# HT99C810/HT99C811
# Cordless Phone Controller

## Features

- Provide mask type or OTP type version
- Operating voltage: 2.4V~5.2V (mask type), 3.0V~5.2V (OTP type)
- 48 bidirectional I/O lines
- One interrupt input
- Two 16-bit programmable timer/event counter with overflow interrupt
- On-chip crystal and RC oscillator
- Watchdog timer
- 8K × 16 program memory ROM
- 224 × 8 data memory RAM
- Halt function and wake-up feature reduce power consumption
- 63 powerful instructions
- Up to 1μs instruction cycle with 4MHz system clock at $V_{DD}$=5V
- All instructions in 1 or 2 machine cycles
- 16-bit table read instructions
- Eight-level subroutine nesting
- Bit manipulation instructions
- Built-in 8-bit D/A converter

## Applications

- Remote controllers
- Fan/light controllers
- Washing machine controllers
- Cordless phone controllers
- Scales
- Toys

## General Description

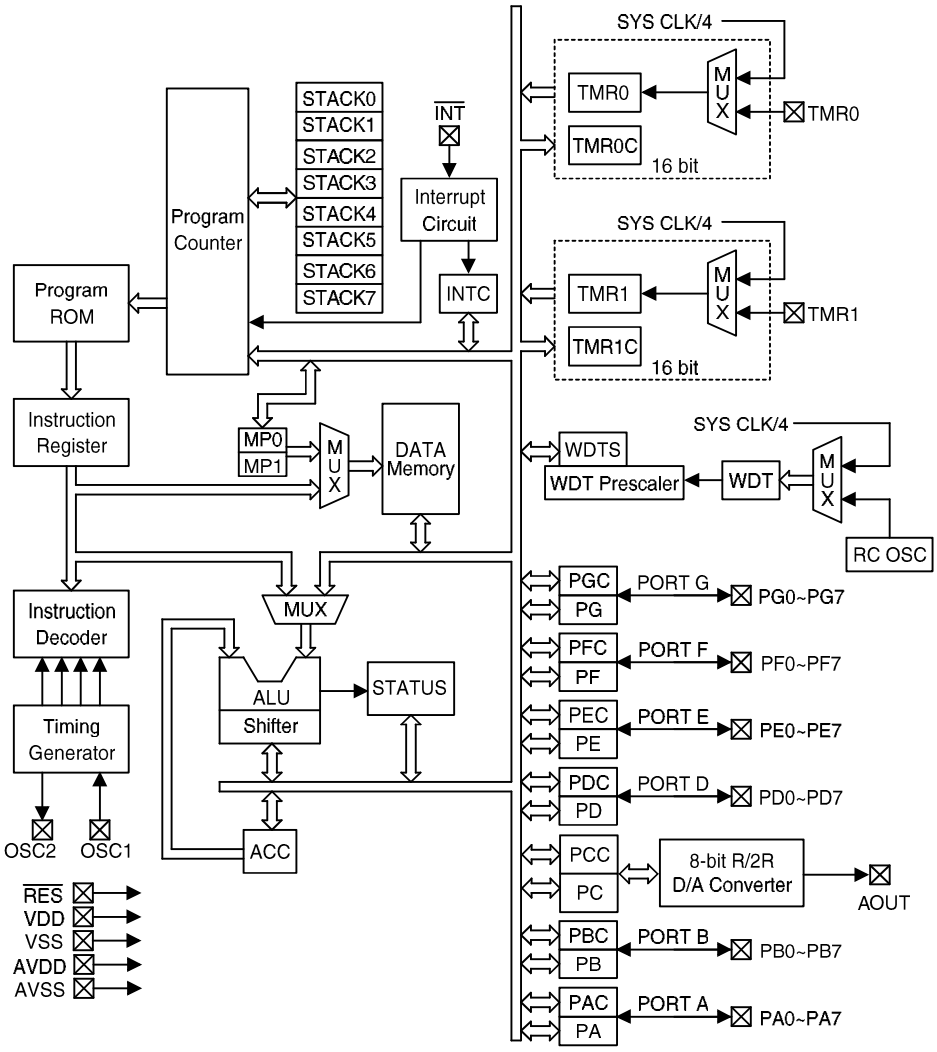
The HT99C810 is an 8-bit high performance RISC-like microcontroller which combines HT48700 8-bit microcontroller and 8-bit D/A converter in one chip. It is specifically designed for multiple I/O product applications. It also provides OTP type version HT99C811 which supports designers in making fast evaluation of private products during development stages.

The device is particularly suitable for use in products such as cordless phone controllers, μC dialers, feature phone controllers, remote controllers, fan/light controllers, washing machine controllers, scales, toys and various subsystem controllers. A halt feature is included to reduce power consumption.

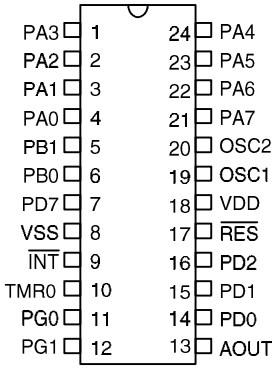
## Selection Table

| Function / Part No. | Type | ROM (bits) | RAM (bits) | I/O (lines) | WDT | Timer/Counter | DAC (bits) |
|---|---|---|---|---|---|---|---|
| HT99C210<br>HT99C211 | mask<br>OTP | 2K×14 | 96×8 | 16 | √ | 1 | 6 |
| HT99C410<br>HT99C411 | mask<br>OTP | 4K×15 | 160×8 | 24 | √ | 2 | 8 |
| HT99C810<br>HT99C811 | mask<br>OTP | 8K×16 | 224×8 | 48 | √ | 2 | 8 |

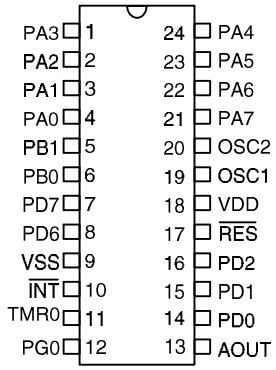
## Block Diagram



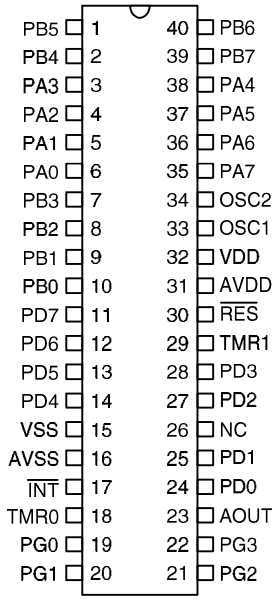
## Pin Assignment



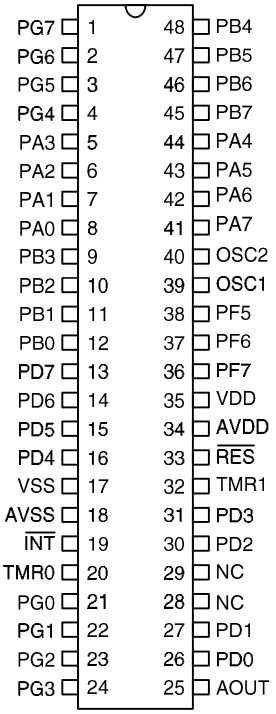

**HT99C810/HT99C811 – 24 SDIP**




**HT99C810/HT99C811 – 24 SOP**




**HT99C810/HT99C811 – 40 DIP**

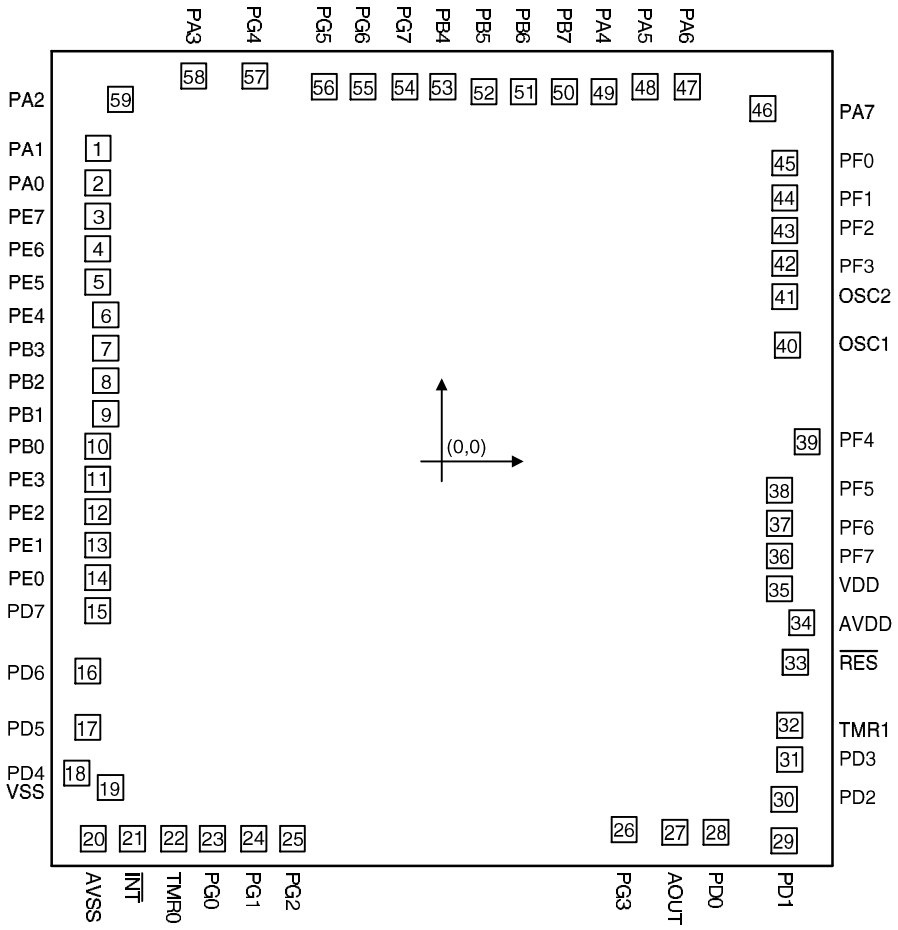



**HT99C810/HT99C811 – 48 SSOP**

* The analog VDD (AVDD) pad and digital VDD pad must be bonded to VDD Pin
* The analog VSS (AVSS) pad and digital VSS pad must be bonded to VSS Pin
* The TMR0 and TMR1 pads must be bonded to VDD or VSS (if not used)

