3V | No load, $f_{SYS}$=4MHz | — | — | 2 | mA |
| | | 5V | | — | — | 5 | mA |
| $I_{OL1}$ | I/O Port Sink Current | 3V | $V_{OL}$=0.1$V_{DD}$ | 7 | — | — | mA |
| | | 5V | | 15 | — | — | mA |
| $I_{OH1}$ | I/O Port Source Current | 3V | $V_{OH}$=0.9$V_{DD}$ | –3.5 | — | — | mA |
| | | 5V | | –8 | — | — | mA |
| $I_{OL2}$ | PWM1/PWM2 Sink Current | 3V | $V_{OL}$=0.1$V_{DD}$ | 50 | — | — | mA |
| | | 5V | | 100 | — | — | mA |
| $I_{OH2}$ | PWM1/PWM2 Source Current | 3V | $V_{OH}$=0.9$V_{DD}$ | –14.5 | — | — | mA |
| | | 5V | | –26 | — | — | mA |
| $V_{IL1}$ | Input Low Voltage for I/O Ports | 3V | — | — | 1 | — | V |
| | | 5V | | — | 2 | — | V |
| $V_{IH1}$ | Input High Voltage for I/O Ports | 3V | — | — | 2 | — | V |
| | | 5V | | — | 3.2 | — | V |
| $V_{IL2}$ | Reset Low Voltage ($\overline{RES}$) | 3V | — | — | 1.5 | — | V |
| | | 5V | | — | 2.5 | — | V |
| $V_{IH2}$ | Reset High Voltage ($\overline{RES}$) | 3V | — | — | 2.1 | — | V |
| | | 5V | | — | 3.5 | — | V |
| $f_{SYS}$ | System Frequency | 3V | $R_{TYPICAL}$=275kΩ | — | 4.0 | — | MHz |
| | | | $R_{TYPICAL}$=144kΩ | — | 8.0 | — | MHz |
| $R_{PH}$ | Pull-high Resistance | 3V | — | 20 | 60 | 100 | kΩ |
| | | 5V | | 10 | 30 | 50 | kΩ |

## A.C. Characteristics

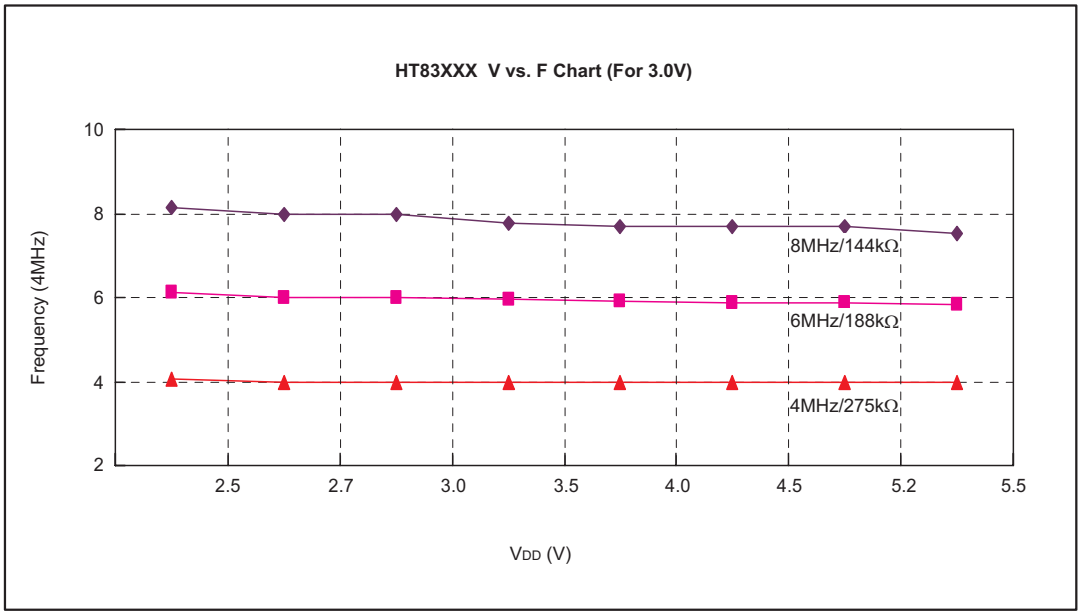
| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $f_{SYS1}$ | System Clock (RC OSC) | — | 2.4V~5.2V | 4 | — | 8 | MHz |
| $f_{SYS2}$ | System Clock (Crystal OSC) | — | 2.4V~5.2V | 4 | — | 8 | MHz |
| $f_{TIMER}$ | Timer Input Frequency | — | 2.4V~5.2V | 0 | — | 8 | MHz |
| $t_{WDTOSC}$ | Watchdog Oscillator Period | 3V | — | 50 | 100 | 200 | μs |
| | | 5V | | 37 | 74 | 148 | μs |
| $t_{WDT1}$ | Watchdog Time-out Period (WDT OSC) | 3V | Without WDT prescaler | 12 | 23 | 46 | ms |
| | | 5V | | 8 | 17 | 33 | ms |
| $t_{WDT2}$ | Watchdog Time-out Period (System Clock) | — | Without WDT prescaler | — | 1024 | — | $t_{SYS}$ |
| $t_{RES}$ | External Reset Low Pulse Width | — | — | 1 | — | — | μs |
| $t_{SST}$ | System Start-up Timer Period | — | Power-up or Wake-up from HALT | — | 1024 | — | $t_{SYS}$ |
| $t_{INT}$ | Interrupt Pulse Width | — | — | 1 | — | — | μs |
| $t_{DRT}$ | Data ROM Access Timer | — | — | 5 | — | — | ms |
| $t_{DRR}$ | Data ROM enable Read | — | Read after data ROM enable | 30 | — | — | ms |

