Cost-Effective I/O 8-Bit OTP MCU

HT48R004
HT48R008

Revision: V1.40 Date: March 31, 2017

www.holtek.com

Table of Contents

Features

CPU Features

- Operating voltage
  - f_{SYS} = 8MHz: V_{LVR}~5.5V
- Up to 0.5μs instruction cycle with 8MHz system clock at V_{DD} =5V
- Power down and wake-up functions to reduce power consumption
- Two oscillators
  - Internal high speed RC – HIRC
  - Internal low speed RC – LIRC
- Fully integrated internal 8MHz oscillator requires no external components
- All instructions executed in one or two instruction cycles
- Table read instruction
- 63 powerful instructions
- 4-level subroutine nesting
- Bit manipulation instruction

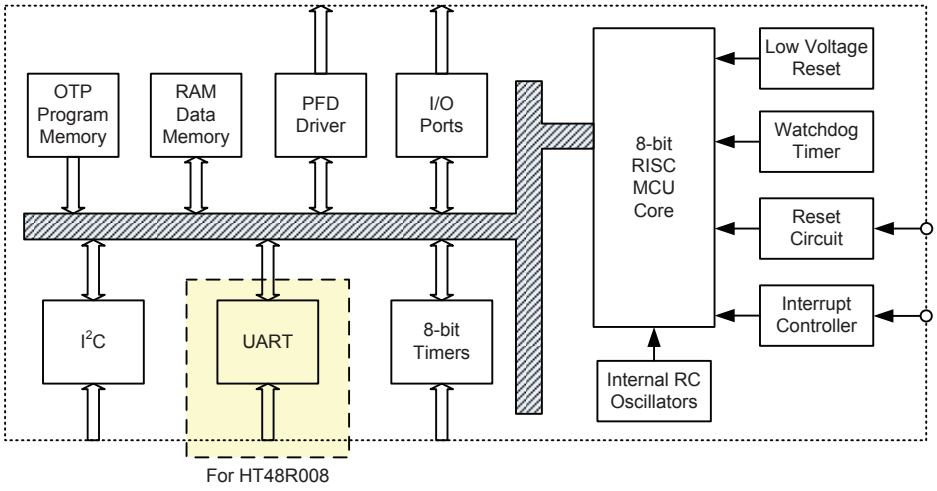
Peripheral Features

- Program Memory: 2K×14 ~ 4K×15
- Data Memory: 64×8 ~ 96×8
- Watchdog Timer function
- Up to 22 bidirectional I/O lines
- External interrupt pin shared with I/O pin
- Two 8-bit programmable Timer/Event Counters with overflow interrupt and prescaler
- Programmable Frequency Divider – PFD
- Universal Asynchronous Receiver Transmitter – UART (only for HT48R008)
- I^2C Function
- Low voltage reset function
- Wide range of available package types

General Description

The series of devices are 8-bit high performance RISC architecture microcontrollers specifically designed for the I/O control. The advantages of low power consumption, I/O flexibility, timer functions, HALT and wake-up functions, watchdog timer, as well as low cost, enhance the versatility of these devices to suit for a wide range of the I/O control application possibilities such as industrial control, consumer products and subsystem controllers, etc.

Block Diagram



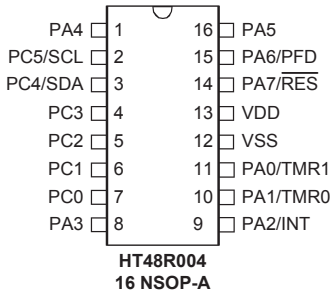
Selection Table

Most features are common to all devices, the main feature distinguishing them are Program Memory and Data Memory capacity, I/O count, UART Interface and package types. The following table summarises the main features of each device.

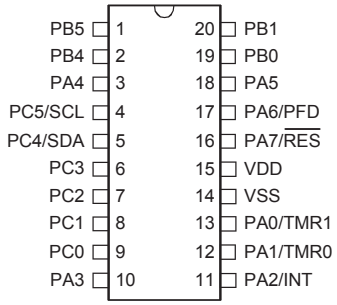
Part No.	Program Memory	Data Memory	I/O	External Interrupt	8-bit Timer	PFD	I²C	UART	Stack	Package
HT48R004	2K×14	64×8	18	1	2	√	√	–	4	16NSOP 20SOP
HT48R008	4K×15	96×8	22	1	2	√	√	√	4	24SOP

Note: As devices exist in more than one package format, the table reflects the situation for the package with the most pins.

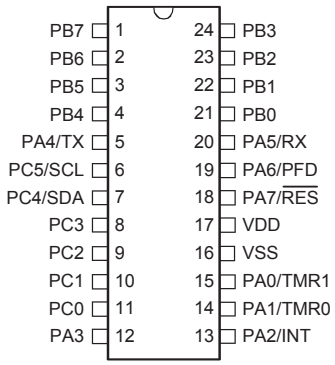
Pin Assignment



HT48R004
16 NSOP-A



HT48R004
20 SOP-A



HT48R008
24 SOP-A

Note: If the pin-shared pin functions have multiple outputs simultaneously, its pin names at the right side of the "/" sign can be used for higher priority.

Pin Description

HT48R004

Pin Name	Function	OPT	I/T	O/T	Description
PA0/TMR1	PA0	PAPU PAWU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	TMR1	TMR1C	ST	—	Timer/Event counter 1 input
PA1/TMR0	PA1	PAPU PAWU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	TMR0	TMR0C	ST	—	Timer/Event counter 0 input
PA2/INT	PA2	PAPU PAWU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	INT	INTC0 CTRL1	ST	—	External interrupt input
PA3~PA5	PA3~PA5	PAPU PAWU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
PA6/PFD	PA6	PAPU PAWU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	PFD	CTRL0	ST	—	PFD output
PA7/$\overline{\text{RES}}$	PA7	PAWU	ST	NMOS	General purpose I/O. Register enabled wake-up.
	$\overline{\text{RES}}$	EXTRESB	ST	—	Reset input
PB0, PB1, PB4, PB5	PB0, PB1, PB4, PB5	PBPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
PC0~PC3	PC0~PC3	PCPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
PC4/SDA	PC4	PCPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
	SDA	—	ST	CMOS	I^2C data line
PC5/SCL	PC5	PCPU	ST	CMOS	General purpose I/O Register enabled pull-up.
	SCL	—	ST	CMOS	I^2C clock line
VDD	VDD	—	PWR	—	Power supply
VSS	VSS	—	PWR	—	Ground

Note: OPT: Optional by register option
I/T: Input type
O/T: Output type
ST: Schmitt Trigger input
CMOS: CMOS output
NMOS: NMOS output
PWR: Power

HT48R008

Pin Name	Function	OPT	I/T	O/T	Description
PA0/TMR1	PA0	PAPU PAWU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	TMR1	TMR1C	ST	—	Timer/Event counter 1 input
PA1/TMR0	PA1	PAPU PAWU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	TMR0	TMR0C	ST	—	Timer/Event counter 0 input
PA2/INT	PA2	PAPU PAWU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	INT	INTC0 CTRL1	ST	—	External interrupt input
PA3	PA3	PAPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
PA4/TX	PA4	PAPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
	TX	UCR2	—	CMOS	UART transmit
PA5/RX	PA5	PAPU PAWU	ST	CMOS	General purpose I/O. Register enabled pull-up and wake-up.
	RX	UCR2	ST	—	UART receive
PA6/PFD	PA6	PAPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
	PFD	CTRL0	ST	—	PFD output
PA7/$\overline{\text{RES}}$	PA7	—	ST	NMOS	General purpose I/O.
	$\overline{\text{RES}}$	EXTRESB	ST	—	Reset input
PB0~PB7	PB0~PB7	PBPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
PC0~PC3	PC0~PC3	PCPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
PC4/SDA	PC4	PCPU	ST	CMOS	General purpose I/O. Register enabled pull-up.
	SDA	—	ST	CMOS	I^2C data line
PC5/SCL	PC5	PCPU	ST	CMOS	General purpose I/O Register enabled pull-up.
	SCL	—	ST	CMOS	I^2C clock line
VDD	VDD	—	PWR	—	Power supply
VSS	VSS	—	PWR	—	Ground

Note: OPT: Optional by register option
I/T: Input type
O/T: Output type
ST: Schmitt Trigger input
CMOS: CMOS output
NMOS: NMOS output
PWR: Power

Absolute Maximum Ratings

Supply Voltage ..... V_{SS} -0.3V to V_{SS} +6.0V
Input Voltage ..... V_{SS} -0.3V to V_{DD} +0.3V
Storage Temperature..... -50°C to 125°C
Operating Temperature ..... -40°C to 85°C

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

D.C. Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	Operating Voltage (HIRC)	—	f_{SYS}=8MHz	2.3	—	5.5	V
I_{DD}	Operating Current (HIRC on)	3V	No load, f_{SYS}=8MHz	—	1.2	1.8	mA
		5V		—	2.4	3.6	mA
I_{STB1}	Standby Current (LIRC on)	3V	No load, system HALT	—	—	5	μA
		5V		—	—	10	μA
I_{STB2}	Standby Current (LIRC off)	3V	No load, system HALT	—	—	1	μA
		5V		—	—	2	μA
V_{IL1}	Input Low Voltage for I/O Ports, TMRn and INT pin	5V	—	0	—	1.5	V
		—		0	—	$0.2V_{DD}$	V
V_{IH1}	Input High Voltage for I/O Ports, TMRn and INT pin	5V	—	3.5	—	5	V
		—		$0.8V_{DD}$	—	V_{DD}	V
V_{IL2}	Input Low Voltage ($\overline{RES}$)	—	—	0	—	$0.4V_{DD}$	V
V_{IH2}	Input High Voltage ($\overline{RES}$)	—	—	$0.9V_{DD}$	—	V_{DD}	V
V_{LVR}	Low Voltage Reset Voltage	—	LVR Enable, voltage select 2.1V	2.0	2.1	2.2	V
I_{OH1}	I/O Source Current (PA, PB, PC, except PA7 for HT48R004; PB, PC3~PC0 for HT48R008)	3V	V_{OH}=$0.9V_{DD}$, PXPS[n+1:n] = 00B (n=0, 2, 4)	-0.67	-1.33	—	mA
		5V		-1.34	-2.67	—	mA
		3V	V_{OH}=$0.9V_{DD}$, PXPS[n+1:n] = 01B (n=0, 2, 4)	-1	-2	—	mA
		5V		-2	-4	—	mA
		3V	V_{OH}=$0.9V_{DD}$, PXPS[n+1:n] = 10B (n=0, 2, 4)	-1.34	-2.67	—	mA
		5V		-2.65	-5.3	—	mA
		3V	V_{OH}=$0.9V_{DD}$, PXPS[n+1:n] = 11B (n=0, 2, 4)	-4	-8	—	mA
		5V		-8	-16	—	mA
I_{OH2}	I/O Source Current (PA, PC5~PC4, except PA7 for HT48R008)	3V	V_{OH}=$0.9V_{DD}$	-4	-8	—	mA
		5V		-8	-16	—	mA
I_{OL1}	I/O Sink Current (I/O Ports except PA7 pin)	3V	V_{OL}=$0.1V_{DD}$	8	16	—	mA
		5V		16	32	—	mA
I_{OL2}	PA7 Sink Current	5V	V_{OL}=$0.1V_{DD}$	2	3	—	mA
R_{PH}	Pull-high Resistance for I/O Ports	3V	—	20	60	100	kΩ
		5V	—	10	30	50	kΩ

A.C. Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
f_{SYS}	System Clock	2.3V~5.5V	—	8	8	8	MHz
f_{HIRC}	System Clock (HIRC)	3V/5V	—	-2%	8	+2%	MHz
		3V/5V	Ta = 0°C~70°C	-5%	8	+5%	MHz
		3.0V~5.5V	Ta = 0°C~70°C	-8%	8	+8%	MHz
		3.0V~5.5V	Ta = -40°C~85°C	-12%	8	+12%	MHz
f_{TIMER}	Timer I/P Frequency (TMRn)	3.3V~5.5V	—	0	—	8	MHz
t_{WDTOSC}	Watchdog oscillator period	3V	—	45	90	180	μs
		5V	—	32	65	130	μs
t_{RES}	External reset low pulse width	—	—	1	—	—	μs
t_{RESF}	External reset low pulse width (with filter)	—	—	—	150	—	ns
t_{SST}	System start-up timer period	—	Wake-up from HALT	—	16	—	t_{SYS}
t_{LVR}	Low Voltage Width to Reset	—	—	0.25	1	2	ms
t_{RSTD}	System Reset Delay Time (All Reset)	—	—	25	50	100	ms

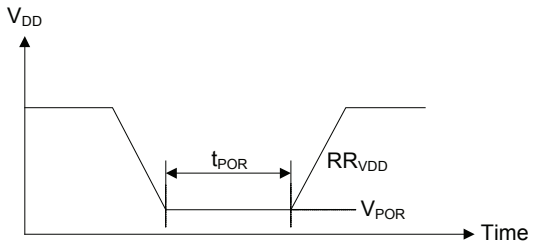
Note: 1. $t_{SYS}= 1/f_{SYS}$

2. To maintain the accuracy of the internal HIRC oscillator frequency, a 0.1μF decoupling capacitor should be connected between VDD and VSS and located as close to the devices as possible.

Power-on Reset Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{POR}	V_{DD} Start Voltage to Ensure Power-on Reset	—	—	—	—	100	mV
RR_{VDD}	V_{DD} Raising Rate to Ensure Power-on Reset	—	—	0.035	—	—	V/ms
t_{POR}	Minimum Time for V_{DD} Stays at V_{POR} to Ensure Power-on Reset	—	—	1	—	—	ms

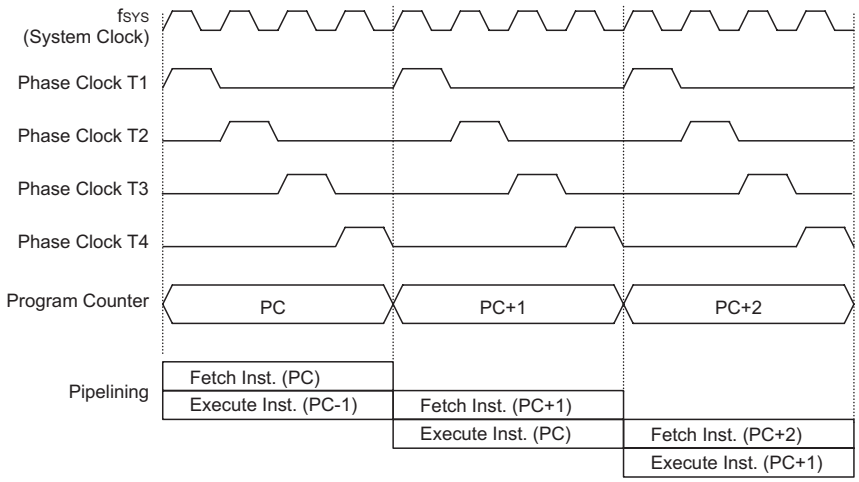


System Architecture

A key factor in the high-performance features of the Holtek range of microcontrollers is attributed to the internal system architecture. These devices take advantage of the usual features found within RISC microcontrollers providing increased speed of operation and enhanced performance. The pipelining scheme is implemented in such a way that instruction fetching and instruction execution are overlapped, hence instructions are effectively executed in one cycle, with the exception of branch or call instructions. An 8-bit wide ALU is used in practically all operations of the instruction set. It carries out arithmetic operations, logic operations, rotation, increment, decrement, branch decisions, etc. The internal data path is simplified by moving data through the Accumulator and the ALU. Certain internal registers are implemented in the Data Memory and can be directly or indirectly addressed. The simple addressing methods of these registers along with additional architectural features ensure that a minimum of external components is required to provide a functional I/O system with maximum reliability and flexibility.

Clocking and Pipelining

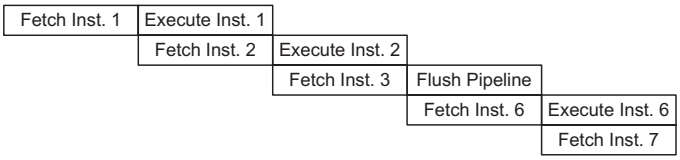
The main system clock, derived from HIRC oscillator is subdivided into four internally generated non-overlapping clocks, T1~T4.The Program Counter is incremented at the beginning of the T1 clock during which time a new instruction is fetched. The remaining T2~T4 clocks carry out the decoding and execution functions. In this way, one T1~T4 clock cycle forms one instruction cycle. Although the fetching and execution of instructions takes place in consecutive instruction cycles, the pipelining structure of the microcontroller ensures that instructions are effectively executed in one instruction cycle. The exception to this are instructions where the contents of the Program Counter are changed, such as subroutine calls or jumps, in which case the instruction will take one more instruction cycle to execute.



System Clocking and Pipelining

For instructions involving branches, such as jump or call instructions, two instruction cycles are required to complete instruction execution. An extra cycle is required as the program takes one cycle to firstly obtain the actual jump or call address and then another cycle to actually execute the branch. The requirement for this extra cycle should be taken into account by programmers in timing sensitive applications.

```
1          MOV A,[12H]
2          CALL DELAY
3          CPL [12H]
4          :
5          :
6  DELAY:  NOP
```



Instruction Fetching

Program Counter – PC

During program execution, the Program Counter is used to keep track of the address of the next instruction to be executed. It is automatically incremented by one each time an instruction is executed except for instructions, such as "JMP" or "CALL" that demand a jump to a non-consecutive Program Memory address. It must be noted that only the lower 8 bits, known as the Program Counter Low Register, are directly addressable by user.

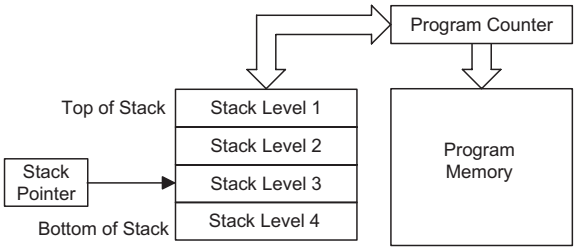
When executing instructions requiring jumping to non-consecutive addresses such as a jump instruction, a subroutine call, interrupt or reset, etc, the microcontroller manages program control by loading the required address into the Program Counter. For conditional skip instructions, once the condition has been met, the next instruction, which has already been fetched during the present instruction execution, is discarded and a dummy cycle takes its place while the correct instruction is obtained.

Device	Program Counter	
	High Byte of Program	Low Byte of Program
HT48R004	PC10~PC8	PCL7~PCL0
HT48R008	PC11~PC8	PCL7~PCL0

The lower byte of the Program Counter, known as the Program Counter Low register or PCL, is available for program control and is a readable and writeable register. By transferring data directly into this register, a short program jump can be executed directly. However, as only this low byte is available for manipulation, the jumps are limited in the present page of memory, which have 256 locations. When such program jumps are executed it should also be noted that a dummy cycle will be inserted. The lower byte of the Program Counter is fully accessible under program control. Manipulating the PCL might cause program branching, so an extra cycle is needed to pre-fetch.

Stack

This is a special part of the memory which is used to save the contents of the Program Counter only. The stack has multiple 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, and is neither readable nor writeable. At a subroutine call or interrupt acknowledge signal, the contents of the Program Counter are pushed onto the stack. At the end of a subroutine or an interrupt routine, signaled by a return instruction, RET or RETI, the Program Counter is restored to its previous value from the stack. After a device reset, the Stack Pointer will point to the top of the stack.



If the stack is full and an enabled interrupt takes place, the interrupt request flag will be recorded but the acknowledge signal 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. However, when the stack is full, a CALL subroutine instruction can still be executed which will result in a stack overflow. Precautions should be taken to avoid such cases which might cause unpredictable program branching.

Arithmetic and Logic Unit – ALU

The arithmetic-logic unit or ALU is a critical area of the microcontroller that carries out arithmetic and logic operations of the instruction set. Connected to the main microcontroller data bus, the ALU receives related instruction codes and performs the required arithmetic or logical operations after which the result will be placed in the specified register. As these ALU calculation or operations may result in carry, borrow or other status changes, the status register will be correspondingly updated to reflect these changes. The ALU supports the following functions:

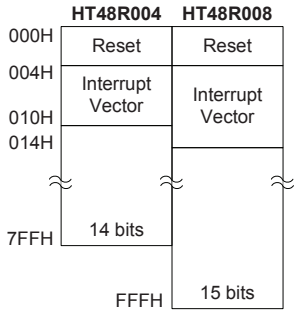
- Arithmetic operations: ADD, ADDM, ADC, ADCM, SUB, SUBM, SBC, SBCM, DAA
- Logic operations: AND, OR, XOR, ANDM, ORM, XORM, CPL, CPLA
- Rotation RRA, RR, RRCA, RRC, RLA, RL, RLCA, RLC
- Increment and Decrement INCA, INC, DECA, DEC
- Branch decision, JMP, SZ, SZA, SNZ, SIZ, SDZ, SIZA, SDZA, CALL, RET, RETI.

Program Memory

The Program Memory is the location where the user code or program is stored. The series of devices are supplied with One-Time Programmable, OTP, memory where users can program their application code into the device. By using the appropriate programming tools, OTP device offers users the flexibility to freely develop their applications which may be useful during debug or for products requiring frequent upgrades or program changes.

Structure

The Program Memory has a capacity of 2k×14 bits to 4k×15 bits. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries information. Table data which can be set in any location within the Program Memory is addressed by separate table pointer registers.



Program Memory Structure

Special Vectors

Within the Program Memory, certain locations are reserved for the reset and interrupts. The location 000H is reserved for use by these devices reset for program initialisation. After a device reset is initiated, the program will jump to this location and begin execution.

Look-up Table

Any location within the Program Memory can be defined as a look-up table where programmers can store fixed data. To use the look-up table, the table pointer must first be set by placing the address of the look up data to be retrieved in the table pointer register, TBLP. This register defines the total address of the look-up table.

After setting up the table pointer, the table data can be retrieved from the Program Memory using the "TABRDC [m]" or "TABRDL [m]" instructions, respectively. When the instruction is executed, the lower order table byte from the Program Memory will be transferred to the user defined Data Memory register [m] as specified in the instruction. The higher order table data byte from the Program Memory will be transferred to the TBLH special register. Any unused bits in this transferred higher order byte will be read as "0".

The accompanying diagram illustrates the addressing data flow of the look-up table.

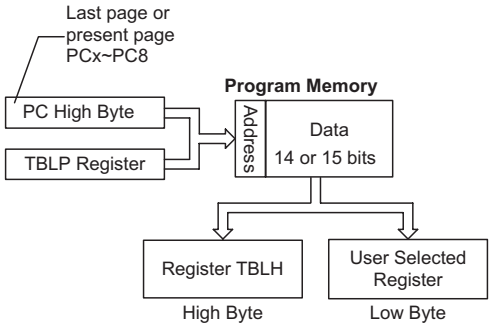


Table Program Example

The accompanying example shows how the table pointer and table data is defined and retrieved from the device. This example uses raw table data located in the last page which is stored there using the ORG statement. The value at this ORG statement is "0F00H" which refers to the start address of the last page within the 4K Program Memory of the microcontroller.

The table pointer is set here to have an initial value of "06H". This will ensure that the first data read from the data table will be at the Program Memory address "0F06H" or 6 locations after the start of the last page. Note that the value for the table pointer is referenced to the first address of the last page if the "TABRDL [m]" instruction is being used. The high byte of the table data which in this case is equal to zero will be transferred to the TBLH register automatically when the "TABRDL [m]"instruction is executed.

Because the TBLH register is a read-only register and cannot be restored, care should be taken to ensure its protection if both the main routine and Interrupt Service Routine use the table read instructions. If using the table read instructions, the Interrupt Service Routines may change the value of TBLH and subsequently cause errors if used again by the main routine. As a rule it is recommended that simultaneous use of the table read instructions should be avoided. However, in situations where simultaneous use cannot be avoided, the interrupts should be disabled prior to the execution of any main routine table-read instructions. Note that all table related instructions require two instruction cycles to complete their operation.

Table Read Program Example

```
tempreg1 db ?            ; temporary register #1
tempreg2 db ?            ; temporary register #2
     :
     :
mov a,06h                ; initialize table pointer - note that this address
                         ; is referenced
mov tblp, a              ; to the last page or present page
     :
     :
tabrdl tempreg1          ; transfers value in table referenced by table pointer
                         ; to tempreg1
                         ; data at prog. memory address "0F06H" transferred to
                         ; to tempreg1 and TBLH
dec tblp                 ; reduce value of table pointer by one
tabrdl tempreg2          ; transfers value in table referenced by table pointer
                         ; to tempreg2
                         ; data at prog. memory address "0F05H" transferred to
                         ; tempreg2 and TBLH
                         ; in this example the data "1AH" is transferred to
                         ; tempreg1 and data "0FH" to register tempreg2
                         ; the value "00H" will be transferred to the high byte
                         ; register TBLH
     :
     :
org 0f00h                ; sets initial address of last page

dc 00Ah, 00Bh, 00Ch, 00Dh, 00Eh, 00Fh, 01Ah, 01Bh
     :
     :
```

Data Memory

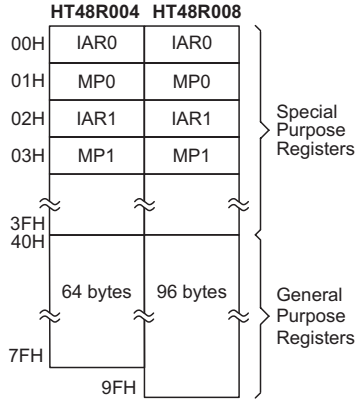
The Data Memory is a volatile area of 8-bit wide RAM internal memory and is the location where temporary information is stored.