## Pad Assignment

### HT99C810




Chip size: $3600 \times 3940$ $(\mu m)^2$

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

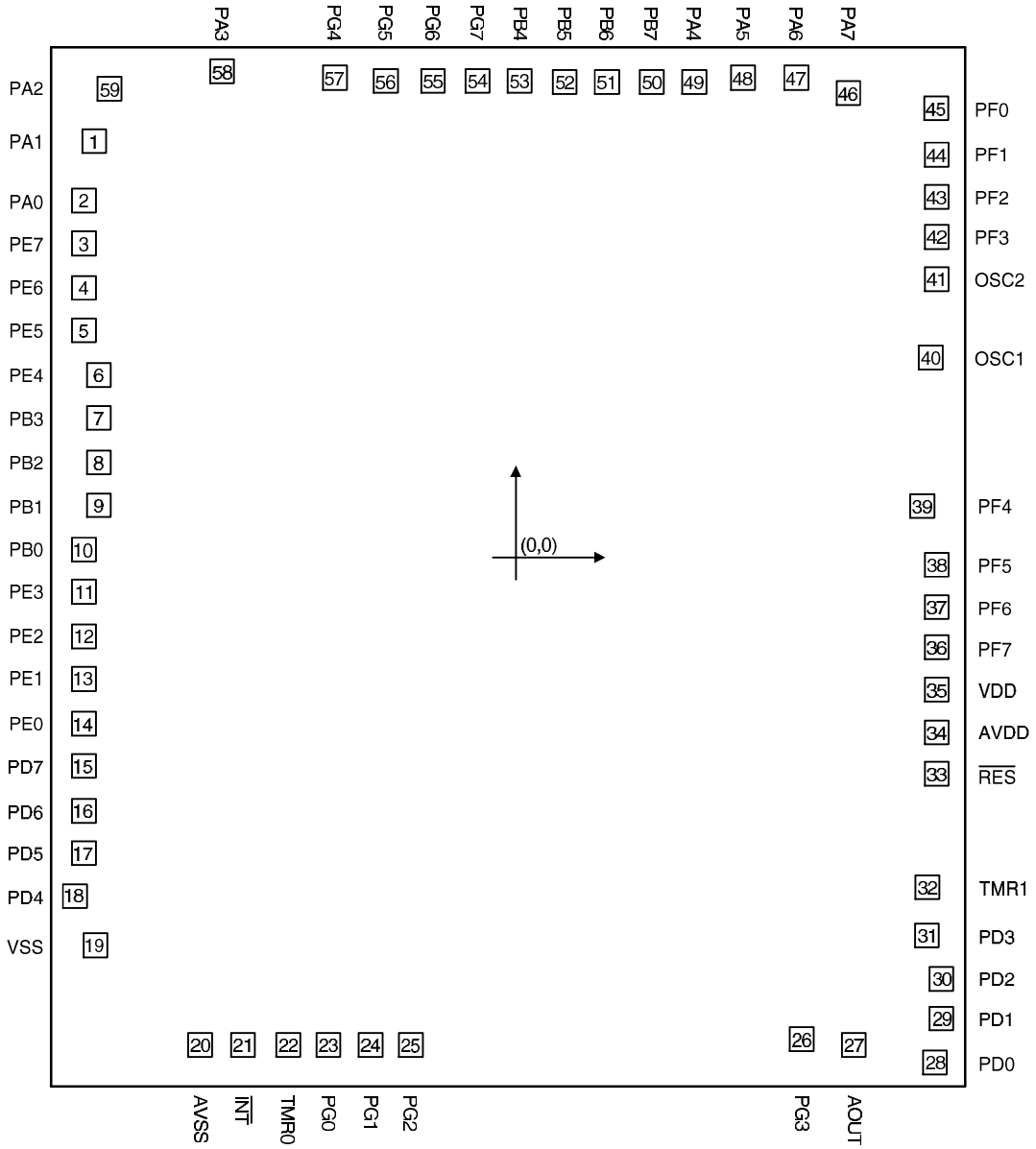
## Pad Coordinates

Unit: μm

| Pad No. | X | Y | Pad No. | X | Y |
|---|---|---|---|---|---|
| 1 | –1559.35 | 1421.85 | 31 | 1581.15 | –1355.65 |
| 2 | –1561.85 | 1264.05 | 32 | 1581.75 | –1199.65 |
| 3 | –1561.85 | 1113.35 | 33 | 1607.85 | –912.35 |
| 4 | –1561.85 | 965.15 | 34 | 1636.05 | –733.85 |
| 5 | –1561.85 | 814.45 | 35 | 1535.85 | –580.95 |
| 6 | –1526.35 | 666.15 | 36 | 1535.85 | –430.85 |
| 7 | –1526.35 | 515.65 | 37 | 1535.85 | –282.05 |
| 8 | –1526.35 | 367.25 | 38 | 1535.85 | –131.95 |
| 9 | –1526.35 | 216.75 | 39 | 1660.45 | 88.05 |
| 10 | –1561.85 | 68.45 | 40 | 1572.35 | 528.65 |
| 11 | –1561.85 | –82.25 | 41 | 1559.05 | 748.15 |
| 12 | –1561.85 | –230.45 | 42 | 1559.05 | 898.25 |
| 13 | –1561.85 | –381.15 | 43 | 1559.05 | 1047.05 |
| 14 | –1561.85 | –529.35 | 44 | 1559.05 | 1197.15 |
| 15 | –1561.85 | –680.05 | 45 | 1559.05 | 1355.25 |
| 16 | –1606.35 | –952.35 | 46 | 1459.05 | 1601.45 |
| 17 | –1606.35 | –1208.95 | 47 | 1116.25 | 1701.45 |
| 18 | –1657.25 | –1417.35 | 48 | 924.45 | 1701.45 |
| 19 | –1504.75 | –1482.65 | 49 | 738.55 | 1677.55 |
| 20 | –1581.15 | –1713.75 | 50 | 558.25 | 1677.55 |
| 21 | –1405.25 | –1713.75 | 51 | 373.85 | 1677.55 |
| 22 | –1217.95 | –1713.75 | 52 | 193.55 | 1677.55 |
| 23 | –1042.05 | –1713.75 | 53 | 6.95 | 1701.45 |
| 24 | –853.25 | –1713.75 | 54 | –168.95 | 1701.45 |
| 25 | –677.35 | –1713.75 | 55 | –357.75 | 1701.45 |
| 26 | 829.85 | –1672.35 | 56 | –533.65 | 1701.45 |
| 27 | 1059.15 | –1685.25 | 57 | –849.95 | 1748.45 |
| 28 | 1247.95 | –1685.25 | 58 | –1127.75 | 1748.45 |
| 29 | 1556.15 | –1723.85 | 59 | –1459.35 | 1643.45 |
| 30 | 1556.15 | –1535.05 | | | |

## Pad Assignment

### HT99C811




Chip size: $3640 \times 4120$ $(\mu m)^2$

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

## Pad Coordinates

Unit: μm

| Pad No. | X | Y | Pad No. | X | Y |
|---|---|---|---|---|---|
| 1 | –1528.90 | 1511.45 | 31 | 1491.55 | –1370.50 |
| 2 | –1566.20 | 1294.60 | 32 | 1493.95 | –1195.00 |
| 3 | –1566.20 | 1140.30 | 33 | 1527.35 | –784.55 |
| 4 | –1566.20 | 978.50 | 34 | 1527.35 | –633.10 |
| 5 | –1566.20 | 824.20 | 35 | 1527.35 | –479.10 |
| 6 | –1512.20 | 663.05 | 36 | 1527.35 | –325.10 |
| 7 | –1512.20 | 507.45 | 37 | 1527.35 | –180.90 |
| 8 | –1512.20 | 346.95 | 38 | 1527.35 | –26.90 |
| 9 | –1512.20 | 191.35 | 39 | 1474.65 | 188.70 |
| 10 | –1566.20 | 30.20 | 40 | 1506.35 | 726.05 |
| 11 | –1566.20 | –124.10 | 41 | 1527.35 | 1010.40 |
| 12 | –1566.20 | –285.90 | 42 | 1527.35 | 1164.40 |
| 13 | –1566.20 | –440.20 | 43 | 1527.35 | 1308.60 |
| 14 | –1566.20 | –602.00 | 44 | 1527.35 | 1462.60 |
| 15 | –1566.20 | –756.30 | 45 | 1525.40 | 1630.85 |
| 16 | –1566.20 | –918.10 | 46 | 1207.40 | 1685.85 |
| 17 | –1566.20 | –1072.40 | 47 | 1017.40 | 1740.85 |
| 18 | –1598.50 | –1227.80 | 48 | 825.40 | 1740.85 |
| 19 | –1524.00 | –1405.15 | 49 | 647.30 | 1724.10 |
| 20 | –1144.05 | –1767.85 | 50 | 494.50 | 1724.10 |
| 21 | –989.75 | –1767.85 | 51 | 331.20 | 1724.10 |
| 22 | –824.30 | –1767.85 | 52 | 178.40 | 1724.10 |
| 23 | –680.10 | –1767.85 | 53 | 15.85 | 1729.80 |
| 24 | –526.10 | –1767.85 | 54 | –138.45 | 1729.80 |
| 25 | –381.90 | –1767.85 | 55 | –300.25 | 1729.80 |
| 26 | 1037.30 | –1746.85 | 56 | –471.35 | 1729.80 |
| 27 | 1226.50 | –1767.85 | 57 | –655.10 | 1740.85 |
| 28 | 1520.10 | –1830.50 | 58 | –1065.60 | 1764.75 |
| 29 | 1544.10 | –1671.70 | 59 | –1473.90 | 1701.45 |
| 30 | 1544.10 | –1527.50 | | | |