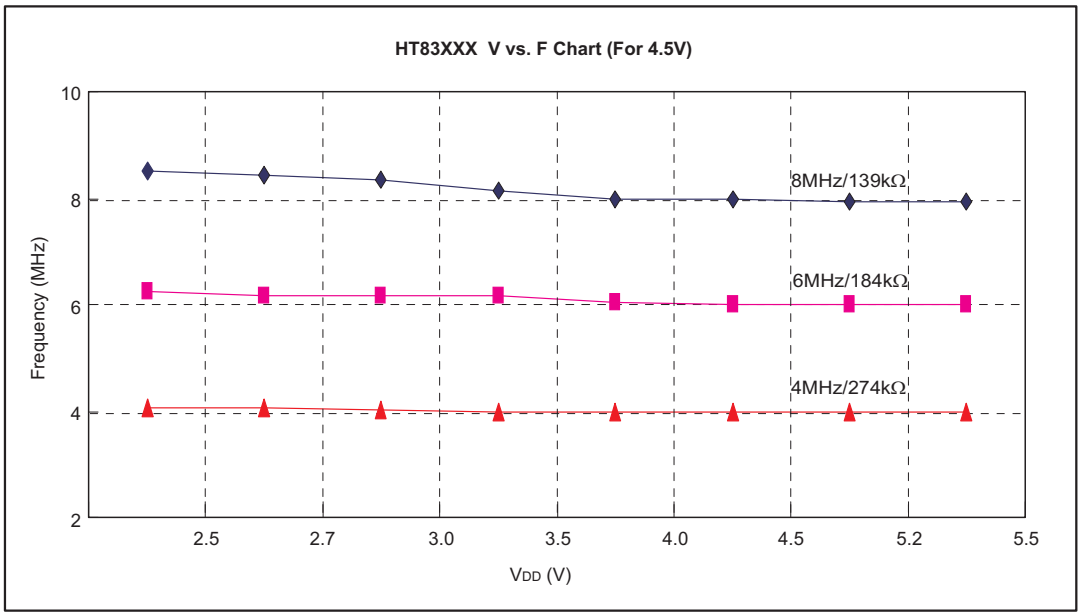
## Characteristics Curves

### R vs. F Characteristics Curve



**V vs. F Characteristics Curve**



## Functional Description

### Execution Flow

The system clock for the HT83XXX is derived from either a crystal or RC oscillator. 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 11-bit program counter (PC) controls the sequence in which the instructions stored in program ROM are executed.

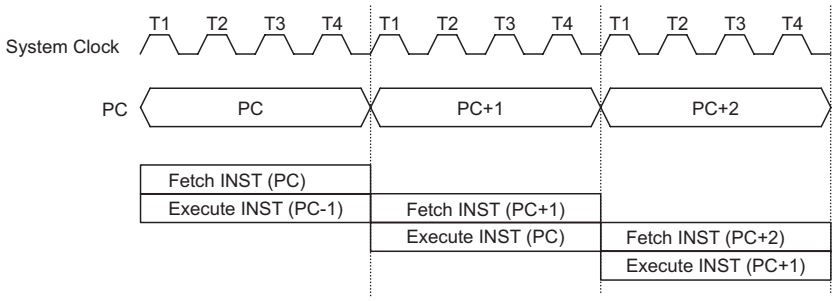
After accessing a program memory word to fetch an instruction code, the contents of the program counter are incremented by one. The program counter then points to the memory word containing the next instruction code.

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

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 takes its place while the correct instruction is obtained.

The lower byte of the program counter (PCL) is a read/write register (06H). Moving data into the PCL performs a short jump. The destination must be within 256 locations.

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




**Execution Flow**

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

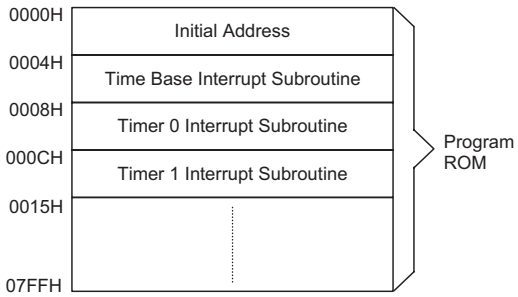
**Program Counter**

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

### Program Memory – ROM

The program memory stores the program instructions that are to be executed. It also includes data, table and interrupt entries, addressed by the program counter along with the table pointer. The program memory size for HT83XXX is 2048×15 bits. Certain locations in the program memory are reserved for special usage:

- Location 000H
  This area is reserved for program initialization. The program always begins execution at location 000H each time the system is reset.
- Location 004H
  This area is reserved for the time base interrupt service program. If the ETBI (intc.1) is activated, and the interrupt is enabled and the stack is not full, the program will jump to location 004H and begins execution.
- Location 008H
  This area is reserved for the 8-bit Timer Counter 0 interrupt service program. If a timer interrupt results from a Timer Counter 0 overflow, and if the interrupt is enabled and the stack is not full, the program will jump to location 008H and begins execution.
- Location 00CH
  This area is reserved for the 8-bit Timer Counter 1 interrupt service program. If a timer interrupt results from a Timer Counter 1 overflow, and if the interrupt is enabled and the stack is not full, the program will jump to location 00CH and begins execution.




**Program Memory**

### Table Location

Any location in the ROM space can be used as look up tables. The instructions "TABRDC [m]" (used for any bank) and "TABRDL [m]" (only used for last page of program ROM) transfer the contents of the lower-order byte to the specified data memory [m], and the higher-order byte to TBLH (08H). Only the destination of the lower-order byte in the table is well-defined. The higher-order bytes of the table word are transferred to the TBLH. The table higher-order byte register (TBLH) is read only.

The table pointer (TBLP) is a read/write register, which indicates the table location.

### Stack Register – Stack

The stack register is a special part of the memory used to save the contents of the Program Counter. This stack is organized into four levels. It is neither part of the data nor part of the program space, and cannot be read or written to. Its activated level is indexed by a stack pointer (SP) and cannot be read or written to. At a subroutine call or interrupt acknowledgment, the contents of the program counter are pushed onto the stack.

The program counter is restored to its previous value from the stack at the end of subroutine or interrupt routine, which is signaled by return instruction (RET or RETI). After a chip resets, SP will point to the top of the stack.

The interrupt request flag will be recorded but the acknowledgment will be inhibited when the stack is full and a non-masked interrupt takes place. After the stack pointer is decremented (by RET or RETI), the interrupt request will be serviced. This feature prevents stack overflow and allows programmers 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 is lost.

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

**Table Location**

Note: *10~*0: Current program ROM table
P10~P8: Bits of current program counter
@7~@0: Write @7~@0 to TBLP pointer register

**Data Memory – RAM**

The data memory is designed with 80×8 bits. The data memory is further divided into two functional groups, namely, special function registers (00H~2AH) and general purpose user data memory (30H~7FH). Although most of them can be read or be written to, some are read only.

