HT46RU26/HT46CU26
A/D Type 8-Bit MCU with UART

Technical Document

- Tools Information
- FAQs
- Application Note
  - HA0003E Communicating between the HT48 & HT46 Series MCUs and the HT93LC46 EEPROM
  - HA0049E Read and Write Control of the HT1380
  - HA0052E Microcontroller Application - Battery Charger
  - HA0075E MCU Reset and Oscillator Circuits Application Note

Features

- Operating voltage:
  f_{SYS}=4MHz: 2.2V~5.5V
  f_{SYS}=8MHz: 3.3V~5.5V
- 48 bidirectional I/O lines
- External interrupt input shared with I/O line
- Single 8-bit Timer/Event Counter
- Two 16-bit Programmable Timer/Event Counters
- 32K ×16 Program Memory in 4 banks
- 768 × 8 byte Data Memory in 4 banks
- Integrated Crystal and RC oscillators
- Watchdog Timer function
- PFD for audio frequency generation
- Power down and wake-up functions to reduce power consumption
- Up to 0.5μs instruction cycle with 8MHz system clock at V_{DD}=5V
- 16-level subroutine nesting
- 8-channel 12-bit resolution A/D converter
- 4-channel 8-bit PWM outputs shared with I/O lines
- Real Time Clock with 8-bit prescaler
- Universal Asynchronous Receiver Transmitter – UART
- I^2C Bus slave function
- SPI Bus
- Bit manipulation instructions
- Table read instructions
- 63 powerful instructions
- All instructions executed in one or two machine cycles
- Low voltage reset function
- 48/56-pin SSOP package types

General Description

The HT46RU26/HT46CU26 is an 8-bit high performance RISC architecture microcontrollers, designed especially for applications that interface directly to analog signals, such as those from sensors.

The device includes an integrated multi-channel Analog to Digital Converter in addition to four Pulse Width Modulator outputs. The usual Holtek MCU features such as power down and wake-up functions, oscillator options, programmable frequency divider, etc. combine to ensure user applications require a minimum of external components.

With a fully integrated UART function and both SPI and I^2C interfaces, a convenient means is provided for easy and efficient interfacing to external personal computers or other external hardware.

The benefits of these combined integrated functions, in addition to low power consumption, high performance, I/O flexibility and low-cost, provide these devices with the versatility to suit a wide range of application possibilities such as sensor signal processing, motor driving, industrial control, consumer products, subsystem controllers, etc.

The HT46CU26 is under development and will be available soon.

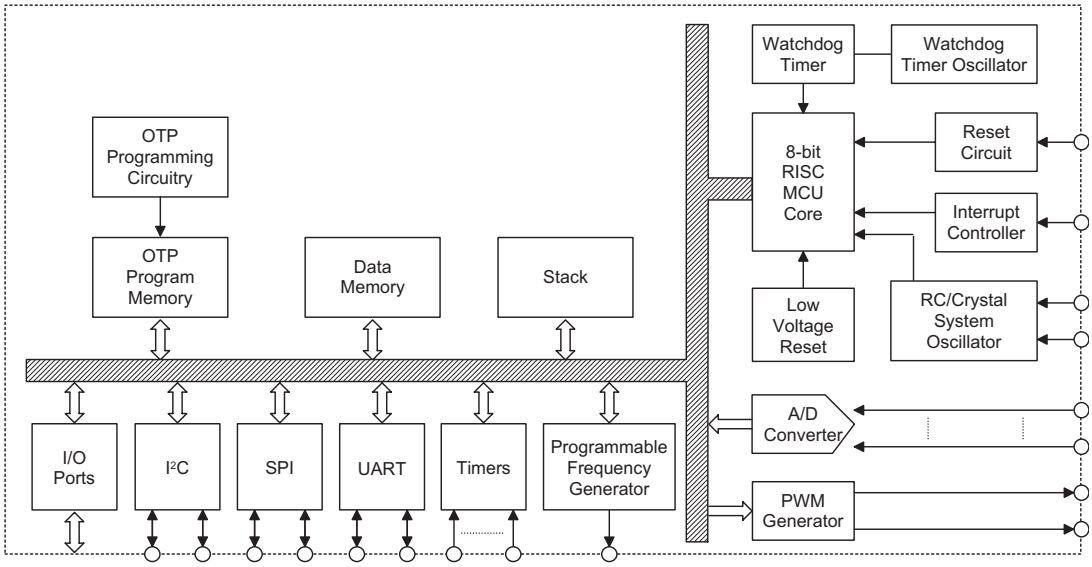
Device Types

Devices which have the letter ″R″ within their part number, indicate that they are OTP devices offering the advantages of easy and effective program updates, using the Holtek range of development and programming tools. These devices provide the designer with the means for fast and low-cost product development cycles. Devices which have the letter ″C″ within their part number indicate that they are mask version devices. These devices offer a complementary device for applications that are at a mature state in their design process and have high volume and low cost demands.

Fully pin and functionally compatible with their OTP sister devices, the mask version devices provide the ideal substitute for products which have gone beyond their development cycle and are facing cost-down demands.

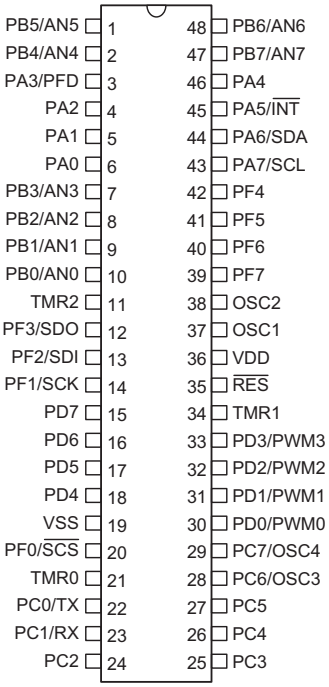
In this datasheet, for convenience, when describing device functions, only the OTP types are mentioned by name, however the same described functions also apply to the Mask type devices.

Block Diagram

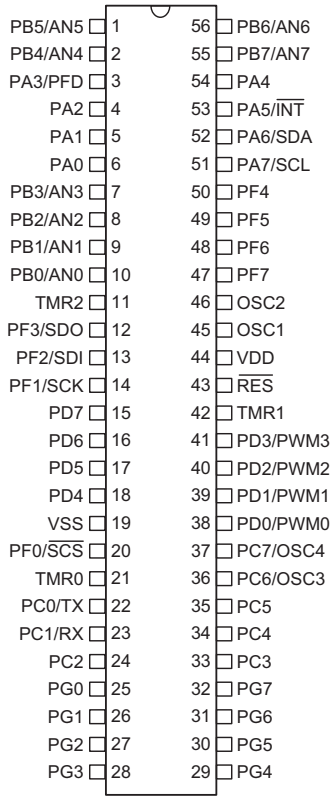


Note: This block diagram represents the OTP devices, for the mask devices there is no Device Programming Circuitry.

Pin Assignment



HT46RU26/HT46CU26
48 SSOP-A



HT46RU26/HT46CU26
56 SSOP-A

Pin Description

Pin Name	I/O	Configuration Options	Description
PA0~PA2 PA3/PFD PA4 PA5/INT PA6/SDA PA7/SCL	I/O	Pull-high Wake-up PFD I^{2}C Bus	Bidirectional 8-bit input/output port. Each bit can be configured as a wake-up input using configuration options. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if the pins have pull-high resistors. Pins PA3 and PA5 are shared with PFD and INT. Pins PA6 and PA7 are shared with I^{2}C Bus pins SDA and SCL.
PB0/AN0~ PB7/AN7	I/O	Pull-high	Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if the pins have pull-high resistors. The A/D inputs are selected via software instructions. Once selected as an A/D input, the I/O function and pull-high resistor options are disabled automatically.
PC0/TX PC1/RX PC2~PC5 PC6/OSC3 PC7/OSC4	I/O	Pull-high OSC3/OSC4	Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if the pins have pull-high resistors. Pins PC0 and PC1 are pin-shared with UART pins TX and RX. Pins PC6 and PC7 are pin-shared with RTC oscillator pins OSC3 and OSC4. The RTC oscillator function is selected via a configuration option. If the RTC oscillator option is selected then a 32768Hz crystal is connected to these two pins.