## Pad Description

| Pad No. | Pin Name | I/O | Mask Option | Function |
|---|---|---|---|---|
| 2,1<br>49~46<br>59,58 | PA0~PA7 | I/O | Wake-Up<br>Pull-High<br>or None | Bidirectional 8-bit Input/Output port. Each bit can be configured as a wake-up input by mask option. Software instructions determine the CMOS output or schmitt trigger input with or without pull-high resistor (mask option). |
| 10~7<br>53~50 | PB0~PB7 | I/O | — | Bidirectional 8-bit Input/Output port. Software instructions determine the NMOS open drain output or schmitt trigger input. |
| 19<br>20 | VSS<br>AVSS | — | — | Negative power supply, GND.<br>Analog negative power supply, AGND |
| 21 | $\overline{INT}$ | I | — | External interrupt schmitt trigger input with pull-high resistor. Edge triggered activated on a high to low transition. |
| 22 | TMR0 | I | — | Schmitt trigger input for timer/event counter 0 |
| 32 | TMR1 | I | — | Schmitt trigger input for timer/event Counter 1 |
| 27 | AOUT | O | — | The D/A converter output can be programmed by D/A controlled register. The register has a total of 8 digits from MSB to LSB, and offers 8-bit resolution for D/A converter and one LSB is 1/256 VDD. |
| 33 | $\overline{RES}$ | I | — | Schmitt trigger reset input. Active low. |
| 35<br>34 | VDD<br>AVDD | — | — | Positive power supply, VDD<br>Analog negative power supply, AVDD |
| 28~31<br>18~15 | PD0~PD3<br>PD4~PD7 | I/O | Pull-High<br>or None<br>(mask type only) | Bidirectional 8-bit Input/Output port. Software instructions determine the CMOS output or schmitt trigger input with or without a pull-high resistor (mask option). |
| 14~11<br>6~3 | PE0~PE3<br>PE4~PE7 | I/O | Pull-High<br>or None<br>(mask type only) | Bidirectional 8-bit Input/Output port. Software instructions determine the CMOS output or schmitt trigger input with or without pull-high resistor (mask option). |
| 45~42<br>39~36 | PF0~PF3<br>PF4~PF7 | I/O | Pull-High<br>or None<br>(mask type only) | Bidirectional 8-bit Input/Output port. Software instructions determine the CMOS output or schmitt trigger input with or without pull-high resistor (mask option). |
| 23~26<br>57~54 | PG0~PG3<br>PG4~PG7 | I/O | Pull-High<br>or None<br>(mask type only) | Bidirectional 8-bit Input/Output port. Software instructions determine the CMOS output or schmitt trigger input with or without pull-high resistor (mask option). |

| Pad No. | Pin Name | I/O | Mask Option | Function |
|---|---|---|---|---|
| 40<br>41 | OSC1<br>OSC2 | I<br>O | Crystal<br>or RC | OSC1, OSC2 are connected to an RC network or a crystal (determined by mask option) for the internal system clock. In the case of RC operation, OSC2 is the output terminal for 1/4 system clock. |

## Absolute Maximum Ratings*

Supply Voltage ..... –0.3V to 5.5V
Storage Temperature..... –50°C to 125°C
Input Voltage..... $V_{SS}$–0.3V to $V_{DD}$+0.3V
Operating Temperature..... –25°C to 70°C

*Note: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only. Functional operation of this device at these or any other conditions above those indicated in the operational sections of this specification is not implied and exposure to absolute maximum rating conditions for extended periods may affect device reliability.

## D.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $V_{DD}$ (mask) | Operating Voltage | — | — | 2.4 | — | 5.2 | V |
| $V_{DD}$ (OTP) | | — | — | 3.0 | — | 5.2 | V |
| $I_{DD1}$ | Operating Current (Crystal OSC) | 3V | No load, $f_{SYS}$=4MHz | — | 0.7 | 1.5 | mA |
| | | 5V | | — | 2 | 5 | mA |
| $I_{DD2}$ | Operating Current (RC OSC) | 3V | No load, $f_{SYS}$=2MHz | — | 0.6 | 1 | mA |
| | | 5V | | — | 1.6 | 5 | mA |
| $I_{STB1}$ | Standby Current (WDT Enabled) | 3V | No load, System HALT | — | — | 5 | μA |
| | | 5V | | — | — | 20 | μA |
| $I_{STB2}$ | Standby Current (WDT Disabled) | 3V | No load, System HALT | — | — | 1 | μA |
| | | 5V | | — | — | 2 | μA |
| $V_{IL}$ | Input Low Voltage for I/O Ports | 3V | — | 0 | — | 0.9 | V |
| | | 5V | — | 0 | — | 1.5 | V |
| $V_{IH}$ | Input High Voltage for I/O Ports | 3V | — | 2.1 | — | 3 | V |
| | | 5V | — | 3.5 | — | 5 | V |
| $V_{IL1}$ | Input Low Voltage ($\overline{RES}$, TMR0, TMR1, $\overline{INT}$) | 3V | — | 0 | — | 0.7 | V |
| | | 5V | — | 0 | — | 1.3 | V |
| $V_{IH1}$ | Input High Voltage ($\overline{RES}$, TMR0, TMR1, $\overline{INT}$) | 3V | — | 2.3 | — | 3 | V |
| | | 5V | — | 3.8 | — | 5 | V |

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $I_{OL}$ | I/O Port Sink Current | 3V | $V_{OL}$=0.3V | 1.5 | 2.5 | — | mA |
| | | 5V | $V_{OL}$=0.5V | 4 | 6 | — | mA |
| $I_{OH}$ | I/O Port Source Current | 3V | $V_{OH}$=2.7V | –1 | –1.5 | — | mA |
| | | 5V | $V_{OH}$=4.5V | –2 | –3 | — | mA |
| $R_{PH}$ | Pull-High Resistance of I/O Ports and $\overline{INT}$ | 3V | — | — | 18 | — | kΩ |
| | | 5V | — | — | 18 | — | kΩ |
| $V_{dac}$ | DAC Output Level | — | — | AVSS | — | AVDD | V |
| $I_{dac}$ | DAC Drive Current | 5V | $V_{OH}$=0.9VDD | — | 50 | — | mA |
| $R_{dac}$ | DAC Output Resistance | — | $V_{OL}$=0.1VDD | — | 10 | — | kΩ |

## A.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $f_{SYS1}$ | System Clock (Crystal OSC) | 3V | — | 400 | — | 4000 | kHz |
| | | 5V | — | 400 | — | 4000 | kHz |
| $f_{SYS2}$ | System Clock (RC OSC) | 3V | — | 400 | — | 2000 | kHz |
| | | 5V | — | 400 | — | 3000 | kHz |
| $f_{TIMER}$ | Timer I/P Frequency (TMR) | 3V | — | 0 | — | 4000 | kHz |
| | | 5V | — | 0 | — | 4000 | kHz |
| $t_{WDTOSC}$ | Watchdog Oscillator | 3V | — | 45 | 90 | 180 | μs |
| | | 5V | — | 35 | 65 | 130 | μs |
| $t_{WDT1}$ | Watchdog Time-Out Period (RC) | 3V | Without WDT Prescaler | 12 | 23 | 45 | ms |
| | | 5V | | 9 | 17 | 35 | ms |
| $t_{WDT2}$ | Watchdog Time-Out Period (Sysstem Clock) | — | Without WDT Prescaler | — | 1024 | — | $t_{SYS}$ |
| $t_{RES}$ | External Reset Low Pulse Width | — | — | 1 | — | — | μs |
| $t_{XST}$ | System Start-Up Timer | — | Power-Up or Wake-Up from Halt | — | 1024 | — | $t_{SYS}$ |
| $t_{INT}$ | Interrupt Pulse Width | — | — | 1 | — | — | μs |

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

For other important system architecture and function description, refer to HT48700 data sheet.

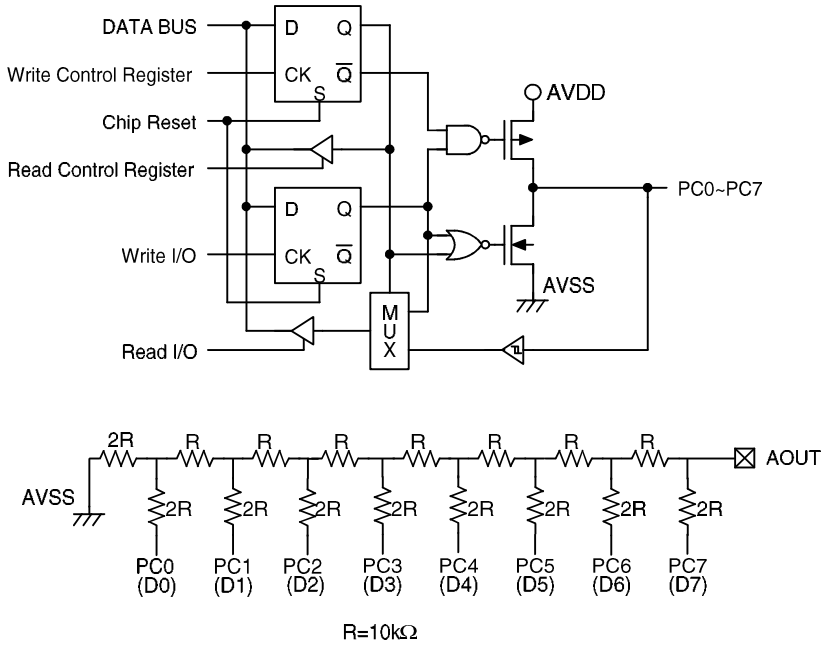
## D/A Converter Description

The HT99C810/HT99C811 built-in 8-bit D/A converter is one of the simple designed methods of the D/A converter. The R/2R lattice method is used in HT99C810/HT99C811 which offers 8-bit resolution. The HT48700 general I/O PORTC is replaced by a D/A converter control register to control the D/A output value as shown below:

| PORTC | PC7 | PC6 | PC5 | PC4 | PC3 | PC2 | PC1 | PC0 |
|---|---|---|---|---|---|---|---|---|
| | (MSB) | | | | | | | (LSB) |
| Determine D/A Value | 1/2 AVDD | 1/4 AVDD | 1/8 AVDD | 1/16 AVDD | 1/32 AVDD | 1/64 AVDD | 1/128 AVDD | 1/256 AVDD |

* D/A converter has isolated power line layout itself, in addition, AVDD and AVSS pads are included.