Structure

Divided into two sections, the first of these is an area of RAM where special function registers are located. These registers have fixed locations and are necessary for correct operation of the device. Many of these registers can be read from and written to directly under program control, however, some remain protected from user manipulation. The second area of Data Memory is reserved for general purpose use. All locations within this area are read and write accessible under program control.

The two sections of Data Memory, the Special Purpose and General Purpose Data Memory are located at consecutive locations. All are implemented in RAM and are 8 bits wide. The start address of the Data Memory for all devices is the address "00H".

All microcontroller programs require an area of read/write memory where temporary data can be stored and retrieved for use later. It is this area of RAM memory that is known as General Purpose Data Memory. This area of Data Memory is fully accessible by the user program for both reading and writing operations. By using the "SET [m].i" and "CLR [m].i" instructions individual bits can be set or reset under program control giving the user a large range of flexibility for bit manipulation in the Data Memory.

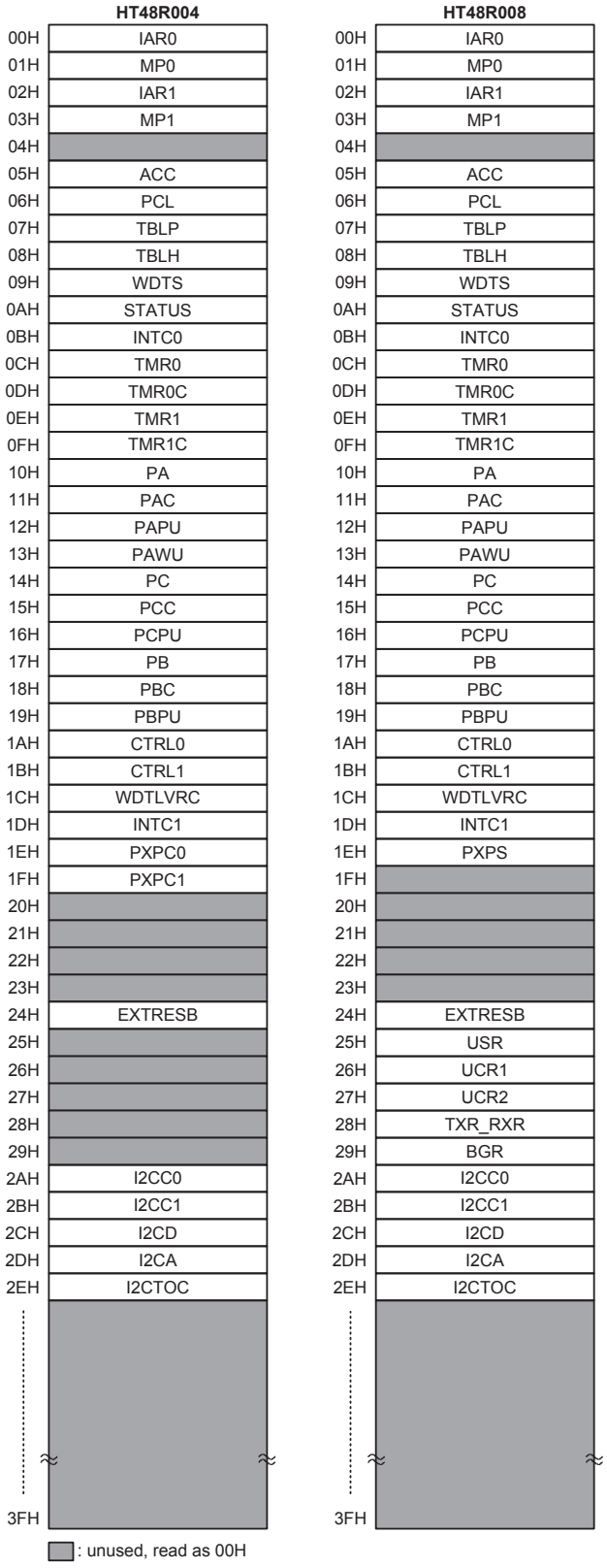


Data Memory Structure

Note: Most of the Data Memory bits can be directly manipulated using the "SET [m].i" and "CLR [m].i" with the exception of a few dedicated bits. The Data Memory can also be accessed via the memory pointer registers.

Special Purpose Data Memory

This area of Data Memory is where registers, necessary for the correct operation of the microcontroller, are stored. Most of the registers are both readable and writeable but some are protected and are readable only, the details of which are located under the relevant Special Function Register section. Note that for locations that are unused, any read instruction to these addresses will return the value "00H".



Special Purpose Data Memory

Special Function Registers

To ensure successful operation of the microcontroller, certain internal registers are implemented in the Data Memory area. These registers ensure correct operation of internal functions such as timer, interrupts, etc., as well as external functions such as I/O data control. The locations of these registers within the Data Memory begin at the address of "00H". Any unused Data Memory locations between these special function registers and the point where the General Purpose Memory begins is reserved and attempting to read data from these locations will return a value of "00H".

Indirect Addressing Registers – IAR0, IAR1

The Indirect Addressing Registers, IAR0 and IAR1, although having their locations in normal RAM register, do not actually physically exist as normal registers. The method of indirect addressing for RAM data manipulation is using these Indirect Addressing Registers and Memory Pointers, in contrast to direct memory addressing, where the actual memory address is specified. Actions on the IAR0 and IAR1 registers will result in no actual read or write operation to these registers but rather to the memory location specified by their corresponding Memory Pointers, MP0 or MP1. As the Indirect Addressing Registers are not physically implemented, reading the Indirect Addressing Registers indirectly will return a result of "00H" and writing to the registers indirectly will result in no operation.

Memory Pointers – MP0, MP1

Two Memory Pointers, known as MP0 and MP1 are provided. These Memory Pointers are physically implemented in the Data Memory and can be manipulated in the same way as normal registers providing a convenient way with which to indirectly address and track data. When any operation to the relevant Indirect Addressing Registers is carried out, the actual address which the microcontroller is directed to is the address specified by the related Memory Pointer. Note that for these devices, the Memory Pointers, MP0 and MP1, are both 8-bit registers and used to access the Data Memory together with their corresponding indirect addressing registers IAR0 and IAR1.

The following example shows how to clear a section of four Data Memory locations already defined as locations adres1 to adres4.

Indirect Addressing Program Example

```
data .section 'data'
adres1   db ?
adres2   db ?
adres3   db ?
adres4   db ?
block    db ?
code .section at 0 code
org 00h
start:
mov a,04h             ; set size of block
mov block,a
mov a,offset adres1   ; Accumulator loaded with first RAM address
mov mp0,a             ; set memory pointer with first RAM address
loop:
clr IAR0              ; clear the data at address defined by MP0
inc mp0               ; increment memory pointer
sdz block             ; check if last memory location has been cleared
jmp loop
continue:
```

The important point to note here is that in the example shown above, no reference is made to specific Data Memory addresses.

Accumulator – ACC

The Accumulator is central to the operation of any microcontroller and is closely related with operations carried out by the ALU. The Accumulator is the place where all intermediate results from the ALU are stored. Without the Accumulator it would be necessary to write the result of each calculation or logical operation such as addition, subtraction, shift, etc., to the Data Memory resulting in higher programming and timing overheads. Data transfer operations usually involve the temporary storage function of the Accumulator; for example, when transferring data between one user-defined register and another, it is necessary to do this by passing the data through the Accumulator as no direct transfer between two registers is permitted.

Program Counter Low Register – PCL

To provide additional program control functions, the low byte of the Program Counter is made accessible to programmers by locating it within the Special Purpose area of the Data Memory. By manipulating this register, direct jumps to other program locations are easily implemented. Loading a value directly into this PCL register will cause a jump to the specified Program Memory location, however as the register is only 8-bit wide only jumps within the current Program Memory page are permitted. When such operations are used, note that a dummy cycle will be inserted.

Status Register – STATUS

This 8-bit register contains the zero flag (Z), carry flag (C), auxiliary carry flag (AC), overflow flag (OV), power down flag (PDF), and watchdog time-out flag (TO). These arithmetic/logical operation and system management flags are used to record the status and operation of the microcontroller.

With the exception of the TO and PDF flags, bits in the status register can be altered by instructions like most other registers. Any data written into the status register will not change the TO or PDF flag. In addition, operations related to the status register may give different results due to the different instruction operations. The TO flag can be affected only by a system power-up, a WDT time-out or by executing the “CLR WDT” or “HALT” instruction. The PDF flag is affected only by executing the “HALT” or “CLR WDT” instruction or during a system power-up.

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

In addition, on entering an interrupt sequence or executing a subroutine call, the status register will not be pushed onto the stack automatically. If the contents of the status registers are important and if the subroutine can corrupt the status register, precautions must be taken to correctly save it. Note that bits 0~3 of the STATUS register are both readable and writeable bits.

STATUS Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	TO	PDF	OV	Z	AC	C
R/W	—	—	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	—	0	0	x	x	x	x

"x": unknown

Bit 7~6 Unimplemented, read as "0"

Bit 5 **TO**: Watchdog Time-Out flag
0: After power up or executing the "CLR WDT" or "HALT" instruction
1: A watchdog time-out occurred.

Bit 4 **PDF**: Power down flag
0: After power up or executing the "CLR WDT" instruction
1: By executing the "HALT" instruction

Bit 3 **OV**: Overflow flag
0: No overflow
1: An operation results in a carry into the highest-order bit but not a carry out of the highest-order bit or vice versa.

Bit 2 **Z**: Zero flag
0: The result of an arithmetic or logical operation is not zero
1: The result of an arithmetic or logical operation is zero

Bit 1 **AC**: Auxiliary flag
0: No auxiliary carry
1: 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

Bit 0 **C**: Carry flag
0: No carry out
1: An operation results in a carry during an addition operation or if a borrow does not take place during a subtraction operation
C is also affected by a rotate through carry instruction

System Control Registers – CTRL0, CTRL1

These registers are used to provide control internal functions such as the PFD function and external interrupt edge trigger type selection.

CTRL0 Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	PFDC	—	—
R/W	—	—	—	—	—	R/W	—	—
POR	—	—	—	—	—	0	—	—

Bit 7~3 Unimplemented, read as "0"

Bit 2 **PFDC**: PA6/PFD selection
0: PA6
1: PFD

Bit 1~0 Unimplemented, read as “0”

CTRL1 Register

Bit	7	6	5	4	3	2	1	0
Name	INTES1	INTES0	—	—	—	—	—	—
R/W	R/W	R/W	—	—	—	—	—	—
POR	1	0	—	—	—	—	—	—

Bit 7~6 **INTES1~INTES0**: External interrupt edge type selection
00: Disable
01: Rising edge trigger
10: Falling edge trigger
11: Dual edge trigger

Bit 5~0 Unimplemented, read as "0"

Oscillator

Various oscillator options offer the user a wide range of functions according to their various application requirements. The flexible features of the oscillator functions ensure that the best optimization can be achieved in terms of speed and power saving.

System Oscillator Overview

In addition to being the source of the main system clock the oscillators also provide clock sources for the Watchdog Timer function and Timer/Event counter.

Type	Name	Freq.
Internal High Speed RC	HIRC	8MHz
Internal Low Speed RC	LIRC	12kHz

Oscillator Types

System Clock Configurations

There is one system oscillator implemented in the device, internal 8MHz RC, HIRC. Also there is an internal 12kHz RC oscillator LIRC used as the clock source for the WDT function and Timer/Event counter. More details are described in the accompany sections.

Internal RC Oscillator – HIRC

The internal RC oscillator is a fully integrated system oscillator requiring no external components. The internal RC oscillator has the frequency of 8MHz .Device trimming during the manufacturing process and the inclusion of internal frequency compensation circuit is used to ensure that the influence of the power supply voltage, temperature and process variations on the oscillation frequency are minimized. Note that this internal system clock option requires no external pins for its operation. Refer to the A.C. Characteristics for more frequency accuracy details.

Internal 12kHz Oscillator – LIRC

The LIRC is a fully self-contained free running on-chip RC oscillator with a typical frequency of 12kHz at 5V, requiring no external components for its implementation. When the devices enter the Sleep Mode, the system clock will stop running but the LIRC oscillator continues to free-run and to keep the watchdog and timer active. However, to preserve power in certain applications the LIRC can be disabled by disabling the WDT function and Timer/Event counter in the HALT mode.

Power Down Mode and Wake-up

Power Down Mode

All of the Holtek microcontrollers have the ability to enter a Power Down Mode, also known as the HALT Mode or Sleep Mode. When the devices enter this mode, the normal operating current will be reduced to an extremely low standby current level. This occurs because when the devices enter the Power Down Mode, the system oscillator is stopped which reduces the power consumption to extremely low levels. However, as these devices maintain their present internal condition, they can be woken up at a later stage and continue running, without requiring a full reset. This feature is extremely important in application areas where the MCUs must have their power supply constantly maintained to keep the devices in a known condition.

Entering the Power Down Mode

There is only one way for the devices to enter the Power Down Mode and that is to execute the "HALT" instruction in the application program. When this instruction is executed, the following will occur:

- The system oscillator will stop running and the application program will stop at the "HALT" instruction.
- The Data Memory contents and registers will maintain their present condition.
- The WDT will be cleared and resume counting if the WDT clock source comes from LIRC oscillator.
- The I/O ports will maintain their present condition.
- In the status register, the Power Down flag, PDF, will be set and the Watchdog time-out flag, TO, will be cleared.

Standby Current Considerations

As the main reason for entering the Sleep Mode is to keep the current consumption of the MCU to as low a value as possible, perhaps only in the order of several micro-amps, there are other considerations which must also be taken into account by the circuit designer if the power consumption is to be minimized.

Special attention must be made to the I/O pins on these devices. All high-impedance input pins must be connected to either a fixed high or low level as any floating input pins could create internal oscillations and result in increased current consumption. Care must also be taken with the loads, which are connected to I/O pins, which are set as outputs. These should be placed in a condition in which minimum current is drawn or connected only to external circuits that do not draw current, such as other CMOS inputs.

Wake-up

After the system enters the Sleep Mode, it can be woken up from one of various sources listed as follows:

- An external reset
- An external falling edge on Port A
- A system interrupt
- A WDT overflow

If the system is woken up by an external reset, the devices will experience a full system reset, however, if the devices are woken up by a WDT overflow, a Watchdog Timer reset will be initiated. Although both of these wake-up methods will initiate a reset operation, the actual source of the wake-up can be determined by examining the TO and PDF flags. The PDF flag is cleared by a system power-up or executing the clear Watchdog Timer instructions and is set when executing the "HALT" instruction. 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 flags remain in their original status.

Pins PA7~PA0 in the HT48R004 and pins PA5, PA2~PA0 in the HT48R008 can be set via the PAWU register to permit a negative transition on the pin to wake-up the system. When a Port A pin wake-up occurs, the program will resume execution at the instruction following the "HALT" instruction.

If the system is woken up by an interrupt, then two possible situations may occur. The first is where the related interrupt is disabled or the interrupt is enabled but the stack is full, in which case the program will resume execution at the instruction following the "HALT" instruction. In this situation, the interrupt which woke-up the devices will not be immediately serviced, but will rather be serviced later when the related interrupt is finally enabled or when a stack level becomes free. The other situation is where the related interrupt is enabled and the stack is not full, in which case the regular interrupt response takes place. If an interrupt request flag is set high before entering the Sleep Mode, the wake-up function of the related interrupt will be ignored.

No matter what the source of the wake-up event is, once a wake-up event occurs, there will be a time delay before normal program execution resumes. Consult the table for the related time

Wake-up Source	Oscillator Type
	HIRC, LIRC
External $\overline{RES}$	$t_{RSTD} + t_{SST}$
PA Port	t_{SST}
Interrupt	
WDT Overflow	

Note: 1. t_{RSTD} (reset delay time), t_{SYS} (system clock)
2. t_{RSTD} is power-on delay, typical time = 50ms
3. $t_{SST} = 16t_{SYS}$
4. PA7~PA0 for HT48R004; PA5, PA2~PA0 for HT48R008

Wake-up Delay Time

Watchdog Timer

The Watchdog Timer, also known as the WDT, is provided to prevent program malfunctions or sequences from jumping to unknown locations, due to certain uncontrollable external events such as electrical noise.

Watchdog Timer Clock Source

The Watchdog Timer clock source is provided by the LIRC, the system clock f_{SYS} or $f_{SYS}/4$ which is sourced from the HIRC oscillator. The Watchdog Timer source clock is then subdivided by a ratio of 2^8 to 2^{15} to give longer timeouts, the actual value being chosen using the WS2~WS0 bits in the WDTS register. The LIRC internal oscillator has an approximate period frequency of 12kHz at a supply voltage of 5V. However, it should be noted that this specified internal clock period can vary with V_{DD}, temperature and process variations.

Watchdog Timer Control Registers

WDTS Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	WS2	WS1	WS0
R/W	—	—	—	—	—	R/W	R/W	R/W
POR	—	—	—	—	—	1	1	1

Bit 7~3 Unimplemented, read as "0"

Bit 2~0 **WS2~WS0**: WDT Time-out period selection
000: $2^8/f_S$
001: $2^9/f_S$
010: $2^{10}/f_S$
011: $2^{11}/f_S$
100: $2^{12}/f_S$
101: $2^{13}/f_S$
110: $2^{14}/f_S$
111: $2^{15}/f_S$
These three bits determine the division ratio of the Watchdog Timer source clock, which in turn determines the timeout period.

WDTLVRC Register

Bit	7	6	5	4	3	2	1	0
Name	WDTCLS1	WDTCLS0	LVREN2	LVREN1	LVREN0	WDTEN2	WDTEN1	WDTEN0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~6 **WDTCLS1~WDTCLS0**: WDT/Timer clock source
00: f_{LIRC}
01: $f_{SYS}/4$
10: f_{SYS}
11: f_{SYS}

Bit 5~3 Described in other section.

Bit 2~0 **WDTEN2~WDTEN0**: WDT enable control
000: Enable
101: Disable
Other values: MCU reset (reset will be active after 2~3 LIRC clock for debounce time)

Watchdog Timer Operation

The Watchdog Timer operates by providing a device reset when its timer overflows. This means that in the application program and during normal operation the user has to strategically clear the Watchdog Timer before it overflows to prevent the Watchdog Timer from executing a reset. This is done using the clear watchdog instruction. Note that if the Watchdog Timer function is not enabled, then any instruction related to the Watchdog Timer will result in no operation.

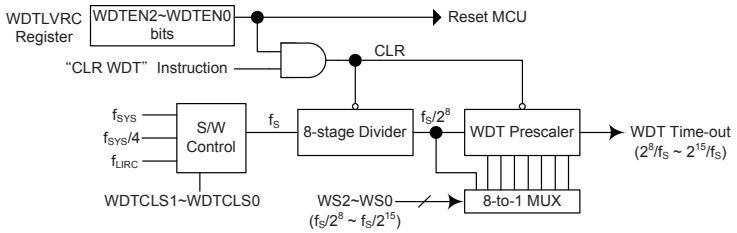
Setting the various Watchdog Timer options are controlled via the internal registers WDTLVRC and WDTS. Enabling the Watchdog Timer can be controlled by the WDTEN bits in the internal WDTLVRC register in the Data Memory.

The Watchdog Timer will be disabled if bits WDTEN2~WDTEN0 in the WDTLVRC register are written with the binary value 101B while the WDT Timer will be enabled if these bits are written with the binary value 000B. If these bits are written with the other values except 000 and 101, the MCU will be reset.

The Watchdog Timer clock can emanate from three different sources, selected by the WDTCLS1~WDTCLS0 bits in the WDTLVRC register. These sources are f_{SYS}, $f_{SYS}/4$ or LIRC. It is important to note that when the system enters the Sleep Mode the instruction clock is stopped, therefore if it has selected f_{SYS} or $f_{SYS}/4$ as the Watchdog Timer clock source, the Watchdog Timer will stop. For systems that operate in noisy environments, it's recommended to use the LIRC as the clock source. The division ratio of the prescaler is determined by bits 0, 1 and 2 of the WDTS register, known as WS0, WS1 and WS2. If the Watchdog Timer internal clock source is selected and with the WS0, WS1 and WS2 bits of the WDTS register all set high, the prescaler division ratio will be 1:32768, which will give a maximum time-out period.

Under normal program operation, a Watchdog Timer time-out will initialize a device reset and set the status bit TO. However, if the system is in the Sleep Mode, when a Watchdog Timer time-out occurs, the devices will be woken up, the TO bit in the status register will be set and only the Program Counter and Stack Pointer will be reset. Four methods can be adopted to clear the contents of the Watchdog Timer. The first is an external hardware reset, which means a low level on the external reset pin, the second is a WDT software reset, which means a certain value except 000B and 101B written into the WDTEN field, the third is using the Clear Watchdog Timer software instructions and the fourth is via a "HALT" instruction.

There is only one method of using software instruction to clear the Watchdog Timer. That is to use the "CLR WDT" instruction to clear the WDT.



Watchdog Timer

Reset and Initialization

A reset function is a fundamental part of any microcontroller ensuring that the devices can be set to some predetermined condition irrespective of outside parameters. The most important reset condition is after power is first applied to the microcontroller. In this case, internal circuitry will ensure that the microcontroller, after a short delay, will be in a well defined state and ready to execute the first program instruction. After this power-on reset, certain important internal registers will be set to defined states before the program commences. One of these registers is the Program Counter, which will be reset to zero forcing the microcontroller to begin program execution from the lowest Program Memory address.

In addition to the power-on reset, situations may arise where it is necessary to forcefully apply a reset condition when the microcontroller is running. One example of this is where after power has been applied and the microcontroller is already running, the $\overline{\text{RES}}$ line is forcefully pulled low. In such a case, known as a normal operation reset, some of the microcontroller registers remain unchanged allowing the microcontroller to deal with normal operation after the reset line is allowed to return high. Another type of reset is when the Watchdog Timer overflows and resets the microcontroller. All types of reset operations result in different register conditions being set.

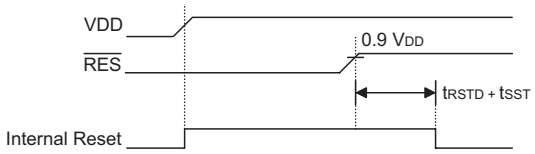
Another reset exists in the form of a Low Voltage Reset, LVR, where a full reset, similar to the $\overline{\text{RES}}$ reset is implemented in situations where the power supply voltage falls below a certain threshold.

Reset Functions

There are five ways in which a microcontroller reset can occur, through events occurring both internally and externally:

Power-on Reset

The most fundamental and unavoidable reset is the one that occurs after power is first applied to the microcontroller. As well as ensuring that the Program Memory begins execution from the first memory address, a power-on reset also ensures that certain other registers are preset to known conditions. All the I/O port and port control registers will power up in a high condition ensuring that all pins will be first set to inputs.



Note: t_{RSTD} is power-on delay, typical time=50ms

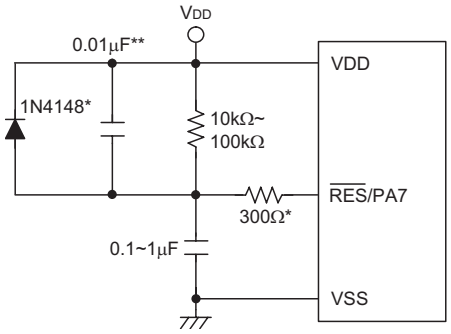
Power-On Reset Timing Chart

$\overline{\text{RES}}$ Pin Reset

Although the microcontroller has an internal RC reset function, if the VDD power supply rise time is not fast enough or does not stabilize quickly at power-on, the internal reset function may be incapable of providing proper reset operation. For this reason it is recommended that an external RC network is connected to the $\overline{\text{RES}}$ pin, whose additional time delay will ensure that the $\overline{\text{RES}}$ pin remains low for an extended period to allow the power supply to stabilize. During this time delay, normal operation of the microcontroller will be inhibited. After the $\overline{\text{RES}}$ line reaches a certain voltage value, the reset delay time t_{RSTD} is invoked to provide an extra delay time after which the microcontroller will begin normal operation. The abbreviation SST in the figures stands for System Start-up Timer.

For most applications a resistor connected between VDD and the $\overline{RES}$ pin and a capacitor connected between VSS and the $\overline{RES}$ pin will provide a suitable external reset circuit. Any wiring connected to the $\overline{RES}$ pin should be kept as short as possible to minimize any stray noise interference.

For applications that operate within an environment where more noise is present the reset circuit shown is recommended.



Note: "*" It is recommended that this component is added for added ESD protection.