Pin Name	I/O	Configuration Options	Description
PD0/PWM0 PD1/PWM1 PD2/PWM2 PD3/PWM3 PD4~PD7	I/O	Pull-high PWM	Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if the pins have pull-high resistors. PD0~PD3 are pin-shared with PWM0~PWM3, the function of each pin is selected via configuration option.
PF0/$\overline{SCS}$ PF1/SCK PF2/SDI PF3/SDO PF4~PF7	I/O	Pull-high SIO	Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if the pins have pull-high resistors. Pins PF0~PF3 are pin-shared with SPI interface pins $\overline{SCS}$, SCK, SDO and SDI.
PG0~PG7	I/O	Pull-high	Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt Trigger input. Configuration options determine if the pins have pull-high resistors.
TMR0	I	—	Timer/Event Counter 0 Schmitt trigger input. No pull-high resistor
TMR1	I	—	Timer/Event Counter 1 Schmitt trigger input. No pull-high resistor
TMR2	I	—	Timer/Event Counter 2 Schmitt trigger input. No pull-high resistor
$\overline{RES}$	I	—	Schmitt trigger reset input. Active low
OSC1 OSC2	I O	Crystal or RC	OSC1, OSC2 are connected to an external RC network or external crystal, determined by configuration option, for the internal system clock. If the RC system clock option is selected, pin OSC2 can be used to measure the system clock at 1/4 frequency.
VSS	—	—	Negative power supply, ground
VDD	—	—	Positive power supply

Note: Individual pins can be selected to have a pull-high resistor.
Port PG does not exist on the 48-pin package

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
I_{OL} Total 150mA
Total Power Dissipation 500mW
Storage Temperature ..... –50°C to 125°C
Operating Temperature ..... –40°C to 85°C
I_{OH} Total ..... –100mA

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

D.C. Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
V_{DD}	Operating Voltage	—	f_{SYS}=4MHz	2.2	—	5.5	V
		—	f_{SYS}=8MHz	3.3	—	5.5	V
I_{DD1}	Operating Current (Crystal OSC, RC OSC)	3V	No load, f_{SYS}=4MHz, ADC Off, UART Off	—	1	2	mA
		5V		—	2.5	5	mA
I_{DD2}	Operating Current (Crystal OSC, RC OSC)	3V	No load, f_{SYS}=4MHz, ADC Off, UART On	—	1.5	3	mA
		5V		—	3	6	mA
I_{DD3}	Operating Current (Crystal OSC, RC OSC)	5V	No load, f_{SYS}=8MHz, ADC Off, UART Off	—	4	8	mA
I_{DD4}	Operating Current (Crystal OSC, RC OSC)	5V	No load, f_{SYS}=8MHz, ADC Off, UART On	—	5	10	mA
I_{DD5}	Operating Current (f_{SYS}=RTC OSC)	3V	No load, ADC Off, UART Off	—	0.3	0.6	mA
		5V		—	0.6	1	mA
I_{STB1}	Standby Current (WDT Enabled, f_S=WDT OSC)	3V	No load, system HALT, UART Off	—	2	5	μA
		5V		—	6	10	μA
I_{STB2}	Standby Current (WDT Enabled, f_S=RTC OSC)	3V	No load, system HALT	—	2.5	5	μA
		5V		—	10	20	μA
I_{STB3}	Standby Current (WDT Disabled)	3V	No load, system HALT, UART Off	—	—	1	μA
		5V		—	—	2	μA
V_{IL1}	Input Low Voltage for I/O Ports, TMR and $\overline{INT}$	—	—	0	—	$0.3V_{DD}$	V
V_{IH1}	Input High Voltage for I/O Ports, TMR and $\overline{INT}$	—	—	$0.7V_{DD}$	—	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	—	Configuration option:2.1V	1.98	2.10	2.22	V
			Configuration option:3.15V	2.98	3.15	3.32	V
			Configuration option:4.2V	3.98	4.20	4.42	V
I_{OL}	I/O Port Sink Current	3V	V_{OL}=0.1V_{DD}	4	8	—	mA
		5V	V_{OL}=0.1V_{DD}	10	20	—	mA
I_{OH}	I/O Port Source Current	3V	V_{OH}=0.9V_{DD}	–2	–4	—	mA
		5V	V_{OH}=0.9V_{DD}	–5	–10	—	mA
R_{PH}	Pull-high Resistance	3V	—	20	60	100	kΩ
		5V	—	10	30	50	kΩ
I_{ADC}	Additional Power Consumption if A/D Converter is Used	3V	—	—	0.5	1	mA
		5V		—	1.5	3	mA
DNL	ADC Differential Non-Linear	5V	t_{AD}=1μs	—	—	±2	LSB
INL	ADC Integral Non-Linear	5V	t_{AD}=1μs	—	±2.5	±4	LSB

A.C. Characteristics

Ta=25°C

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
		V_{DD}	Conditions				
f_{SYS}	System Clock	—	2.2V~5.5V	400	—	4000	kHz
		—	3.3V~5.5V	400	—	8000	kHz
f_{TIMER}	Timer I/P Frequency (TMR)	—	2.2V~5.5V	0	—	4000	kHz
		—	3.3V~5.5V	0	—	8000	kHz
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_{SST}	System Start-up Timer Period	—	Wake-up from Power Down	—	1024	—	*t_{SYS}
t_{LVR}	Low Voltage Reset Time	—	—	0.25	1	2	ms
t_{INT}	Interrupt Pulse Width	—	—	1	—	—	μs
t_{AD}	A/D Clock Period	—	—	1	—	—	μs
t_{ADC}	A/D Conversion Time	—	—	—	80	—	t_{AD}
t_{ADCS}	A/D Sampling Time	—	—	—	32	—	t_{AD}
t_{IIC}	I^2C Bus Clock Period	—	—	64	—	—	*t_{SYS}

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

System Architecture

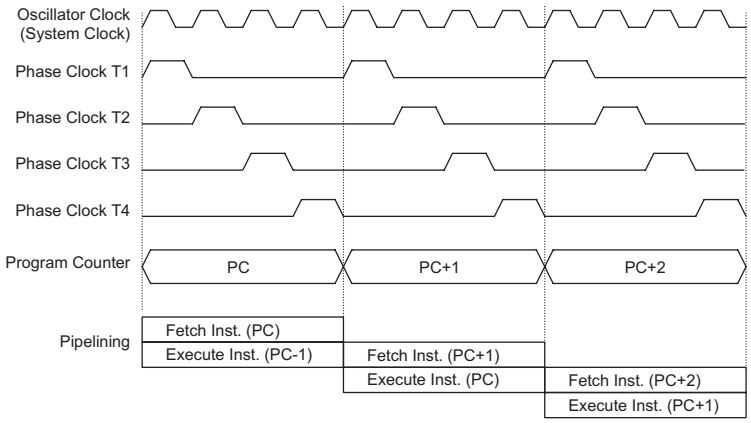
A key factor in the high-performance features of the Holtek range of microcontrollers is attributed to the internal system architecture. The range of devices take advantage of the usual features found within RISC microcontrollers providing increased speed of operation and enhanced performance. The pipelining 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 and A/D control system with maximum reliability and flexibility. This makes these devices suitable for low-cost, high-volume production for controller applications that have to interface to external analog inputs such as those from sensors.

Clocking and Pipelining

The main system clock, derived from either a Crystal/Resonator or RC 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.

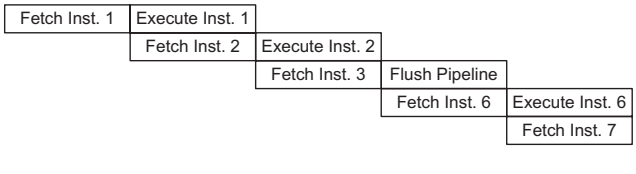
When the RC oscillator is used, OSC2 is freed for use as a T1 phase clock synchronizing pin. This T1 phase clock has a frequency of $f_{SYS}/4$ with a 1:3 high/low duty cycle.

For instructions involving branches, such as jump or call instructions, two machine cycles are required to complete instruction execution. An extra cycle is required as the program takes one cycle to first 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



System Clocking and Pipelining

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



Instruction Fetching

Program Counter

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. However, 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 jumps 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.

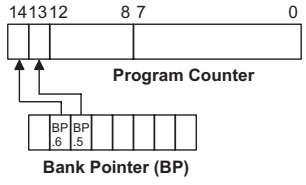
The lower byte of the Program Counter, known as the Program Counter Low register or PCL, is available for program control and is a readable and writeable register. By transferring data directly into this register, a short program jump can be executed directly, however, as only this low byte is available for manipulation, the jumps are limited to the present page of memory, that is 256 locations. When such program jumps are executed it should also be noted that a dummy cycle will be inserted.

As the Program Memory is stored in four banks, note that the Bank Selection is under the control of bits 5 and 6 of the Bank Pointer. It is these two Bank Pointer bits that control the highest address bits of the Program Counter as shown in the diagram.