## D/A Converter Circuit



## Application Circuit

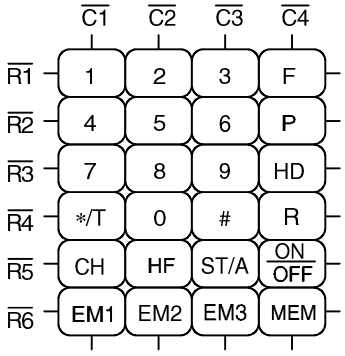
### Cordless phone controller arrangement

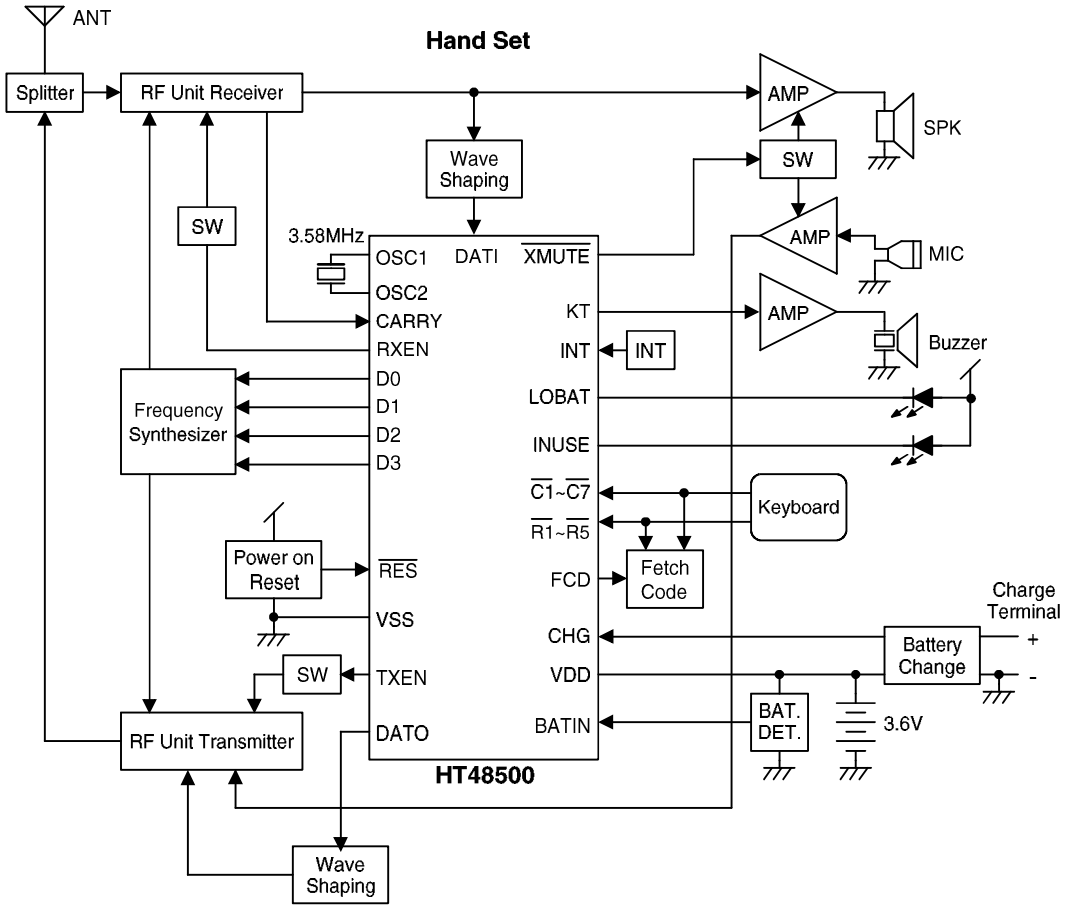
- Base unit: HT99C810/HT99C811

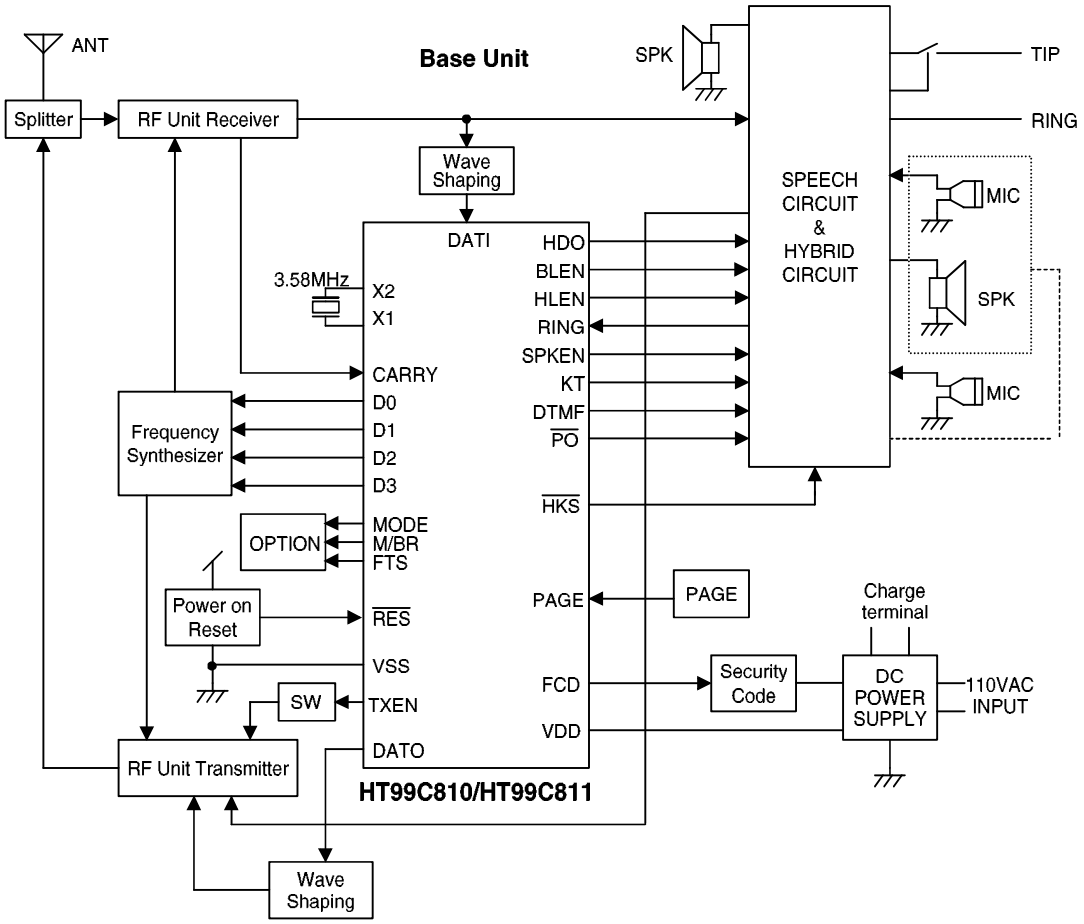
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **PA0** | $\overline{PO}$ | **PB0** | INUSE | **PD0** | MODE | **AOUT** | DTMF |
| **PA1** | DATI | **PB1** | KT | **PD1** | M/BR | | |
| **PA2** | DATO | **PB2** | RING | **PD2** | CARRY | | |
| **PA3** | TXEN | **PB3** | $\overline{HKS}$ | **PD3** | LED | | |
| **PA4** | PWDN | **PB4** | HDO | **PD4** | D0 | | |
| **PA5** | FTS | **PB5** | HLEN | **PD5** | D1 | | |
| **PA6** | FCD | **PB6** | BLEN | **PD6** | D2 | | |
| **PA7** | INT/PAGE | **PB7** | SPKEN | **PD7** | D3 | | |

- Hand set: HT48500

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **PA0** | $\overline{C1}$ | **PB0** | $\overline{R1}$ | **PC0** | $\overline{XMUTE}$ | **PD0** | NC |
| **PA1** | $\overline{C2}$ | **PB1** | $\overline{R2}$ | **PC1** | LOBAT | **PD1** | NC |
| **PA2** | $\overline{C3}$ | **PB2** | $\overline{R3}$ | **PC2** | BATIN | **PD2** | RXEN |
| **PA3** | $\overline{C4}$ | **PB3** | $\overline{R4}$ | **PC3** | KT | **PD3** | TXEN |
| **PA4** | $\overline{C5}$ | **PB4** | $\overline{R5}$ | **PC4** | CARRY | **PD4** | D0 |
| **PA5** | $\overline{C6}$ | **PB5** | $\overline{R6}$ | **PC5** | CHG | **PD5** | D1 |
| **PA6** | $\overline{C7}$ | **PB6** | DATI | **PC6** | LED | **PD6** | D2 |
| **PA7** | INT/PAGE | **PB7** | DATO | **PC7** | FCD | **PD7** | D3 |




ANT
Hand Set
Splitter
RF Unit Receiver
Wave Shaping
AMP
SPK
SW
SW
3.58MHz
OSC1
DATI
XMUTE
AMP
MIC
OSC2
CARRY
KT
AMP
RXEN
INT
INT
Buzzer
D0
D1
D2
D3
LOBAT
INUSE
Frequency Synthesizer
C1~C7
R1~R5
Keyboard
Power on Reset
RES
FCD
Fetch Code
Charge Terminal
VSS
CHG
Battery Change
+
-
SW
TXEN
VDD
BAT. DET.
3.6V
RF Unit Transmitter
DATO
BATIN
HT48500
Wave Shaping


Base Unit
ANT
Splitter
RF Unit Receiver
Wave Shaping
DATI
3.58MHz
X2
X1
CARRY
D0
D1
D2
D3
MODE
M/BR
FTS
RES
VSS
TXEN
DATO
HDO
BLEN
HLEN
RING
SPKEN
KT
DTMF
PO
HKS
PAGE
FCD
VDD
Frequency Synthesizer
OPTION
Power on Reset
SW
RF Unit Transmitter
Wave Shaping
HT99C810/HT99C811
SPK
SPEECH CIRCUIT & HYBRID CIRCUIT
TIP
RING
MIC
SPK
MIC
PAGE
Security Code
Charge terminal
DC POWER SUPPLY
110VAC INPUT

## System Architecture

### Execution flow

The system clock for the HT99C810/HT99C811 is derived from either a crystal or an RC oscillator. The system clock is internally divided into four non-overlapping clocks. One instruction cycle consists of four system clock cycles.

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

### Program counter – PC

The 13-bit program counter (PC) controls the sequence in which the instructions stored in program ROM are executed and its contents specify a maximum of 8192 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, internal interrupt, external interrupt or return from subroutine, the PC manipulates the program transfer by loading the address corresponding to each instruction.

The conditional skip is activated by 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.

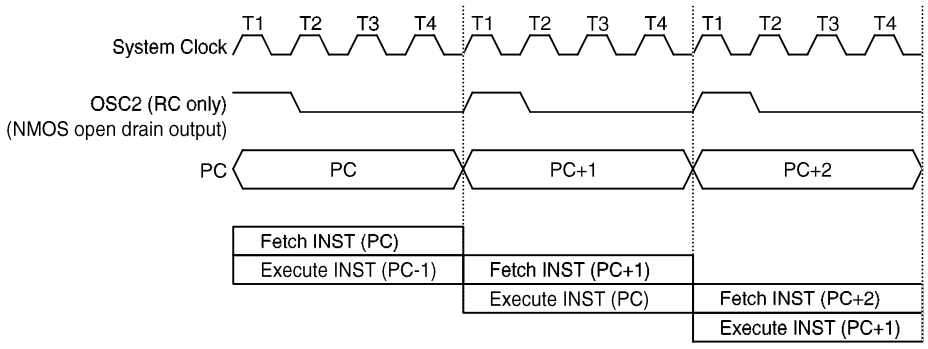
When a control transfer takes place, an additional dummy cycle is required.

### Program memory – ROM

The program memory is used to store the program instructions which are to be executed. It also contains data, table, and interrupt entries, and is organized into 8192 × 16 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 chip reset, the program always begins execution at location 000H.
- Location 004H
  This area is reserved for the external interrupt service program. If the $\overline{\text{INT}}$ input pin is activated, and the interrupt is enabled and the stack is not full, the program begins execution at location 004H.