The general purpose data memory, addressed from 30H~7FH, is used for data and control information under instruction commands.

The areas in the RAM can directly handle the arithmetic, logic, increment, decrement and rotate operations. Except some dedicated bits, each bit in the RAM can be set and reset by ″SET [m].i″ and ″CLR [m].i″. They are also indirectly accessible through the Memory Pointer register 0 (MP0:01H).

**Indirect Addressing Register**

Location 00H is indirect addressing registers that are not physically implemented. Any read/write operation of [00H] accesses the RAM pointed to by MP0 (01H) respectively. Reading location 00H indirectly returns the result 00H. While, writing it indirectly leads to no operation.

**Accumulator – ACC (05H)**

The accumulator (ACC) is related to the ALU operations. It is also mapped to location 05H of the RAM and is capable of operating with immediate data. The data movement between two data memory locations must pass through the ACC.

**Arithmetic and Logic Unit – ALU**

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

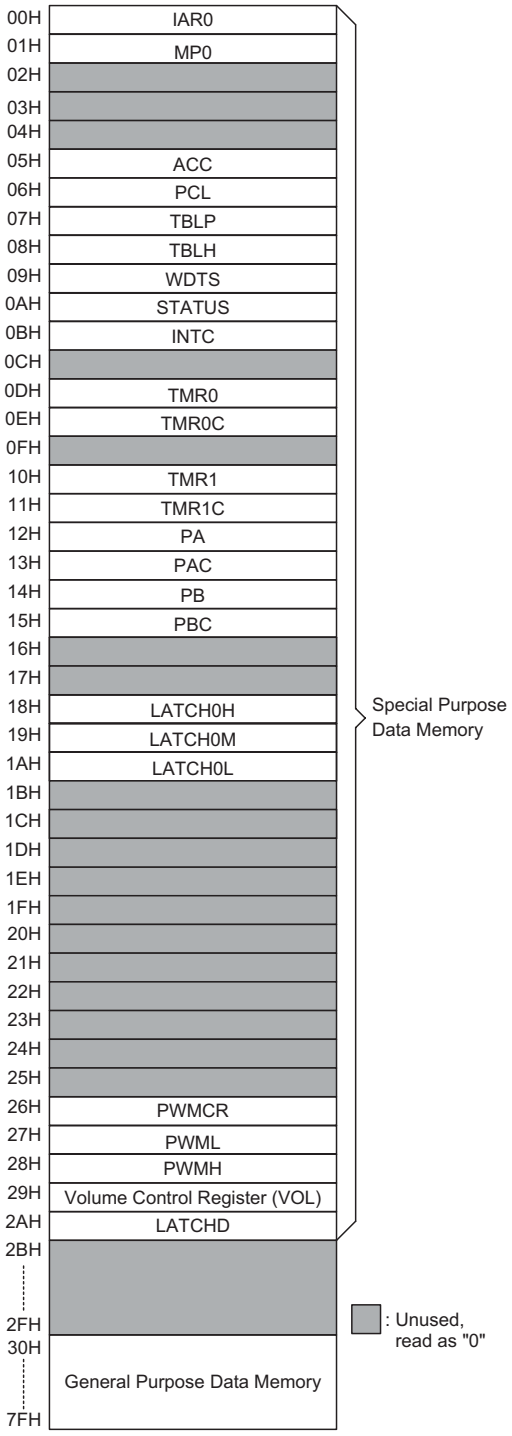
- Arithmetic operations (ADD, ADC, SUB, SBC, DAA)
- Logic operations (AND, OR, XOR, CPL)
- Rotation (RL, RR, RLC, RRC)
- Increment and Decrement (INC, DEC)
- Branch decision (SZ, SNZ, SIZ, SDZ etc)

**Status Register – STATUS (0AH)**

This 8-bit STATUS register (0AH) consists of a zero flag (Z), carry flag (C), auxiliary carry flag (AC), overflow flag (OV), power down flag (PDF), watchdog time-out flag (TO). It also records the status information and controls the operation sequence.

Except the TO and PDF flags, bits in the status register can be altered by instructions similar to other registers. Data written into the status register does not alter the TO or PDF flags. Operations related to the status register, however, may yield different results from those intended. The TO and PDF flags can only be changed by a Watchdog Timer overflow, chip power-up, or clearing the Watchdog Timer and executing the ″HALT″ instruction. The Z, OV, AC, and C flags reflect the status of the latest operations.

On entering the interrupt sequence or executing the subroutine call, the status register will not be automatically pushed onto the stack. If the contents of the status is important, and if the subroutine is likely to corrupt the status register, the programmer should take precautions and save it properly.




**RAM Mapping**

| Address | RAM Mapping | Read/Write | Description |
|---|---|---|---|
| 00H | IAR0 | R/W | Indirect Addressing Register 0 |
| 01H | MP0 | R/W | Memory Pointer 0 |
| 05H | ACC | R/W | Accumulator |
| 06H | PCL | R/W | Program counter lower-order byte address |
| 07H | TBLP | R/W | Table pointer lower-order byte register |
| 08H | TBLH | R | Table higher-order byte content register |
| 09H | WDTS | R/W | Watchdog Timer option setting register |
| 0AH | STATUS | R/W | Status register |
| 0BH | INTC | R/W | Interrupt control register 0 |
| 0DH | TMR0 | R/W | Timer Counter 0 register |
| 0EH | TMR0C | R/W | Timer Counter 0 control register |
| 10H | TMR1 | R/W | Timer Counter 1 register |
| 11H | TMR1C | R/W | Timer Counter 1 control register |
| 12H | PA | R/W | Port A I/O data register |
| 13H | PAC | R/W | Port A I/O control register |
| 14H | PB | R/W | Port B I/O data register |
| 15H | PBC | R/W | Port B I/O control register |
| 18H | LATCH0H | R/W | Voice ROM address latch 0 [A17, A16] |
| 19H | LATCH0M | R/W | Voice ROM address latch 0 [A15~A8] |
| 1AH | LATCH0L | R/W | Voice ROM address latch 0 [A7~A0] |
| 26H | PWMCR | R/W | PWM control register |
| 27H | PWML | R/W, higher-nibble available only | PWM output data P3~P0 to PWML7~PWML4 |
| 28H | PWMH | R/W | PWM output data P11~P4 to PWMH7~PWMH0 |
| 29H | VOL | R/W, higher-nibble available only | Volume control register and volume controlled by VOL8~VOL4 |
| 2AH | LATCHD | R | Voice ROM data register |
| 2BH~2FH | Unused | | |
| 30H~7FH | User data RAM | R/W | User data RAM |

Note: R: Read only
W: Write only
R/W: Read/Write

**Interrupts**

The HT83XXX provides two 8-bit programmable timer interrupts, and a time base interrupt. The Interrupt Control registers (INTC:0BH) contain the interrupt control bits to set to enable/disable and the interrupt request flags.