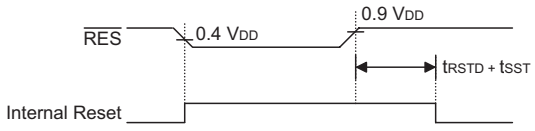
"**" It is recommended that this component is added in environments where power line noise is significant.

External $\overline{RES}$ Circuit

More information regarding external reset circuits is located in Application Note HA0075E on the Holtek website.

As the reset pin is shared with the PA7 pin, the reset function must be selected using the RESBEN2~RESBEN0 bits in the EXTRESB control register.

This type of reset occurs when the microcontroller is already running and the $\overline{RES}$ pin is forcefully pulled low by external hardware such as an external switch. In this case as in the case of other reset, the Program Counter will reset to zero and program execution initiated from this point.



Note: t_{RSTD} is power-on delay, typical time=50ms

$\overline{RES}$ Reset Timing Chart

- **EXTRESB Register**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	—	RESBEN2	RESBEN1	RESBEN0
R/W	—	—	—	—	—	R/W	R/W	R/W
POR	—	—	—	—	—	0	0	0

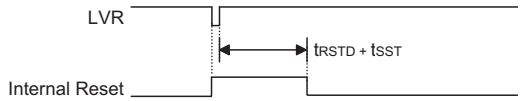
Bit 7~3 Unimplemented, read as "0"

Bit 2~0 **RESBEN2~RESBEN0**: PA7/$\overline{RES}$ selection
000: PA7
101: $\overline{RES}$
Other values: MCU reset

Low Voltage Reset – LVR

The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device. The LVR function is enabled/disabled by the LVREN2~LVREN0 bits in the WDTLVRC register. This voltage is fixed at 2.1V (V_{LVR}). If the supply voltage of the devices drops to within a range of 0.9V~V_{LVR} such as might occur when changing a battery, the LVR will automatically reset the devices internally.

The LVR includes the following specifications: For a valid LVR signal, a low voltage, i.e., a voltage in the range between 0.9V~V_{LVR} must exist for greater than the value t_{LVR} specified in the A.C. characteristics. If the low voltage state does not exceed t_{LVR}, the LVR will ignore it and will not perform a reset function.



Note: t_{RSTD} is power-on delay, typical time=50ms

Low Voltage Reset Timing Chart

- **WDTLVRC Register**

Bit	7	6	5	4	3	2	1	0
Name	WDTCLS1	WDTCLS0	LVREN2	LVREN1	LVREN0	WDTEN2	WDTEN1	WDTEN0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

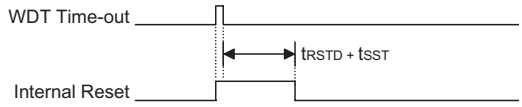
Bit 7~6 Described in other section.

Bit 5~3 **LVREN2~LVREN0**: LVR enable control
000: Enable
101: Disable
Other values: MCU reset (reset will be active after 2~3 LIRC clock for debounce time)

Bit 2~0 Described in other section.

Watchdog Time-out Reset during Normal Operation

The Watchdog time-out Reset during normal operation is the same as a hardware $\overline{RES}$ pin reset except that the Watchdog time-out flag TO will be set to "1".

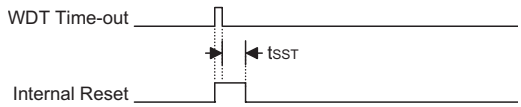


Note: t_{RSTD} is power-on delay, typical time=50ms

WDT Time-out Reset during Normal Operation Timing Chart

Watchdog Time-out Reset during Sleep Mode

The Watchdog time-out Reset during Sleep Mode is a little different from other kinds of reset. Most of the conditions remain unchanged except that the Program Counter and the Stack Pointer will be cleared to "0" and the TO flag will be set to "1". Refer to the A.C. Characteristics for t_{SST} details.



Note: t_{SST} is 16 clock cycles for the system clock source is provided by HIRC.

WDT Time-out Reset during Sleep Timing Chart

Reset Initial Conditions

The different types of reset described affect the reset flags in different ways. These flags, known as PDF and TO are located in the status register and are controlled by various microcontroller operations, such as the Sleep Mode function or Watchdog Timer. The reset flags are shown in the table:

TO	PDF	RESET Conditions
0	0	Power-on reset
u	u	$\overline{RES}$ or LVR reset during NORMAL Mode operation
1	u	WDT time-out reset during NORMAL Mode operation
1	1	WDT time-out reset during Sleep Mode operation

Note: "u" stands for unchanged

The following table indicates the way in which the various components of the microcontroller are affected after a power-on reset occurs.

Item	Condition After RESET
Program Counter	Reset to zero
Interrupts	All interrupts will be disabled
WDT	Clear after reset, WDT begins counting
Timer/Event Counter	Timer Counter will be turned off
Input/Output Ports	I/O ports will be set as inputs
Stack Pointer	Stack Pointer will point to the top of the stack

The different kinds of resets all affect the internal registers of the microcontroller in different ways. To ensure reliable continuation of normal program execution after a reset occurs, it is important to know what condition the microcontroller is in after a particular reset occurs. The following table describes how each type of reset affects the microcontroller internal registers.

HT48R004

Register	Reset (Power On)	$\overline{RES}$ Reset (Normal Operation)	$\overline{RES}$ Reset (HALT)	WDT Time-out (Normal Operation)	WDT Time-out (HALT)
PCL	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000
IAR0	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
MP0	1xxx xxxx	1uuu uuuu	1uuu uuuu	1uuu uuuu	1uuu uuuu
IAR1	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
MP1	1xxx xxxx	1uuu uuuu	1uuu uuuu	1uuu uuuu	1uuu uuuu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
PCL	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	--xx xxxx	--uu uuuu	--uu uuuu	--uu uuuu	--uu uuuu
WDTS	---- -111	---- -111	---- -111	---- -111	---- -uuu
STATUS	--00 xxxx	--uu uuuu	--01 uuuu	--1u uuuu	--11 uuuu
INTC0	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
INTC1	---0 ---0	---0 ---0	---0 ---0	---0 ---0	---u ---u
TMR0	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR0C	00-0 1000	00-0 1000	00-0 1000	00-0 1000	uu-u uu
TMR1	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR1C	00-0 1000	00-0 1000	00-0 1000	00-0 1000	uu-u uu
PA	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAWU	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PAPU	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
PB	--11 --11	--11 --11	--11 --11	--11 --11	--uu --uu
PBC	--11 --11	--11 --11	--11 --11	--11 --11	--uu --uu
PBPU	--00 --00	--00 --00	--00 --00	--00 --00	--uu --uu
PC	--11 1111	--11 1111	--11 1111	--11 1111	--uu uuuu
PCC	--11 1111	--11 1111	--11 1111	--11 1111	--uu uuuu
PCPU	--00 0000	--00 0000	--00 0000	--00 0000	--uu uuuu
CTRL0	---- -0--	---- -0--	---- -0---	---- -0--	---- -u--
CTRL1	10-- ----	10-- ----	10-- ----	10-- ----	uu-- ----
WDTLVRC	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000
EXTRESB	---- -000	---- -000	---- -000	---- -000	---- -uuu
I2CC0	---- 000-	---- 000-	---- 000-	---- 000-	---- uuu-
I2CC1	1000 0001	1000 0001	1000 0001	1000 0001	uuuu uuuu
I2CD	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
I2CA	0000 000-	0000 000-	0000 000-	0000 000-	uuuu uuu-
I2CTOC	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
PXPC0	--01 0101	--01 0101	--01 0101	--01 0101	--uu uuuu
PXPC1	---- 0101	---- 0101	---- 0101	---- 0101	---- uuuu

Note: "u" stands for unchanged
"x" stands for unknown
"-" stands for unimplemented

HT48R008

Register	Power-on Reset	$\overline{RES}$ Reset (Normal operation)	$\overline{RES}$ Reset (HALT)	WDT Time-out (Normal Operation)	WDT Time-out (HALT)
PCL	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000
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	--xx xxxx	--uu uuuu	--uu uuuu	--uu uuuu	--uu uuuu
WDTS	---- -111	---- -111	---- -111	---- -111	---- -uuu
STATUS	--00 xxxx	--uu uuuu	--01 uuuu	--1u uuuu	--11 uuuu
INTC0	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
INTC1	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu	uuuu uuuu
TMR0	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR0C	00-0 1000	00-0 1000	00-0 1000	00-0 1000	uu-u uu
TMR1	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR1C	00-0 1000	00-0 1000	00-0 1000	00-0 1000	uu-u uu
PA	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAWU	--0- -000	--0- -000	--0- -000	--0- -000	--u- -uuu
PAPU	-000 0000	-000 0000	-000 0000	-000 0000	-uuu uuuu
PB	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBC	1111 1111	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBPU	0000 0-00	0000 0-00	0000 0-00	0000 0-00	uuuu u-uu
PC	--11 1111	--11 1111	--11 1111	--11 1111	--uu uuuu
PCC	--11 1111	--11 1111	--11 1111	--11 1111	--uu uuuu
PCPU	--00 0000	--00 0000	--00 0000	--00 0000	--uu uuuu
CTRL0	---- -0--	---- -0--	---- -0---	---- -0--	---- -u--
CTRL1	10-- ----	10-- ----	10-- ----	10-- ----	uu-- ----
WDTLVRC	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000
PXPS	--01 0101	--01 0101	--01 0101	--01 0101	--uu uuuu
USR	0000 1011	0000 1011	0000 1011	0000 1011	uuuu uuuu
UCR1	0000 00x0	0000 00x0	0000 00x0	0000 00x0	uuuu uuuu
UCR2	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu
TXR_RXR	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
BRG	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
EXTRESB	---- -000	---- -000	---- -000	---- -000	---- -uuu
I2CC0	---- 000-	---- 000-	---- 000-	---- 000-	---- uuu-
I2CC1	1000 0001	1000 0001	1000 0001	1000 0001	uuuu uuuu
I2CD	xxxx xxxx	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
I2CA	0000 000-	0000 000-	0000 000-	0000 000-	uuuu uuu-
I2CTOC	0000 0000	0000 0000	0000 0000	0000 0000	uuuu uuuu

Note: “u” stands for unchanged
“x” stands for unknown
“-” stands for unimplemented

Input/Output Ports

Holtek microcontrollers offer considerable flexibility on their I/O ports. Most pins can have either an input or output designation under user program control. Additionally, as there are pull-high resistors and wake-up software configurations, the user is provided with an I/O structure to meet the needs of a wide range of application possibilities.

These devices provide bidirectional input/output lines labeled with port names PA~PC. These I/O ports are mapped to the RAM Data Memory with specific addresses as shown in the Special Purpose Data Memory table. All of these I/O ports can be used for input and output operations. For input operation, these ports are non-latching, which means the inputs must be ready at the T2 rising edge of instruction "MOV A, [m]", where m denotes the port address. For output operation, all the data is latched and remains unchanged until the output latch is rewritten.

Register Name	Bit							
	7	6	5	4	3	2	1	0
PA	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
PAC	PAC7	PAC6	PAC5	PAC4	PAC3	PAC2	PAC1	PAC0
PAPU	—	PAPU6	PAPU5	PAPU4	PAPU3	PAPU2	PAPU1	PAPU0
PAWU	PAWU7	PAWU6	PAWU5	PAWU4	PAWU3	PAWU2	PAWU1	PAWU0
PB	—	—	PB5	PB4	—	—	PB1	PB0
PBC	—	—	PBC5	PBC4	—	—	PBC1	PBC0
PBPU	—	—	PBPU5	PBPU4	—	—	PBPU1	PBPU0
PC	—	—	PC5	PC4	PC3	PC2	PC1	PC0
PCC	—	—	PCC5	PCC4	PCC3	PCC2	PCC1	PCC0
PCPU	—	—	PCPU5	PCPU4	PCPU3	PCPU2	PCPU1	PCPU0

I/O Register List – HT48R004

Register Name	Bit							
	7	6	5	4	3	2	1	0
PA	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
PAC	PAC7	PAC6	PAC5	PAC4	PAC3	PAC2	PAC1	PAC0
PAPU	—	PAPU6	PAPU5	PAPU4	PAPU3	PAPU2	PAPU1	PAPU0
PAWU	—	—	PAWU5	—	—	PAWU2	PAWU1	PAWU0
PB	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
PBC	PBC7	PBC6	PBC5	PBC4	PBC3	PBC2	PBC1	PBC0
PBPU	PBPU7	PBPU6	PBPU5	PBPU4	PBPU3	PBPU2	PBPU1	PBPU0
PC	—	—	PC5	PC4	PC3	PC2	PC1	PC0
PCC	—	—	PCC5	PCC4	PCC3	PCC2	PCC1	PCC0
PCPU	—	—	PCPU5	PCPU4	PCPU3	PCPU2	PCPU1	PCPU0

I/O Register List – HT48R008

Pull-high Resistors

Many product applications require pull-high resistors for their switch inputs usually requiring the use of an external resistor. To eliminate the need for these external resistors, all I/O pins, when configured as an input have the capability of being connected to an internal pull-high resistor. These pull-high resistors are selected using the registers PAPU~PCPU located in the Data Memory. The pull-high resistors are implemented using weak PMOS transistors. Note that pin PA7 does not have a pull-high resistor selection.

PAPU Register

Bit	7	6	5	4	3	2	1	0
Name	—	PAPU6	PAPU5	PAPU4	PAPU3	PAPU2	PAPU1	PAPU0
R/W	—	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	0	0	0	0	0	0	0

Bit 7 Unimplemented, read as "0"

Bit 6~0 **PAPU6~PAPU0**: Port A bit 6~bit 0 pull-high control
0: Disable
1: Enable

PBPU Register

- **HT48R004**

Bit	7	6	5	4	3	2	1	0
Name	—	—	PBPU5	PBPU4	—	—	PBPU1	PBPU0
R/W	—	—	R/W	R/W	—	—	R/W	R/W
POR	—	—	0	0	—	—	0	0

Bit 7~6 Unimplemented, read as "0"

Bit 5~4 **PBPU5~PBPU4**: Port B bit 5~bit 4 pull-high control
0: Disable
1: Enable

Bit 3~2 Unimplemented, read as “0”

Bit 1~0 **PBPU1~PBPU0**: Port B bit 1~bit 0 pull-high control
0: Disable
1: Enable

- **HT48R008**

Bit	7	6	5	4	3	2	1	0
Name	PBPU7	PBPU6	PBPU5	PBPU4	PBPU3	PBPU2	PBPU1	PBPU0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **PBPU7~PBPU0**: Port B bit 7~bit 0 pull-high control
0: Disable
1: Enable

PCPU Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	PCPU5	PCPU4	PCPU3	PCPU2	PCPU1	PCPU0
R/W	—	—	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	—	0	0	0	0	0	0

Bit 7~6 Unimplemented, read as "0"

Bit 5~0 **PCPU5~PCPU0**: Port C bit 5~bit 0 pull-high control
0: Disable
1: Enable

Port A Wake-up

If the HALT instruction is executed, the devices will enter the Sleep Mode, where the system clock will stop resulting in power being conserved, a feature that is important for battery and other low-power applications. Various methods exist to wake-up the microcontroller, one of which is to change the logic condition on one of the Port A pins from high to low. After a HALT instruction forces the microcontroller into entering the Sleep Mode, the processor will remain in a low-power state until the logic condition of the selected wake-up pin on Port A changes from high to low. This function is especially suitable for applications that can be woken up via external switches. Note that each of pins PA7~PA0 in the HT48R004 and pins PA5, PA2~PA0 in the HT48R008 can be selected individually to have this wake-up feature using an internal register known as PAWU, located in the Data Memory.

PAWU Register

- **HT48R004**

Bit	7	6	5	4	3	2	1	0
Name	PAWU7	PAWU6	PAWU5	PAWU4	PAWU3	PAWU2	PAWU1	PAWU0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7~0 **PAWU7~PAWU0**: Port A bit 7~bit 0 wake-up control
0: Disable
1: Enable

- **HT48R008**

Bit	7	6	5	4	3	2	1	0
Name	—	—	PAWU5	—	—	PAWU2	PAWU1	PAWU0
R/W	—	—	R/W	—	—	R/W	R/W	R/W
POR	—	—	0	—	—	0	0	0

Bit 7~6 Unimplemented, read as "0"

Bit 5 **PAWU5**: Port A bit 5 wake-up control
0: Disable
1: Enable

Bit 4~3 Unimplemented, read as “0”

Bit 2~0 **PAWU2~PAWU0**: Port A bit 2~bit 0 wake-up control
0: Disable
1: Enable

I/O Port Control Registers

Each port has its own control register known as PAC~PCC, which control the input/output configuration. With this control register, each I/O pin with or without pull-high resistors can be reconfigured dynamically under software control. For the I/O pin to function as an input, the corresponding bit of the control register must be written as a "1". This will then allow the logic state of the input pin to be directly read by instructions. When the corresponding bit of the control register is written as a "0", the I/O pin will be set as a CMOS output. If the pin is currently set as an output, instructions can still be used to read the output register. However, it should be noted that the program will in fact only read the status of the output data latch and not the actual logic status of the output pin.

PAC Register

Bit	7	6	5	4	3	2	1	0
Name	PAC7	PAC6	PAC5	PAC4	PAC3	PAC2	PAC1	PAC0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	1	1	1	1	1	1	1

Bit 7~0 **PAC7~PAC0**: Port A bit 7~bit 0 Input/Output control
0: Output
1: Input

PBC Register

- **HT48R004**

Bit	7	6	5	4	3	2	1	0
Name	—	—	PBC5	PBC4	—	—	PBC1	PBC0
R/W	—	—	R/W	R/W	—	—	R/W	R/W
POR	—	—	1	1	—	—	1	1

Bit 7~6 Unimplemented, read as "0"

Bit 5~4 **PBC5~PBC4**: Port B bit 5~bit 4 Input/Output control
0: Output
1: Input

Bit 3~2 Unimplemented, read as "0"

Bit 1~0 **PBC1~PBC0**: Port B bit 1~bit 0 Input/Output control
0: Output
1: Input

- **HT48R008**

Bit	7	6	5	4	3	2	1	0
Name	PBC7	PBC6	PBC5	PBC4	PBC3	PBC2	PBC1	PBC0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	1	1	1	1	1	1	1	1

Bit 7~0 **PBC7~PBC0**: Port B bit 7~bit 0 Input/Output control
0: Output
1: Input

PCC Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	PCC5	PCC4	PCC3	PCC2	PCC1	PCC0
R/W	—	—	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	—	1	1	1	1	1	1

Bit 7~6 Unimplemented, read as "0"

Bit 5~0 **PCC5~PCC0**: Port C bit 5~ bit 0 Input/Output control
0: Output
1: Input

Source Current Selection Registers

To enhance the I/O driving ability, PA0~PA6, PB0~PB1, PB4~PB5, PC0~PC5 pins in the HT48R004 and PB0~PB7, PC0~PC3 pins in the HT48R008 can be setup to have a choice of various source current using specific registers, which are the PXPC0, PXPC1 registers for the HT48R004 and the PXPS register for the HT48R008.

PXPC0 Register — HT48R004

Bit	7	6	5	4	3	2	1	0
Name	—	—	PBPS1	PBPS0	PAPS3	PAPS2	PAPS1	PAPS0
R/W	—	—	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	—	0	1	0	1	0	1

Bit 7~6 Unimplemented, read as "0"

Bit 5~4 **PBPS1~PBPS0**: PB5~PB4, PB1~PB0 source current select
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3(max.)

Bit 3~2 **PAPS3~PAPS2**: PA6~PA4 source current select
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3(max.)

Bit 1~0 **PAPS1~PAPS0**: PA3~PA0 source current select
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3(max.)

Note: Users should refer to the D.C. Characteristirs section to obtain the exact value for different applications.

PXPC1 Register — HT48R004

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	PCPS3	PCPS2	PCPS1	PCPS0
R/W	—	—	—	—	R/W	R/W	R/W	R/W
POR	—	—	—	—	0	1	0	1

Bit 7~4 Unimplemented, read as "0"

Bit 3~2 **PCPS3~PCPS2**: PC5~PC4 source current select
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3(max.)

Bit 1~0 **PCPS1~PCPS0**: PC3~PC0 source current select
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3(max.)

Note: Users should refer to the D.C. Characteristirs section to obtain the exact value for different applications.

PXPS Register — HT48R008

Bit	7	6	5	4	3	2	1	0
Name	—	—	PXPS5	PXPS4	PXPS3	PXPS2	PXPS1	PXPS0
R/W	—	—	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	—	0	1	0	1	0	1

Bit 7~6 Unimplemented, read as "0"

Bit 5~4 **PXPS5~PXPS4**: PC3~PC0 source current selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3(max.)

Bit 3~2 **PXPS3~PXPS2**: PB7~PB4 source current selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3(max.)

Bit 1~0 **PXPS1~PXPS0**: PB3~PB0 source current selection
00: Source current = Level 0 (min.)
01: Source current = Level 1
10: Source current = Level 2
11: Source current = Level 3(max.)

Note: Users should refer to the D.C. Characteristirs section to obtain the exact value for different applications.

Pin-shared Functions

The flexibility of the microcontroller range is greatly enhanced by the use of pins that have more than one function. Limited numbers of pins can force serious design constraints on designers but by supplying pins with multi-functions, many of these difficulties can be overcome. For some pins, the chosen function of the multi-function I/O pins is set by application program control.

External Interrupt Input

The external interrupt pin, INT, is pin-shared with an I/O pin. To use the pin as an external interrupt input the correct bits in the INTC0 register must be programmed. The pin must also be set as an input by setting the corresponding bit in the Port Control Register. A pull-high resistor can also be selected via the appropriate port pull-high resistor register. Note that even if the pin is set as an external interrupt input the I/O function still remains.

External Timer/Event Counter Input

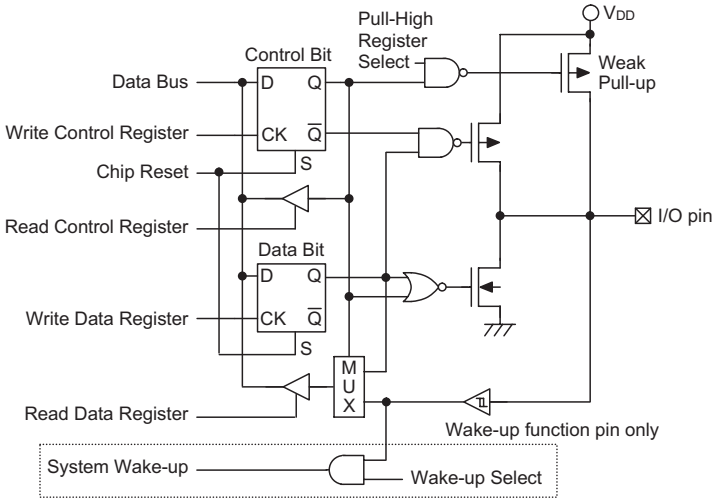
The Timer/Event Counter pins are pin-shared with I/O pins. For these shared pins to be used as Timer/Event Counter input, the Timer/Event Counter must be configured to be in the Event Counters or Pulse Width Capture Mode. This is achieved by setting the appropriate bits in the Timer/Event Counter Control Register. The pin must also be set as input by setting the appropriate bit in the Port Control Register. Pull-high resistor options can also be selected using the port pull-high resistor registers. Note that even if the pin is set as an external timer input the I/O function still remains.

PFD Output

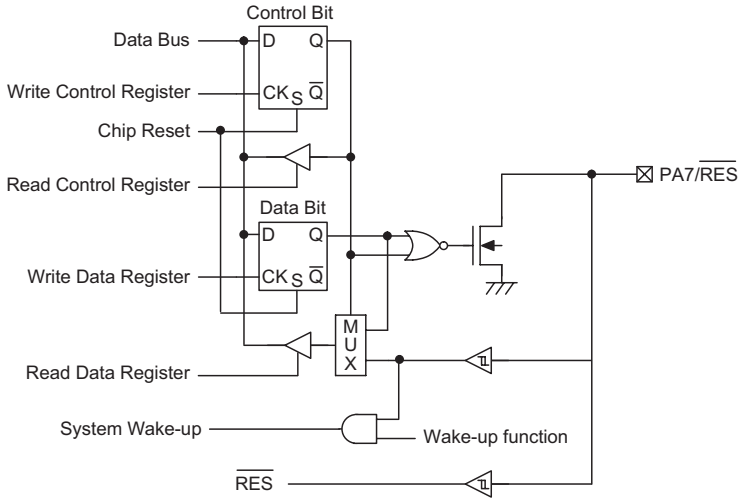
The PFD function output is pin-shared with an I/O pin. The output function of this pin is chosen using the CTRL0 register. Note that the corresponding bit of the port control register must be set the pin as an output to enable the PFD output. If the port control register has set the pin as an input, then the pin will function as a normal logic input with the usual pull-high selection, even if the PFD function has been selected.

I/O Pin Structures

The accompanying diagrams illustrate the I/O pin internal structures. As the exact logical construction of the I/O pin may differ from these drawings, they are supplied as a guide only to assist with the functional understanding of the I/O pins.