Mode	Program Counter														
	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
Initial Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
External, A/D Converter or SPI Interrupt - configuration option select	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Timer/Event Counter 0 Overflow	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
Timer/Event Counter 1 Overflow	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
UART Bus Interrupt	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
I^2C Bus or SPI Interrupt - configuration option select	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0
Multi-function Interrupt	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
Skip	Program Counter + 2 (within the current bank)														
Loading PCL	PC14	PC13	PC12	PC11	PC10	PC9	PC8	@7	@6	@5	@4	@3	@2	@1	@0
Jump, Call Branch	BP.6	BP.5	#12	#11	#10	#9	#8	#7	#6	#5	#4	#3	#2	#1	#0
Return from Subroutine	S14	S13	S12	S11	S10	S9	S8	S7	S6	S5	S4	S3	S2	S1	S0

Program Counter

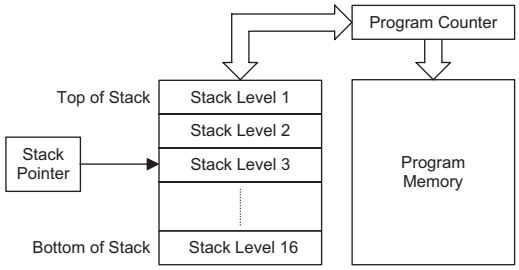
Note: Configuration Options select the function of some interrupt vectors
PC14~PC8: Current Program Counter bits
@7~@0: PCL bits
BP.5, PB.6: Bank Pointer bit.
#12~#0: Instruction code address bits
S14~S0: Stack register bits



Stack

This is a special part of the memory which is used to save the contents of the Program Counter only. The stack has 16 levels and is neither part of the data nor part of the program space, and is neither readable nor writeable. The activated level is indexed by the Stack Pointer, SP, and is neither readable nor writeable. At a subroutine call or interrupt acknowledge signal, the contents of the Program Counter are pushed onto the stack. At the end of a subroutine or an interrupt routine, signaled by a return instruction, RET or RETI, the Program Counter is restored to its previous value from the stack. After a 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 device contains One-Time Programmable, OTP, memory where users can program their application code into the device. By using the appropriate programming tools, OTP devices offer users the flexibility to freely develop their applications which may be useful during debug or for products requiring frequent upgrades or program changes. OTP devices are also applicable for use in applications that require low or medium volume production runs.

Structure

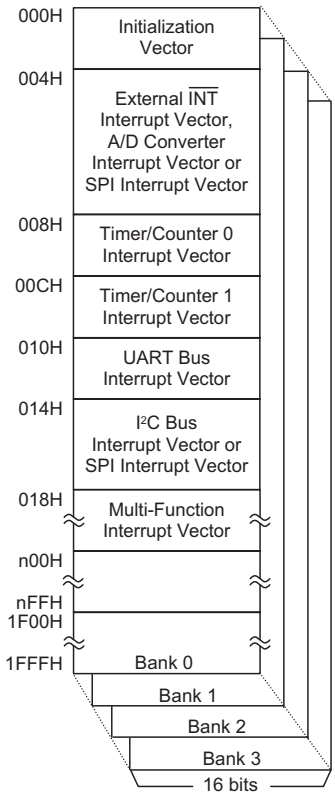
The Program Memory has a capacity of 32K by 16 bits. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries. Table data, which can be setup in any location within the Program Memory, is addressed by separate table pointer registers. The Program Memory is subdivided into four individual banks each of 8K capacity. The Program Memory Bank is selected using the Bank Pointer. Care must exercised when manipulating the Bank Pointer Register as it is also used to control the Data Memory Bank Pointer.

Special Vectors

Within the Program Memory, certain locations are reserved for special usage such as reset and interrupts.

- Location 000H
  This vector is reserved for use by the device reset for program initialisation. After a device reset is initiated, the program will jump to this location and begin execution.
- Location 004H
  This vector is used by the external interrupt, the AD converter interrupt and the SPI interrupt. One of these three interrupt sources must be chosen to use this vector location using a configuration option. If the external interrupt pin on the device goes low, an A/D conversion finishes or 8-bits of data have been transferred on the SPI bus, the program will jump to this location and begin execution if the corresponding interrupt is enabled and the stack is not full.
- Location 008H
  This internal vector is used by Timer/Event Counter 0. If a Timer/Event Counter 0 overflow occurs, the program will jump to this location and begin execution if the Timer/Event Counter 0 interrupt is enabled and the stack is not full.
- Location 00CH
  This internal vector is used by Timer/Event Counter 1. If a Timer/Event Counter 1 overflow occurs, the program will jump to this location and begin execution if the Timer/Event Counter 1 interrupt is enabled and the stack is not full.

- Location 010H
  This internal vector is used by the UART. If a UART interrupt occurs, resulting from a transmit data register empty, received data available, transmission idle, overrun error or address detected, the program will jump to this location and begin execution if the UART interrupt is enabled and the stack is not full.
- Location 014H
  This internal vector is used by the I^2C interrupt and the SPI interrupt. One of these two interrupt sources must be chosen to use this vector location using a configuration option. If an I^2C or SPI interrupt occurs, resulting from an I^2C slave address match or when 8-bits of data have been transferred on the I^2C/SPI bus, the program will jump to this location and begin execution if the corresponding interrupt is enabled and the stack is not full.
- Location 018H
  This area is reserved for the Multi-function interrupt. If a Multi-function interrupt occurs, resulting from a Timer/Event Counter 2 overflow, a real time clock time-out, or a Time base time-out, the program will jump to this location and begin execution if the corresponding interrupt is enabled and the stack is not full.



Program Memory Structure

Managing Multiple Banks

As the Program Memory is divided up into several Memory banks, there are some special considerations that have to be taken into account. First, the sections of program which are to be located into different banks are placed using the ROMBANK directive. When using the CALL instruction to call routines located in a different bank, or when using the JMP instructions to directly jump to a location in a different bank, the target bank must be first selected by correctly setting up the Bank Pointer prior to executing the CALL or JMP instruction. This of course can be achieved by directly controlling Bit 5 of the Bank Pointer, BP, but can also be done by using the BANK directive as shown in the example. Then, when a CALL or JMP instruction is executed, the Bank Pointer value stored in the BP register will be automatically loaded into the Program Counter. When the RET instruction is encountered in a subroutine called from a different bank, the program will automatically return to the original bank, however, the BP value will not be changed and will remain at the value where the subroutine is located. For this reason the BP must be carefully managed when moving between banks. The following example illustrates how to use the CALL and JMP instructions between different banks:

```
include   HT46RU26.inc
     :
     :
rombank 0 codesec0               ; define rombank 0
rombank 1 codesec1               ; define rombank 1
     :
     :
codesec0 .section at 000h 'code'  ; locates the following program section into Bank 0
     clr  bp                     ; re-initializing the BP
     jmp start
     :
     :

start:
     :
     :
lab0:
     :
     :
     mov a, BANK routb1          ; routine "routb1" is located in Bank 1
     mov bp,a                    ; load bank number for routb1 into BP
     call routb1                 ; call subroutine located in Bank 1
     clr bp                      ; program will return to this location
                                 ; after RET in Bank 1
     :                           ; but BP will retain Bank 1 value
     :                           ; so clear the BP
codesec1 .section at 000h 'code'  ; locates following program section into Bank 1
     :
     :
routb1 proc
     :
     :
     ret                         ; return program to Bank 0 but BP will
                                 ; retain Bank 1 value
routb1 endp
     :
     :
```

When managing interrupts, care has to be exercised in supervising the Bank Pointer. Irrespective of what Bank the program is presently running in, when an interrupt occurs, whether it be an external interrupt or internal interrupt, the program will immediately jump to its respective interrupt vector located in Bank 0. Note however that, although in all cases the program will jump to Bank 0, the Bank Pointer will retain its original value and not indicate Bank 0. For this reason, after entering the interrupt subroutine, in addition to the usual backup of the accumulator and status register, it is important to backup its original value immediately and also clear the Bank Pointer to indicate Bank 0 especially if other calls or jumps are encountered within Bank 0. Before the RETI instruction in the interrupt subroutine is executed, the Bank Pointer, along with the accumulator and status register, must be restored to ensure the program returns to the correct Bank and point from where the subroutine was called. The following example illustrates how interrupts can be managed:

```
include   HT46RU26.inc
     :
     :
rombank 0 codesec0               ; define rombank 0
rombank 1 codesec1               ; define rombank 1
     :
     :
codesec0 .section at 000h 'code'  ; locates the following program section into Bank 0
     clr bp                      ; clear bank pointer after power-on reset
     :
     :
     org 004h                    ; jump here from any bank when ext0 int.
                                 ; occurs - BP retains original value

     mov accbuf0,a               ; backup accumulator
     mov a,bp                    ; backup bank pointer
     clr bp                      ; clear bp to indicate Bank 0 otherwise
                                 ; original BP value will remain and give
                                 ; rise to false jmp or call addresses
     jmp ext0_int                ; jump to external 0 interrupt subroutine
     :
     :
     org 008h                    ; jump here from any bank when ext1_int.
                                 ; occurs - BP retains original value
     mov accbuf1,a               ; backup accumulator
     mov a,bp                    ; backup bank pointer
     clr bp                      ; clear bp to indicate Bank 0 otherwise
                                 ; original BP value will remain and give rise to false jmp or
```

```
                                ; call addresses
     jmp ext1_int               ; jump to timer 0 interrupt subroutine
     :
     :
     org 00Ch                   ; jump here from any bank when timer 0 int.
                                ; occurs - BP retains original value
     :
     :
ext0_int:                       ; external interrupt subroutine
     mov bp_exti,a              ; backup bank pointer
     mov  a,status              ; backup status register
     mov  statusbuf0,a          ; backup status register
     :
     :
     mov a,statusbuf0           ; restore status register
     mov status,a
     mov a,bp_exti              ; restore bank pointer
     mov bp,a
     mov  a,accbuf0             ; restore accumulator

                                ; return to main program and original calling bank
     :
     :
ext1_int:                       ; ext1_int interrupt subroutine
     mov bp_tmr0,a              ; backup bank pointer
     mov  a,status              ; backup status register
     mov  statusbuf1,a
     :
     :
     mov a,statusbuf1           ; restore status register
     mov status,a
     mov a,bp_tmr0              ; restore bank pointer
     mov bp,a
     mov a,accbuf1              ; restore accumulator
reti                            ; return to main program and original calling bank
     :
     :
```

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 pointers must first be setup by placing the lower order address of the look up data to be retrieved in the table low pointer register, TBLP, and the higher order address in the table high pointer register, TBHP. These registers define the full address of the look-up table in any bank.