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

As an interrupt is serviced, a control transfer occurs by pushing the program counter onto the stack and then branching to subroutines at the specified location(s) in the program memory. Only the program counter is pushed onto the stack. The programmer must save the contents of the register or status register (STATUS) in advance if they are altered by an interrupt service program which corrupts the desired control sequence.

| Bit No. | Label | Function |
|---|---|---|
| 0 | C | C is set if an operation results in a carry during an addition operation or if a borrow does not take place during a subtraction operation; otherwise C is cleared. C is also affected by a rotate through carry instruction. |
| 1 | AC | AC is set if an operation results in a carry out of the low nibbles in addition or no borrow from the high nibble into the low nibble in subtraction; otherwise AC is cleared. |
| 2 | Z | Z is set if the result of an arithmetic or logical operation is zero; otherwise Z is cleared. |
| 3 | OV | OV is set if an operation results in a carry into the highest-order bit but not a carry out of the highest-order bit, or vice versa; otherwise OV is cleared. |
| 4 | PDF | PDF is cleared by system power-up or executing the ″CLR WDT″ instruction.<br>PDF is set by executing the ″HALT″ instruction. |
| 5 | TO | TO is cleared by system power-up or executing the ″CLR WDT″ or ″HALT″ instruction.<br>TO is set by a WDT time-out. |
| 6~7 | — | Unused bit, read as ″0″ |

**Status (0AH) Register**

**The Internal Timer Counter 0 Interrupt** is initialized by setting the Timer Counter 0 interrupt request flag (T0F:bit 5 of INTC), caused by a Timer Counter 0 overflow. When the interrupt is enabled, and 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 Counter 1 Interrupt** is initialized by setting the Timer Counter 1 interrupt request flag (T1F:bit 6 of INTC), caused by a Timer Counter 1 overflow. When the interrupt is enabled, and the stack is not full and the T1F bit is set, a subroutine call to location 0CH will occur. The related interrupt request flag (T1F) will be reset and the EMI bit cleared to disable further interrupts.

**Time Base Interrupt** is triggered by set INTC.1 (ETBI) which sets the related interrupt request flag (TBF:bit 4 of INTC). When the interrupt is enabled, and the stack is not full and the external interrupt is active, a subroutine call to location 04H will occur. The interrupt request flag (TBF) and EMI bits will be cleared to disable other interrupts.

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

The Timer Counter 0/1 interrupt request flag (T0F/T1F) which enables Timer Counter 0/1 control bit (ET0I/ ET1I), the time base interrupt request flag (TBF) which enables time base control bit (ETBI) from the interrupt control register (INTC:0BH) EMI, ETBI, ET0I, ET1I are used to control the enabling/disabling of interrupts. These bits prevent the requested interrupt begin serviced. Once the interrupt request flags (T0F, T1F, TBF) are set, they will remain in the INTC register until the interrupts are serviced or cleared by a software instruction.

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

| Bit No. | Label | Function |
|---|---|---|
| 0 | EMI | Controls the master (global) interrupt<br>(1= enabled; 0= disabled) |
| 1 | ETBI | Controls the time base interrupt<br>(1= enabled; 0= disabled) |
| 2 | ET0I | Controls the timer 0 interrupt<br>(1= enabled; 0= disabled) |
| 3 | ET1I | Controls the timer 1 interrupt<br>(1= enabled; 0= disabled) |
| 4 | TBF | Time base interrupt request flag<br>(1= active; 0= inactive) |
| 5 | T0F | Timer 0 request flag<br>(1= active; 0= inactive) |
| 6 | T1F | Timer 1 request flag<br>(1= active; 0= inactive) |
| 7 | — | Unused bit, read as ″0″ |

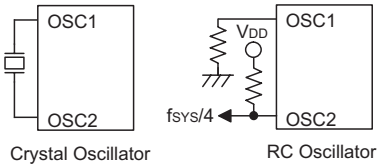
**INTC (0BH) Register**

| Interrupt Source | Priority | Vector |
|---|---|---|
| Time Base Interrupt | 1 | 04H |
| Timer Counter 0 Overflow | 2 | 08H |
| Timer Counter 1 Overflow | 3 | 0CH |

**Oscillator Configuration**

The HT83XXX provides two oscillator circuits for system clock, i.e., RC oscillator and Crystal oscillator. No matter what type of oscillator.. The signal is used for the system clock. The HALT mode stops the system oscillator to conserve power. If the RC oscillator is used, an external resistor between OSC1 and VSS is required, and the range of the resistance should be from 144kΩ to 275kΩ. The system clock, divided by 4. The RC oscillator provides the most cost effective solution. However, the frequency of the oscillation may vary with VDD, temperature, and the chip itself due to process variations. It is therefore not suitable for timing sensitive operations where accurate oscillator frequency is desired.

On the other hand, if the crystal oscillator is selected, a crystal across OSC1 and OSC2 is needed to provide the feedback and phase shift required for the oscillator, and no other external components are required. A resonator may be connected between OSC1 and OSC2 to replace the crystal and to get a frequency reference, but two external capacitors in OSC1 and OSC2 are required.




**System Oscillator**

**Watchdog Timer – WDT**

The WDT clock source is implemented by a dedicated RC oscillator (WDT oscillator) or instruction clock (system clock divided by 4), decided by mask options. This 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 mask 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 period 78μs normally) is selected, it is first divided by 256 (8-stages) to get the nominal time-out period of approximately 20ms. This time-out period may vary with temperature, VDD and process variations. By invoking the WDT prescaler, longer time-out period can be realized. Writing data to WS2, WS1, WS0 (bit 2,1,0 of WDTS(09H)) can give different time-out period.