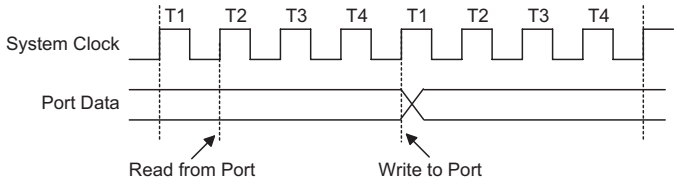
Generic Input/Output Ports



PA7 NMOS Input/Output Port

Programming Considerations

Within the user program, one of the things first to consider is port initialization. After a reset, all of the I/O data and port control registers will be set to high. This means that all I/O pins will be defaulted to an input state, the level of which depends on the other connected circuitry and whether pull-high selections have been chosen. If the port control registers are then programmed to set some pins as outputs, these output pins will have an initial high output value unless the associated port data registers are first programmed. Selecting which pins are inputs and which are outputs can be achieved byte-wide by loading the correct values into the appropriate port control register or by programming individual bits in the port control register using the "SET [m].i" and "CLR [m].i" instructions. Note that when using these bit control instructions, a read-modify-write operation takes place. The microcontroller must first read in the data on the entire port, modify it to the required new bit values and then rewrite this data back to the output ports.



Read Modify Write Timing

Each of pins PA7~PA0 in the HT48R004 and pins PA5, PA2~PA0 in the HT48R008 has wake-up function, selected via the PAWU register. When the devices are in the Sleep Mode, various methods are available to wake these devices up. One of these is a high to low transition of any pins. Single or multiple pins on Port A can be set to have this function.

Timer/Event Counters

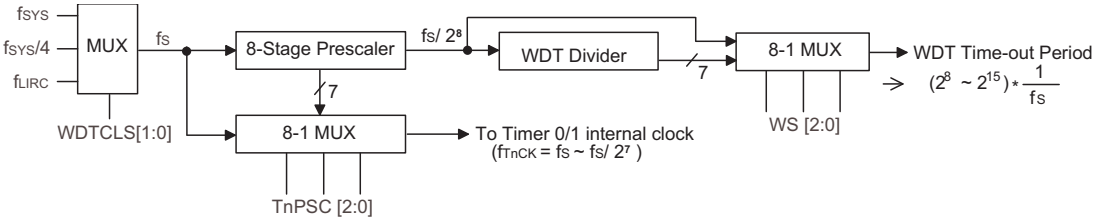
The provision of timers form an important part of any microcontroller, giving the designer a means of carrying out time related functions. The series of devices contain two 8-bit count-up timers. As the timers have three different operating modes, they can be configured to operate as a general timer, an external event counter or as a pulse width capture device. The provision of an internal prescaler to the clock circuitry on gives added range to the timers.

There are two types of registers related to the Timer/Event Counters. The first is the registers that contain the actual value of the timer and into which an initial value can be preloaded, TMR0 and TMR1. Reading from these registers retrieves the contents of the Timer/Event Counter. The second type of associated registers is the Timer Control Register which defines the timer options and determines how the timer is to be used. The devices can have the timer clock configured to come from the internal clock source. In addition, the timer clock source can also be configured to come from an external timer pin.

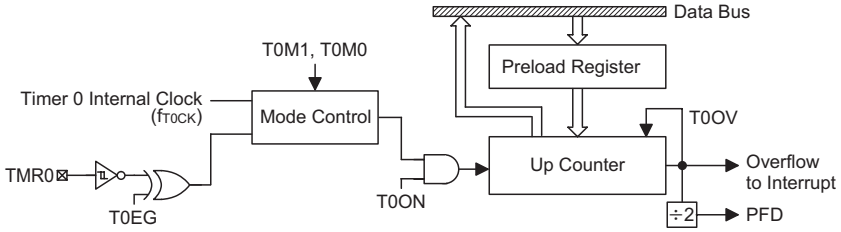
Configuring the Timer/Event Counter Input Clock Source

The Timer/Event Counter clock source can originate from various sources, an internal clock or an external pin. The internal clock source is used when the timer is in the timer mode. For the Timer/Event Counter 0/1, this internal clock source is first divided by a prescaler, the division ratio of which is conditioned by the Timer Control Register bits TnPSC2~TnPSC0. The internal clock source can be derived from the system clock f_{SYS} or from the instruction clock $f_{SYS}/4$ or the internal low speed oscillator LIRC for Timer/Event Counter selected by the clock selection bits WDTCLS1~WDTCLS0 in the register WDTLVRC.

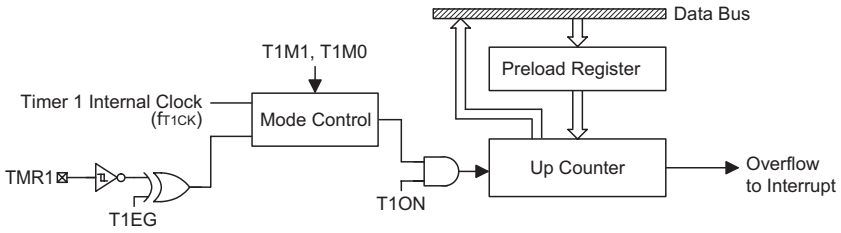
An external clock source is used when the Timer/Event Counter is in the event counting mode, the clock source being provided on an external timer pin. Depending upon the condition of the TnEG bit, each high to low, or low to high transition on the external timer pin will increment the counter by one.



Clock Source for Timer/WDT



8-bit Timer/Event Counter 0 Structure



8-bit Timer/Event Counter 1 Structure

Timer Register – TMR0, TMR1

The timer registers are special function registers located in the Special Purpose Data Memory and is the place where the actual timer value is stored. The register is known as TMR0 and TMR1. The value in the timer register increases by one each time an internal clock pulse is received or an external transition occurs on the external timer pin. The timer will count from the initial value loaded by the preload register to the full count of FFH at which point the timer overflows and an internal interrupt signal is generated. The timer value will then reset with the initial preload register value and continue counting.

Note that to achieve a maximum full range count of FFH, the preload register must first be cleared. It should be noted that after power-on, the preload register will be in an unknown condition. Note that if the Timer/Event Counter is in an OFF condition and data is written to its preload register, this data will be immediately written into the actual counter. However, if the counter is enabled and counting, any new data written into the preload data register during this period will remain in the preload register and will only be written into the actual counter the next time an overflow occurs.

Timer Control Register – TMR0C, TMR1C

The flexible features of the Holtek microcontroller Timer/Event Counters enable them to operate in three different modes, the options of which are determined by the contents of their respective control register.

The Timer Control Register is known as TMRnC. It is the Timer Control Register together with its corresponding timer register that controls the full operation of the Timer/Event Counter. Before the timer can be used, it is essential that the Timer Control Register is fully programmed with the right data to ensure its correct operation, a process that is normally carried out during program initialization.

To choose which of the three modes the timer is to operate in, either in the timer mode, the event counting mode or the pulse width capture mode, bits 7 and 6 of the Timer Control Register, which are known as the bit pair TnM1/TnM0, must be set to the required logic levels. The timer-on bit, which is bit 4 of the Timer Control Register and known as TnON, provides the basic on/off control of the respective timer. Setting the bit to high allows the counter to run. Clearing the bit stops the counter. Bits 0~2 of the Timer Control Register determine the division ratio of the input clock prescaler. The prescaler bit settings have no effect if an external clock source is used. If the timer is in the event count or pulse width capture mode, the active transition edge level type is selected by the logic level of bit 3 of the Timer Control Register which is known as TnEG.

TMR0C Register

Bit	7	6	5	4	3	2	1	0
Name	T0M1	T0M0	—	T0ON	T0EG	T0PSC2	T0PSC1	T0PSC0
R/W	R/W	R/W	—	R/W	R/W	R/W	R/W	R/W
POR	0	0	—	0	1	0	0	0

Bit 7~6 **T0M1~T0M0**: Timer operation mode selection
00: No mode available
01: Event counter mode
10: Timer mode
11: Pulse width capture mode

Bit 5 Unimplemented, read as "0"

Bit 4 **T0ON**: Timer/event counter counting enable
0: Disable
1: Enable

Bit 3 **T0EG**: Timer/Event Counter active edge selection
In event counter mode (T0M1~T0M0 = 01)
0: Count on rising edge
1: Count on falling edge
In pulse width measurement mode (T0M1~T0M0 = 11)
0: Start counting on falling edge, stop on the rising edge
1: Start counting on rising edge, stop on the falling edge

Bit 2~0 **T0PSC2~T0PSC0**: Timer prescalar rate selection
000: f_S
001: $f_S/2$
010: $f_S/4$
011: $f_S/8$
100: $f_S/16$
101: $f_S/32$
110: $f_S/64$
111: $f_S/128$

TMR1C Register

Bit	7	6	5	4	3	2	1	0
Name	T1M1	T1M0	—	T1ON	T1EG	T1PSC2	T1PSC1	T1PSC0
R/W	R/W	R/W	—	R/W	R/W	R/W	R/W	R/W
POR	0	0	—	0	1	0	0	0

Bit 7~6 **T1M1~T1M0**: Timer operation mode selection
00: No mode available
01: Event counter mode
10: Timer mode
11: Pulse width capture mode

Bit 5 Unimplemented, read as "0"

Bit 4 **T1ON**: Timer/event counter counting enable
0: Disable
1: Enable

Bit 3 **T1EG**: Timer/Event Counter active edge selection
In event counter mode (T1M1~T1M0 = 01)
0: Count on rising edge
1: Count on falling edge
In pulse width measurement mode (T1M1~T1M0 = 11)
0: Start counting on falling edge, stop on the rising edge
1: Start counting on rising edge, stop on the falling edge

Bit 2~0 **T1PSC2~T1PSC0**: Timer prescalar rate selection
000: f_S
001: $f_S/2$
010: $f_S/4$
011: $f_S/8$
100: $f_S/16$
101: $f_S/32$
110: $f_S/64$
111: $f_S/128$

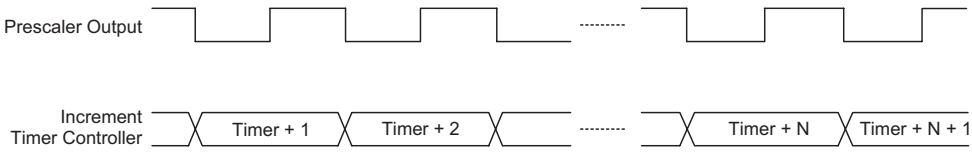
Timer Mode

In this mode, the Timer/Event Counter can be utilized to measure fixed time intervals, providing an internal interrupt signal each time the Timer/Event Counter overflows. To operate in this mode, the Operating Mode Select bit pair, TnM1/TnM0, in the Timer Control Register must be set to the correct value as shown.

Bit7	Bit6
1	0

Control Register Operating Mode Select Bits for the Timer Mode

In this mode the internal clock is used as the timer clock. The timer input clock source is f_{SYS} or $f_{SYS}/4$. However, this timer clock source is further divided by a prescaler, the value of which is determined by the bits TnPSC2~TnPSC0 in the Timer Control Register. The timer-on bit, TnON must be set high to enable the timer to run. Each time an internal clock high to low transition occurs, the timer increments by one. When the timer is full and overflows, an interrupt signal is generated and the timer will reload the value already loaded into the preload register and continue counting. A timer overflow condition and corresponding internal interrupts are two of the wake-up sources. However, the internal interrupts can be disabled by ensuring that the TnE bits of the INTC0 register are reset to zero.



Timer Mode Timing Chart

Event Counter Mode

In this mode, a number of externally changing logic events, occurring on the external timer TMRn pin, can be recorded by the Timer/Event Counter. To operate in this mode, the Operating Mode Select bit pair, TnM1/TnM0, in the Timer Control Register must be set to the correct value as shown.

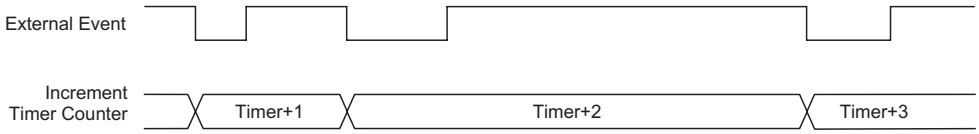
Bit7	Bit6
0	1

Control Register Operating Mode Select Bits for the Timer Mode

In this mode, the external timer TMRn pin is used as the Timer/Event Counter clock source, however it is not divided by the internal prescaler. After the other bits in the Timer Control Register have been set, the enable bit TnON, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter to run. If the Active Edge Select bit, TnEG, which is bit 3 of the Timer Control Register, is low, the Timer/Event Counter will increment each time the external timer pin receives a low to high transition. If the TnEG is high, the counter will increment each time the external timer pin receives a high to low transition. When it is full and overflows, an interrupt signal is generated and the Timer/Event Counter will reload the value already loaded into the preload

register and continue counting. The interrupt can be disabled by ensuring that the Timer/Event Counter Interrupt Enable bit in the corresponding Interrupt Control Register. It is reset to zero.

As the external timer pin is shared with an I/O pin, to ensure that the pin is configured to operate as an event counter input pin, two things have to happen. The first is to ensure that the Operating Mode Select bits in the Timer Control Register place the Timer/Event Counter in the Event Counting Mode. The second is to ensure that the port control register configures the pin as an input. It should be noted that in the event counting mode, even if the microcontroller is in the Sleep Mode, the Timer/Event Counter will continue to record externally changing logic events on the timer input TMRn pin. As a result when the timer overflows it will generate a timer interrupt and corresponding wake-up source.



Event Counter Mode Timing Chart (TnEG=1)

Pulse Width Capture Mode

In this mode, the Timer/Event Counter can be utilised to measure the width of external pulses applied to the external timer pin. To operate in this mode, the Operating Mode Select bit pair, TnM1/TnM0, in the Timer Control Register must be set to the correct value as shown.

Bit7	Bit6
1	1

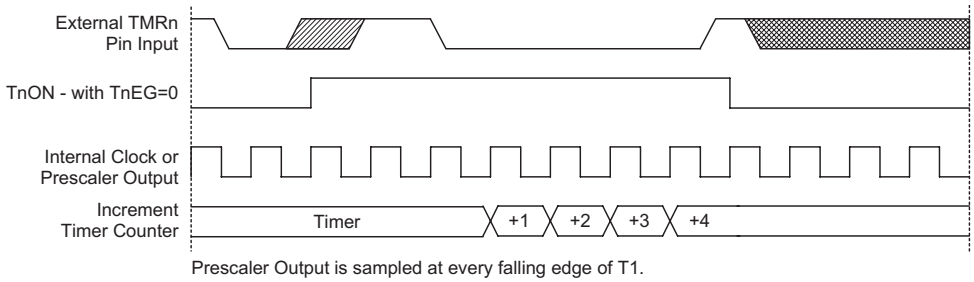
Control Register Operating Mode Select Bits for the Pulse Width Capture Mode

In this mode the internal clock, f_{SYS}, $f_{SYS}/4$ or f_{LIRC} is used as the internal clock for the 8-bit Timer/Event Counter. However, the clock source, f_S, for the 8-bit timer is further divided by a prescaler, the value of which is determined by the Prescaler Rate Select bits TnPSC2~TnPSC0, which are bit 2~0 of the Timer Control Register, After other bits in the Timer Control Register have been set, the enable bit TnON, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter, however it will not actually start counting until an active edge is received on the external timer pin.

If the Active Edge Select bit TnEG which is bit 3 of the Timer Control Register is low, once a high to low transition has been received on the external timer pin, the Timer/Event Counter will start counting until the external timer pin returns to its original high level. At this point the enable bit will be automatically reset to zero and the Timer/Event Counter will stop counting. If the Active Edge Select bit is high, the Timer/Event Counter will begin counting once a low to high transition has been received on the external timer pin and stop counting when the external timer pin returns to its original low level. As before, the enable bit will be automatically reset to zero and the Timer/Event Counter will stop counting. It is important to note that in the pulse width capture mode, the enable bit is automatically reset to zero when the external control signal on the external timer pin returns to its original level, whereas in the other two modes the enable bit can only be reset to zero under program control.

The residual value in the Timer/Event Counter, which can now be read by the program, therefore represents the length of the pulse received on the TMRn pin. As the enable bit has now been reset, any further transitions on the external timer pin will be ignored. The timer cannot begin further pulse width capture until the enable bit is set high again by the program. In this way, single shot pulse measurements can be easily made. It should be noted that in this mode the Timer/Event Counter is controlled by logical transitions on the external timer pin and not by the logic level.

When the Timer/Event Counter is full and overflows, an interrupt signal is generated and the Timer/Event Counter will reload the value already loaded into the preload register and continue counting. The interrupt can be disabled by ensuring that the Timer/Event Counter Interrupt Enable bit in the corresponding Interrupt Control Register, it is reset to zero. As the TMRn pin is shared with an I/O pin, to ensure that the pin is configured to operate as a pulse width capture pin, two things have to be implemented. The first is to ensure that the Operating Mode Select bits in the Timer Control Register place the Timer/Event Counter in the pulse width capture mode, the second is to ensure that the port control register configure the pin as an input.



Pulse Width Capture Mode Timing Chart (TnEG=0)

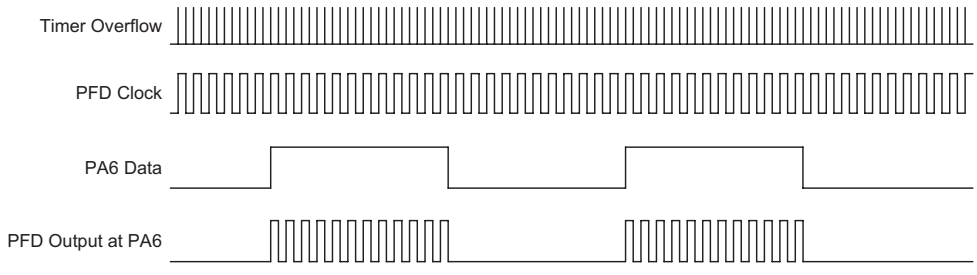
Prescaler

Bits TnPSC2~TnPSC0 of the TMRnC register can be used to define a division ratio for the internal clock source of the Timer/Event Counter enabling longer time out periods to be set.

PFD Function

The Programmable Frequency Divider provides a means of producing a variable frequency output suitable for application, such as some interfaces requiring a precise frequency generator.

The Timer/Event Counter overflow signal is the clock source for the PFD function, which is controlled by PFDC bit in CTRL0. For these devices the clock source can come from Timer/Event Counter 0. The output frequency is controlled by loading the required values into the timer prescaler and timer registers to give the required division ratio. The counter will begin to count-up from this preload register value until full, at which point an overflow signal is generated, causing both the PFD outputs to change state. Then the counter will be automatically reloaded with the preload register value and continue counting-up. If the CTRL0 register has selected the PFD function, then for PFD output to operate, it is essential for the Port A control register PAC to set the PFD pins as outputs. PA6 must be set high to activate the PFD. The output data bits can be used as the on/off control bit for the PFD outputs. Note that the PFD outputs will all be low if the output data bit is cleared to zero.



PFD Function

I/O Interfacing

The Timer/Event Counter, when configured to run in the event counter or pulse width capture mode, requires the use of an external timer pin for its operation. As this pin is a shared pin it must be configured correctly to ensure that it is set for use as a Timer/Event Counter input pin. This is achieved by ensuring that the mode selects bits in the Timer/Event Counter control register, either the event counter or pulse width capture mode. Additionally the corresponding Port Control Register bit must be set high to ensure that the pin is set as an input. Any pull-high resistor connected to this pin will remain valid even if the pin is used as a Timer/Event Counter input.

Programming Considerations

When running in the timer mode, the internal system clock is used as the timer clock source and is therefore synchronised with the overall operation of the microcontroller. In this mode when the appropriate timer register is full, the microcontroller will generate an internal interrupt signal directing the program flow to the respective internal interrupt vector. For the pulse width capture mode, the internal system clock is also used as the timer clock source but the timer will only run when the correct logic condition appears on the external timer input pin. As this is an external event and not synchronised with the internal timer clock, the microcontroller will only see this external event when the next timer clock pulse arrives. As a result, there may be small differences in measured values requiring programmers to take this into account during programming. The same applies if the timer is configured to be in the event counting mode, which again is an external event and not synchronised with the internal system or timer clock.

When the Timer/Event Counter is read, or if data is written to the preload register, the clock is inhibited to avoid errors, however as this may result in a counting error, this should be taken into account by the programmer. Care must be taken to ensure that the timers are properly initialised before using them for the first time. The associated timer enable bits in the interrupt control register must be properly set otherwise the internal interrupt associated with the timer will remain inactive. The edge select, timer mode and clock source control bits in timer control register must also be correctly set to ensure the timer is properly configured for the required application. It is also important to ensure that an initial value is first loaded into the timer registers before the timer is switched on; this is because after power-on the initial values of the timer registers are unknown. After the timer has been initialised the timer can be turned on and off by controlling the enable bit in the timer control register.

When the Timer/Event Counter overflows, its corresponding interrupt request flag in the interrupt control register will be set. If the Timer/Event Counter interrupt is enabled this will in turn generate an interrupt signal. However irrespective of whether the interrupts are enabled or not, a Timer/Event Counter overflow will also generate a wake-up signal if the devices are in a Power-down condition. This situation may occur if the Timer/Event Counter is in the Event Counting Mode and if the external signal continues to change state. In such a case, the Timer/Event Counter will continue to count these external events and if an overflow occurs the devices will be woken up from its Power-down condition. To prevent such a wake-up from occurring, the timer interrupt request flag should first be set high before issuing the “HALT” instruction to enter the Sleep Mode.

Timer Program Example

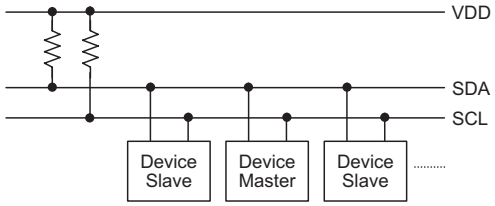
The program shows how the Timer/Event Counter registers are set along with how the interrupts are enabled and managed. Note how the Timer/Event Counter is turned on, by setting bit 4 of the Timer Control Register. The Timer/Event Counter can be turned off in a similar way by clearing the same bit. This example program sets the Timer/Event Counters to be in the timer mode, which uses the internal system clock as their clock source.

PFD Programming Example

```
org  04h              ; external interrupt vector
org  08h              ; Timer Counter 0 interrupt vector
jmp  tmr0int          ; jump here when Timer 0 overflows
     :
     :
org  20h              ; main program
     :
     :
                      ; internal Timer 0 interrupt routine
tmr0int:
     :
                      ; Timer 0 main program placed here
     :
     :
begin:
                      ; set Timer 0 registers
mov  a,09bh           ; set Timer 0 preload value
mov  tmr0,a
mov  a,081h           ; set Timer 0 control register
mov  tmr0c,a          ; timer mode and prescaler set to /2 set interrupt register
mov  a, 0c0H          ; select fSYS for the TMR0 clock source
mov  wdtlvrc, a
mov  a,05h            ; enable master interrupt and both timer interrupts
mov  intc0,a
     :
     :
set  tmr0c.4          ; start Timer 0
     :
     :
```

I²C Interface

The I²C interface is used to communicate with external peripheral devices such as sensors, EEPROM memory etc. Originally developed by Philips, it is a two line low speed serial interface for synchronous serial data transfer. The advantage of only two lines for communication, relatively simple communication protocol and the ability to accommodate multiple devices on the same bus has made it an extremely popular interface type for many applications.



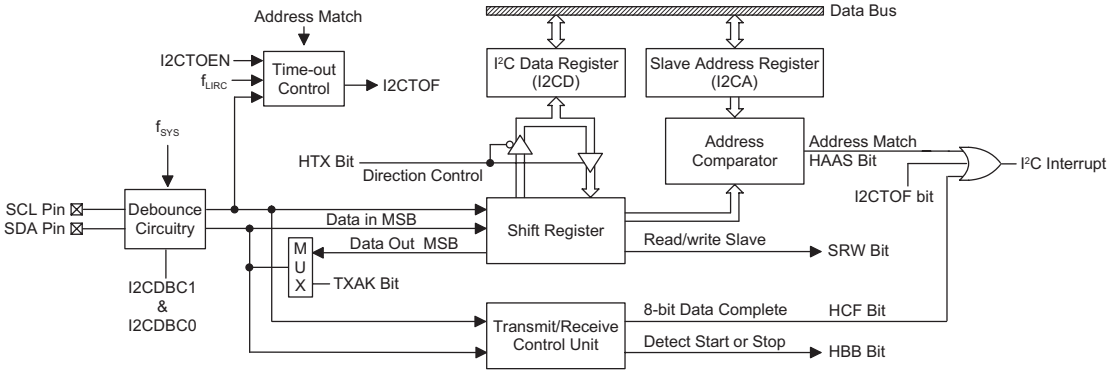
I²C Master/Slave Bus Connection

I²C Interface Operation

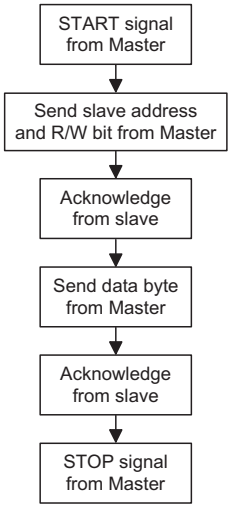
The I²C serial interface is a two line interface, a serial data line, SDA, and serial clock line, SCL. As many devices may be connected together on the same bus, their outputs are both open drain types. For this reason it is necessary that external pull-high resistors are connected to these outputs. Note that no chip select line exists, as each device on the I²C bus is identified by a unique address which will be transmitted and received on the I²C bus.