Execution flow

- Location 008H

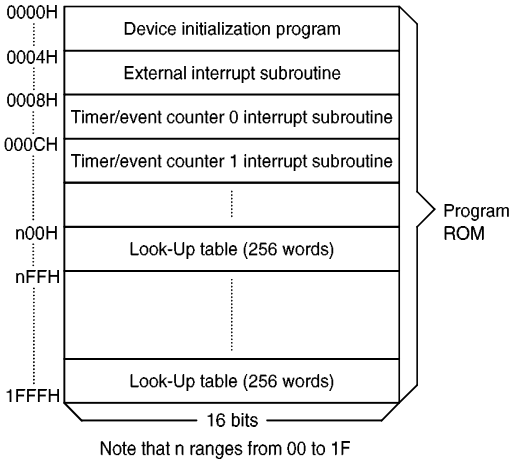
  This area is reserved for the timer/event counter 0 interrupt service program. If timer interrupt results from a timer/event counter 0 overflow, and if the interrupt is enabled and the stack is not full, the program begins execution at location 008H.

- Location 00CH

  This area is reserved for the timer/event counter 1 interrupt service program. If timer interrupt resulting from a timer/event counter 1 overflow, and if the interrupt is enabled and the stack is not full, the program begins execution at location 00CH.

- Table location

  Any location in the ROM space can be used as look–up tables. The instructions TABRDC [m] (the current page, 1 page=256 words) and TABRDL [m] (the last page) transfer the contents of the lower-order byte to the specified data memory, and the higher-order byte to TBLH (08H). Only the destination of the lower-order byte in the table is well-defined, the higher-order byte of the table word are transferred to the TBLH. The Table Higher-order byte register (TBLH) is read only. The




Program memory

Table Pointer (TBLP) is a read/write register (07H), which indicates the table location. Before accessing the table, the location must be placed in TBLP. The TBLH is read only and cannot be restored. If the main routine and the ISR (Interrupt Service Routine) both employ the table read instruction, the contents of the TBLH in the main routine are likely to be changed by the table read instruction used in

| Mode | Program Counter | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ***12** | ***11** | ***10** | ***9** | ***8** | ***7** | ***6** | ***5** | ***4** | ***3** | ***2** | ***1** | ***0** |
| Initial reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| External interrupt | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| Timer/event counter 0 overflow | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| Timer/event counter 1 overflow | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| Skip | PC+2 | | | | | | | | | | | | |
| Loading PCL | *12 | *11 | *10 | *9 | *8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| Jump, call branch | #12 | #11 | #10 | #9 | #8 | #7 | #6 | #5 | #4 | #3 | #2 | #1 | #0 |
| Return from subroutine | S12 | S11 | S10 | S9 | S8 | S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 |

Program counter

Notes: *12~*0: Program counter bits
#12~#0: Instruction code bits
S12~S0: Stack register bits
@7~@0: PCL bits

the ISR. Errors can occur. In other words, simultaneously using the table read instruction in the main routine and the ISR should be avoided. However, if the table read instruction has to be applied in both the main routine and the ISR, the interrupt(s) is supposed to be disabled prior to the table read instruction and will not be enabled until the TBLH has been backed-up. All table related instructions need two cycles to complete the operation. These areas may function as normal program memory depending upon the requirements.

### Stack register – STACK

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

If the stack is full and a non-masked interrupt takes place, the interrupt request flag will be recorded but the acknowledgment will be inhibited. When the stack pointer is decremented (by RET or RETI), the interrupt will be serviced. This feature prevents stack overflow allowing the programmer to use the structure more easily. In a similar case, if the stack is full and a CALL is subsequently executed, stack overflow occurs and the first entry will be lost (only the most recent eight return addresses are stored).

### Data memory – RAM

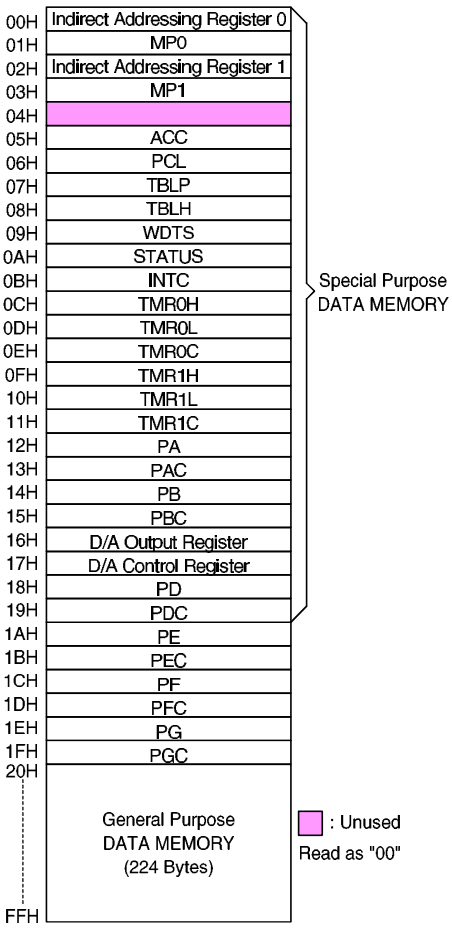
The data memory is designed with $255 \times 8$ bits. The data memory is divided into two functional groups: special function registers and general purpose data memory (224×8). Most of them are read/write, but some are read only.

The special function registers include the indirect addressing register 0 (00H), the memory pointer register 0 (MP0;01H), the indirect addressing register 1 (02H), the memory pointer register 1 (MP1;03H), the accumulator (ACC;05H), the program counter lower-byte register (PCL;06H), the table pointer (TBLP;07H), the table higher-order byte register (TBLH;08H), the watchdog timer option setting register (WDTS;09H), the status register (STATUS;0AH), the interrupt control register (INTC;0BH), the timer/event counter 0 higher-order byte register (TMR0H;0CH), the timer/event counter 0 lower-order byte register (TMR0L;0DH), the timer/event counter 0 control register (TMR0C;0EH),the timer/event counter 1 higher-order byte register (TMR1H;0FH), the timer/event counter 1 lower-order byte register (TMR1L;10H), the timer/event counter 1 control register (TMR1C;11H), the I/O registers (PA;12H, PB;14H, PD;18H, PE;1AH, PF;1CH, PG;1EH) and the I/O control registers (PAC;13H, PBC;15H, PDC;19H, PEC;1BH, PFC;1DH, PGC;1FH). The remaining space before the 20H is reserved for future expanded usage and reading these locations will return the result to 00H. The general purpose data memory, addressed from 20H to FFH, is used for data and control

| Instruction(s) | Table Location | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *12 | *11 | *10 | *9 | *8 | *7 | *6 | *5 | *4 | *3 | *2 | *1 | *0 |
| TABRDC [m] | P12 | P11 | P10 | P9 | P8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| TABRDL [m] | 1 | 1 | 1 | 1 | 1 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |

Table location

Notes: *12~*0: Table location bits
@7~@0: Table pointer bits
P12~P8: Current program counter bits




RAM mapping

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 registers (MP0;01H, MP1;03H).

### Indirect addressing register

Location 00H and 02H are indirect addressing registers that are not physically implemented. Any read/write operation of [00H] and [02H] access data memory pointed to by MP0 (01H) and MP1 (03H) respectively. Reading location 00H or 02H indirectly will return the result to 00H. Writing indirectly results in no operation.

The function of data movement between two indirect addressing registers, is not supported. The memory pointer registers, MP0 and MP1, are 8-bit registers which can be used to access the data memory by combining corresponding indirect addressing registers.

### 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 these two data memories 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 a zero flag (Z), a carry flag (C), an auxiliary carry flag (AC), an overflow flag (OV), a power down flag (PD) and a watchdog time-out flag (TO). The status register not only records the status information but also controls the operation sequence.

With the exception of the TO and PD flags, bits in the status register can be altered by instructions like most other register. Any data written into the status register will not change the TO or PD flags. It should be noted that operations related to the status register may give different results from those intended. The TO and PD

flags can only be changed by system power up, watchdog timer overflow, executing the HALT instruction and clearing the watchdog timer.

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

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

### Interrupt

The HT99C810/HT99C811 provides an external interrupt and internal timer/event counter interrupts. The interrupt control register (INTC;0BH) contains the interrupt control bits to set the enable/disable and the interrupt request flags.

Once an interrupt subroutine is serviced, all other interrupts will be blocked (by clearing the EMI bit). This scheme may prevent any further interrupt nesting. Other interrupt requests may occur during this interval but only the interrupt request flag is recorded. If a certain interrupt needs servicing within the service routine, the EMI bit and the corresponding bit of the INTC may be set to permit interrupt nesting. If the stack is full, the interrupt request will not be acknowledged, even if the related interrupt is enabled, until the SP is decremented. If immediate service is desired, the stack must be prevented from becoming full.

All these kinds of interrupt have a wake-up capability. As an interrupt is serviced, a control transfer occurs by pushing the program counter onto the stack, followed by a branch to a subroutine at specified location in the program memory. Only the the contents of the program counter is pushed onto the stack. If the contents of the register and Status register (STATUS) are altered by the interrupt service program which corrupts the desired control sequence, the contents should be saved in advance.

External interrupt is triggered by a high to low transition of $\overline{\text{INT}}$ and the related interrupt request flag (EIF; bit 4 of INTC) will be set. When the interrupt is enabled, and the stack is not

| Labels | Bits | Function |
|---|---|---|
| C | 0 | C is set if the operation results in a carry during an addition operation or if a borrow does not take place during a subtraction operation; otherwise C is cleared. C is also affected by a rotate through carry instruction. |
| AC | 1 | AC is set if the operation results in a carry out of the low nibbles in addition or a borrow does not take place from the high nibble into the low nibble in subtraction; otherwise AC is cleared. |
| Z | 2 | Z is set if the result of an arithmetic or logic operation is zero; otherwise Z is cleared. |
| OV | 3 | OV is set if the operation results in a carry into the highest-order bit but not a carry out of the highest-order bit, or vice versa; otherwise OV is cleared. |
| PD | 4 | PD is cleared when either a system power-up or executing the CLR WDT instruction. PD is set by executing the HALT instruction. |
| TO | 5 | 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 | Undefined, read as “0” |
| – | 7 | Undefined, read as “0” |

STATUS register

full and the external interrupt is active, a subroutine call to location 04H will occur. The interrupt request flag (EIF) and EMI bits will be cleared to disable other interrupts.

The internal timer/event counter 0 interrupt is initialized by setting the timer/event counter 0 interrupt request flag (T0F; bit 5 of INTC), which is normally caused by a timer/event 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 timer/event counter 1 interrupt is operated in the same manner as timer/event counter 0. The related interrupt control bits ET1I and T1F of timer/event counter 1 are bit 3 and bit 6 of the INTC respectively.