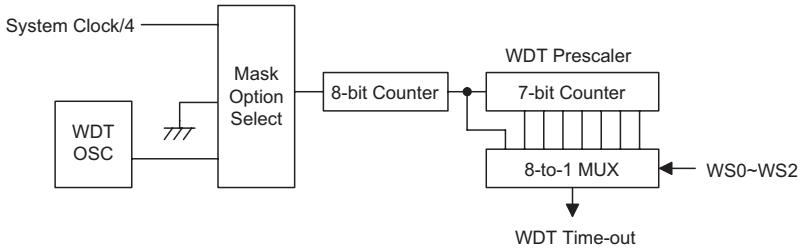
If WS2, WS1, WS0 all equal to 1, the division ratio is up to 1:128, and the maximum time-out period is 2.6 seconds.

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.

The WDT overflow under normal operation will initialize a "chip reset" and set the status bit "TO". Whereas in the HALT mode, the overflow will initialize a "warm reset" only the Program Counter and SP are reset to zero. To clear the contents of the WDT (including the WDT prescaler), three methods are adopted; external reset (external reset (a low level to $\overline{RES}$), software instructions, or a "HALT" instruction. The software instruction is "CLR WDT" and execution of the "CLR WDT" instruction will clear the WDT.

| WS7 | WS6 | WS5 | WS4 | WS3 | WS2 | WS1 | WS0 | Division Ratio |
|---|---|---|---|---|---|---|---|---|
| — | — | — | — | — | 0 | 0 | 0 | 1:1 |
| — | — | — | — | — | 0 | 0 | 1 | 1:2 |
| — | — | — | — | — | 0 | 1 | 0 | 1:4 |
| — | — | — | — | — | 0 | 1 | 1 | 1:8 |
| — | — | — | — | — | 1 | 0 | 0 | 1:16 |
| — | — | — | — | — | 1 | 0 | 1 | 1:32 |
| — | — | — | — | — | 1 | 1 | 0 | 1:64 |
| — | — | — | — | — | 1 | 1 | 1 | 1:128 |

**WDTS (09H) Register**




**Watchdog Timer**

**Power Down – HALT**

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

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

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". By examining the TO and PDF flags, the reason for the chip reset can be determined. The PDF flag is cleared when the system powers-up or executes the "CLR WDT" instruction, and is set when the "HALT" instruction is executed. The TO flag is set if a WDT time-out occurs, and causes a wake-up that only resets the Program Counter and Stack Pointer. The other maintain 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 awakening from an interrupt, two sequence may occur. If the related interrupt is disabled or the interrupt is enabled by 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.

Once a wake-up event occurs, it takes 1024 system clock period to resume normal operation. In other words, a dummy cycle period will be inserted after a wake-up. If the wake-up results from an interrupt acknowledge, the actual interrupt subroutine will be delayed by one more cycle. If the wake-up results in next instruction execution, this will be executed immediately after a dummy period is finished. 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. To minimize power consumption, all I/O pins should be carefully managed before entering the HALT status.

**Reset**

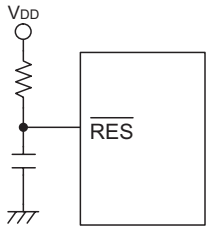
There are 3 ways in which a reset can occur:

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

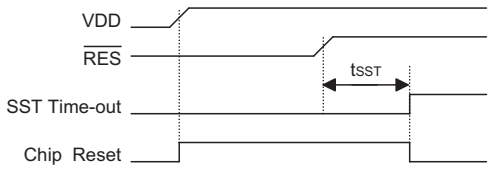
The WDT time-out during HALT is different from other chip reset conditions, since it can perform a "warm reset" that resets only the Program Counter and SP, leaving the other circuits in their original state. Some registers remain unchanged during any other reset conditions. Most registers are reset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

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

Note: "u" stands for "unchanged"



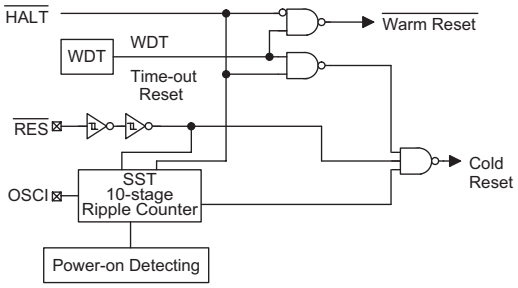

**Reset Circuit**




**Reset Timing Chart**

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 after a system power up or when awakening from a HALT state.

When a system power up occurs, the SST delay is added during the reset period. But when the reset comes from the $\overline{RES}$ pin, the SST delay is disabled. Any wake-up from HALT will enable the SST delay.




**Reset Configuration**

The functional unit chip reset status are shown below.

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

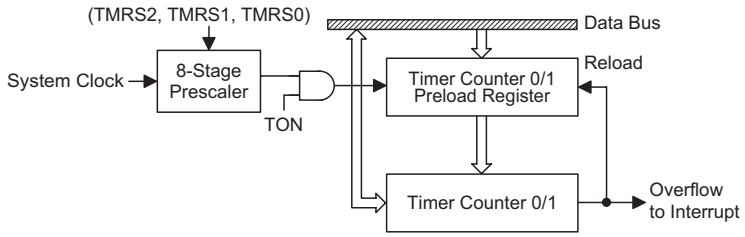
**Timer Counter 0/1**

The TMR0/TMR1 is internal clock source only, i.e. (TM1, TM0) = (0, 1). There is a 3-bit prescaler (TMRS2, TMRS1, TMRS0) which defines different division ratio of TMR0/TMR1′s clock source.

| Bit No. | Label | Function |
|---|---|---|
| 0~2 | TMRS2, TMRS1, TMRS0 | Defines the operating clock source (TMRS2, TMRS1, TMRS0)<br>000: clock source/2<br>001: clock source/4<br>010: clock source/8<br>011: clock source/16<br>100: clock source/32<br>101: clock source/64<br>110: clock source/128<br>111: clock source/256 |
| 3 | TE | Defines the TMR0/TMR1 active edge of Timer Counter |
| 4 | TON | Enable/disable timer counting (0=disabled; 1=enabled) |
| 5 | — | Unused bit, read as ″0″ |
| 6<br>7 | TM0,<br>TM1 | Defines the operating mode (TM1, TM0) |

**TMR0C (0EH)/TMR1C (11H) Register**

Note: TMR0C/TMR1C bit 3 always write ″0″
TMR0C/TMR1C bit 5 always write ″0″
TMR0C/TMR1C bit 6 always write ″1″
TMR0C/TMR1C bit 7 always write ″0″




**Timer Counter 0/1**

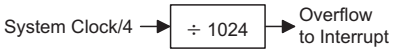
The TMR0C is the Timer Counter 0 control register, which defines the Timer Counter 0 options. The Timer Counter 1 has the same options as the Timer Counter 0 and is defined by TMR1C.