When two devices communicate with each other on the bidirectional I²C bus, one is known as the master device and one as the slave device. Both master and slave can transmit and receive data. However, it is the master device that has overall control of the bus. For this device, which only operates in slave mode, there are two methods of transferring data on the I²C bus, the slave transmit mode and the slave receive mode.

It is suggested that the user shall not enter the micro processor to HALT mode by application program during processing I²C communication.



I²C Block Diagram



I²C Registers

There are four control registers associated with the I²C bus, I2CC0, I2CC1, I2CA and I2CTOC and one data register, I2CD. The I2CD register is used to store the data being transmitted and received on the I²C bus. Before the microcontroller writes data to the I²C bus, the actual data to be transmitted must be placed in the I2CD register. After the data is received from the I²C bus, the microcontroller can read it from the I2CD register. Any transmission or reception of data from the I²C bus must be made via the I2CD register.

Register Name	Bit							
	7	6	5	4	3	2	1	0
I2CC0	—	—	—	—	I2CDBNC1	I2CDBNC0	I2CEN	—
I2CC1	HCF	HAAS	HBB	HTX	TXAK	SRW	IAMWU	RXAK
I2CD	D7	D6	D5	D4	D3	D2	D1	D0
I2CA	A6	A5	A4	A3	A2	A1	A0	—
I2CTOC	I2CTOEN	I2CTOF	I2CTOS5	I2CTOS4	I2CTOS3	I2CTOS2	I2CTOS1	I2CTOS0

I²C Registers List

I2CC0 Register

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	—	I2CDBC1	I2CDBC0	I2CEN	—
R/W	—	—	—	—	R/W	R/W	R/W	—
POR	—	—	—	—	0	0	0	—

Bit 7~4 Unimplemented, read as "0"

Bit 3~2 **I2CDBC1~I2CDBC0**: I²C Debounce Time Selection
00: No debounce
01: 2 system clock debounce
10: 4 system clock debounce
11: 4 system clock debounce

Bit 1 **I2CEN**: I²C enable
0: Disable
1: Enable

Bit 0 Unimplemented, read as "0"

I2CC1 Register

Bit	7	6	5	4	3	2	1	0
Name	HCF	HAAS	HBB	HTX	TXAK	SRW	IAMWU	RXAK
R/W	R	R	R	R/W	R/W	R	R/W	R
POR	1	0	0	0	0	0	0	1

Bit 7 **HCF**: I²C Bus data transfer completion flag
0: Data is being transferred
1: Completion of an 8-bit data transfer

The HCF flag is the data transfer flag. This flag will be zero when data is being transferred. Upon completion of an 8-bit data transfer the flag will go high and an interrupt will be generated.

Below is an example of the flow of a two-byte I²C data transfer.

First, I²C slave device receive a start signal from I²C master and then HCF bit is automatically cleared to zero.

Second, I²C slave device finish receiving the 1st data byte and then HCF bit is automatically set to one.

Third, user read the 1st data byte from I2CD register by the application program and then HCF bit is automatically cleared to zero.

Fourth, I²C slave device finish receiving the 2nd data byte and then HCF bit is automatically set to one and so on.

Finally, I²C slave device receive a stop signal from I²C master and then HCF bit is automatically set to one.

Bit 6 **HAAS**: I²C Bus address match flag
0: Not address match
1: Address match

The HAAS flag is the address match flag. This flag is used to determine if the slave device address is the same as the master transmit address. If the addresses match then this bit will be high, if there is no match then the flag will be low.

Bit 5 **HBB**: I²C Bus busy flag
0: I²C Bus is not busy
1: I²C Bus is busy

The HBB flag is the I²C busy flag. This flag will be "1" when the I²C bus is busy which will occur when a START signal is detected. The flag will be set to "0" when the bus is free which will occur when a STOP signal is detected.

Bit 4 **HTX**: Select I²C slave device is transmitter or receiver
0: Slave device is the receiver
1: Slave device is the transmitter

Bit 3 **TXAK**: I²C Bus transmit acknowledge flag
0: Slave send acknowledge flag
1: Slave do not send acknowledge flag

The TXAK bit is the transmit acknowledge flag. After the slave device receipt of 8-bits of data, this bit will be transmitted to the bus on the 9th clock from the slave device. The slave device must always set TXAK bit to "0" before further data is received.

Bit 2 **SRW**: I²C Slave Read/Write flag
0: Slave device should be in receive mode
1: Slave device should be in transmit mode

The SRW flag is the I²C Slave Read/Write flag. This flag determines whether the master device wishes to transmit or receive data from the I²C bus. When the transmitted address and slave address is match, that is when the HAAS flag is set high, the slave device will check the SRW flag to determine whether it should be in transmit mode or receive mode. If the SRW flag is high, the master is requesting to read data from the bus, so the slave device should be in transmit mode. When the SRW flag is zero, the master will write data to the bus, therefore the slave device should be in receive mode to read this data.

Bit 1 **IAMWU**: I²C Address Match Wake-up Control
0: Disable
1: Enable – must be cleared by the application program after wake-up
The I²C module can run without using internal clock, and generate an interrupt if the I²C interrupt is enabled, which can be used in SLEEP Mode, NORMAL(SLOW) Mode. This bit should be set to "1" to enable the I²C address match wake up from the SLEEP or IDLE Mode. If the IAMWU bit has been set before entering either the SLEEP or IDLE mode to enable the I²C address match wake up, then this bit must be cleared by the application program after wake-up to ensure correction device operation.

Bit 0 **RXAK**: I²C Bus Receive acknowledge flag
0: Slave receive acknowledge flag
1: Slave do not receive acknowledge flag
The RXAK flag is the receiver acknowledge flag. When the RXAK flag is "0", it means that a acknowledge signal has been received at the 9th clock, after 8 bits of data have been transmitted. When the slave device in the transmit mode, the slave device checks the RXAK flag to determine if the master receiver wishes to receive the next byte. The slave transmitter will therefore continue sending out data until the RXAK flag is "1". When this occurs, the slave transmitter will release the SDA line to allow the master to send a STOP signal to release the I²C Bus.

The I2CD register is used to store the data being transmitted and received. The same register is used by both the SPI and I²C functions. Before the device writes data to the I²C bus, the actual data to be transmitted must be placed in the I2CD register. After the data is received from the I²C bus, the device can read it from the I2CD register. Any transmission or reception of data from the I²C bus must be made via the I2CD register.

I2CD Register

Bit	7	6	5	4	3	2	1	0
Name	D7	D6	D5	D4	D3	D2	D1	D0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	x	x	x	x	x	x	x	x

"x" unknown

Bit 7~0 **D7~D0**: I2C Data Buffer bit 7~bit 0

I2CA Register

Bit	7	6	5	4	3	2	1	0
Name	A6	A5	A4	A3	A2	A1	A0	—
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	—
POR	x	x	x	x	x	x	x	—

"x" unknown

Bit 7~1 **A6~A0**: I²C slave address
A6~ A0 is the I²C slave address bit 6~bit 0.
The I2CA register is the location where the 7-bit slave address of the slave device is stored. Bits 7~ 1 of the I2CA register define the device slave address. Bit 0 is not defined.
When a master device, which is connected to the I²C bus, sends out an address, which matches the slave address in the I2CA register, the slave device will be selected.

Bit 0 Unimplemented, read as "0"

I²C Bus Communication

Communication on the I²C bus requires four separate steps, a START signal, a slave device address transmission, a data transmission and finally a STOP signal. When a START signal is placed on the I²C bus, all devices on the bus will receive this signal and be notified of the imminent arrival of data on the bus. The first seven bits of the data will be the slave address with the first bit being the MSB. If the address of the slave device matches that of the transmitted address, the HAAS bit in the I2CC1 register will be set and an I²C interrupt will be generated. After entering the interrupt service routine, the slave device must first check the condition of the HAAS bit to determine whether the interrupt source originates from an address match or from the completion of an 8-bit data transfer. During a data transfer, note that after the 7-bit slave address has been transmitted, the following bit, which is the 8th bit, is the read/write bit whose value will be placed in the SRW bit. This bit will be checked by the slave device to determine whether to go into transmit or receive mode. Before any transfer of data to or from the I²C bus, the microcontroller must initialise the bus. The following are steps to achieve this:

Step 1

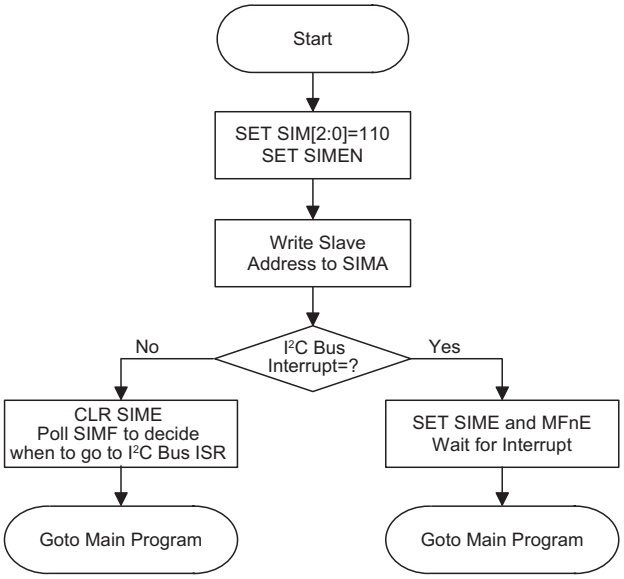
Set I2CEN bit in the I2CC0 register to "1" to enable the I²C bus.

Step 2

Write the slave address of the device to the I²C bus address register I2CA.

Step 3

Set the IICE interrupt enable bit of the interrupt control register to enable the I²C interrupt.



I²C Bus Initialisation Flow Chart

I²C Bus Start Signal

The START signal can only be generated by the master device connected to the I²C bus and not by the slave device. This START signal will be detected by all devices connected to the I²C bus. When detected, this indicates that the I²C bus is busy and therefore the HBB bit will be set. A START condition occurs when a high to low transition on the SDA line takes place when the SCL line remains high.

Slave Address

The transmission of a START signal by the master will be detected by all devices on the I²C bus. To determine which slave device the master wishes to communicate with, the address of the slave device will be sent out immediately following the START signal. All slave devices, after receiving this 7-bit address data, will compare it with their own 7-bit slave address. If the address sent out by the master matches the internal address of the microcontroller slave device, then an internal I²C bus interrupt signal will be generated. The next bit following the address, which is the 8th bit, defines the read/write status and will be saved to the SRW bit of the I2CC1 register. The slave device will then transmit an acknowledge bit, which is a low level, as the 9th bit. The slave device will also set the status flag HAAS when the addresses match.

As an I²C bus interrupt can come from two sources, when the program enters the interrupt subroutine, the HAAS bit should be examined to see whether the interrupt source has come from a matching slave address or from the completion of a data byte transfer. When a slave address is matched, the device must be placed in either the transmit mode and then write data to the I2CD register, or in the receive mode where it must implement a dummy read from the I2CD register to release the SCL line.

I²C Bus Read/Write Signal

The SRW bit in the I2CC1 register defines whether the slave device wishes to read data from the I²C bus or write data to the I²C bus. The slave device should examine this bit to determine if it is to be a transmitter or a receiver. If the SRW flag is "1" then this indicates that the master device wishes to read data from the I²C bus, therefore the slave device must be setup to send data to the I²C bus as a transmitter. If the SRW flag is "0" then this indicates that the master wishes to send data to the I²C bus, therefore the slave device must be setup to read data from the I²C bus as a receiver.

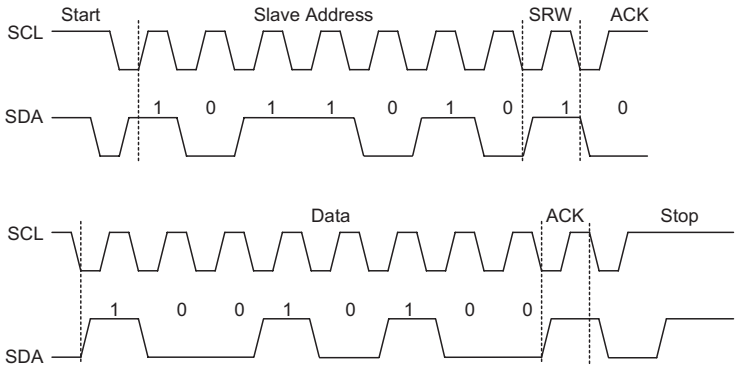
I²C Bus Slave Address Acknowledge Signal

After the master has transmitted a calling address, any slave device on the I²C bus, whose own internal address matches the calling address, must generate an acknowledge signal. The acknowledge signal will inform the master that a slave device has accepted its calling address. If no acknowledge signal is received by the master then a STOP signal must be transmitted by the master to end the communication. When the HAAS flag is high, the addresses have matched and the slave device must check the SRW flag to determine if it is to be a transmitter or a receiver. If the SRW flag is high, the slave device should be setup to be a transmitter so the HTX bit in the I2CC1 register should be set to "1". If the SRW flag is low, then the microcontroller slave device should be setup as a receiver and the HTX bit in the I2CC1 register should be set to "0".

I²C Bus Data and Acknowledge Signal

The transmitted data is 8-bits wide and is transmitted after the slave device has acknowledged receipt of its slave address. The order of serial bit transmission is the MSB first and the LSB last. After receipt of 8-bits of data, the receiver must transmit an acknowledge signal, level "0", before it can receive the next data byte. If the slave transmitter does not receive an acknowledge bit signal from the master receiver, then the slave transmitter will release the SDA line to allow the master to send a STOP signal to release the I²C Bus. The corresponding data will be stored in the I2CD register. If setup as a transmitter, the slave device must first write the data to be transmitted into the I2CD register. If setup as a receiver, the slave device must read the transmitted data from the I2CD register.

When the slave receiver receives the data byte, it must generate an acknowledge bit, known as TXAK, on the 9th clock. The slave device, which is setup as a transmitter will check the RXAK bit in the I2CC1 register to determine if it is to send another data byte, if not then it will release the SDA line and await the receipt of a STOP signal from the master.

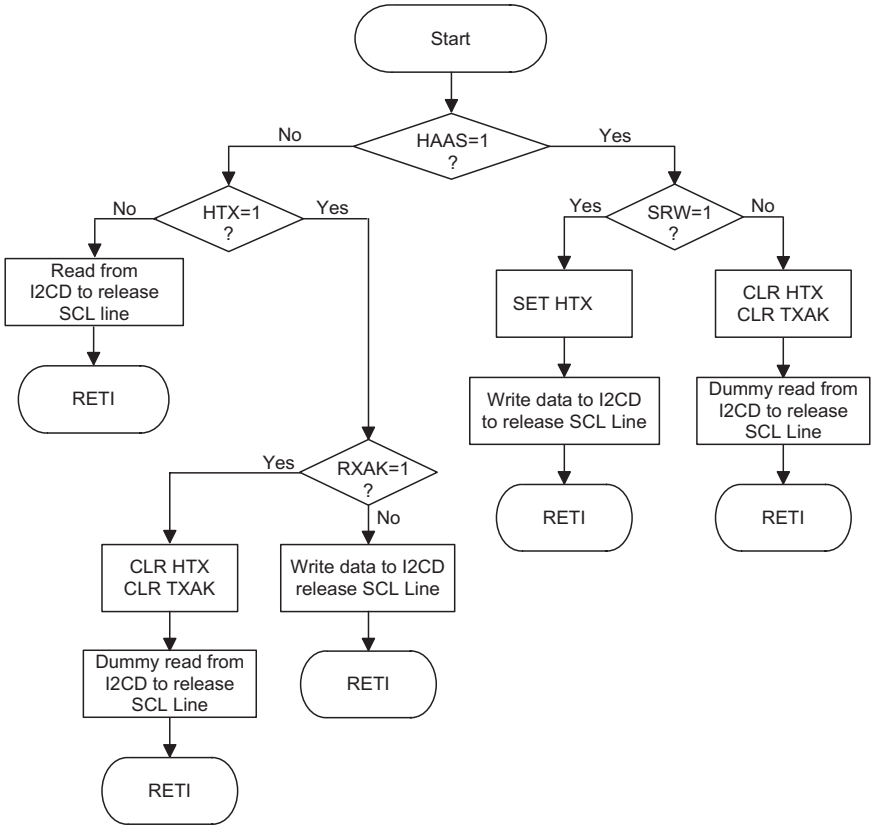


S=Start (1 bit)
SA=Slave Address (7 bits)
SR=SRW bit (1 bit)
M=Slave device send acknowledge bit (1 bit)
D=Data (8 bits)
A=ACK (RXAK bit for transmitter, TXAK bit for receiver 1 bit)
P=Stop (1 bit)

S	SA	SR	M	D	A	D	A	--------	S	SA	SR	M	D	A	D	A	--------	P

I²C Communication Timing Diagram

Note: *When a slave address is matched, the device must be placed in either the transmit mode and then write data to the I2CD register, or in the receive mode where it must implement a dummy read from the I2CD register to release the I²C SCL line.

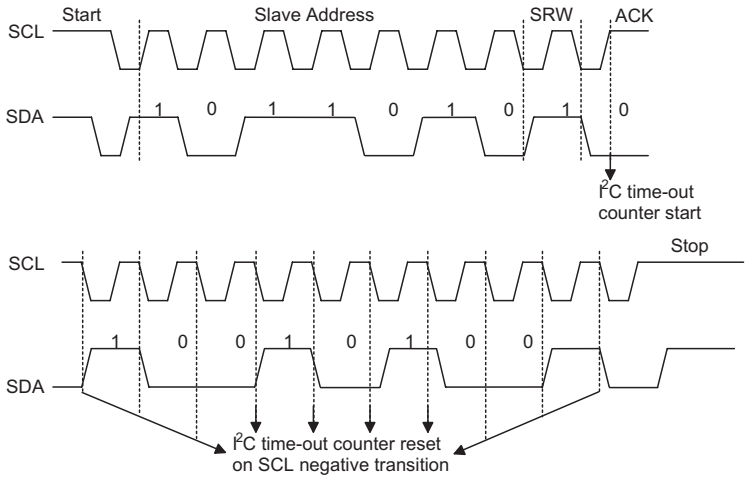


I²C Bus ISR Flow Chart

I²C Time-out Control

In order to reduce the problem of I²C lockup due to reception of erroneous clock sources, a time-out function is provided. If the clock source to the I²C is not received then after a fixed time period, the I²C circuitry and registers will be reset.

The time-out counter starts counting on an I²C bus "START" & "address match" condition, and is cleared by an SCL falling edge. Before the next SCL falling edge arrives, if the time elapsed is greater than the time-out setup by the I2CTOC register, then a time-out condition will occur. The time-out function will stop when an I²C "STOP" condition occurs.



I²C Time-out Control

When an I²C time-out counter overflow occurs, the counter will stop and the I2CTOEN bit will be cleared to zero and the I2CTOF bit will be set high to indicate that a time-out condition has occurred. The time-out condition will also generate an interrupt which uses the I²C interrupt vector. When an I²C time-out occurs, the I²C internal circuitry will be reset and the registers will be reset into the following condition:

Register	After I²C Time-out
I2CD, I2CA, I2CC0	No change
I2CC1	Reset to POR condition

I²C Registers after Time-out

The I2CTOF flag can be cleared by the application program. There are 64 time-out periods which can be selected using bits in the I2CTOC register. The time-out time is given by the formula:

$$((1\sim64) \times 32) / f_{LIRC}$$

This gives a range of about 1ms to 64ms. Note also that the LIRC oscillator is continuously enabled.

I2CTOC Register

Bit	7	6	5	4	3	2	1	0
Name	I2CTOEN	I2CTOF	I2CTOS5	I2CTOS4	I2CTOS3	I2CTOS2	I2CTOS1	I2CTOS0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **I2CTOEN**: I²C Time-out Control
0: Disable
1: Enable

Bit 6 **I2CTOF**: Time-out flag (set by time-out and clear by software)
0: No time-out
1: Time-out occurred

Bit 5~0 **I2CTOS5~I2CTOS0**: Time-out Definition
I²C time-out clock source is $f_{LIRC}/32$.
I²C time-out time is given by: ([I2CTOS5: I2CTOS0] + 1) × $(32/f_{LIRC})$

UART Module Serial Interface

The HT48R008 device contains an integrated full-duplex asynchronous serial communications UART interface that enables communication with external devices that contain a serial interface. The UART function has many features and can transmit and receive data serially by transferring a frame of data with eight or nine data bits per transmission as well as being able to detect errors when the data is overwritten or incorrectly framed. The UART function possesses its own internal interrupt which can be used to indicate when a reception occurs or when a transmission terminates.

The integrated UART function contains the following features:

- Full-duplex, asynchronous communication
- 8 or 9 bits character length
- Even, odd or no parity options
- One or two stop bits
- Baud rate generator with 8-bit prescaler
- Parity, framing, noise and overrun error detection
- Support for interrupt on address detect (last character bit=1)
- Separately enabled transmitter and receiver
- 2-byte Deep FIFO Receive Data Buffer
- Transmit and receive interrupts
- Interrupts can be initialized by the following conditions:
  - Transmitter Empty
  - Transmitter Idle
  - Receiver Full
  - Receiver Overrun
  - Address Mode Detect

UART External Pin Interfacing

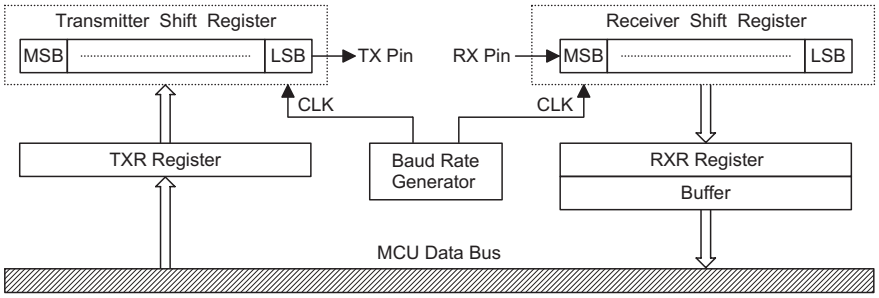
To communicate with an external serial interface, the internal UART has two external pins known as TX and RX. The TX pin is the UART transmitter pin, which can be used as a general purpose I/O pin if the pin is not configured as a UART transmitter, which occurs when the TXEN bit value is equal to zero. Similarly, the RX pin is the UART receiver pin, which can also be used as a general purpose I/O pin, if the pin is not configured as a receiver, which occurs if the RXEN bit in the UCR2 register is equal to zero. Along with the UARTEN bit, the TXEN and RXEN bits, if set, will automatically setup these I/O pins to their respective TX output and RX input conditions and disable any pull-high resistor option which may exist on the RX pin.

UART Data Transfer Scheme

The following block diagram shows the overall data transfer structure arrangement for the UART. The actual data to be transmitted from the MCU is first transferred to the TXR register by the application program. The data will then be transferred to the Transmit Shift Register from where it will be shifted out, LSB first, onto the TX pin at a rate controlled by the Baud Rate Generator. Only the TXR register is mapped onto the MCU Data Memory, the Transmit Shift Register is not mapped and is therefore inaccessible to the application program.

Data to be received by the UART is accepted on the external RX pin, from where it is shifted in, LSB first, to the Receiver Shift Register at a rate controlled by the Baud Rate Generator. When the shift register is full, the data will then be transferred from the shift register to the internal RXR register, where it is buffered and can be manipulated by the application program. Only the RXR register is mapped onto the MCU Data Memory, the Receiver Shift Register is not mapped and is therefore inaccessible to the application program.

It should be noted that the actual register for data transmission and reception, although referred to in the text, and in application programs, as separate TXR and RXR registers, only exists as a single shared register in the Data Memory. This shared register known as the TXR_RXR register is used for both data transmission and data reception.



UART Data Transfer Scheme

UART Status and Control Registers

There are four control registers associated with the UART function. The USR, UCR1 and UCR2 registers control the overall function of the UART, while the BRG register controls the Baud rate. The actual data to be transmitted and received on the serial interface is managed through the TXR_RXR data registers.

Register Name	Bit							
	7	6	5	4	3	2	1	0
USR	PERR	NF	FERR	OERR	RIDLE	RXIF	TIDLE	TXIF
UCR1	UARTEN	BNO	PREN	PRT	STOPS	TXBRK	RX8	TX8
UCR2	TXEN	RXEN	BRGH	ADDEN	WAKE	RIE	TIIE	TEIE
TXR_RXR	TXRX7	TXRX6	TXRX5	TXRX4	TXRX3	TXRX2	TXRX1	TXRX0
BRG	BRG7	BRG6	BRG5	BRG4	BRG3	BRG2	BRG1	BRG0

UART Registers Summary

USR Register

The USR register is the status register for the UART, which can be read by the program to determine the UART present status. All flags within the USR register are read only. Further explanation on each of the flags is given below.