After setting up the table pointers, the table data can be retrieved from the current Program Memory page or last Program Memory page using the "TABRDC[m]" or "TABRDL [m]" instructions, respectively. When these instructions are 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 following diagram illustrates the addressing/data flow of the look-up table:

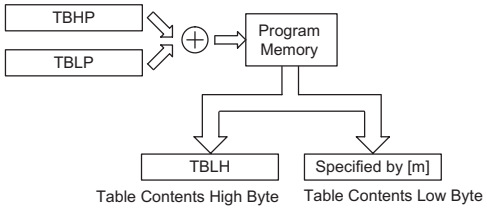


Table Program Example

The following example shows how the table pointer and table data is defined and retrieved from the microcontroller. 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 000H, however, this only indicates the offset value from the start address of Bank 1 which in this case is 2000H. The table pointer high byte is setup to have a value of 20H while the value of the table pointer low byte is setup here to have an initial value of 05H. This will ensure that the data byte read from the data table will be located at the Program Memory address 2005H, or 5 locations after the first address defined by the ORG statement. When the TABRDC [m] instruction is executed, the table data low byte which has a value of FFH, will be transferred to the user defined temp register, while the table data high byte, which has a value of 55H, will be transferred to the TBLH register.

```
include HT46RU26.inc
     :
     :
data .section 'data'
     temp db ?
     :
     :
rombank 0 codesec0                ; Bank 0 definition
rombank 1 codesec1                ; Bank 1 definition
     :
     :
codesec0 .section at 0 code
     jmp  start
     :
     org  010h
start:
     :
     :
     mov         a,020h           ; setup table high byte address
     mov         tbhp,a
     :
     :
     mov         a,005h           ; setup table low byte address
     mov         tblp,a           ; table pointer address is now 2005H
     tabrdc      temp             ; read table data from PC address 2005H
     nop                          ; FFH will be placed in the temp
                                  ; register and 55H will be placed in the TBLH register
codesec1 .section at 000h code    ; Bank 1 code located here
     org 0000h                    ; this defines the offset from the start address of Bank 1
                                  ; which is 2000H
dc 000AAh, 011BBh, 022CCh, 033DDh, 044EEh, 055FFh
     :
     :
```

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 table read instructions. If using the table read instructions, the Interrupt Service Routines may change the value of the 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.

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. 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. This General Purpose area of the Data Memory is divided into four separate banks, known as Bank 0~Bank 3. Switching between banks is accomplished by setting the Bank Pointer to the correct value.

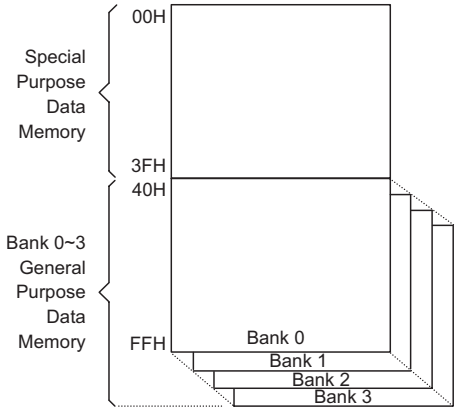
Instruction	Table Location Bits								
	b14~b8	b7	b6	b5	b4	b3	b2	b1	b0
TABRDC [m]	TBHP	@7	@6	@5	@4	@3	@2	@1	@0
TABRDL [m]	1111111	@7	@6	@5	@4	@3	@2	@1	@0

Table Location

Note: PC14~PC8: Current Program Counter bits
@7~@0: Table Pointer TBLP bits

Structure

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 is the address ″00H″. The Special Purpose Data Memory is mapped into each bank and can therefore be read from within any bank.



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 through the memory pointers MP0 and MP1.

General Purpose Data Memory

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 read and write 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. As the General Purpose Data Memory is divided into four banks it is necessary to first setup the Bank Pointer with the correct value before accessing the General Purpose Data Memory.

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″.

00H	IAR0
01H	MP0
02H	IAR1
03H	MP1
04H	BP
05H	ACC
06H	PCL
07H	TBLP
08H	TBLH
09H	RTCC
0AH	STATUS
0BH	INTC0
0CH	TMR0H
0DH	TMR0L
0EH	TMR0C
0FH	TMR1H
10H	TMR1L
11H	TMR1C
12H	PA
13H	PAC
14H	PB
15H	PBC
16H	PC
17H	PCC
18H	PD
19H	PDC
1AH	PWM0
1BH	PWM1
1CH	PWM2
1DH	PWM3
1EH	INTC1
1FH	TBHP
20H	HADR
21H	HCR
22H	HSR
23H	HDR
24H	ADRL
25H	ADRH
26H	ADCR
27H	ACSR
28H	PF
29H	PFC
2AH	PG
2BH	PGC
2CH	
2DH	TMR2
2EH	TMR2C
2FH	MFIC
30H	USR
31H	UCR1
32H	UCR2
33H	TXR/RXR
34H	BRG
35H	
36H	
37H	
38H	SBCR
39H	SBDR
3AH	
3BH	
3CH	
3DH	
3EH	
3FH	

: Unused, read as "00"

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 timers, interrupts, etc., as well as external functions such as I/O data control and A/D converter operation. The location of these registers within the Data Memory begins at the address 00H. Any unused Data Memory locations between these special function registers and the point where the General Purpose Memory begins is reserved, 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 Data Memory register space, do not actually physically exist as normal registers. The method of indirect addressing for Data Memory manipulation uses 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 Pointer, MP0 or MP1. Acting as a pair, IAR0 and MP0 can together only access data from Bank 0, while the IAR1 and MP1 register pair can access data from any bank. 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 Pointer – MP0, MP1

Two 8-bit 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 address and track data. When any operation to the relevant Indirect Addressing Registers is carried out, the actual address that the microcontroller is directed to, is the address specified by the related Memory Pointer. MP0, together with Indirect Addressing Register, IAR0, are used to access data from Bank 0 only, while MP1 and IAR1 are used to access data from any bank.