To enable the counting operation, the Timer ON bit (TON; bit 4 of TMR0C/TMR1C) should be set to "1". The overflow of the timer counter is one of the wake-up sources. No matter what the operation mode is, writing a 0 to ET0I/ET1I can disable the corresponding interrupt service.

The TMR0/1 is internal clock source only. There is a 3-bit prescaler (TMRS2, TMRS1, TMRS0) which defines different division ratio of TMR0/1's clock source.

**Time Base**

The time base enables the counting operation by INTC.1 (ETBI) bit. The overflow to interrupt as set INTC.4. The time base is internal clock source only. Time base of 1ms to overflow as system clock is 4MHz. Time base of 0.5ms to overflow as system clock is 8MHz.




**Time Base**

The registers states are summarized in the following table.

| Register | Reset (Power-on) | WDT Time-out (Normal Operation) | $\overline{\text{RES}}$ Reset (Normal Operation) | $\overline{\text{RES}}$ Reset (HALT) | WDT Time-out (HALT) |
|---|---|---|---|---|---|
| MP0 | 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 |
| | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TBLH | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| WDTS | 0000 0111 | 0000 0111 | 0000 0111 | 0000 0111 | uuuu uuuu |
| STATUS | --00 xxxx | --1u uuuu | --uu uuuu | --01 uuuu | --11 uuuu |
| INTC | -000 0000 | -000 0000 | -000 0000 | -000 0000 | -uuu uuuu |
| TMR0 | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx |
| TMR0C | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx |
| TMR1 | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx |
| TMR1C | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx | xxxx xxxx |
| 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 | ---- uuuu |
| PBC | ---- 1111 | ---- 1111 | ---- 1111 | ---- 1111 | ---- uuuu |
| LATCH0H | ---- --xx | ---- --uu | ---- --uu | ---- --uu | ---- --uu |
| LATCH0M | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| LATCH0L | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| PWMCR | 0--- 00-0 | u--- uu-u | u--- uu-u | u--- uu-u | u--- uu-u |
| PWML | xxxx ---- | uuuu ---- | uuuu ---- | uuuu ---- | uuuu ---- |
| PWMH | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu | uuuu uuuu |
| VOL | xxxx ---- | uuuu ---- | uuuu ---- | uuuu ---- | uuuu ---- |
| LATCHD | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |

Note: "u" means "unchanged"
"x" means "unknown"
"–" means "undefined"

**Input/Output Ports**

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

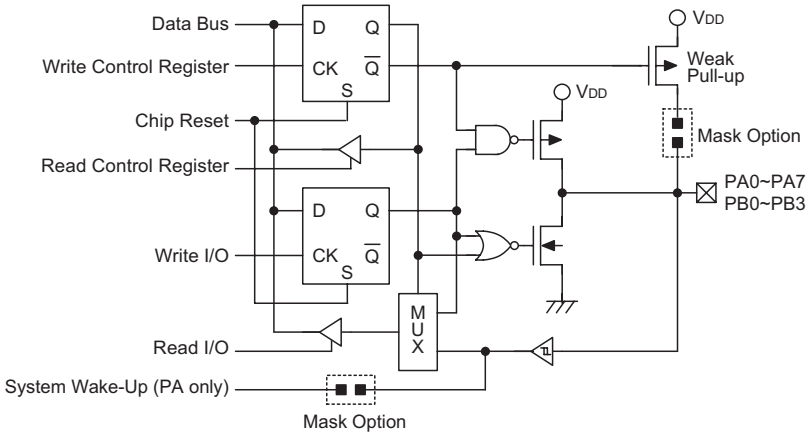
Each I/O line has its own control register (PAC, PBC) to control the input/output configuration. With this control register, CMOS output or Schmitt trigger input with or without pull-high resistor structures can be reconfigured dynamically under software control. To function as an input, the corresponding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

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

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

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

Each line of port A has the capability of waking-up the device. The wake-up capability of port A is determined by mask option. There is a pull-high option available for all I/O lines. Once the pull-high option is selected, all I/O lines have pull-high resistors. Otherwise, the pull-high resistors are absent. It should be noted that a non-pull-high I/O line operating in input mode will cause a floating state.




**Input/Output Ports**

**Pulse Width Modulation Output – PWML/PWMH (27H/28H)**

The HT83XXX provide one 12-bit PWM interface for driving an external 8Ω speaker. The programmer must write the voice data to register PWML/PWMH (27H/28H)

**Pulse Width Modulation Control Register – PWMCR (26H)**

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 (R/W) | Bit 2 (R/W) | Bit 1 | Bit 0 (R/W) |
|---|---|---|---|---|---|---|---|
| MSB_SIGN | — | — | — | Single_PWM | VROMC | — | PWMC |

PWMC: Start bit of PWM output

- PWM start counter: 0 to 1
- PWM stop counter: 1 to 0

After waiting one cycle end , stop the PWM counter and keep in low signal

VROMC: Enable voice ROM power circuit (1=enable; 0=disable)

Single_PWM: Driving PWM signal by PWM1 output. (1=PWM1 output; 0=PWM1/PWM2 output)

The HT83XXX provide an 12-bit (bit 7 is a sign bit, if Single_PWM = 0) PWM interface. The PWM provides two pad outputs: PWM1, PWM2 which can directly drive a piezo or an 8Ω speaker without adding any external element (green mode), or using only port PWM1 (Set Single_PWM = 1) to drive piezo or an 8Ω speaker with external element.

When Setting Single_PWM= 1, choose voice data7~ data1 as the output data (no sign bit on it).

If the sign bit is 0, then the signal is output to PWM1and the PWM2 will get a GND level voltage after setting start bit to 1. If the sign bit is 1, then the signal is output to PWM2 and the PWM1 will get a GND level voltage after setting start bit to 1.

PWM output Initial low level , and stop in low level

If PWMC from low to high then start PWM output latch new data , if no update then keep the old value.