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Name	PERR	NF	FERR	OERR	RIDLE	RXIF	TIDLE	TXIF
R/W	R	R	R	R	R	R	R	R
POR	0	0	0	0	0	0	0	0

Bit 7 **PERR**: Parity error flag
0: No parity error is detected
1: Parity error is detected

The PERR flag is the parity error flag. When this read only flag is "0", it indicates a parity error has not been detected. When the flag is "1", it indicates that the parity of the received word is incorrect. This error flag is applicable only if Parity mode (odd or even) is selected. The flag can also be cleared by a software sequence which involves a read to the status register USR followed by an access to the RXR data register.

Bit 6 **NF**: Noise flag
0: No noise is detected
1: Noise is detected

The NF flag is the noise flag. When this read only flag is "0", it indicates no noise condition. When the flag is "1", it indicates that the UART has detected noise on the receiver input. The NF flag is set during the same cycle as the RXIF flag but will not be set in the case of as overrun. The NF flag can be cleared by a software sequence which will involve a read to the status register USR followed by an access to the RXR data register.

Bit 5 **FERR**: Framing error flag
0: No framing error is detected
1: Framing error is detected

The FERR flag is the framing error flag. When this read only flag is "0", it indicates that there is no framing error. When the flag is "1", it indicates that a framing error has been detected for the current character. The flag can also be cleared by a software sequence which will involve a read to the status register USR followed by an access to the RXR data register.

Bit 4 **OERR**: Overrun error flag
0: No overrun error is detected
1: Overrun error is detected

The OERR flag is the overrun error flag which indicates when the receiver buffer has overflowed. When this read only flag is "0", it indicates that there is no overrun error. When the flag is "1", it indicates that an overrun error occurs which will inhibit further transfers to the RXR receive data register. The flag is cleared by a software sequence, which is a read to the status register USR followed by an access to the RXR data register.

Bit 3 **RIDLE**: Receiver status
0: Data reception is in progress (data being received)
1: No data reception is in progress (receiver is idle)

The RIDLE flag is the receiver status flag. When this read only flag is "0", it indicates that the receiver is between the initial detection of the start bit and the completion of the stop bit. When the flag is "1", it indicates that the receiver is idle. Between the completion of the stop bit and the detection of the next start bit, the RIDLE bit is "1" indicating that the UART receiver is idle and the RX pin stays in logic high condition.

Bit 2 **RXIF**: Receive RXR data register status
0: RXR data register is empty
1: RXR data register has available data, at least one more character can be read.
The RXIF flag is the receive data register status flag. When this read only flag is "0", it indicates that the RXR read data register is empty. When the flag is "1", it indicates that the RXR read data register contains new data. When the contents of the shift register are transferred to the RXR register, an interrupt is generated if RIE=1 in the UCR2 register. If one or more errors are detected in the received word, the appropriate receive-related flags NF, FERR, and/or PERR are set within the same clock cycle.
The RXIF flag is cleared when the USR register is read with RXIF set, followed by a read from the RXR register, and if the RXR register has no data available.

Bit 1 **TIDLE**: Transmission idle
0: Data transmission is in progress (data being transmitted)
1: No data transmission is in progress (transmitter is idle)
The TIDLE flag is known as the transmission complete flag. When this read only flag is "0", it indicates that a transmission is in progress. This flag will be set to "1" when the TXIF flag is "1" and when there is no transmit data or break character being transmitted. When TIDLE is equal to "1", the TX pin becomes idle with the pin state in logic high condition. The TIDLE flag is cleared by reading the USR register with TIDLE set and then writing to the TXR register. The flag is not generated when a data character or a break is queued and ready to be sent.

Bit 0 **TXIF**: Transmit TXR data register status
0: Character is not transferred to the transmit shift register
1: Character has transferred to the transmit shift register (TXR data register is empty)
The TXIF flag is the transmit data register empty flag. When this read only flag is "0", it indicates that the character is not transferred to the transmitter shift register. When the flag is "1", it indicates that the transmitter shift register has received a character from the TXR data register. The TXIF flag is cleared by reading the UART status register (USR) with TXIF set and then writing to the TXR data register. Note that when the TXEN bit is set, the TXIF flag bit will also be set since the transmit data register is not yet full.

UCR1 Register

The UCR1 register together with the UCR2 register are the two UART control registers that are used to set the various options for the UART function, such as overall on/off control, parity control, data transfer bit length etc. Further explanation on each of the bits is given below:

Bit	7	6	5	4	3	2	1	0
Name	UARTEN	BNO	PREN	PRT	STOPS	TXBRK	RX8	TX8
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R	W
POR	0	0	0	0	0	0	x	0

"x" unknown

Bit 7 **UARTEN**: UART function enable control
0: Disable UART. TX and RX pins are as I/O pins
1: Enable UART. TX and RX pins function as UART pins
The UARTEN bit is the UART enable bit. When this bit is equal to "0", the UART will be disabled and the RX pin as well as the TX pin will be as General Purpose I/O pins. When the bit is equal to "1", the UART will be enabled and the TX and RX pins will function as defined by the TXEN and RXEN enable control bits.

When the UART is disabled, it will empty the buffer so any character remaining in the buffer will be discarded. In addition, the value of the baud rate counter will be reset. If the UART is disabled, all error and status flags will be reset. Also the TXEN, RXEN, TXBRK, RXIF, OERR, FERR, PERR and NF bits will be cleared, while the TIDLE, TXIF and RIDLE bits will be set. Other control bits in UCR1, UCR2 and BRG registers will remain unaffected. If the UART is active and the UARTEN bit is cleared, all pending transmissions and receptions will be terminated and the module will be reset as defined above. When the UART is re-enabled, it will restart in the same configuration.

Bit 6 **BNO**: Number of data transfer bits selection
0: 8-bit data transfer
1: 9-bit data transfer

This bit is used to select the data length format, which can have a choice of either 8-bit or 9-bit format. When this bit is equal to "1", a 9-bit data length format will be selected. If the bit is equal to "0", then an 8-bit data length format will be selected. If 9-bit data length format is selected, then bits RX8 and TX8 will be used to store the 9th bit of the received and transmitted data respectively.

Bit 5 **PREN**: Parity function enable control
0: Parity function is disabled
1: Parity function is enabled

This is the parity enable bit. When this bit is equal to "1", the parity function will be enabled. If the bit is equal to "0", then the parity function will be disabled.

Bit 4 **PRT**: Parity type selection bit
0: Even parity for parity generator
1: Odd parity for parity generator

This bit is the parity type selection bit. When this bit is equal to "1", odd parity type will be selected. If the bit is equal to "0", then even parity type will be selected.

Bit 3 **STOPS**: Number of Stop bits selection
0: One stop bit format is used
1: Two stop bits format is used

This bit determines if one or two stop bits are to be used. When this bit is equal to "1", two stop bits are used. If this bit is equal to "0", then only one stop bit is used.

Bit 2 **TXBRK**: Transmit break character
0: No break character is transmitted
1: Break characters transmit

The TXBRK bit is the Transmit Break Character bit. When this bit is "0", there are no break characters and the TX pin operates normally. When the bit is "1", there are transmit break characters and the transmitter will send logic zeros. When this bit is equal to "1", after the buffered data has been transmitted, the transmitter output is held low for a minimum of a 13-bit length and until the TXBRK bit is reset.

Bit 1 **RX8**: Receive data bit 8 for 9-bit data transfer format (read only)

This bit is only used if 9-bit data transfers are used, in which case this bit location will store the 9th bit of the received data known as RX8. The BNO bit is used to determine whether data transfers are in 8-bit or 9-bit format.

Bit 0 **TX8**: Transmit data bit 8 for 9-bit data transfer format (write only)

This bit is only used if 9-bit data transfers are used, in which case this bit location will store the 9th bit of the transmitted data known as TX8. The BNO bit is used to determine whether data transfers are in 8-bit or 9-bit format.

UCR2 Register

The UCR2 register is the second of the two UART control registers and serves several purposes. One of its main functions is to control the basic enable/disable operation of the UART Transmitter and Receiver as well as enabling the various UART interrupts. The register also serves to control the baud rate speed, receiver wake-up enable and the address detect enable. Further explanation on each of the bits is given below:

Bit	7	6	5	4	3	2	1	0
Name	TXEN	RXEN	BRGH	ADDEN	WAKE	RIE	TIIE	TEIE
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	0	0	0	0	0	0	0	0

Bit 7 **TXEN**: UART Transmitter enabled control
0: UART transmitter is disabled
1: UART transmitter is enabled

The bit named TXEN is the Transmitter Enable Bit. When this bit is equal to "0", the transmitter will be disabled with any pending data transmissions being aborted. In addition the buffers will be reset. In this situation the TX pin will be as GPIO PORT.

If the TXEN bit is equal to "1" and the UARTEN bit is also equal to "1", the transmitter will be enabled and the TX pin will be controlled by the UART. Clearing the TXEN bit during a transmission will cause the data transmission to be aborted and will reset the transmitter. If this situation occurs, the TX pin will be as GPIO PORT.

Bit 6 **RXEN**: UART Receiver enabled control
0: UART receiver is disabled
1: UART receiver is enabled

The bit named RXEN is the Receiver Enable Bit. When this bit is equal to "0", the receiver will be disabled with any pending data receptions being aborted. In addition the receive buffers will be reset. In this situation the RX pin will be as GPIO PORT. If the RXEN bit is equal to "1" and the UARTEN bit is also equal to "1", the receiver will be enabled and the RX pin will be controlled by the UART. Clearing the RXEN bit during a reception will cause the data reception to be aborted and will reset the receiver. If this situation occurs, the RX pin will be as GPIO PORT.

Bit 5 **BRGH**: Baud Rate speed selection
0: Low speed baud rate
1: High speed baud rate

The bit named BRGH selects the high or low speed mode of the Baud Rate Generator. This bit, together with the value placed in the baud rate register BRG, controls the Baud Rate of the UART. If this bit is equal to "1", the high speed mode is selected. If the bit is equal to "0", the low speed mode is selected.

Bit 4 **ADDEN**: Address detect function enable control
0: Address detect function is disabled
1: Address detect function is enabled

The bit named ADDEN is the address detect function enable control bit. When this bit is equal to "1", the address detect function is enabled. When it occurs, if the 8^{th} bit, which corresponds to RX7 if BNO=0 or the 9^{th} bit, which corresponds to RX8 if BNO=1, has a value of "1", then the received word will be identified as an address, rather than data. If the corresponding interrupt is enabled, an interrupt request will be generated each time the received word has the address bit set, which is the 8^{th} or 9^{th} bit depending on the value of BNO. If the address bit known as the 8^{th} or 9^{th} bit of the received word is "0" with the address detect function being enabled, an interrupt will not be generated and the received data will be discarded.

Bit 3 **WAKE**: RX pin falling edge wake-up function enable control
0: RX pin wake-up function is disabled
1: RX pin wake-up function is enabled

This bit enables or disables the receiver wake-up function. If this bit is equal to "1" and the MCU is in Power-down mode, a falling edge on the RX input pin will wake-up the device. If this bit is equal to "0" and the MCU is in Power-down mode, any edge transitions on the RX pin will not wake-up the device.

Bit 2 **RIE**: Receiver interrupt enable control
0: receiver related interrupt is disabled
1: receiver related interrupt is enabled
This bit enables or disables the receiver interrupt. If this bit is equal to "1" and when the receiver overrun flag OERR or receive data available flag RXIF is set, the UART interrupt request flag will be set. If this bit is equal to "0", the UART interrupt request flag will not be influenced by the condition of the OERR or RXIF flags.

Bit 1 **TIIE**: Transmitter Idle interrupt enable control
0: Transmitter idle interrupt is disabled
1: Transmitter idle interrupt is enabled
This bit enables or disables the transmitter idle interrupt. If this bit is equal to "1" and when the transmitter idle flag TIDLE is set, due to a transmitter idle condition, the UART interrupt request flag will be set. If this bit is equal to "0", the UART interrupt request flag will not be influenced by the condition of the TIDLE flag.

Bit 0 **TEIE**: Transmitter Empty interrupt enable control
0: Transmitter empty interrupt is disabled
1: Transmitter empty interrupt is enabled
his bit enables or disables the transmitter empty interrupt. If this bit is equal to "1" and when the transmitter empty flag TXIF is set, due to a transmitter empty condition, the UART interrupt request flag will be set. If this bit is equal to "0", the UART interrupt request flag will not be influenced by the condition of the TXIF flag.

TXR_RXR Register

Bit	7	6	5	4	3	2	1	0
Name	TXRX7	TXRX6	TXRX5	TXRX4	TXRX3	TXRX2	TXRX1	TXRX0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	x	x	x	x	x	x	x	x

"x" means unknown

Bit 7~0 **TXRX7~TXRX0**: UART Transmit/receive data bit

Baud Rate Generator

To setup the speed of the serial data communication, the UART function contains its own dedicated baud rate generator. The baud rate is controlled by its own internal free running 8-bit timer, the period of which is determined by two factors. The first of these is the value placed in the baud rate register BRG and the second is the value of the BRGH bit with the control register UCR2. The BRGH bit decides if the baud rate generator is to be used in a high speed mode or low speed mode, which in turn determines the formula that is used to calculate the baud rate. The value N in the BRG register which is used in the following baud rate calculation formula determines the division factor. Note that N is the decimal value placed in the BRG register and has a range of between 0 and 255.

UCR2 BRGH Bit	0	1
Baud Rate (BR)	f_{SYS} / [64 (N+1)]	f_{SYS} / [16 (N+1)]

By programming the BRGH bit which allows selection of the related formula and programming the required value in the BRG register, the required baud rate can be setup. Note that because the actual baud rate is determined using a discrete value, N, placed in the BRG register, there will be an error associated between the actual and requested value. The following example shows how the BRG register value N and the error value can be calculated.

Calculating the baud rate and error values

For a clock frequency of 4 MHz, and with BRGH set to "0" determine the BRG register value N, the actual baud rate and the error value for a desired baud rate of 4800.

From the above table the desired baud rate BR = f_{SYS} / [64 (N+1)]

Re-arranging this equation gives N = [f_{SYS} / (BR×64) / 64] - 1

Giving a value for N = [(8000000 / 9600) / 64] - 1 = 12.0208

To obtain the closest value, a decimal value of 12 should be placed into the BRG register. This gives an actual or calculated baud rate value of BR = 4000000 / [64 (12 + 1)] = 4808

Therefore the error is equal to (4808 - 4800) / 4800 = 0.16%

The following tables show actual values of baud rate and error values for the two values of BRGH.

Baud Rate K/BPS	Baud Rates for BRGH=0								
	f_{CLKI}=4MHz			f_{CLKI}=3.579545MHz			f_{CLKI}=7.159MHz		
	BRG	Kbaud	Error(%)	BRG	Kbaud	Error(%)	BRG	Kbaud	Error(%)
0.3	207	0.300	0.16	185	0.300	0.00	—	—	—
1.2	51	1.202	0.16	46	1.190	-0.83	92	1.203	0.23
2.4	25	2.404	0.16	22	2.432	1.32	46	2.380	-0.83
4.8	12	4.808	0.16	11	4.661	-2.90	22	4.863	1.32
9.6	6	8.929	-6.99	5	9.321	-2.90	11	9.332	-2.90
19.2	2	20.833	8.51	2	18.643	-2.90	5	18.643	-2.90
38.4	—	—	—	—	—	—	2	32.286	-2.90
57.6	0	62.500	8.51	0	55.930	-2.90	1	55.930	-2.90
115.2	—	—	—	—	—	—	0	111.859	-2.90

Baud Rates and Error Values for BRGH = 0

Baud Rate K/BPS	Baud Rates for BRGH=1								
	f_{CLKI}=4MHz			f_{CLKI}=3.579545MHz			f_{CLKI}=7.159MHz		
	BRG	Kbaud	Error(%)	BRG	Kbaud	Error(%)	BRG	Kbaud	Error(%)
0.3	—	—	—	—	—	—	—	—	—
1.2	207	1.202	0.16	185	1.203	0.23	—	—	—
2.4	103	2.404	0.16	92	2.406	0.23	185	2.406	0.23
4.8	51	4.808	0.16	46	4.76	-0.83	92	4.811	0.23
9.6	25	9.615	0.16	22	9.727	1.32	46	9.520	-0.83
19.2	12	19.231	0.16	11	18.643	-2.90	22	19.454	1.32
38.4	6	35.714	-6.99	5	37.286	-2.90	11	37.286	-2.90
57.6	3	62.5	8.51	3	55.930	-2.90	7	55.930	-2.90
115.2	1	125	8.51	1	111.86	-2.90	3	111.86	-2.90
250	0	250	0	—	—	—	—	—	—

Baud Rates and Error Values for BRGH = 1

BRG Register

Bit	7	6	5	4	3	2	1	0
Name	BRG7	BRG6	BRG5	BRG4	BRG3	BRG2	BRG1	BRG0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	x	x	x	x	x	x	x	x

"x" means unknown

Bit 7~0 **BRG7~BRG0**: Baud Rate values

By programming the BRGH bit in UCR2 Register which allows selection of the related formula described above and programming the required value in the BRG register, the required baud rate can be setup.

Note: Baud rate= f_{SYS}/[64*(N+1)] if BRGH=0

Baud rate= f_{SYS}/[16*(N+1)] if BRGH=1

UART Setup and Control

For data transfer, the UART function utilizes a non-return-to-zero, more commonly known as NRZ, format. This is composed of one start bit, eight or nine data bits, and one or two stop bits. Parity is supported by the UART hardware, and can be setup to be even, odd or no parity. For the most common data format, 8 data bits along with no parity and one stop bit, denoted as 8, N, 1, is used as the default setting, which is the setting at power-on. The number of data bits and stop bits, along with the parity, are setup by programming the corresponding BNO, PRT, PREN, and STOPS bits in the UCR1 register. The baud rate used to transmit and receive data is setup using the internal 8-bit baud rate generator, while the data is transmitted and received LSB first. Although the UART transmitter and receiver are functionally independent, they both use the same data format and baud rate. In all cases stop bits will be used for data transmission.

- Enabling/disabling the UART interface

  The basic on/off function of the internal UART function is controlled using the UARTEN bit in the UCR1 register. As the UART transmit and receive pins, TX and RX respectively, are pin-shared with normal I/O pins. One of the basic functions of the UARTEN control bit is to control the UART function of these two pins. If the UARTEN, TXEN and RXEN bits are set, then these two I/O pins will be setup as a TX output pin and an RX input pin respectively, in effect disabling the normal I/O pin function. If no data is being transmitted on the TX pin then it will default to a logic high value.

  Clearing the UARTEN bit will disable the TX and RX pins and allow these two pins to be used as normal I/O pins. When the UART function is disabled the buffer will be reset to an empty condition, at the same time discarding any remaining residual data. Disabling the UART will also reset the error and status flags with bits TXEN, RXEN, TXBRK, RXIF, OERR, FERR, PERR and NF being cleared while bits TIDLE, TXIF and RIDLE will be set. The remaining control bits in the UCR1, UCR2 and BRG registers will remain unaffected. If the UARTEN bit in the UCR1 register is cleared while the UART is active, then all pending transmissions and receptions will be immediately suspended and the UART will be reset to a condition as defined above. If the UART is then subsequently re-enabled, it will restart again in the same configuration.

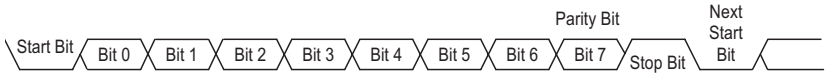
- Data, parity and stop bit selection

  The format of the data to be transferred is composed of various factors such as data bit length, parity on/off, parity type, address bits and the number of stop bits. These factors are determined by the setup of various bits within the UCR1 register. The BNO bit controls the number of data bits which can be set to either 8 or 9, the PRT bit controls the choice of odd or even parity, the PREN bit controls the parity on/off function and the STOPS bit decides whether one or two stop bits are to be used. The following table shows various formats for data transmission. The address bit identifies the frame as an address character. The number of stop bits, which can be either one or two, is independent of the data length.

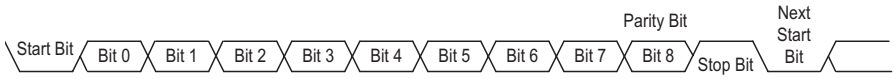
Start Bit	Data Bits	Address Bits	Parity Bits	Stop Bit
Example of 8-bit Data Formats				
1	8	0	0	1
1	7	0	1	1
1	7	1	0	1
Example of 9-bit Data Formats				
1	9	0	0	1
1	8	0	1	1
1	8	1	0	1

Transmitter Receiver Data Format

The following diagram shows the transmit and receive waveforms for both 8-bit and 9-bit data formats.



8-Bit Data Format



9-Bit Data Format

UART Transmitter

Data word lengths of either 8 or 9 bits can be selected by programming the BNO bit in the UCR1 register. When BNO bit is set, the word length will be set to 9 bits. In this case the 9th bit, which is the MSB, needs to be stored in the TX8 bit in the UCR1 register. At the transmitter core lies the Transmitter Shift Register, more commonly known as the TSR, whose data is obtained from the transmit data register, which is known as the TXR register. The data to be transmitted is loaded into this TXR register by the application program. The TSR register is not written to with new data until the stop bit from the previous transmission has been sent out. As soon as this stop bit has been transmitted, the TSR can then be loaded with new data from the TXR register, if it is available. It should be noted that the TSR register, unlike many other registers, is not directly mapped into the Data Memory area and as such is not available to the application program for direct read/write operations. An actual transmission of data will normally be enabled when the TXEN bit is set, but the data will not be transmitted until the TXR register has been loaded with data and the baud rate generator has defined a shift clock source. However, the transmission can also be initiated by first loading data into the TXR register, after which the TXEN bit can be set. When a transmission of data begins, the TSR is normally empty, in which case a transfer to the TXR register will result in an immediate transfer to the TSR. If during a transmission the TXEN bit is cleared, the transmission will immediately cease and the transmitter will be reset. The TX output pin will then return to having a normal general purpose I/O pin function.

- Transmitting data

  When the UART is transmitting data, the data is shifted on the TX pin from the shift register, with the least significant bit LSB first. In the transmit mode, the TXR register forms a buffer between the internal bus and the transmitter shift register. It should be noted that if 9-bit data format has been selected, then the MSB will be taken from the TX8n bit in the UCR1 register. The steps to initiate a data transfer can be summarized as follows:

  - Make the correct selection of the BNO, PRT, PREN and STOPS bits to define the required word length, parity type and number of stop bits.
  - Setup the BRG register to select the desired baud rate.
  - Set the TXEN bit to ensure that the UART transmitter is enabled and the TX pin is used as a UART transmitter pin.
  - Access the USR register and write the data that is to be transmitted into the TXR register. Note that this step will clear the TXIF bit.

  This sequence of events can now be repeated to send additional data. It should be noted that when TXIF=0, data will be inhibited from being written to the TXR register. Clearing the TXIF flag is always achieved using the following software sequence:

  1. A USR register access
  2. A TXR register write execution

  The read-only TXIF flag is set by the UART hardware and if set indicates that the TXR register is empty and that other data can now be written into the TXR register without overwriting the previous data. If the TEIE bit is set, then the TXIF flag will generate an interrupt. During a data transmission, a write instruction to the TXR register will place the data into the TXR register, which will be copied to the shift register at the end of the present transmission. When there is no data transmission in progress, a write instruction to the TXR register will place the data directly into the shift register, resulting in the commencement of data transmission, and the TXIF bit being immediately set. When a frame transmission is complete, which happens after stop bits are sent or after the break frame, the TIDLE bit will be set. To clear the TIDLE bit the following software sequence is used:

  1. A USR register access
  2. A TXR register write execution

  Note that both the TXIF and TIDLE bits are cleared by the same software sequence.

- Transmit break

  If the TXBRK bit is set, then the break characters will be sent on the next transmission. Break character transmission consists of a start bit, followed by 13xN “0” bits, where N=1, 2, etc. if a break character is to be transmitted, then the TXBRK bit must be first set by the application program and then cleared to generate the stop bits. Transmitting a break character will not generate a transmit interrupt. Note that a break condition length is at least 13 bits long. If the TXBRK bit is continually kept at a logic high level, then the transmitter circuitry will transmit continuous break characters. After the application program has cleared the TXBRK bit, the transmitter will finish transmitting the last break character and subsequently send out one or two stop bits. The automatic logic high at the end of the last break character will ensure that the start bit of the next frame is recognized.

UART Receiver

The UART is capable of receiving word lengths of either 8 or 9 bits can be selected by programming the BNO bit in the UCR1 register. When BNO bit is set, the word length will be set to 9 bits. In this case the 9th bit, which is the MSB, will be stored in the RX8 bit in the UCR1 register. At the receiver core lines the Receiver Shift Register more commonly known as the RSR. The data which is received on the RX external input pin is sent to the data recovery block. The data recovery block operating speed is 16 times that of the baud rate, while the main receive serial shifter operates at the baud rate. After the RX pin is sampled for the stop bit, the received data in RSR is transferred to the receive data register, if the register is empty. The data which is received on the external RX input pin is sampled three times by a majority detect circuit to determine the logic level that has been placed onto the RX pin. It should be noted that the RSR register, unlike many other registers, is not directly mapped into the Data Memory area and as such is not available to the application program for direct read/write operations.