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

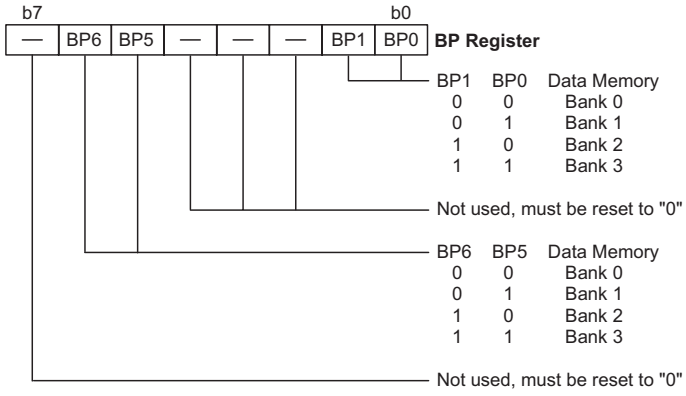
```
data .section  'data'
adres1    db ?
adres2    db ?
adres3    db ?
adres4    db ?
block     db ?
code .section at 0 'code'
org  00h

start:
    mov  a,04h              ; setup size of block
    mov  block,a
    mov  a,offset adres1    ; Accumulator loaded with first RAM address
    mov  mp0,a              ; setup 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.



Bank Pointer Register

Bank Pointer

The Program Memory and Data Memory are each divided into four separate banks. To select which bank is to be accessed a Bank Pointer register is used. Bits 5 and 6 of the Bank Pointer register select the Program Memory bank while bits 0 and 1 select the Data Memory bank. The Program and Data Memory are initialised to Bank 0 after a reset, except for the WDT time-out reset in the Power Down Mode, in which case, the bank remains unchanged. It should be noted that the Special Function Data Memory is not affected by the bank selection, which means that the Special Function Registers can be accessed from within any Data Memory bank.

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.

Look-up Table Registers – TBLP, TBHP, TBLH

These three special function registers are used to control operation of the look-up table which is stored in the Program Memory. TBLP and TBHP are the table low and high byte pointers that are used to indicate the full address where the table data is located. Their values must be setup before any table read commands are executed. Their values can be changed, for example using the ″INC″ or ″DEC″ instructions, allowing for easy table data pointing and reading. TBLH is the location where the high order byte of the table data is stored after a table read data instruction has been executed. Note that the lower order table data byte is transferred to a user defined location.

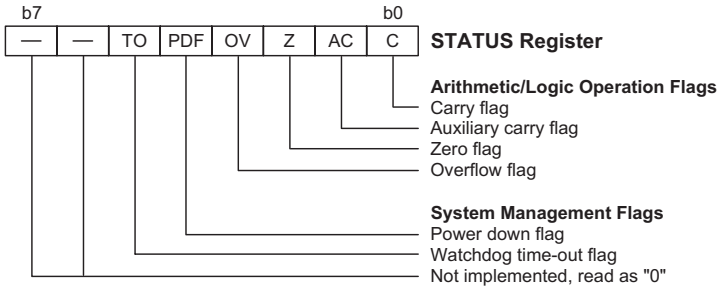
Status Register – STATUS

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.

- **C** is set if an operation results in a carry during an addition operation or if a borrow does not take place



Status Register

during a subtraction operation; otherwise C is cleared. C is also affected by a rotate through carry instruction.

- **AC** is set if an operation results in a carry out of the low nibbles in addition, or no borrow from the high nibble into the low nibble in subtraction; otherwise AC is cleared.
- **Z** is set if the result of an arithmetic or logical operation is zero; otherwise Z is cleared.
- **OV** is set if an operation results in a carry into the highest-order bit but not a carry out of the highest-order bit, or vice versa; otherwise OV is cleared.
- **PDF** is cleared by a system power-up or executing the ″CLR WDT″ instruction. PDF is set by executing the ″HALT″ instruction.
- **TO** is cleared by a system power-up or executing the ″CLR WDT″ or ″HALT″ instruction. TO is set by a WDT time-out.

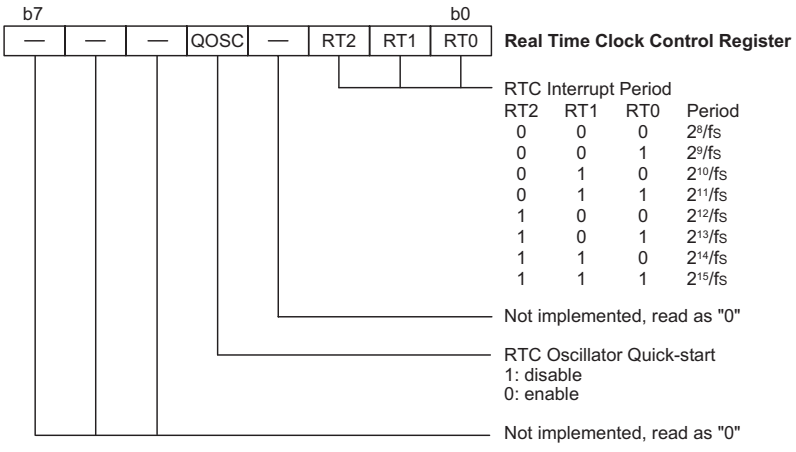
In addition, on entering an interrupt sequence or executing a subroutine call, the status register will not be pushed onto the stack automatically. If the contents of the status registers are important and if the subroutine can corrupt the status register, precautions must be taken to correctly save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

Real Time Clock Control Register – RTCC

The RTCC register controls two internal functions one of which is the Real Time Clock (RTC) interrupt, whose function is to provide an internal interrupt signal at regular fixed intervals. The driving clock for the RTC interrupt comes from the internal clock source, known as f_S, which is then further divided to give longer time values, which in turn generates the interrupt signal. The value of this division ratio is determined by the value programmed into bits 2~0, known as RT2~RT0, of the RTCC register. By writing a value directly into these RTCC register bits, time-out values from $2^8/f_S$ to $2^{15}/f_S$ can be generated. The RTCC register also controls the quick start up function of the RTC oscillator. This oscillator, which has a fixed frequency of 32768Hz, can be made to start up at a quicker rate by setting bit 4, known as the QOSC bit to ″0″. This bit will be set to a ″0″ value when the device is powered on, however, as some extra power is consumed, the QOSC bit should be set to ″1″ after about 2 seconds to reduce power consumption.

Interrupt Control Register – INTC0, INTC1

These 8-bit registers control the operation of both external and internal timer interrupts. By setting various bits within this register using standard bit manipulation instructions, the enable/disable function of each interrupt can be independently controlled. A master interrupt bit within this register, the EMI bit, acts like a global enable/disable and is used to set all of the interrupt enable bits on or off. This bit is cleared when an interrupt routine is entered to disable further interrupt and is set by executing the ″RETI″ instruction.



RTCC Register

Timer/Event Counter Registers