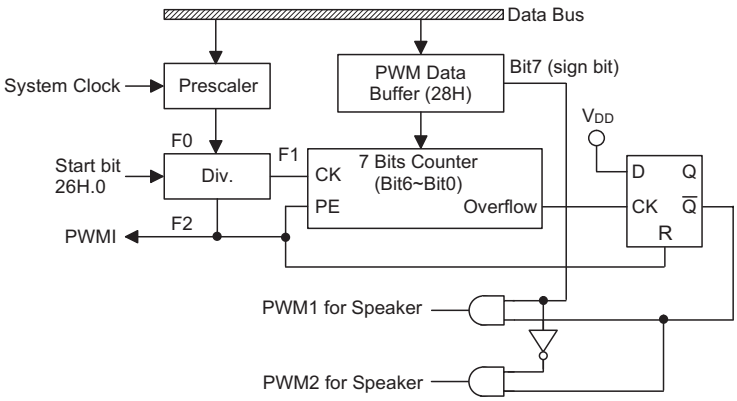
If PWMC from high to low, in duty end, stop PWM output and stop the counter.

**Voice ROM Data Address Latch Counter**

The voice ROM data address latch counter is the handshaking between the microcontroller and voice ROM, where the voice codes are stored. One 8-bit of voice ROM data will be addressed by setting 18-bit address latch counter LATCH0H/LATCH0M/LATCH0L. After the 8-bit voice ROM data is addressed, a few instruction cycles (4μs at least) will be generated to latch the voice ROM data, then the microcontroller can read the voice data from LATCHD (2AH).

Example: Read an 8-bit voice ROM data which is located at address 000007H by address latch 0

```
set   [26H].2      ; Enable voice ROM circuit
mov   A, 07H       ;
mov   LATCH0L, A   ; Set LATCH0L to 07H
mov   A, 00H       ;
mov   LATCH0M, A   ; Set LATCH0M to 00H
mov   A, 00H       ;
mov   LATCH0H, A   ; Set LATCH0H to 00H
call  Delay Time   ; Delay a short period of time
mov   A, LATCHD    ; Get voice data at 000007H
```




**PWM**

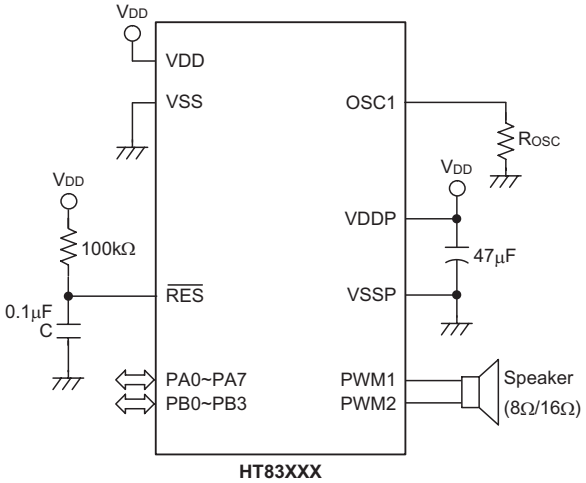
**Mask Option**

| Mask Option | Description |
|---|---|
| PA Wake-up | Enable or disable PA wake-up function |
| Watchdog Timer (WDT) | Enable or disable WDT function<br>WDT clock source is from WDTOSC or T1 |
| PA Pull-high | Enable or disable PA pull-high |
| PB Pull-high | Enable or disable PB pull-high |
| OSC Option | Crystal or Resistor type |

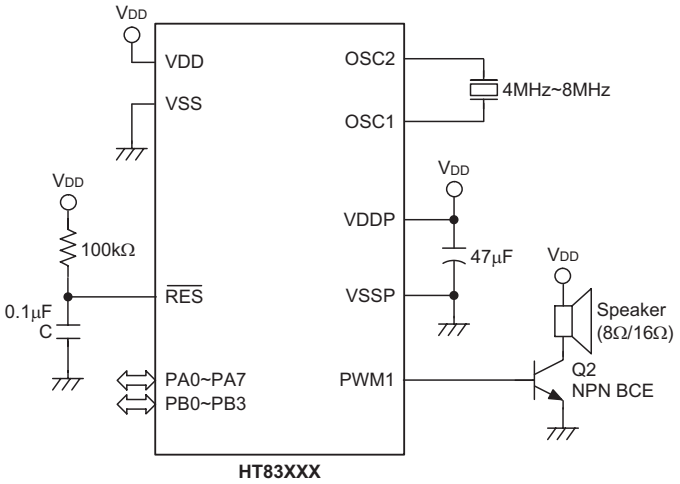
**$f_{OSC}$ – $R_{TYPICAL}$ Table ($V_{DD}$=3V)**

| $f_{OSC}$ | $R_{TYPICAL}$ |
|---|---|
| 4MHz±10% | 275kΩ |
| 6MHz±10% | 188kΩ |
| 8MHz±10% | 144kΩ |

## Application Circuits




**Single PWM Mode**



## 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 data memory 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,PDF |
| CLR WDT1 | Pre-clear Watchdog Timer | 1 | TO(4),PDF(4) |
| CLR WDT2 | Pre-clear Watchdog Timer | 1 | TO(4),PDF(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,PDF |

Note: x: 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 and PDF are cleared.
Otherwise the TO and PDF 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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 ← Program Counter+1
Program Counter ← addr

Affected flag(s)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**CLR WDT** Clear Watchdog Timer

Description The WDT is cleared (clears the WDT). The power down bit (PDF) and time-out bit (TO) are cleared.

Operation WDT ← 00H
PDF and TO ← 0

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| 0 | 0 | — | — | — | — |

**CLR WDT1** Preclear Watchdog Timer

Description Together with CLR WDT2, clears the WDT. PDF and TO are also cleared. 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 PDF flags remain unchanged.

Operation WDT ← 00H*
PDF and TO ← 0*

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| 0* | 0* | — | — | — | — |

**CLR WDT2** Preclear Watchdog Timer

Description Together with CLR WDT1, clears the WDT. PDF and TO are also cleared. 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 PDF flags remain unchanged.

Operation WDT ← 00H*
PDF and TO ← 0*

Affected flag(s)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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 (PDF) is set and the WDT time-out bit (TO) is cleared.