- Receiving data

  When the UART receiver is receiving data, the data is serially shifted in on the external RX input pin to the shift register, with the least significant bit LSB first. The RXR register is a two byte deep FIFO data buffer, where two bytes can be held in the FIFO while the third byte can continue to be received. Note that the application program must ensure that the data is read from RXR before the third byte has been completely shifted in, otherwise the third byte will be discarded and an overrun error OERR will be subsequently indicated. The steps to initiate a data transfer can be summarized as follows:

  - Make the correct selection of the BNO, PRT, PREN and STOPS bits to define the required word length, parity type and number of stop bits.
  - Setup the BRG register to select the desired baud rate.
  - Set the RXEN bit to ensure that the UART receiver is enabled and the RX pin is used as a UART receiver pin.

  At this point the receiver will be enabled which will begin to look for a start bit.

  When a character is received, the following sequence of events will occur:

  - The RXIF bit in the USR register will be set then RXR register has data available, at least three more character can be read.
  - When the contents of the shift register have been transferred to the RXR register and if the RIE bit is set, then an interrupt will be generated.
  - If during reception, a frame error, noise error, parity error or an overrun error has been detected, and then the error flags can be set.

  The RXIF bit can be cleared using the following software sequence:

  1. A USR register access
  2. A RXR register read execution

- Receiving break

  Any break character received by the UART will be managed as a framing error. The receiver will count and expect a certain number of bit times as specified by the values programmed into the BNO and STOPS bits. If the break is much longer than 13 bit times, the reception will be considered as complete after the number of bit times specified by BNO and STOPS. The RXIF bit is set, FERR is set, zeros are loaded into the receive data register, interrupts are generated if appropriate and the RIDLE bit is set. If a long break signal has been detected and the receiver has received a start bit, the data bits and the invalid stop bit, which sets the FERR flag, the receiver must wait for a valid stop bit before looking for the next start bit. The receiver will not make the assumption that the break condition on the line is the next start bit. A break is regarded as a character that contains only zeros with the FERR flag set. The break character will be loaded

into the buffer and no further data will be received until stop bits are received. It should be noted that the RIDLE read only flag will go high when the stop bits have not yet been received. The reception of a break character on the UART registers will result in the following:

- The framing error flag, FERR, will be set.
- The receive data register, RXR, will be cleared.
- The OERR, NF, PERR, RIDLE or RXIF flags will possibly be set.

- Idle status
  When the receiver is reading data, which means it will be in between the detection of a start bit and the reading of a stop bit, the receiver status flag in the USR register, otherwise known as the RIDLE flag, will have a zero value. In between the reception of a stop bit and the detection of the next start bit, the RIDLE flag will have a high value, which indicates the receiver is in an idle condition.
- Receiver interrupt
  The read only receive interrupt flag RXIF in the USR register is set by an edge generated by the receiver. An interrupt is generated if RIE=1, when a word is transferred from the Receive Shift Register, RSR, to the Receive Data Register, RXR. An overrun error can also generate an interrupt if RIE=1.

Managing Receiver Errors

Several types of reception errors can occur within the UART module, the following section describes the various types and how they are managed by the UART.

- Overrun Error – OERR
  The RXR register is composed of a two byte deep FIFO data buffer, where two bytes can be held in the FIFO register, while a third byte can continue to be received. Before the third byte has been entirely shifted in, the data should be read from the RXR register. If this is not done, the overrun error flag OERR will be consequently indicated.
  In the event of an overrun error occurring, the following will happen:
  - The OERR flag in the USR register will be set.
  - The RXR contents will not be lost.
  - The shift register will be overwritten.
  - An interrupt will be generated if the RIE bit is set.

  The OERR flag can be cleared by an access to the USR register followed by a read to the RXR register.
- Noise Error – NF Flag
  Over-sampling is used for data recovery to identify valid incoming data and noise. If noise is detected within a frame, the following will occur:
  - The read only noise flag, NF, in the USR register will be set on the rising edge of the RXIF bit.
  - Data will be transferred from the shift register to the RXR register.
  - No interrupt will be generated. However this bit rises at the same time as the RXIF bit which itself generates an interrupt.

  Note that the NF flag is reset by a USR register read operation followed by an RXR register read operation.
- Framing Error – FERR
  The read only framing error flag, FERR, in the USR register, is set if a zero is detected instead of stop bits. If two stop bits are selected, both stop bits must be high. Otherwise the FERR flag will be set. The FERR flag is buffered along with the received data and is cleared in any reset.

- Parity Error – PERR

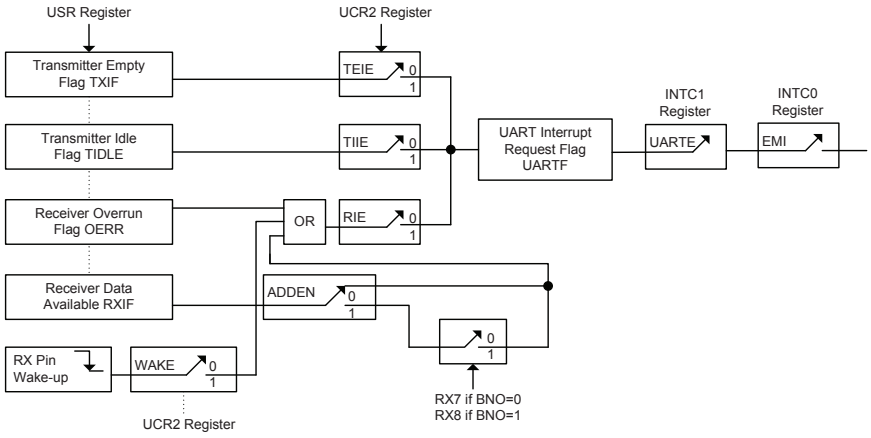
  The read only parity error flag, PERR, in the USR register, is set if the parity of the received word is incorrect. This error flag is only applicable if the parity function is enabled, PREN=1, and if the parity type, odd or even, is selected. The read only PERR flag is buffered along with the received data bytes. It is cleared on any reset, it should be noted that the FERR and PERR flags are buffered along with the corresponding word and should be read before reading the data word.

UART Interrupt Structure

Several individual UART conditions can generate a UART interrupt. When these conditions exist, a low pulse will be generated to get the attention of the microcontroller. These conditions are a transmitter data register empty, transmitter idle, receiver data available, receiver overrun, address detect and an RX pin wake-up. When any of these conditions are created, if its corresponding interrupt control is enabled and the stack is not full, the program will jump to its corresponding interrupt vector where it can be serviced before returning to the main program. Four of these conditions have the corresponding USR register flags which will generate a UART interrupt if its associated interrupt enable control bit in the UCR2 register is set. The two transmitter interrupt conditions have their own corresponding enable control bits, while the two receiver interrupt conditions have a shared enable control bit. These enable bits can be used to mask out individual UART interrupt sources.

The address detect condition, which is also a UART interrupt source, does not have an associated flag, but will generate a UART interrupt when an address detect condition occurs if its function is enabled by setting the ADDEN bit in the UCR2 register. An RX pin wake-up, which is also a UART interrupt source, does not have an associated flag, but will generate a UART interrupt if the microcontroller is woken up by a falling edge on the RX pin, if the WAKE and RIE bits in the UCR2 register are set. Note that in the event of an RX wake-up interrupt occurring, there will be a certain period of delay, commonly known as the System Start-up Time, for the oscillator to restart and stabilize before the system resumes normal operation.

Note that the USR register flags are read only and cannot be cleared or set by the application program, neither will they be cleared when the program jumps to the corresponding interrupt servicing routine, as is the case for some of the other interrupts. The flags will be cleared automatically when certain actions are taken by the UART, the details of which are given in the UART register section. The overall UART interrupt can be disabled or enabled by the related interrupt enable control bits in the interrupt control registers of the microcontroller to decide whether the interrupt requested by the UART module is masked out or allowed.



UART Interrupt Scheme

Address Detect Mode

Setting the Address Detect function enable control bit, ADDEN, in the UCR2 register, enables this special function. If this bit is set to 1, then an additional qualifier will be placed on the generation of a Receiver Data Available interrupt, which is requested by the RXIF flag. If the ADDEN bit is equal to 1, then when the data is available, an interrupt will only be generated, if the highest received bit has a high value. Note that the related interrupt enable control bit and the EMI bit of the microcontroller must also be enabled for correct interrupt generation. The highest address bit is the 9^{th} bit if the bit BNO=1 or the 8^{th} bit if the bit BNO=0. If the highest bit is high, then the received word will be defined as an address rather than data. A Data Available interrupt will be generated every time the last bit of the received word is set. If the ADDEN bit is equal to 0, then a Receive Data Available interrupt will be generated each time the RXIF flag is set, irrespective of the data last bit status. The address detection and parity functions are mutually exclusive functions. Therefore, if the address detect function is enabled, then to ensure correct operation, the parity function should be disabled by resetting the parity function enable bit PREN to zero.

ADDEN	Bit 9 if BNO=1, Bit 8 if BNO=0	UART Interrupt Generated
0	0	√
	1	√
1	0	×
	1	√

ADDEN Bit Function

UART Power Down Mode and Wake-up

When the MCU is in the Power Down Mode, the UART will cease to function. When the device enters the Power Down Mode, all clock sources to the module are shutdown. If the MCU enters the Power Down Mode while a transmission is still in progress, then the transmission will be paused until the UART clock source derived from the microcontroller is activated. In a similar way, if the MCU enters the Power Down Mode while receiving data, then the reception of data will likewise be paused. When the MCU enters the Power Down Mode, note that the USR, UCR1, UCR2, transmit and receive registers, as well as the BRG register will not be affected. It is recommended to make sure first that the UART data transmission or reception has been finished before the microcontroller enters the Power Down mode.

The UART function contains a receiver RX pin wake-up function, which is enabled or disabled by the WAKE bit in the UCR2 register. If this bit, along with the UART enable bit, UARTEN, the receiver enable bit, RXEN and the receiver interrupt bit, RIE, are all set before the MCU enters the Power Down Mode, then a falling edge on the RX pin will wake up the MCU from the Power Down Mode. Note that as it takes certain system clock cycles after a wake-up, before normal microcontroller operation resumes, any data received during this time on the RX pin will be ignored.

For a UART wake-up interrupt to occur, in addition to the bits for the wake-up being set, the global interrupt enable bit, EMI, and the UART interrupt enable bit, UARTE, must also be set. If these two bits are not set then only a wake up event will occur and no interrupt will be generated. Note also that as it takes certain system clock cycles after a wake-up before normal microcontroller resumes, the UART interrupt will not be generated until after this time has elapsed.

Interrupts

Interrupts are an important part of any microcontroller system. When an external event or an internal function such as a Timer/Event Counter requires microcontroller attention, their corresponding interrupt will enforce a temporary suspension of the main program allowing the microcontroller to direct attention to their respective needs.

The series of devices contain only one external interrupt and multiple internal interrupts. The external interrupts are controlled by the action of the external interrupt pin, while the internal interrupt is controlled by the Timer/Event Counter.

Interrupt Register

Overall interrupt control, which means interrupt enabling and request flag setting, is controlled by using the registers, INTC0 and INTC1. By controlling the appropriate enable bits in the register each individual interrupt can be enabled or disabled. Also when an interrupt occurs, the corresponding request flag will be set by the microcontroller. The global enable flag cleared to zero will disable all interrupts.

Function	Enable Bit	Request Flag
Global	EMI	—
INT Pin	INTE	INTF
Timer 0	T0E	T0F
Timer 1	T1E	T1F
I^2C	IICE	IICF
UART	UARTE	UARTF

Note: The UART Interrupt is only for the HT48R008 device.

INTC0 Register

Bit	7	6	5	4	3	2	1	0
Name	—	T1F	T0F	INTF	T1E	T0E	INTE	EMI
R/W	—	R/W	R/W	R/W	R/W	R/W	R/W	R/W
POR	—	0	0	0	0	0	0	0

Bit 7 Unimplemented, read as "0"

Bit 6 **T1F**: Timer/Event Counter 1 request flag
0: No request
1: Interrupt request

Bit 5 **T0F**: Timer/Event Counter 0 request flag
0: No request
1: Interrupt request

Bit 4 **INTF**: INT pin interrupt request flag
0: No request
1: Interrupt request

Bit 3 Unimplemented, read as “0”

Bit 2 **T0E**: Timer/Event Counter 0 interrupt control
0: Disable
1: Enable

Bit 1 **INTE**: INT interrupt control
0: Disable
1: Enable

Bit 0 **EMI**: Global interrupt control
0: Disable
1: Enable

INTC1 Register

- **HT48R004**

Bit	7	6	5	4	3	2	1	0
Name	—	—	—	IICF	—	—	—	IICE
R/W	—	—	—	R/W	—	—	—	R/W
POR	—	—	—	0	—	—	—	0

Bit 7~5 Unimplemented, read as "0"

Bit 4 **IICF**: I²C interrupt request flag
0: No request
1: Interrupt request

Bit 3~1 Unimplemented, read as "0"

Bit 0 **IICE**: I²C interrupt control
0: Disable
1: Enable

- **HT48R008**

Bit	7	6	5	4	3	2	1	0
Name	—	—	UARTF	IICF	—	—	UARTE	IICE
R/W	—	—	R/W	R/W	—	—	R/W	R/W
POR	—	—	0	0	—	—	0	0

Bit 7~6 Unimplemented, read as "0"

Bit 5 **UARTF**: UART request flag
0: No request
1: Interrupt request

Bit 4 **IICF**: I²C interrupt request flag
0: No request
1: Interrupt request

Bit 3~2 Unimplemented, read as "0"

Bit 1 **UARTE**: UART interrupt control
0: Disable
1: Enable

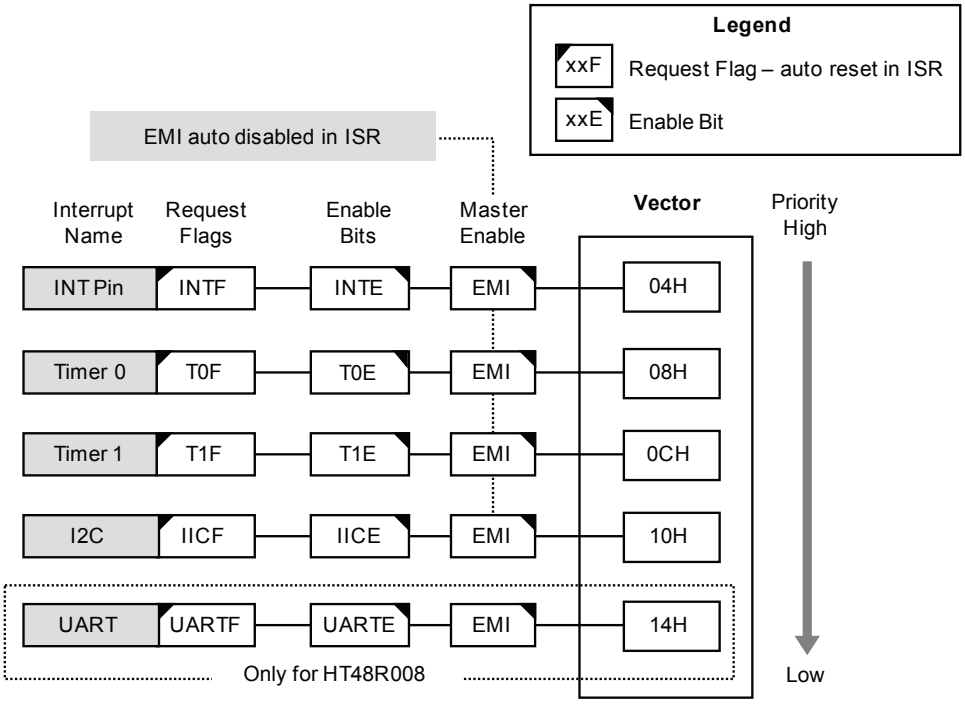
Bit 0 **IICE**: I²C interrupt control
0: Disable
1: Enable

Interrupt Operation

A Timer/Event Counter overflow or an active edge on the external interrupt pin will all generate an interrupt request by setting their corresponding request flag, if their appropriate interrupt enable bit is set. When this happens, the Program Counter, which stores the address of the next instruction to be executed, will be transferred onto the stack. The Program Counter will then be loaded with a new address which will be the value of the corresponding interrupt vector. The microcontroller will then fetch its next instruction from this interrupt vector.

The instruction at this vector will usually be a JMP statement which will jump to another section of program which is known as the interrupt service routine. Here is located the code to control the appropriate interrupt. The interrupt service routine must be terminated with a RETI instruction, which retrieves the original Program Counter address from the stack and allows the microcontroller to continue with normal execution at the point where the interrupt occurred.

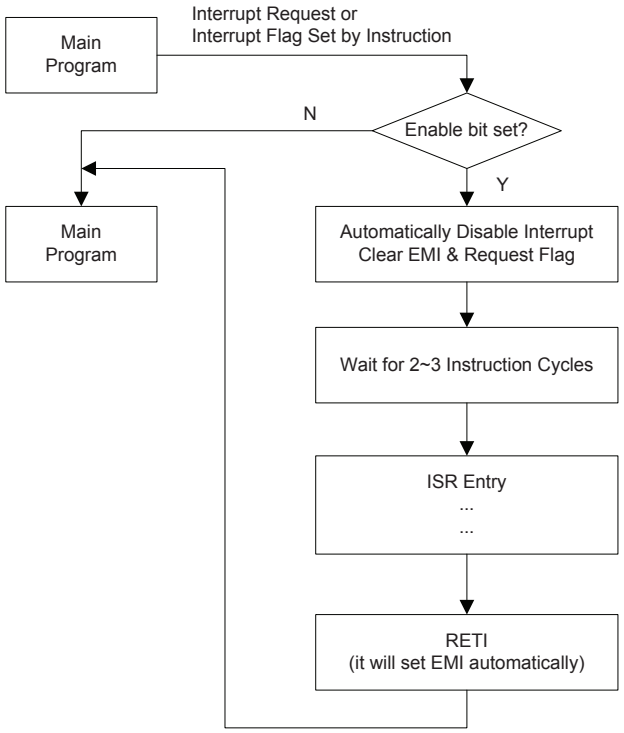
The various interrupt enable bits, together with their associated request flags, are shown in the following diagram with their order of priority.



Interrupt Scheme

Once an interrupt subroutine is serviced, all the other interrupts will be blocked, as the EMI bit will be cleared automatically. This will prevent any further interrupt nesting from occurring. However, if other interrupt requests occur during this interval, although the interrupt will not be immediately serviced, the request flag will still be recorded. If an interrupt requires immediate servicing while the program is already in another interrupt service routine, the EMI bit should be set after entering the routine, 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.

When an interrupt request is generated it takes 2 or 3 instruction cycles before the program jumps to the interrupt vector. If the devices are in the Sleep Mode and are woken up by an interrupt request then it will take 3 cycles before the program jumps to the interrupt vector.



Interrupt Flow

Interrupt Priority

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

Interrupt Source	Priority	Vector
External interrupt	1	04H
Timer/Event Counter 0 overflow	2	08H
Timer/Event Counter 1 overflow	3	0CH
I²C interrupt	4	10H
UART interrupt	5	14H

Note: The UART Interrupt is only for the HT48R008 device.

In cases where both external and internal interrupts are enabled and where an external and internal interrupt occur simultaneously, the external interrupt will always have priority and will therefore be serviced first. Suitable masking of the individual interrupts using the interrupt registers can prevent simultaneous occurrences.

External Interrupt

For an external interrupt to occur, the global interrupt enable bit, EMI, and external interrupt enable bit, INTE, must first be set. An actual external interrupt will take place when the external interrupt request flag, INTF is set, a situation that will occur when an edge transition appears on the external INT line. The type of transition that will trigger an external interrupt, whether high to low, low to high or both is determined by the INTES0 and INTES1 bits, which are bits 6 and 7 respectively in the CTRL1 control register. These two bits can also disable the external interrupt function.

INTES1	INTES0	Request Flag
0	0	External interrupt disable
0	1	Rising edge trigger
1	0	Falling edge trigger
1	1	Dual edge trigger

The external interrupt pin is pin-shared with the I/O pin PA2 and can only be used as an external interrupt pin if the corresponding external interrupt enable bit in the INTC0 register has been set and the edge trigger type has been selected using the CTRL1 register. The pin must also be set as an input by setting the corresponding PAC.2 bit in the port control register. When the interrupt is enabled, the stack is not full and a transition appears on the external interrupt pin, a subroutine call to the external interrupt vector at location 04H, will take place. When the interrupt is serviced, the external interrupt request flag, INTF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts. Note that any pull-high resistor connections on this pin will remain valid even if the pin is used as an external interrupt input.

Timer/Event Counter Interrupt

For a Timer/Event Counter interrupt to occur, the global interrupt enable bit, EMI and the corresponding timer interrupt enable bit TnE must first be set. An actual Timer/Event Counter interrupt will take place when the Timer/Event Counter request flag TnF is set, a situation that will occur when the relevant Timer/Event Counter overflows. When the interrupt is enabled, the stack is not full and a Timer/Event Counter overflow occurs, a subroutine call to the relevant timer interrupt vector, will take place. When the interrupt is serviced, the timer interrupt request flag TnF will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

UART Interrupt

The UART interrupt is only contained in the HT48R008 device. Several individual UART conditions can generate a UART interrupt. When these conditions exist, a low pulse will be generated to get the attention of the microcontroller. These conditions are a transmitter data register empty, transmitter idle, receiver data available, receiver overrun, address detect and an RX pin wake-up. To allow the program to branch to the respective interrupt vector addresses, the global interrupt enable bit, EMI, and UART interrupt enable bit, UARTE, must first be set. When the interrupt is enabled, the stack is not full and any of these conditions are created, a subroutine call to the UART Interrupt vector will take place. When the interrupt is serviced, the UART Interrupt flag, UARTF, will be automatically cleared. The EMI bit will also be automatically cleared to disable other interrupts. However, the USR register flags will be cleared automatically when certain actions are taken by the UART, the details of which are given in the UART section.

I²C Interrupt

An I²C Interrupt request will take place when the I²C Interrupt request flag, IICF, is set, which occurs when a byte of data has been received or transmitted by the I²C interface, a slave address is matched, or an I²C time-out condition has occurred. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and the Serial Interface Interrupt enable bit, IICE, must first be set. When the interrupt is enabled, the stack is not full and any of these conditions are created, a subroutine call to the respective Interrupt vector, will take place. When the I²C Interface Interrupt is serviced, the interrupt request flag, IICF, will be automatically reset and the EMI bit will be cleared to disable other interrupts.

Interrupt Wake-up Function

Each of the interrupt functions has the capability of waking up the microcontroller when in the Sleep Mode. A wake-up is generated when an interrupt request flag changes from low to high and is independent of whether the interrupt is enabled or not. Therefore, even though the devices are in the Sleep Mode and its system oscillator is stopped, situations such as external edge transitions on the external interrupt pins or timer/event counter overflow may cause their respective interrupt flag to be set high and consequently generate an interrupt. Care must therefore be taken if spurious wake-up situations are to be avoided. If an interrupt wake-up function is to be disabled then the corresponding interrupt request flag should be set high before the devices enter the Sleep Mode. The interrupt enable bits have no effect on the interrupt wake-up function.

Programming Considerations

By disabling the relevant interrupt enable bits, a requested interrupt can be prevented from being serviced, however, once an interrupt request flag is set, it will remain in this condition in the interrupt register until the corresponding interrupt is serviced or until the request flag is cleared by the application program.

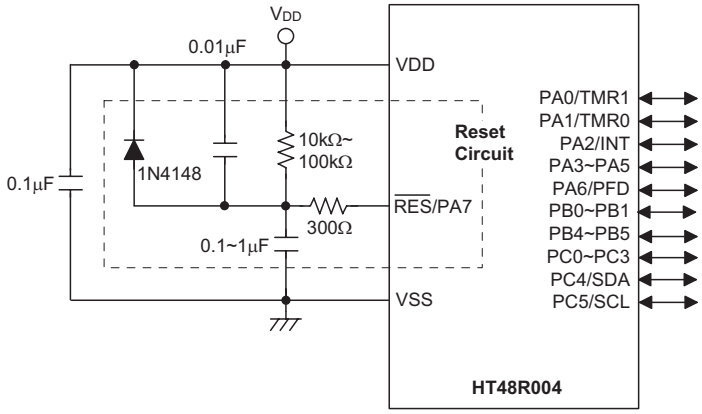
It is recommended that programs do not use the “CALL” instruction within the interrupt service subroutine. Interrupts often occur in an unpredictable manner or need to be serviced immediately. If only one stack is left and the interrupt is not well controlled, the original control sequence will be damaged once a CALL subroutine is executed in the interrupt subroutine.