The device contains a single 8-bit and two 16-bit Timer/Event Counters. Each Timer/Event Counter has an associated register or register pair where the timer′s 8 or 16-bit value is located. Timer/Event Counter 2 is 8-bits wide whose register is TMR2. Timer/Event Counter 0 and 1 are 16-bits wide and have register pairs TMR0L/TMR0H and TMR1H/TMR1H. Three control registers, known as TMR0C, TMR1C and TMR2C, contains the setup information for the Timer/Event Counters, and determine in what mode the Timer/Event Counter is to be used as well as containing the timer on/off control function.

Input/Output Ports and Control Registers

Within the area of Special Function Registers, the I/O registers and their associated control registers play a prominent role. All I/O ports have a designated register correspondingly labeled as PA, PB, PC, PD, PF and PG. These labeled I/O registers are mapped to specific addresses within the Data Memory as shown in the Data Memory table, which are used to transfer the appropriate output or input data on that port. With each I/O port there is an associated control register labeled PAC, PBC, PCC, PDC, PFC and PGC, also mapped to specific addresses with the Data Memory. The control register specifies which pins of that port are set as inputs and which are set as outputs. To setup a pin as an input, the corresponding bit of the control register must be set high, for an output it must be set low. During program initialisation, it is important to first setup the control registers to specify which pins are outputs and which are inputs before reading data from or writing data to the I/O ports. One flexible feature of these registers is the ability to directly program single bits using the ″SET [m].i″ and ″CLR [m].i″ instructions. The ability to change I/O pins from output to input and vice versa by manipulating specific bits of the I/O control registers during normal program operation is a useful feature of these devices.

Pulse Width Modulator Registers – PWM0, PWM1, PWM2, PWM3

Each PWM has its own related independent control register. The 8-bit contents of these registers, defines the duty cycle value for the modulation cycle of the corresponding Pulse Width Modulator.

A/D Converter Registers – ADRL, ADRH, ADCR, ACSR

The device contains an 8-channel 12-bit A/D converter. The correct operation of the A/D requires the use of two data registers, a control register and a clock source register. These are the register locations where the digital value is placed after the completion of an analog to digital conversion cycle. The channel selection and configuration of the A/D converter is setup via the control register ADCR while the A/D clock frequency is defined by the clock source register, ACSR.

Input/Output Ports

Holtek microcontrollers offer considerable flexibility on their I/O ports. With the input or output designation of every pin fully under user program control, pull-high options for all ports and wake-up options on certain pins, the user is provided with an I/O structure to meet the needs of a wide range of application possibilities.

The microcontroller provides 48 bidirectional input/output lines labeled with port names PA, PB, PC, PD, PF and PG. 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.

Pull-high Resistors

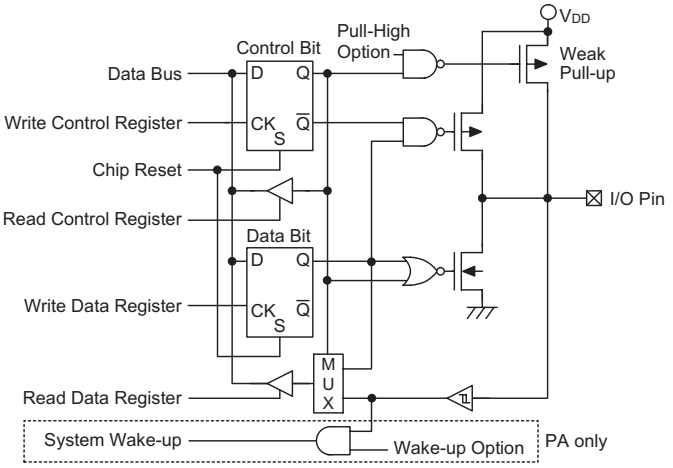
Many product applications require pull-high resistors for their switch inputs usually requiring the use of an external resistor. To eliminate the need for these external resistors, all I/O pins, when configured as an input have the capability of being connected to an internal pull-high resistor. These pull-high resistors are selectable via configuration options and are implemented using a weak PMOS transistor.

Port A Wake-up

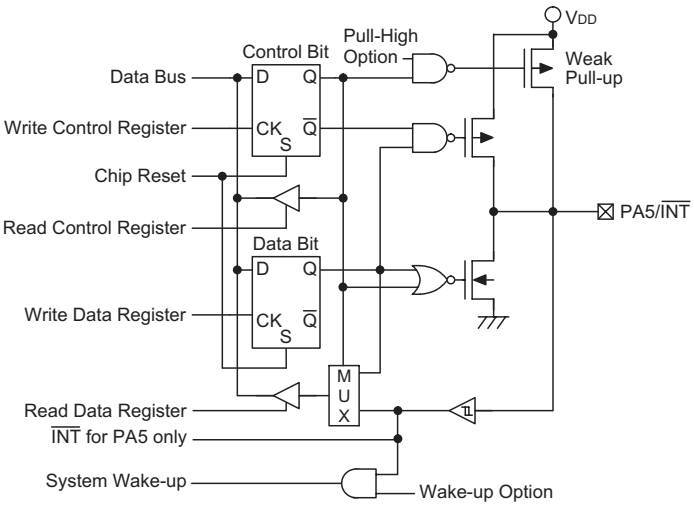
The instruction set includes a HALT instruction which if executed forces the microcontroller to enter a Power-down Mode and preserve power, a feature that is important for battery and other low-power applications. Various methods exist to wake-up the microcontroller, one of which is to change the logic condition on one of the Port A pins from high to low. After a HALT instruction forces the microcontroller into the Power-down Mode, a high to low transition on any of the configuration option selected wake-up pins on Port A will wake up the device. This function is especially suitable for applications that can be woken up via external switches. Note that each pin on Port A can be selected individually using configuration options to have this wake-up feature.

I/O Port Control Registers

Each I/O port has its own control register PAC, PBC, PCC, PDC, PFC and PGC, to control the input/output configuration. With this control register, each CMOS output or input with or without pull-high resistor structures can be reconfigured dynamically under software control. Each pin of the I/O ports is directly mapped to a bit in its associated port control register. For the I/O pin



Non-pin-shared Function Input/Output Ports



PA5 Input/Output Port

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 setup as a CMOS output. If the pin is currently setup 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.

Pin-shared Functions

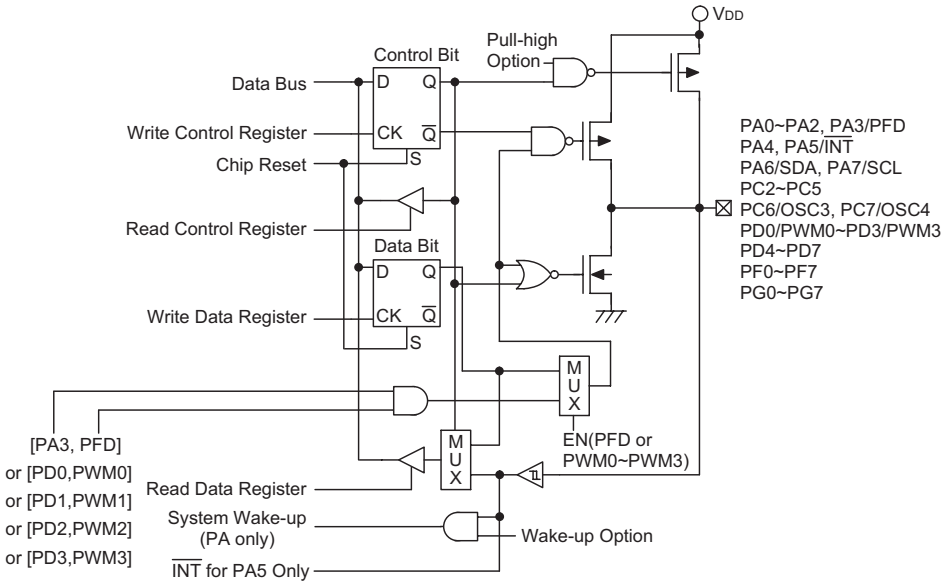
The flexibility of the device is greatly enhanced by the use of pins that have more than one function. Limited numbers of pins can force serious design constraints on designers but by supplying pins with multi-functions, many of these difficulties can be overcome. For some pins, the chosen function of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

- External Interrupt Input