Operation Program Counter ← Program Counter+1
PDF ← 1
TO ← 0

Affected flag(s)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

**JMP addr** Directly jump

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

Operation Program Counter ←addr

Affected flag(s)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | — | — | — |

**NOP** No operation

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

Operation Program Counter ← Program Counter+1

Affected flag(s)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

**RET** Return from subroutine

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

Operation Program Counter ← Stack

Affected flag(s)

| TO | PDF | 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 Program Counter ← Stack
ACC ← x

Affected flag(s)

| TO | PDF | 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.

Operation Program Counter ← Stack
EMI ← 1

Affected flag(s)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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 ← ROM code (high byte)

Affected flag(s)

| TO | PDF | 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)

| TO | PDF | 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)

| TO | PDF | 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.

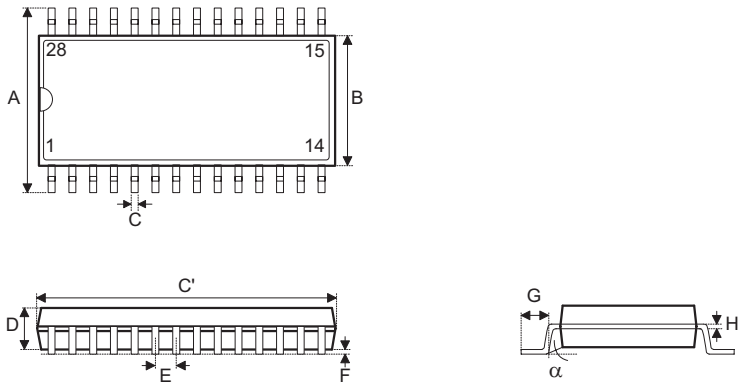
Operation ACC ← ACC ″XOR″ x

Affected flag(s)

| TO | PDF | OV | Z | AC | C |
|---|---|---|---|---|---|
| — | — | — | √ | — | — |

## Package Information

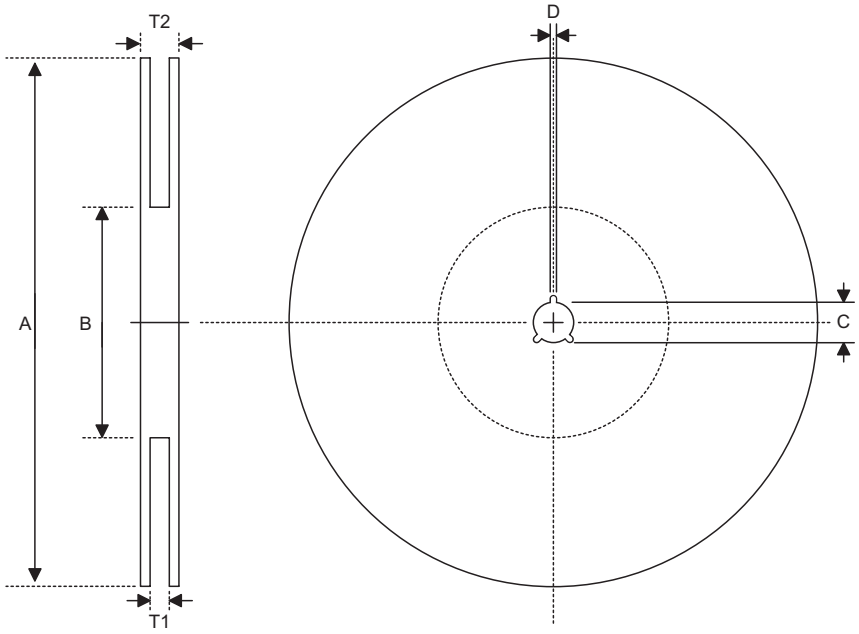
**28-pin SOP (300mil) Outline Dimensions**




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

## Product Tape and Reel Specifications

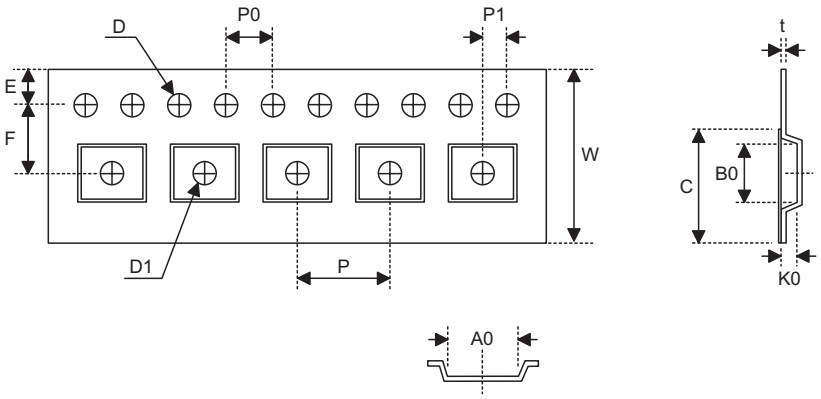
**Reel Dimensions**




SOP 28W (300mil)

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

**Carrier Tape Dimensions**




SOP 28W (300mil)

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

**Holtek Semiconductor Inc. (Headquarters)**
No.3, Creation Rd. II, Science Park, Hsinchu, Taiwan
Tel: 886-3-563-1999
Fax: 886-3-563-1189
http://www.holtek.com.tw

**Holtek Semiconductor Inc. (Taipei Sales Office)**
4F-2, No. 3-2, YuanQu St., Nankang Software Park, Taipei 115, Taiwan
Tel: 886-2-2655-7070
Fax: 886-2-2655-7373
Fax: 886-2-2655-7383 (International sales hotline)

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

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

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

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

**Holtek Semiconductor (USA), Inc. (North America Sales Office)**
46729 Fremont Blvd., Fremont, CA 94538
Tel: 1-510-252-9880
Fax: 1-510-252-9885
http://www.holtek.com

Copyright © 2007 by HOLTEK SEMICONDUCTOR INC.

The information appearing in this Data Sheet is believed to be accurate at the time of publication. However, Holtek assumes no responsibility arising from the use of the specifications described. The applications mentioned herein are used solely for the purpose of illustration and Holtek makes no warranty or representation that such applications will be suitable without further modification, nor recommends the use of its products for application that may present a risk to human life due to malfunction or otherwise. Holtek's products are not authorized for use as critical components in life support devices or systems. Holtek reserves the right to alter its products without prior notification. For the most up-to-date information, please visit our web site at http://www.holtek.com.tw.