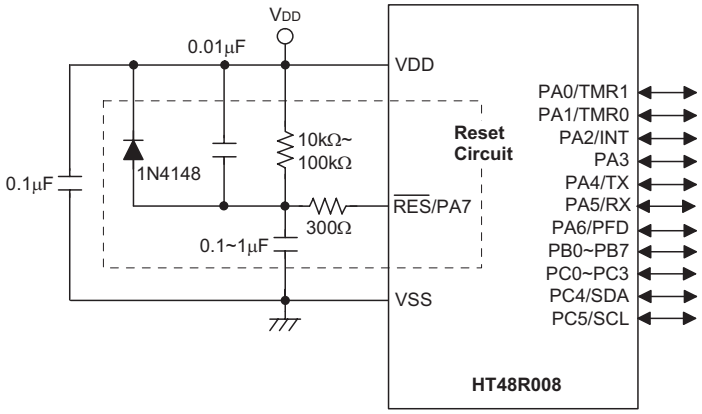
All of these interrupts have the capability of waking up the microcontroller when it is in Sleep Mode, the wake up being generated when the interrupt request flag changes from low to high. If it is required to prevent a certain interrupt from waking up the microcontroller then its respective request flag should be first set high before entering the Sleep Mode.

As only the Program Counter is pushed onto the stack, then if the contents of the accumulator, status register or other registers are altered by the interrupt service program, which may corrupt the desired control sequence, then the contents should be saved in advance.

To return from an interrupt subroutine, either a RET or RETI instruction may be executed. The RETI instruction in addition to executing a return to the main program also automatically sets the EMI bit high to allow further interrupts. The RET instruction however only executes a return to the main program leaving the EMI bit in its present zero state and therefore disabling the execution of further interrupts.

Application Circuits





Instruction Set

Introduction

Central to the successful operation of any microcontroller is its instruction set, which is a set of program instruction codes that directs the microcontroller to perform certain operations. In the case of Holtek microcontroller, a comprehensive and flexible set of over 60 instructions is provided to enable programmers to implement their application with the minimum of programming overheads.

For easier understanding of the various instruction codes, they have been subdivided into several functional groupings.

Instruction Timing

Most instructions are implemented within one instruction cycle. The exceptions to this are branch, call, or table read instructions where two instruction cycles are required. One instruction cycle is equal to 4 system clock cycles, therefore in the case of an 8MHz system oscillator, most instructions would be implemented within 0.5μs and branch or call instructions would be implemented within 1μs. Although instructions which require one more cycle to implement are generally limited to the JMP, CALL, RET, RETI and table read instructions, it is important to realize that any other instructions which involve manipulation of the Program Counter Low register or PCL will also take one more cycle to implement. As instructions which change the contents of the PCL will imply a direct jump to that new address, one more cycle will be required. Examples of such instructions would be "CLR PCL" or "MOV PCL, A". For the case of skip instructions, it must be noted that if the result of the comparison involves a skip operation then this will also take one more cycle, if no skip is involved then only one cycle is required.

Moving and Transferring Data

The transfer of data within the microcontroller program is one of the most frequently used operations. Making use of several kinds of MOV instructions, data can be transferred from registers to the Accumulator and vice-versa as well as being able to move specific immediate data directly into the Accumulator. One of the most important data transfer applications is to receive data from the input ports and transfer data to the output ports.

Arithmetic Operations

The ability to perform certain arithmetic operations and data manipulation is a necessary feature of most microcontroller applications. Within the Holtek microcontroller instruction set are a range of add and subtract instruction mnemonics to enable the necessary arithmetic to be carried out. Care must be taken to ensure correct handling of carry and borrow data when results exceed 255 for addition and less than 0 for subtraction. The increment and decrement instructions such as INC, INCA, DEC and DECA provide a simple means of increasing or decreasing by a value of one of the values in the destination specified.

Logical and Rotate Operation

The standard logical operations such as AND, OR, XOR and CPL all have their own instruction within the Holtek microcontroller instruction set. As with the case of most instructions involving data manipulation, data must pass through the Accumulator which may involve additional programming steps. In all logical data operations, the zero flag may be set if the result of the operation is zero. Another form of logical data manipulation comes from the rotate instructions such as RR, RL, RRC and RLC which provide a simple means of rotating one bit right or left. Different rotate instructions exist depending on program requirements. Rotate instructions are useful for serial port programming applications where data can be rotated from an internal register into the Carry bit from where it can be examined and the necessary serial bit set high or low. Another application which rotate data operations are used is to implement multiplication and division calculations.

Branches and Control Transfer

Program branching takes the form of either jumps to specified locations using the JMP instruction or to a subroutine using the CALL instruction. They differ in the sense that in the case of a subroutine call, the program must return to the instruction immediately when the subroutine has been carried out. This is done by placing a return instruction "RET" in the subroutine which will cause the program to jump back to the address right after the CALL instruction. In the case of a JMP instruction, the program simply jumps to the desired location. There is no requirement to jump back to the original jumping off point as in the case of the CALL instruction. One special and extremely useful set of branch instructions are the conditional branches. Here a decision is first made regarding the condition of a certain data memory or individual bits. Depending upon the conditions, the program will continue with the next instruction or skip over it and jump to the following instruction. These instructions are the key to decision making and branching within the program perhaps determined by the condition of certain input switches or by the condition of internal data bits.

Bit Operations

The ability to provide single bit operations on Data Memory is an extremely flexible feature of all Holtek microcontrollers. This feature is especially useful for output port bit programming where individual bits or port pins can be directly set high or low using either the "SET [m].i" or "CLR [m].i" instructions respectively. The feature removes the need for programmers to first read the 8-bit output port, manipulate the input data to ensure that other bits are not changed and then output the port with the correct new data. This read-modify-write process is taken care of automatically when these bit operation instructions are used.

Table Read Operations

Data storage is normally implemented by using registers. However, when working with large amounts of fixed data, the volume involved often makes it inconvenient to store the fixed data in the Data Memory. To overcome this problem, Holtek microcontrollers allow an area of Program Memory to be setup as a table where data can be directly stored. A set of easy to use instructions provides the means by which this fixed data can be referenced and retrieved from the Program Memory.

Other Operations

In addition to the above functional instructions, a range of other instructions also exist such as the "HALT" instruction for Power-down operations and instructions to control the operation of the Watchdog Timer for reliable program operations under extreme electric or electromagnetic environments. For their relevant operations, refer to the functional related sections.

Instruction Set Summary

The following table depicts a summary of the instruction set categorised according to function and can be consulted as a basic instruction reference using the following listed conventions.

Table Conventions

x: Bits immediate data
m: Data Memory address
A: Accumulator
i: 0~7 number of bits
addr: Program memory address

Mnemonic	Description	Cycles	Flag Affected
Arithmetic			
ADD A,[m]	Add Data Memory to ACC	1	Z, C, AC, OV
ADDM A,[m]	Add ACC to Data Memory	1Note	Z, C, AC, OV
ADD A,x	Add immediate data to ACC	1	Z, C, AC, OV
ADC A,[m]	Add Data Memory to ACC with Carry	1	Z, C, AC, OV
ADCM A,[m]	Add ACC to Data memory with Carry	1Note	Z, C, AC, OV
SUB A,x	Subtract immediate data from the ACC	1	Z, C, AC, OV
SUB A,[m]	Subtract Data Memory from ACC	1	Z, C, AC, OV
SUBM A,[m]	Subtract Data Memory from ACC with result in Data Memory	1Note	Z, C, AC, OV
SBC A,[m]	Subtract Data Memory from ACC with Carry	1	Z, C, AC, OV
SBCM A,[m]	Subtract Data Memory from ACC with Carry, result in Data Memory	1Note	Z, C, AC, OV
DAA [m]	Decimal adjust ACC for Addition with result in Data Memory	1Note	C
Logic Operation			
AND A,[m]	Logical AND Data Memory to ACC	1	Z
OR A,[m]	Logical OR Data Memory to ACC	1	Z
XOR A,[m]	Logical XOR Data Memory to ACC	1	Z
ANDM A,[m]	Logical AND ACC to Data Memory	1Note	Z
ORM A,[m]	Logical OR ACC to Data Memory	1Note	Z
XORM A,[m]	Logical XOR ACC to Data Memory	1Note	Z
AND A,x	Logical AND immediate Data to ACC	1	Z
OR A,x	Logical OR immediate Data to ACC	1	Z
XOR A,x	Logical XOR immediate Data to ACC	1	Z
CPL [m]	Complement Data Memory	1Note	Z
CPLA [m]	Complement Data Memory with result in ACC	1	Z
Increment & Decrement			
INCA [m]	Increment Data Memory with result in ACC	1	Z
INC [m]	Increment Data Memory	1Note	Z
DECA [m]	Decrement Data Memory with result in ACC	1	Z
DEC [m]	Decrement Data Memory	1Note	Z
Rotate			
RRA [m]	Rotate Data Memory right with result in ACC	1	None
RR [m]	Rotate Data Memory right	1Note	None
RRCA [m]	Rotate Data Memory right through Carry with result in ACC	1	C
RRC [m]	Rotate Data Memory right through Carry	1Note	C
RLA [m]	Rotate Data Memory left with result in ACC	1	None
RL [m]	Rotate Data Memory left	1Note	None
RLCA [m]	Rotate Data Memory left through Carry with result in ACC	1	C
RLC [m]	Rotate Data Memory left through Carry	1Note	C

Mnemonic	Description	Cycles	Flag Affected
Data Move			
MOV A,[m]	Move Data Memory to ACC	1	None
MOV [m],A	Move ACC to Data Memory	1Note	None
MOV A,x	Move immediate data to ACC	1	None
Bit Operation			
CLR [m].i	Clear bit of Data Memory	1Note	None
SET [m].i	Set bit of Data Memory	1Note	None
Branch			
JMP addr	Jump unconditionally	2	None
SZ [m]	Skip if Data Memory is zero	1Note	None
SZA [m]	Skip if Data Memory is zero with data movement to ACC	1Note	None
SZ [m].i	Skip if bit i of Data Memory is zero	1Note	None
SNZ [m].i	Skip if bit i of Data Memory is not zero	1Note	None
SIZ [m]	Skip if increment Data Memory is zero	1Note	None
SDZ [m]	Skip if decrement Data Memory is zero	1Note	None
SIZA [m]	Skip if increment Data Memory is zero with result in ACC	1Note	None
SDZA [m]	Skip if decrement Data Memory is zero with result in ACC	1Note	None
CALL addr	Subroutine call	2	None
RET	Return from subroutine	2	None
RET A,x	Return from subroutine and load immediate data to ACC	2	None
RETI	Return from interrupt	2	None
Table Read			
TABRD [m]	Read table (specific page) to TBLH and Data Memory	2Note	None
TABRDC [m]	Read table (current page) to TBLH and Data Memory	2Note	None
TABRDL [m]	Read table (last page) to TBLH and Data Memory	2Note	None
Miscellaneous			
NOP	No operation	1	None
CLR [m]	Clear Data Memory	1Note	None
SET [m]	Set Data Memory	1Note	None
CLR WDT	Clear Watchdog Timer	1	TO, PDF
CLR WDT1	Pre-clear Watchdog Timer	1	TO, PDF
CLR WDT2	Pre-clear Watchdog Timer	1	TO, PDF
SWAP [m]	Swap nibbles of Data Memory	1Note	None
SWAPA [m]	Swap nibbles of Data Memory with result in ACC	1	None
HALT	Enter power down mode	1	TO, PDF

Note: 1. For skip instructions, if the result of the comparison involves a skip then two cycles are required, if no skip takes place only one cycle is required.
2. Any instruction which changes the contents of the PCL will also require 2 cycles for execution.
3. For the “CLR WDT1” and “CLR WDT2” instructions the TO and PDF flags may be affected by the execution status. The TO and PDF flags are cleared after both “CLR WDT1” and “CLR WDT2” instructions are consecutively executed. Otherwise the TO and PDF flags remain unchanged.

Instruction Definition

ADC A,[m] Add Data Memory to ACC with Carry

Description: The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the Accumulator.

Operation: ACC ← ACC + [m] + C

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

ADCM A,[m] Add ACC to Data Memory with Carry

Description: The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the specified Data Memory.

Operation: [m] ← ACC + [m] + C

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

ADD A,[m] Add Data Memory to ACC

Description: The contents of the specified Data Memory and the Accumulator are added. The result is stored in the Accumulator.

Operation: ACC ← ACC + [m]

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

ADD A,x Add immediate data to ACC

Description: The contents of the Accumulator and the specified immediate data are added. The result is stored in the Accumulator.

Operation: ACC ← ACC + x

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

ADDM A,[m] Add ACC to Data Memory

Description: The contents of the specified Data Memory and the Accumulator are added. The result is stored in the specified Data Memory.

Operation: [m] ← ACC + [m]

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

AND A,[m] Logical AND Data Memory to ACC

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

Operation: ACC ← ACC "AND" [m]

Affected flag(s): Z

AND A,x Logical AND immediate data to ACC

Description: Data in the Accumulator and the specified immediate data perform a bit wise logical AND operation. The result is stored in the Accumulator.

Operation: ACC ← ACC "AND" x

Affected flag(s): Z

ANDM A,[m] Logical AND ACC to Data Memory

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

Operation: [m] ← ACC "AND" [m]

Affected flag(s): Z

CALL addr Subroutine call

Description Unconditionally calls a subroutine at the specified address. The Program Counter then increments by 1 to obtain the address of the next instruction which is then pushed onto the stack. The specified address is then loaded and the program continues execution from this new address. As this instruction requires an additional operation, it is a two cycle instruction.

Operation Stack ← Program Counter + 1
Program Counter ← addr

Affected flag(s) None

CLR [m] Clear Data Memory

Description Each bit of the specified Data Memory is cleared to 0.

Operation [m] ← 00H

Affected flag(s) None

CLR [m].i Clear bit of Data Memory

Description Bit i of the specified Data Memory is cleared to 0.

Operation [m].i ← 0

Affected flag(s) None

CLR WDT Clear Watchdog Timer

Description The TO, PDF flags and the WDT are all cleared.

Operation WDT cleared
TO ← 0
PDF ← 0

Affected flag(s) TO, PDF

CLR WDT1 Pre-clear Watchdog Timer

Description The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT2 and must be executed alternately with CLR WDT2 to have effect. Repetitively executing this instruction without alternately executing CLR WDT2 will have no effect.

Operation WDT cleared
TO ← 0
PDF ← 0

Affected flag(s) TO, PDF

CLR WDT2 Pre-clear Watchdog Timer

Description The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT1 and must be executed alternately with CLR WDT1 to have effect. Repetitively executing this instruction without alternately executing CLR WDT1 will have no effect.

Operation WDT cleared
TO ← 0
PDF ← 0

Affected flag(s) TO, PDF

CPL [m] Complement Data Memory

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

Operation [m] ← $\overline{[m]}$

Affected flag(s) Z

CPLA [m] Complement Data Memory with result in ACC

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

Operation ACC ← $\overline{[m]}$

Affected flag(s) Z

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

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

Operation [m] ← ACC + 00H or
[m] ← ACC + 06H or
[m] ← ACC + 60H or
[m] ← ACC + 66H

Affected flag(s) C

DEC [m] Decrement Data Memory

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

Operation [m] ← [m] − 1

Affected flag(s) Z

DECA [m] Decrement Data Memory with result in ACC

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

Operation ACC ← [m] − 1

Affected flag(s) Z

HALT Enter power down mode

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

Operation TO ← 0
PDF ← 1

Affected flag(s) TO, PDF

INC [m] Increment Data Memory

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

Operation [m] ← [m] + 1

Affected flag(s) Z

INCA [m] Increment Data Memory with result in ACC

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

Operation ACC ← [m] + 1

Affected flag(s) Z

JMP addr Jump unconditionally

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

Operation Program Counter ← addr

Affected flag(s) None

MOV A,[m] Move Data Memory to ACC

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

Operation ACC ← [m]

Affected flag(s) None

MOV A,x Move immediate data to ACC

Description The immediate data specified is loaded into the Accumulator.

Operation ACC ← x

Affected flag(s) None

MOV [m],A Move ACC to Data Memory

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

Operation [m] ← ACC

Affected flag(s) None

NOP No operation

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

Operation No operation

Affected flag(s) None

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

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

Operation ACC ← ACC ″OR″ [m]

Affected flag(s) Z

OR A,x Logical OR immediate data to ACC

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

Operation ACC ← ACC ″OR″ x

Affected flag(s) Z

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

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

Operation [m] ← ACC ″OR″ [m]

Affected flag(s) Z

RET Return from subroutine

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

Operation Program Counter ← Stack

Affected flag(s) None

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

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

Operation Program Counter ← Stack
ACC ← x

Affected flag(s) None

RETI Return from interrupt

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

Operation Program Counter ← Stack
EMI ← 1

Affected flag(s) None

RL [m] Rotate Data Memory left

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

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

Affected flag(s) None

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

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

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

Affected flag(s) None

RLC [m] Rotate Data Memory left through Carry

Description The contents of the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into bit 0.

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

Affected flag(s) C

RLCA [m] Rotate Data Memory left through Carry with result in ACC

Description Data in the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into the bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

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

Affected flag(s) C

RR [m] Rotate Data Memory right

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

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

Affected flag(s) None

RRA [m] Rotate Data Memory right with result in ACC

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

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

Affected flag(s): None

RRC [m] Rotate Data Memory right through Carry

Description: The contents of the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7.

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

Affected flag(s): C

RRCA [m] Rotate Data Memory right through Carry with result in ACC

Description: Data in the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged.

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

Affected flag(s): C

SBC A,[m] Subtract Data Memory from ACC with Carry

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

Operation: ACC ← ACC − [m] − C

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

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

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

Operation: [m] ← ACC − [m] − C

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

SDZ [m] Skip if decrement Data Memory is 0

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

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

Affected flag(s): None

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

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

Operation ACC ← [m] − 1
Skip if ACC=0

Affected flag(s) None

SET [m] Set Data Memory

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

Operation [m] ← FFH

Affected flag(s) None

SET [m].i Set bit of Data Memory

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

Operation [m].i ← 1

Affected flag(s) None

SIZ [m] Skip if increment Data Memory is 0

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

Operation [m] ← [m] + 1
Skip if [m]=0

Affected flag(s) None

SIZA [m] Skip if increment Data Memory is zero with result in ACC

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

Operation ACC ← [m] + 1
Skip if ACC=0

Affected flag(s) None

SNZ [m].i Skip if bit i of Data Memory is not 0

Description If bit i of the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction.

Operation Skip if [m].i ≠ 0

Affected flag(s) None

SUB A,[m] Subtract Data Memory from ACC

Description The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation ACC ← ACC − [m]

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

SUBM A,[m] Subtract Data Memory from ACC with result in Data Memory

Description The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation [m] ← ACC − [m]

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

SUB A,x Subtract immediate data from ACC

Description The immediate data specified by the code is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1.

Operation ACC ← ACC − x

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

SWAP [m] Swap nibbles of Data Memory

Description The low-order and high-order nibbles of the specified Data Memory are interchanged.

Operation [m].3~[m].0 ↔ [m].7~[m].4

Affected flag(s) None

SWAPA [m] Swap nibbles of Data Memory with result in ACC

Description The low-order and high-order nibbles of the specified Data Memory are interchanged. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.

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

Affected flag(s) None

SZ [m] Skip if Data Memory is 0

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

Operation Skip if [m]=0

Affected flag(s) None

SZA [m] Skip if Data Memory is 0 with data movement to ACC

Description The contents of the specified Data Memory are copied to the Accumulator. If the value is zero, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction.

Operation ACC ← [m]
Skip if [m]=0

Affected flag(s) None

SZ [m].i Skip if bit i of Data Memory is 0

Description If bit i of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction.

Operation Skip if [m].i=0

Affected flag(s) None

TABRD [m] Read table (specific page) to TBLH and Data Memory

Description The low byte of the program code (specific page) addressed by the table pointer pair (TBHP and TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.

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

Affected flag(s) None

TABRDC [m] Read table (current page) to TBLH and Data Memory

Description The low byte of the program code (current page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.

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

Affected flag(s) None

TABRDL [m] Read table (last page) to TBLH and Data Memory

Description The low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH.

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

Affected flag(s) None

XOR A,[m] Logical XOR Data Memory to ACC

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

Operation ACC ← ACC ″XOR″ [m]

Affected flag(s) Z

XORM A,[m] Logical XOR ACC to Data Memory

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

Operation [m] ← ACC ″XOR″ [m]

Affected flag(s) Z

XOR A,x Logical XOR immediate data to ACC

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

Operation ACC ← ACC ″XOR″ x

Affected flag(s) Z

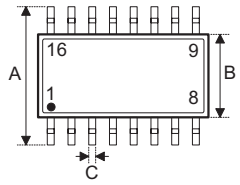
Package Information

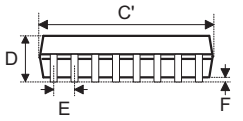
Note that the package information provided here is for consultation purposes only. As this information may be updated at regular intervals users are reminded to consult the Holtek website for the latest version of the Package/Carton Information.

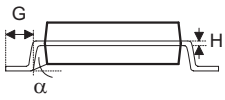
Additional supplementary information with regard to packaging is listed below. Click on the relevant section to be transferred to the relevant website page.

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

16-pin NSOP (150mil) Outline Dimensions



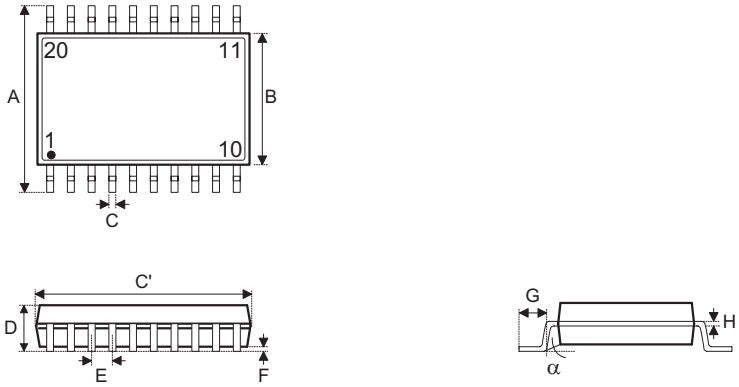




Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	—	0.236 BSC	—
B	—	0.154 BSC	—
C	0.012	—	0.020
C'	—	0.390 BSC	—
D	—	—	0.069
E	—	0.050 BSC	—
F	0.004	—	0.010
G	0.016	—	0.050
H	0.004	—	0.010
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	6.0 BSC	—
B	—	3.9 BSC	—
C	0.31	—	0.51
C'	—	9.9 BSC	—
D	—	—	1.75
E	—	1.27 BSC	—
F	0.10	—	0.25
G	0.40	—	1.27
H	0.10	—	0.25
α	0°	—	8°

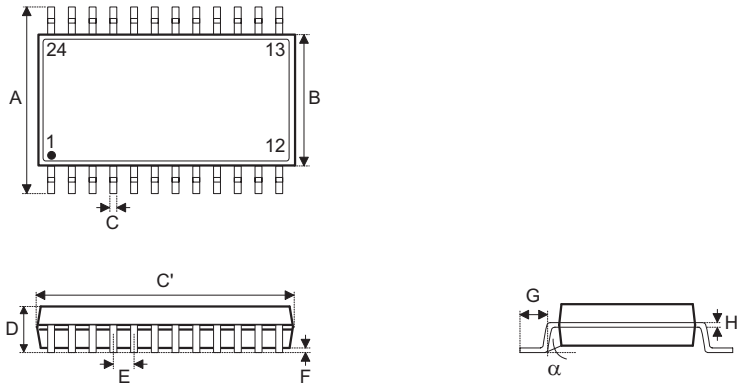
20-pin SOP (300mil) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	—	0.406 BSC	—
B	—	0.295 BSC	—
C	0.012	—	0.020
C’	—	0.504 BSC	—
D	—	—	0.104
E	—	0.050 BSC	—
F	0.004	—	0.012
G	0.016	—	0.050
H	0.008	—	0.013
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	10.30 BSC	—
B	—	7.5 BSC	—
C	0.31	—	0.51
C’	—	12.8 BSC	—
D	—	—	2.65
E	—	1.27 BSC	—
F	0.10	—	0.30
G	0.40	—	1.27
H	0.20	—	0.33
α	0°	—	8°

24-pin SOP (300mil) Outline Dimensions



Symbol	Dimensions in inch		
	Min.	Nom.	Max.
A	—	0.406 BSC	—
B	—	0.295 BSC	—
C	0.012	—	0.020
C’	—	0.606 BSC	—
D	—	—	0.104
E	—	0.050 BSC	—
F	0.004	—	0.012
G	0.016	—	0.050
H	0.008	—	0.013
α	0°	—	8°

Symbol	Dimensions in mm		
	Min.	Nom.	Max.
A	—	10.30 BSC	—
B	—	7.5 BSC	—
C	0.31	—	0.51
C’	—	15.4 BSC	—
D	—	—	2.65
E	—	1.27 BSC	—
F	0.10	—	0.30
G	0.40	—	1.27
H	0.20	—	0.33
α	0°	—	8°