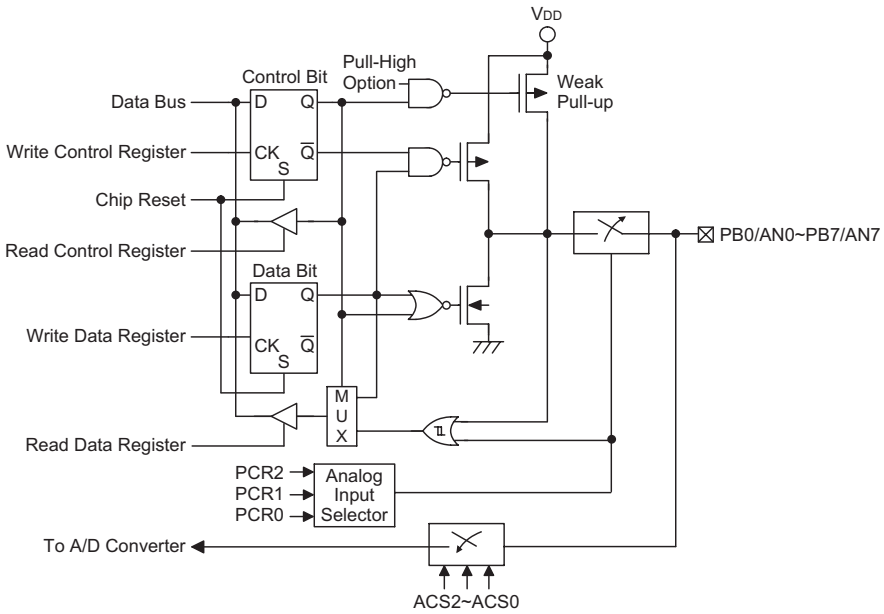
  The external interrupt pin, $\overline{INT}$, is pin-shared with the I/O pin, PA5.To be used as an external interrupt pin the external interrupt enable bit in the INTC0 register must be enabled. The corresponding bit of the port control register, PAC.5, must also setup the pin as an input for correct external interrupt operation. Any pull-high configuration options selected for this pin will remain valid if the pin is used as an external interrupt. If the PAC port control register has setup the pin as an output, then the pin will function as a normal logic output, even if the external interrupt enable bit in the INTC0 register is enabled.

- PFD Output

  The device contains a Programmable Frequency Divider, PFD, function whose single output is pin-shared with PA3. The output function of this pin is chosen via a configuration option and remains fixed after the device is programmed. Note that the corresponding bit of the port control register, PAC.3, must setup the pin as an output to enable the PFD output. If the PAC port control register has setup the pin as an input, then the



Input/Output Ports



PB Input/Output Ports

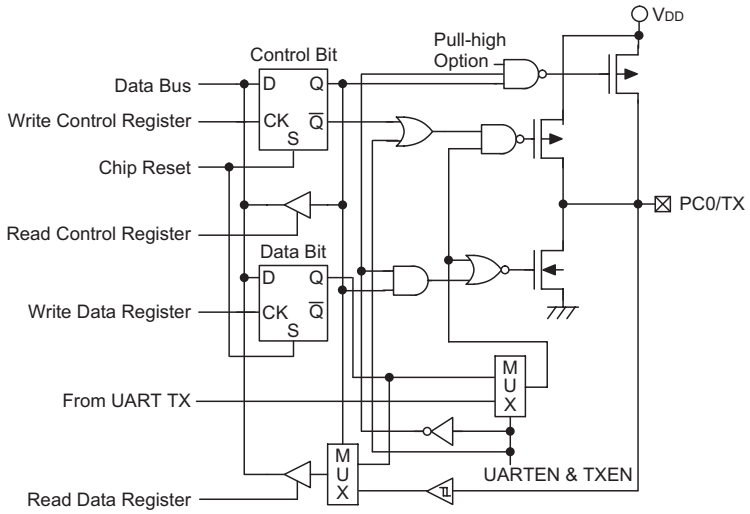
pin will function as a normal logic input with the usual pull-high option, even if the PFD configuration option has been selected.

- PWM Outputs

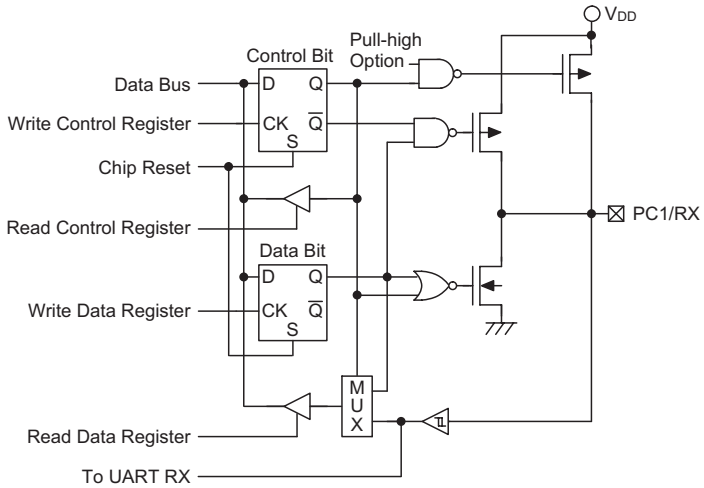
  The device contains four Pulse Width Modulator outputs PWM0, PWM1, PWM2 and PWM3, shared with pins PD0, PD1, PD2 and PD3. The PWM output functions are chosen via configuration options and remain fixed after the device is programmed. Note that the corresponding bit or bits of the port control register, PDC, must setup the pin as an output to enable the PWM output. If the PDC port control register has setup the pin as an input, then the pin will function as a normal logic input with the usual pull-high option, even if the PWM configuration option has been selected.

- A/D Inputs

  The device has 8 A/D converter inputs. All of these analog inputs are pin-shared with I/O pins on Port B. If these pins are to be used as A/D inputs and not as normal I/O pins then the corresponding bits in the A/D Converter Control Register, ADCR, must be properly set. There are no configuration options associated with the A/D function. If used as I/O pins, then full pull-high resistor configuration options remain, however if used as A/D inputs then any pull-high resistor options associated with these pins will be automatically disconnected.



PC0/TX Input/Output Ports



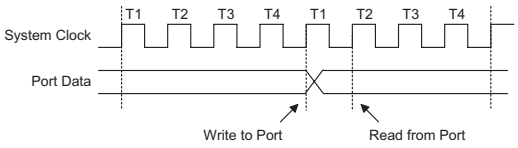
PC1/RX Input/Output Ports

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.

Programming Considerations

Within the user program, one of the first things to consider is port initialization. After a reset, all of the I/O data and port control registers will be set high. This means that all I/O pins will default to an input state, the level of which depends on the other connected circuitry and whether pull-high options have been selected. If the port control registers, PAC, PBC, PCC, PDC, PFC and PGC, are then programmed to setup some pins as outputs, these output pins will have an initial high output value unless the associated port data registers, PA, PB, PC, PD, PF and PG, 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/Write Timing

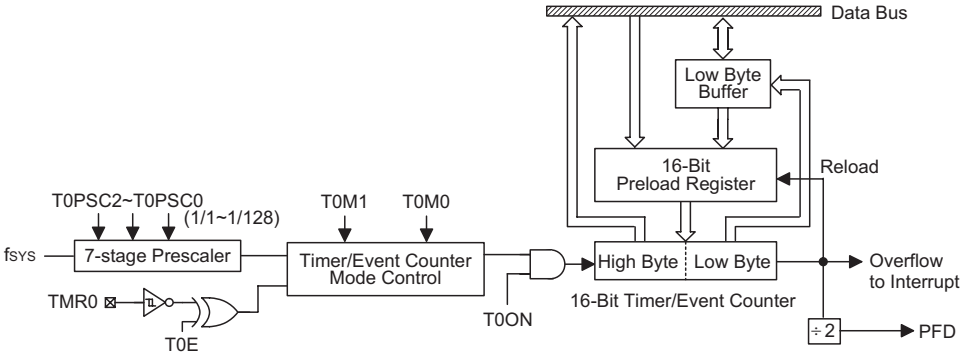
Port A has the additional capability of providing wake-up functions. When the device is in the Power Down Mode, various methods are available to wake the device up. One of these is a high to low transition of any of the Port A pins. Single or multiple pins on Port A can be setup to have this function.

The smaller package types will have some internal chip pins which are not connected to external pins. If these pins are setup as inputs they may oscillate and increase power consumption, especially notable if the device is in the Power Down Mode. It is therefore recommended that these pins should be setup as outputs, or if setup as inputs, then they should be connected to pull-high resistors.

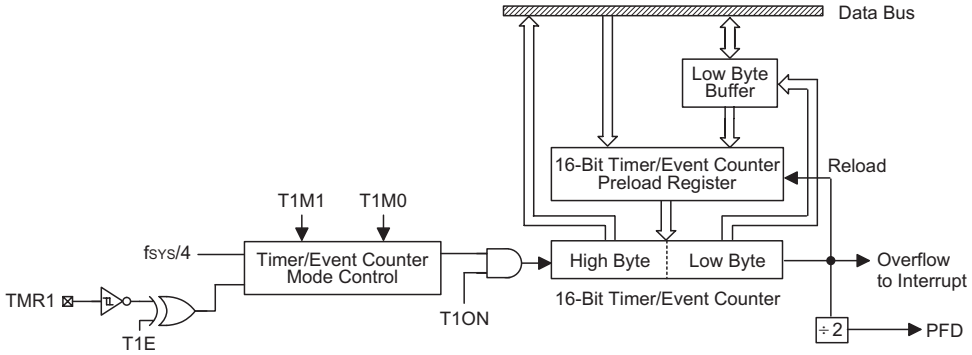
Timer/Event Counters

The provision of timers form an important part of any microcontroller giving the designer a means of carrying out time related functions. The device contains two 16-bit and one 8-bit count-up Timer/Event Counters. With three operating modes, the timers can be configured to operate as a general timer, external event counter or as a pulse width measurement device. The provision of an internal prescaler on some of the timer clock circuitry provides additional timer range.

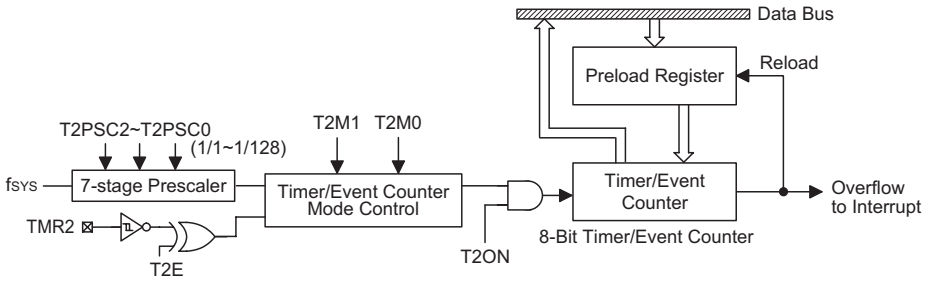
Each Timer/Event Counter has an associated register or register pair where its 8 or 16-bit value is located. Timer/Event Counter 2 is 8-bits wide whose register is TMR2. Timer/Event Counter 0 and 1 are 16-bits wide and have register pairs TMR0L/TMR0H and TMR1H/TMR1H. Three control registers, known as TMR0C, TMR1C and TMR2C, contains the setup information for the Timer/Event Counters, and determine in what mode the Timer/Event Counter is to be used as well as containing the timer on/off control function.



16-bit Timer/Event Counter 0 Structure



16-bit Timer/Event Counter 1 Structure



8-bit Timer/Event Counter 2 Structure

Configuring the Timer/Event Counter Input Clock Source