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

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

The timer/event counter 0/1 interrupt request flag (T0F/T1F), external interrupt request flag (EIF), enable timer/event counter 0/1 bit (ET0I/ET1I), enable external interrupt bit (EEI) and enable master interrupt bit (EMI) constitute an interrupt control register (INTC) which is located at 0BH in the data memory.

| Register | Bit No. | Label | Function |
|---|---|---|---|
| INTC (0BH) | 0 | EMI | Controls the master (global) interrupt (1=enabled; 0=disabled) |
| | 1 | EEI | Controls the external interrupt (1=enabled; 0=disabled) |
| | 2 | ET0I | Controls the timer/event counter 0 interrupt (1=enabled; 0=disabled) |
| | 3 | ET1I | Controls the timer/event counter1 interrupt (1=enabled; 0=disabled |
| | 4 | EIF | External interrupt request flag (1=active; 0=inactive) |
| | 5 | T0F | Internal timer/event counter 0 request flag. (1=active; 0=inactive) |
| | 6 | T1F | Internal timer/event counter1 request flag. (1=active; 0=inactive) |
| | 7 | — | Unused bit, read as “0” |

INTC register

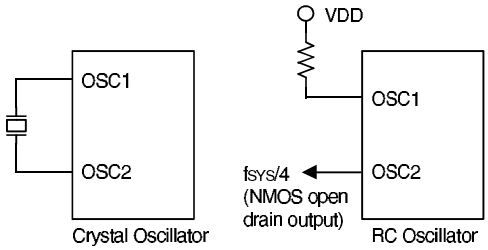
EMI, EEI, ET0I, ET1I are used to control the enabling/disabling of interrupts. These bits prevent the requested interrupt from being serviced. Once the interrupt request flags (T0F, T1F, EIF) are set, they will remain in the INTC register until the interrupts are serviced or cleared by a software instruction.

It is suggested that a program does not use the "CALL subroutine" within the interrupt subroutine. Because interrupts often occur in an unpredictable manner or need to be serviced immediately in some applications, if only one stack is left and enabling the interrupt is not well controlled, once the "CALL subroutine" operates in the interrupt subroutine, it will damage the original control sequence.

### Oscillator configuration

There are two oscillator circuits in the HT99C810/HT99C811. Both are designed for system clocks; the RC oscillator and the crystal oscillator, which are determined by mask options. No matter what oscillator type is selected, the signal provides the system clock. The HALT mode stops the system oscillator and ignores the external signal to conserve power.

If an RC oscillator is used, an external resistor between OSC1 and VDD is needed and the resistance must range from 51kΩ to 1MΩ. The system clock, divided by four, is available on OSC2, which can be used to synchronize external logic. The RC oscillator provides the most cost effective solution. However, the frequency of 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.



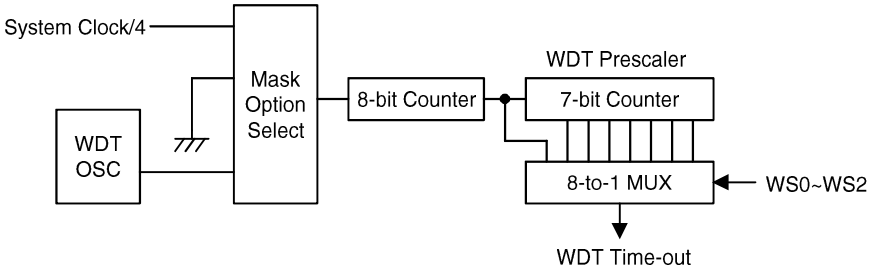
System oscillator

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

The WDT oscillator is a free running on-chip RC oscillator, and no external components are required. Even if the system enters the power down mode, the system clock is stopped, but the WDT oscillator still works with a period of approximately 78 μs. The WDT oscillator can be disabled by mask option to conserve power.

### Watchdog timer – WDT

The clock source of the WDT 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 the program sequence from jumping to an unknown location with unpredictable results. The watch-



Watchdog timer

dog timer can be disabled by mask option. If the watchdog timer is disabled, all the executions related to the WDT result to 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 20 ms. This time-out period may vary with temperature, VDD and process variations. By invoking the WDT prescaler, longer time-out periods can be realized. Writing data to WS2, WS1, WS0 (bit 2,1,0 of the WDTS) can give different time-out periods. If WS2, WS1, WS0 are all equal to 1, the division ratio is up to 1:128, and the maximum time-out period is 2.6 seconds.

If the WDT oscillator is disabled, the WDT clock may still come from the instruction clock and operate in the same manner except that in the HALT state the WDT may stop counting and lose its protection purpose. In this situation the logic can only be restarted by external logic. The high nibble and bit 3 of the WDTS are reserved for user defined flags, which can be used to indicate some specified status.

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

WDTS register

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 overflow of the WDT under normal operation will initialize “chip reset” and set the status bit "TO". An overflow in the HALT mode initializes a “warm reset” only when the PC and SP are reset to zero. To clear the contents of the WDT (including the WDT prescaler ), there are three methods to be adopted; external reset (a low level to $\overline{\text{RES}}$), software instruction(s), or a HALT instruction. The software instruction include CLR WDT and CLR WDT1/CLR WDT2. Of these two types of instruction, only one can be active depending on the mask option — “CLR WDT times selection option”. If the “CLR WDT” is selected (i.e.. CLRWDT times equal one), any execution of the CLR WDT instruction will clear the WDT. In case “CLR WDT1” and “CLR WDT2” are chosen (i.e.. CLRWDT times equal two), these two instructions must be executed to clear the WDT; otherwise, the WDT may reset the chip due to a time-out.

### Power down operation – HALT

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

- The system oscillator will turn 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 counted again (if the WDT clock is from the WDT oscillator).
- All I/O ports maintain their original status.
- The PD flag is set and the TO flag is cleared.

The system can quit 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”. Examining the TO and PD flags, the reason for chip reset can be determined. The PD flag is cleared when system power-up or executing the CLR WDT instruction and is set when the HALT instruction is executed. The TO flag is set if the WDT time-out occurs, which causes a wake-up that only resets the PC and SP, and leaves the others in their original status.

The port A wake-up and interrupt methods can be considered as a continuation of normal execution. Each bit in port A can be independently selected to wake-up the device by mask option. Awakening from an I/O port stimulus, the program resumes execution of the next instruction. However, if the program awakens from an inter-

rupt, two sequences may occur. The program will resume execution at the next instruction if the related interrupt(s) is (are) disabled or the interrupt(s) is enabled but the stack is full. A regular interrupt response may take place if the interrupt is enabled and the stack is not full.

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

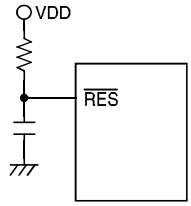
To minimize power consumption, all I/O pins should be carefully managed before entering the HALT status.

### Reset

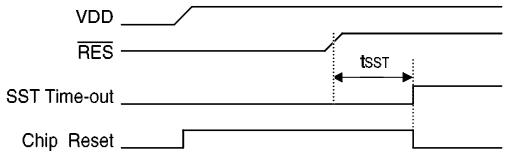
There are three ways in which a reset can occur:

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

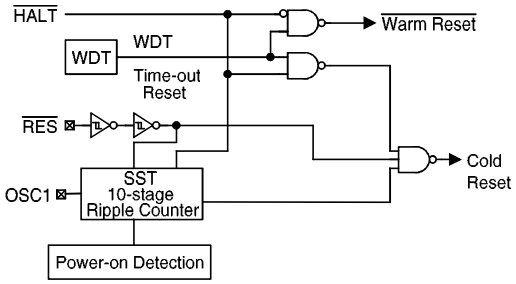
The WDT time-out during HALT is different from other chip reset conditions, since it can perform a "warm reset" that just resets the PC and SP, leaving the other circuits in their original state. Some registers remain unchanged during other reset conditions. Most registers are reset to the "initial condition"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




Reset circuit




Reset timing chart




Reset configuration

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

Note: "u" means "unchanged"

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

When the 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.

The states of the registers are summarized in the following table:

| Register | Reset (Power On) | WDT Time-Out (Normal Operation) | $\overline{RES}$ Reset (Normal Operation) | $\overline{RES}$ Reset (HALT) | WDT Time-Out (HALT) |
|---|---|---|---|---|---|
| TMR1H | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TMR1L | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TMR1C | 00-0 1--- | 00-0 1--- | 00-0 1--- | 00-0 1--- | uu-u u--- |
| TMR0H | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TMR0L | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TMR0C | 00-0 1--- | 00-0 1--- | 00-0 1--- | 00-0 1--- | uu-u u--- |
| PC | 0000H | 0000H | 0000H | 0000H | 0000H* |
| MP0 | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| MP1 | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| ACC | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TBLP | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TBLH | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| STATUS | --00 xxxx | --1u uuuu | --uu uuuu | --01 uuuu | --11 uuuu |
| INTC | -000 0000 | -000 0000 | -000 0000 | -000 0000 | -uuu uuuu |
| WDTS | 0000 0111 | 0000 0111 | 0000 0111 | 0000 0111 | uuuu uuuu |
| PA | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PAC | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PB | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PBC | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| D/A Output Register | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| D/A Control Register | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PD | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PDC | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PE | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PEC | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PF | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PFC | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PG | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PGC | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |

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

The functional unit chip reset status is shown in the following table.

| | |
|---|---|
| PC | 000H |
| Interrupt | Disabled |
| Prescaler | Cleared |
| WDT | Cleared. After master reset, WDT starts counting |
| Timer/Event Counter (0/1) | Off |
| Input/Output ports | Input mode |
| SP | Points to the top of the stack |

### Timer/Event Counter

Two timer/event counters are implemented in the HT99C810/HT99C811. The timer/event counter 0 and timer/event counter 1 contain 16-bit programmable count-up counters and the clock may come from an external source or the system clock divided by 4.

Using the internal instruction clock, there is only one reference time-base. The external clock input allows the user to count external events, measure time intervals or pulse width, or generate an accurate time base.

There are three registers related to the timer/event counter 0; TMR0H (0CH), TMR0L (0DH), TMR0C (0EH). Writing TMR0L only writes the data into a low byte buffer, and writing TMR0H will simultaneously write the data and the contents of the low byte buffer into the timer/event counter 0 preload register (16-bit). The timer/event counter 0 preload register is changed by writing TMR0H operations and writing TMR0L will keep the timer/event counter 0 preload register unchanged.

Reading TMR0H will also latch the TMR0L into the low byte buffer to avoid the false timing problem. Reading TMR0L returns the contents of the low byte buffer. In other words, the low byte of timer/event counter 0 cannot be read directly. It must read the TMR0H first to make the low byte contents of timer/event counter 0 latched into the buffer.

There are three registers related to timer/event counter 1; TMR1H (0FH), TMR1L (10H), TMR1C (11H). The timer/event counter 1 operates in the same manner as with timer/event counter 0.

The TMR0C is the timer/event counter 0 control register, which defines the timer/event counter 0 options. The timer/event counter 1 has the same options as the timer/event counter 0 and is defined by TMR1C.

The timer/event counter control registers define the operation mode, counting enable or disable and active edge.

The TM0, TM1 bits define the operation mode. The event count mode is used to count external

| Label (TMRC) | Bits | Function |
|---|---|---|
| — | 0~2 | Unused bits, read as “0” |
| TE | 3 | To define the TMR active edge of the timer/event counter<br>(0=active on low to high; 1=active on high to low) |
| TON | 4 | To enable/disable timer counting<br>(0=disabled; 1=enabled) |
| — | 5 | Unused bits, read as “0” |
| TM0<br>TM1 | 6<br>7 | To define the operation mode<br>01=Event count mode (external clock)<br>10=Timer mode (internal clock)<br>11=Pulse width measurement mode<br>00=Unused |

TMR0C/TMR1C register

events, which means the clock source comes from an external (TMR0/TMR1) pin. The timer mode functions as a normal timer with the clock source coming from the instruction clock. The pulse width measurement mode can be used to count the high or low level duration of the external signal (TMR0/TMR1). The counting is based on the instruction clock.

In the Event count or Timer mode, once the timer/event counter starts counting, it will count from the current contents in the timer/event counter to FFFFH. Once overflow occurs, the counter is reloaded from the timer/event counter preload register and generates the corresponding interrupt request flag (T0F/T1F; bit 5/6 of INTC) at the same time.

In the pulse width measurement mode, the TON and TE bits are equal to one. Once the TMR0/TMR1 has received a transient from low to high (or high to low; if the TE bit is 0) it will start counting until the TMR0/TMR1 returns to the original level and resets the TON. The measured result will remain in the timer/event counter even if the activated transient occurs again. In other words, only one cycle measurements can be made until the TON is set. The cycle measurement will function again as long as it receives further transient pulse. Note that, in this operation mode, the timer/event counter starts counting not according to the logic level but according to the transient edges. In the case of counter overflows, the counter is reloaded from the timer/event counter preload register and issues an interrupt request similar to the other two modes.

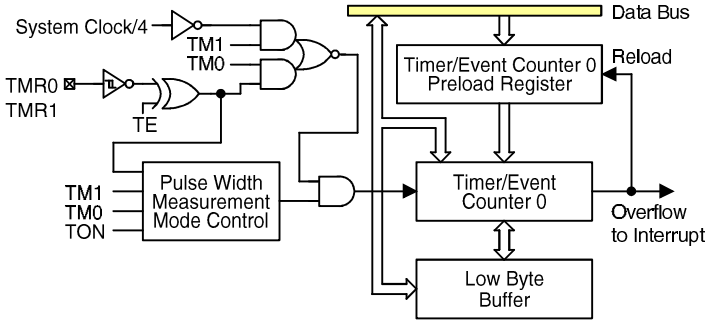
To enable the counting operation, the Timer ON bit (TON; bit 4 of TMR0C/TMR1C) should be set to 1. In the pulse width measurement mode, the TON will automatically be cleared after the measurement cycle is completed. But in the other two modes, the TON can only be reset by instruction. The overflow of the timer/event counter is one of the wake-up sources. No matter what type of operation mode is, writing a 0 to ET0I/ET1I can disable the corresponding interrupt service.

In the case of timer/event counter OFF condition, writing data to the timer/event counter preload register will also reload that data to timer/event counter. But if the timer/event counter is turned on, data written to the timer/event counter will only be kept in the timer/event counter preload register. The timer/event counter will still operate until an overflow occurs.

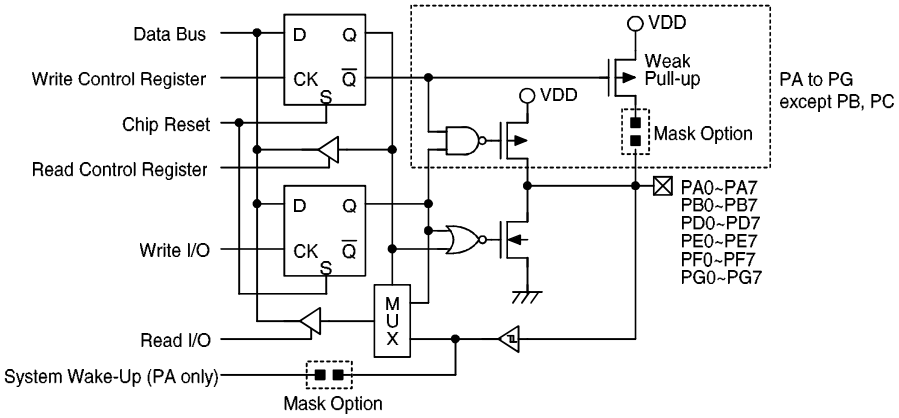
When the timer/event counter (reading TMR0H/TMR1H) is read, the clock will be blocked to avoid errors. As this may results in a counting error, this must be taken into consideration by the programmer.

### Input/output ports

There are 48 bidirectional input/output lines in the HT99C810/HT99C811, labeled from PA to PG except PC, which are mapped to the data memory of [12H], [14H], [18H], [1AH], [1CH] and [1EH] respectively. All 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




Timer/Event Counter0/1




Input/output ports

rising edge of instruction MOV A,[m] (m=12H, 14H, 18H, 1AH, 1CH or 1EH). For output operation, all data is latched and remains unchanged until the output latch is rewritten.

Each I/O line has its own control register (PAC, PBC, PDC, PEC, PFC, PGC) to control the input/output configuration. With this control register, CMOS output or schmitt trigger input with or without pull-high resistor (mask option) structures can be reconfigured dynamically (i.e., on-the-fly) under software control. To function as an input, the corresponding latch of the control register must write a "1". The pull-high resistance will exhibit automatically if the pull-high option is selected. The input source(s) also depend(s) on the control register. If the control register bit is "1", 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 "read-modify-write" instruction. For output function, CMOS is the only configuration. These control registers are mapped to locations 13H, 15H, 19H, 1BH, 1DH and 1FH.

After a chip reset, these input/output lines stay at high levels or floating (mask option). Each bit of these input/output latches can be set or cleared by the SET [m].i or CLR [m].i (m=12H, 14H, 18H, 1AH, 1CH or 1EH) instruction.

Some instructions first input data and then follow the output operations. For example, the SET [m].i, CLR [m].i, CPL [m] and CPLA [m] instructions 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 to wake-up the device.

The 8-bit D/A output register is mapped to the data memory of [16H] and its corresponding control register is mapped to location [17H] which must be set to "0" after initialization, when using the D/A function.

### Mask option

The following table shows five kinds of mask options in the HT99C810/HT99C811. All the mask options must be defined to ensure proper system functioning.

| No. | Mask Option |
|---|---|
| 1 | OSC type selection. This option is to determine whether an RC or Crystal oscillator is chosen as system clock. |
| 2 | WDT source selection. There are three types of selection: on-chip RC oscillator, instruction clock or disable the WDT. |
| 3 | CLRWDT 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 that only if both of the CLR WDT1 and CLR WDT2 instructions have been executed, then the WDT can be cleared. |
| 4 | Wake-up selection. This option defines the activity of the wake-up function. External I/O pins (PA only) all have the capability to wake-up the chip from a HALT. |
| 5 | Pull-high selection. This option is to determine whether the pull-high resistance is visible or not in the input mode of the I/O ports. Each bit of an I/O port can be independently selected. (See Note) |

Note: There are no pull-high selections in port B of HT99C810/HT99C811.

There are pull-high selections in port A, port D, port E, port F and port G of HT99C810 but they are always pulled-high in port A, port D, port E, port F and port G of HT99C811.

There are no mask option in port C of HT99C810/HT99C811.

### HT99C811 PROM programming and verification

The program memory used in the HT99C811 is arranged into a 8K×16 bits program PROM and a 1×14 bits option PROM. The program code and option code are stored in the program PROM and option PROM. The programming of PROM can be summarized in nine steps as described below:

- Power on
- Set VPP ($\overline{RES}$) to 12.5V
- Set $\overline{CS}$ (PA5) to low

Let PA3~PA0 (AD3~AD0) be the address and data bus and the PA4 (CLK) be the clock input. The data on the AD3~AD0 pins will be clocked into or out the HT99C811 on the falling edge of PA4(CLK) for PROM programming and verification.

The address data contains the code address (11 bits) and two option bits. A complete write cycle will contain 4 CLK cycles. The first cycle, bits 0~3 of the address are latched into the HT99C811. The second and third cycles, bits 4~7 and bits 8~10 are latched respectively. The fourth cycle, bit 2 is the TSEL option bit and bit 3 is the OSEL option bit. Bit 3 in the third cycle and bit 0~1 in the fourth cycle are undefined. If the TSEL is "1" and the OSEL is "0", the TEST memory will be read. If the TSEL is "0" and the OSEL is "1", the option PROM will be accessed. If both the TSEL and OSEL are "0", the program PROM will be managed.

The code data is 14 bits wide. A complete read/write cycle contains 4 CLK cycles. In the first cycle, bits 0~3 of the code data are accessed. In the second and third, bits 4~7 and bits 8~11 are accessed respectively. In the fourth cycle, bits 12~13 are accessed. Bits 14~15 are undefined. During code verification, reading will return the result "00".

Select the TSEL and OSEL to program and verify the program PROM and the option PROM. Use the R/$\overline{W}$(PA6) to select programming or verification.

The address is automatically incremented by one after a code verification cycle. If discontinued address programming or verification is carried out, the automatic addressing increment is disabled. For discontinued address programming and verification, the $\overline{CS}$ pin must return to a high level for a programming or verification cycle, i.e. if a discontinued address is implemented, the programming or verification cycle must be interrupted and restarted as well.

The related pins of PROM programming and verification are listed in the following table.

| Pin Name | Function | Description |
|---|---|---|
| PA0 | AD0 | Bit 0 of address/data bus |
| PA1 | AD1 | Bit 1 of address/data bus |
| PA2 | AD2 | Bit 2 of address/data bus |
| PA3 | AD3 | Bit 3 of address/data bus |
| PA4 | CLK | Serial clock input for address and data |
| PA5 | $\overline{CS}$ | Chip select, active low |
| PA6 | R/$\overline{W}$ | Read/write control input |
| $\overline{RES}$ | VPP | Programming power supply |

The timing charts of programming and verification are as shown. There is a LOCK signal for code protection. If the LOCK is "1", reading the code will return the result "1". However, if the LOCK is "0", the code protection is disabled and the code always can be read until the LOCK is programmed as "1".

## Instruction Set Summary

| Mnemonic | Description | Flag Affected |
|---|---|---|
| Arithmetic | | |
| ADD A,[m] | Add data memory to ACC | Z,C,AC,OV |
| ADDM A,[m] | Add ACC to data memory | Z,C,AC,OV |
| ADD A,x | Add immediate data to ACC | Z,C,AC,OV |
| ADC A,[m] | Add data memory to ACC with carry | Z,C,AC,OV |
| ADCM A,[m] | Add ACC to register with carry | Z,C,AC,OV |
| SUB A,x | Subtract immediate data from ACC | Z,C,AC,OV |
| SUB A,[m] | Subtract data memory from ACC | Z,C,AC,OV |
| SUBM A,[m] | Subtract data memory from ACC with result in data memory | Z,C,AC,OV |
| SBC A,[m] | Subtract data memory from ACC with carry | Z,C,AC,OV |
| SBCM A,[m] | Subtract data memory from ACC with carry, result in data memory | Z,C,AC,OV |
| DAA [m] | Decimal adjust ACC for addition with result in data memory | C |
| Logic Operation | | |
| AND A,[m] | AND data memory to ACC | Z |
| OR A,[m] | OR data memory to ACC | Z |
| XOR A,[m] | Exclusive-OR data memory to ACC | Z |
| ANDM A,[m] | AND ACC to data memory | Z |
| ORM A,[m] | OR ACC to data memory | Z |
| XORM A,[m] | Exclusive-OR ACC to data memory | Z |
| AND A,x | AND immediate data to ACC | Z |
| OR A,x | OR immediate data to ACC | Z |
| XOR A,x | Exclusive-OR immediate data to ACC | Z |
| CPL [m] | Complement data memory | Z |
| CPLA [m] | Complement data memory with result in ACC | Z |
| Increment and Decrement | | |
| INCA [m] | Increment data memory with result in ACC | Z |
| INC [m] | Increment data memory | Z |
| DECA [m] | Decrement data memory with result in ACC | Z |
| DEC [m] | Decrement data memory | Z |
| Rotate | | |
| RRA [m] | Rotate data memory right with result in ACC | None |
| RR [m] | Rotate data memory right | None |
| RRCA [m] | Rotate data memory right through carry with result in ACC | C |
| RRC [m] | Rotate data memory right through carry | C |
| RLA [m] | Rotate data memory left with result in ACC | None |
| RL [m] | Rotate data memory left | None |
| RLCA [m] | Rotate data memory left through carry with result in ACC | C |
| RLC [m] | Rotate data memory left through carry | C |

| Mnemonic | Description | Flag Affected |
|---|---|---|
| Data Move | | |
| MOV A,[m] | Move data memory to ACC | None |
| MOV [m],A | Move ACC to data memory | None |
| MOV A,x | Move immediate data to ACC | None |
| Bit Operation | | |
| CLR [m].i | Clear bit of data memory | None |
| SET [m].i | Set bit of data memory | None |
| Branch | | |
| JMP addr | Jump unconditional | None |
| SZ [m] | Skip if data memory is zero | None |
| SZA [m] | Skip if data memory is zero with data movement to ACC | None |
| SZ [m].i | Skip if bit i of data memory is zero | None |
| SNZ [m].i | Skip if bit i of data memory is not zero | None |
| SIZ [m] | Skip if increment data memory is zero | None |
| SDZ [m] | Skip if decrement data memory is zero | None |
| SIZA [m] | Skip if increment data memory is zero with result in ACC | None |
| SDZA [m] | Skip if decrement data memory is zero with result in ACC | None |
| CALL addr | Subroutine call | None |
| RET | Return from subroutine | None |
| RET A,x | Return from subroutine and load immediate data to ACC | None |
| RETI | Return from interrupt | None |
| Table Read | | |
| TABRDC [m] | Read ROM code (current page) to data memory and TBLH | None |
| TABRDL [m] | Read ROM code (last page) to data memory and TBLH | None |
| Miscellaneous | | |
| NOP | No operation | None |
| CLR [m] | Clear data memory | None |
| SET [m] | Set data memory | None |
| CLR WDT | Clear the watchdog timer | TO,PD |
| CLR WDT1 | Pre-clear the watchdog timer | TO*,PD* |
| CLR WDT2 | Pre-clear the watchdog timer | TO*,PD* |
| SWAP [m] | Swap nibbles of data memory | None |
| SWAPA [m] | Swap nibbles of data memory with result in ACC | None |
| HALT | Enter power down mode | TO,PD |

Notes:

x = 8 bits immediate data
m = 8 bits data memory address
A = accumulator
i = 0...7 number of bits
addr = 12 bits program memory address
√ = Flag(s) is affected
– = Flag(s) is not affected
* = Flag(s) may be affected by the execution status

## Instruction Definition

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

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

Operation ACC ← ACC+[m]+C

Affected flag(s)

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

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

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

Operation [m] ← ACC+[m]+C

Affected flag(s)

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

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

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

Operation ACC ← ACC+[m]

Affected flag(s)

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

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

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

Operation ACC ← ACC+x

Affected flag(s)

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

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

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

Operation [m] ← ACC+[m]

Affected flag(s)

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

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

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

Operation ACC ← ACC "AND" [m]

Affected flag(s)

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

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

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

Operation ACC ← ACC "AND" x

Affected flag(s)

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

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

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

Operation [m] ← ACC "AND" [m]

Affected flag(s)

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

**CALL addr** Subroutine call

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

Operation Stack ← PC+1
PC ← addr

Affected flag(s)

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

**CLR [m]** Clear data memory

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

Operation [m] ← 00H

Affected flag(s)

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

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

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

Operation [m].i ← 0

Affected flag(s)

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

**CLR WDT** Clear the watchdog timer

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

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

Affected flag(s)

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

**CLR WDT1** Preclear the watchdog timer

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

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

Affected flag(s)

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

**CLR WDT2** Preclear the watchdog timer

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

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

Affected flag(s)

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

**CPL [m]** Complement data memory

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

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

Affected flag(s)

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

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

Description Each bit of the specified data memory is logically complemented (1's complement). Bits which previously contained a one are changed to zero 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)

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

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

Description The accumulator value is adjusted to the BCD (Binary Code 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
If (ACC.7~ACC.4)+AC1 >9 or C=1
then ([m].7~[m].4) ← (ACC.7~ACC.4)+6+AC1, C=1
else ([m].7~[m].4) ← (ACC.7~ACC.4)+AC1, C=C

Affected flag(s)

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

**DEC [m]** Decrement data memory

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

Operation [m] ← [m]–1

Affected flag(s)

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

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

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

Operation ACC ← [m]–1

Affected flag(s)

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

**HALT** Enter power down mode

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

Operation PC ← PC+1
PD ← 1
TO ← 0

Affected flag(s)

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

**INC [m]** Increment data memory

Description Data in the specified data memory is incremented by one.

Operation [m] ← [m]+1

Affected flag(s)

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

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

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

Operation ACC ← [m]+1

Affected flag(s)

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

**JMP addr** Direct Jump

Description Bits 0~11 of the program counter are replaced with the directly–specified address unconditionally, and control passed to this destination.

Operation PC ← addr

Affected flag(s)

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

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

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

Operation ACC ← [m]

Affected flag(s)

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

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

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

Operation ACC ← x

Affected flag(s)

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

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

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

Operation [m] ← ACC

Affected flag(s)

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

**NOP** No operation

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

Operation PC ← PC+1

Affected flag(s)

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

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

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

Operation ACC ← ACC "OR" [m]

Affected flag(s)

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

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

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

Operation ACC ← ACC "OR" x

Affected flag(s)

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

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

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

Operation [m] ← ACC "OR" [m]

Affected flag(s)

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

**RET** Return from subroutine

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

Operation PC ← Stack

Affected flag(s)

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

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

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

Operation PC ← Stack
ACC ← x

Affected flag(s)

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

**RETI** Return from interrupt

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

Operation PC ← Stack
EMI ← 1

Affected flag(s)

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

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

Description The contents of the specified data memory is rotated left, one bit with bit 7 rotated into bit 0.

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

**SET [m]** Set data memory

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

Operation [m] ← FFH

Affected flag(s)

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

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

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

Operation [m].i ← 1

Affected flag(s)

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

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

Description The contents of the specified data memory is incremented by one. If the result is zero, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction. This is a 2-cycle instruction. Otherwise proceed to the next instruction.

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

Affected flag(s)

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

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

Description The contents of the specified data memory is incremented by one. If the result is zero, the next instruction is skipped and the result stored in the accumulator. The data memory remains unchanged. If the result is zero, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle replaced to get the proper instruction. This is a 2-cycle instruction. Otherwise proceed to the next instruction.

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

Affected flag(s)

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

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

Description If bit i of the specified data memory is not zero, the next instruction is skipped. If bit i of the data memory is not zero, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction. This is a 2-cycle instruction. Otherwise proceed to the next instruction.

Operation Skip if [m].i≠0

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

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

Affected flag(s)

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

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

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

Operation ACC ← ACC+$\overline{x}$+1

Affected flag(s)

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

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

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

Operation [m].3~[m].0 ↔ [m].7~[m].4

Affected flag(s)

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

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

Description The low-order and high-order nibbles of the specified data memory are interchanged, writing the result to the accumulator. The contents of the data memory remain unchanged.

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

Affected flag(s)

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

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

Description If the contents of the specified data memory is zero, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction. This is a 2-cycle instruction. Otherwise proceed to the next instruction.

Operation Skip if [m]=0

Affected flag(s)

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

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

Description The contents of the specified data memory is copied to accumulator. If the contents is zero, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction. This is a 2-cycle instruction. Otherwise proceed to the next instruction.

Operation Skip if [m]=0

Affected flag(s)

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

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

Description If bit i of the specified data memory is zero, the following instruction, fetched during the current instruction execution, is discarded and a dummy cycle is replaced to get the proper instruction. This is a 2-cycle instruction. Otherwise proceed to the next instruction.

Operation Skip if [m].i=0

Affected flag(s)

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

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

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

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

Affected flag(s)

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

**TABRDL [m]** Move 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)

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

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

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

Operation ACC ← ACC "XOR" [m]

Affected flag(s)

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

**XORM A,[m]** Logical XOR data memory with 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 zero flag is affected.

Operation [m] ← ACC "XOR" [m]

Affected flag(s)

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

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

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

Operation ACC ← ACC "XOR" x

Affected flag(s)

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