The Timer/Event Counter clock source can originate from either the system clock or from an external clock source. The system clock input source is used when the Timer/Event Counter is in the timer mode or in the pulse width measurement mode.

An external clock source is used when the Timer/Event Counter is in the event counting mode, the clock source being provided on the external timer pin TMR0, TMR1 or TMR2. Depending upon the condition of the T0E, T1E or T2E bit, each high to low, or low to high transition on the external timer pin will increment the Timer/Event Counter by one.

Timer Register – TMR0L/TMR0H, TMR1L/TMR1H, TMR2

The timer registers are special function registers located in the Special Purpose Data Memory and is the place where the actual Timer/Event Counter value is stored. For Timer/Event Counter 0 and 1, which are 16-bits wide, a pair of 8-bit registers is required to store the 16-bit value. These register pairs are known as TMR0L/TMR0H and TMR1L/TMR1H. For Timer/Event Counter 2, which is an 8-bit timer, a register known as TMR2 is provided.

The value in the timer registers 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 value of FFH for the 8-bit Timer/Event Counter or FFFFH for the 16-bit Timer/Event Counters, at which point the timer overflows and an internal interrupt signal generated. The timer value will then be reset with the initial preload register value and continue counting.

For a maximum full range count of 00H to FFH or FFFFH, the preload registers must first be cleared to all zeros. 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 not running and data is written to its preload registers, 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 registers during this period will remain in the preload registers and will only be written into the actual counter the next time an overflow occurs.

For the 16-bit Timer/Event Counters which hve both low byte and high byte timer registers, accessing these registers is carried out in a specific way. It must be noted when using instructions to preload data into the low byte timer registers, namely TMR0L or TMR1L, the data will only be placed in a low byte buffer and not directly into the low byte timer register. The actual transfer of the data into the low byte timer register is only carried out when a write to its associated high byte timer register, namely TMR0H or TMR1H, is executed. On the other hand, using instructions to preload data into the high byte timer register will result in the data being directly written to the high byte timer register. At the same time the data in the low byte buffer will be transferred into its associated low byte timer register. For this reason, the low byte timer register should be written first when preloading data into the 16-bit timer registers. It must also be noted that to read the contents of the low byte timer register, a read to the high byte timer register must be executed first to latch the contents of the low byte timer register into its associated low byte buffer. After this has been done, the low byte timer register can be read in the normal way. Note that reading the low byte timer register will result in reading the previously latched contents of the low byte buffer and not the actual contents of the low byte timer register.

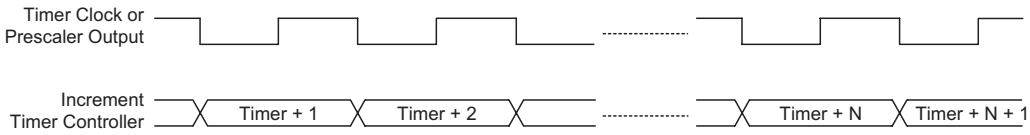
Timer Control Register – TMR0C, TMR1C, TMR2C

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 the Timer Control Registers TMR0C, TMR1C and TMR2C. It is the Timer Control Register together with its corresponding timer register that control the full operation of the Timer/Event Counter. Before the Timer/Event Counter 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 initialisation.

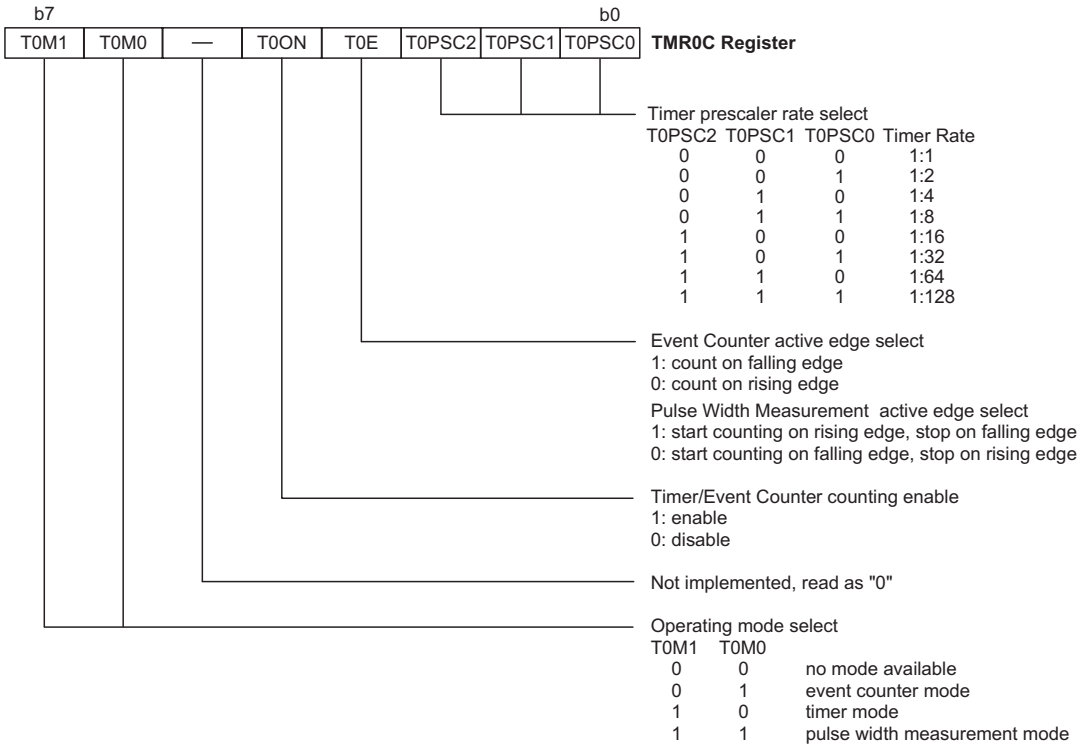
To choose which of the three modes the timer is to operate in, the timer mode, the event counting mode or the pulse width measurement mode, bits 7 and 6 of the Timer Control Register, which are known as the bit pair T0M0/T0M1, T1M0/T1M1 and T2M0/T2M1 must be set to the required logic levels. The Timer/Event Counter on/off bit, which is bit 4 of the Timer Control Register and known as T0ON, T1ON or T2ON, provides the basic on/off control of the Timer/Event Counter. Setting the bit high allows the Timer/Event Counter to run, clearing the bit stops it running. Bits 0~2 of the TMR0C and TMR2C register determine the division ratio of the input clock prescaler for the respective Timer/Event Counter. The prescaler bit settings have no effect if an external clock source is used. If the Timer/Event Counter is in the event count or pulse width measurement 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 T0E, T1E or T2E.

Configuring the Timer Mode

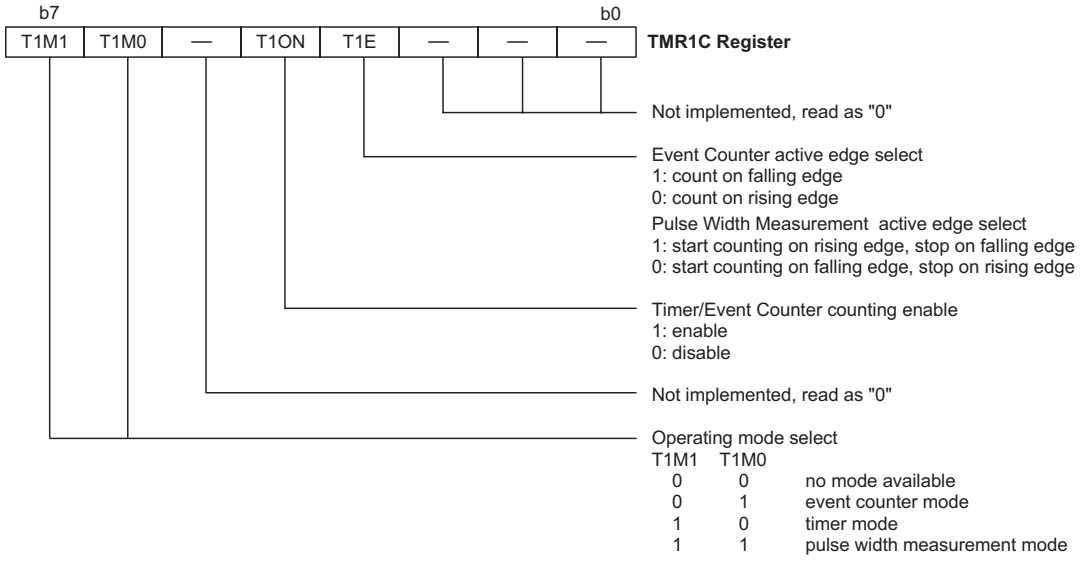
In this mode, the Timer/Event Counter can be utilised to measure fixed time intervals, providing an internal interrupt signal each time the Timer/Event Counter overflows. To operate in this mode, the Operating Mode Select bit pair, T0M1/T0M0, T1M1/T1M0 or



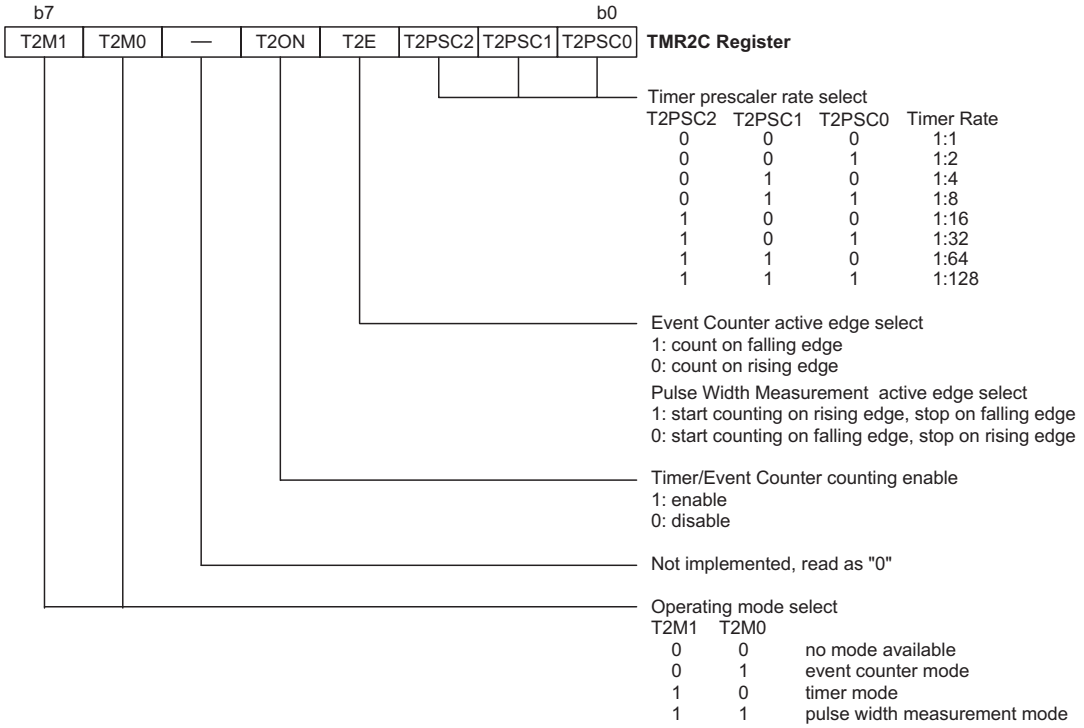
Timer Mode Timing Chart



Timer/Event Counter 0 Control Register



Timer/Event Counter 1 Control Register



Timer/Event Counter 2 Control Register

T2M1/T2M0, in the Timer Control Register must be set to the correct value as shown.

Control Register Operating Mode Select Bits for the Timer Mode	Bit7	Bit6
	1	0

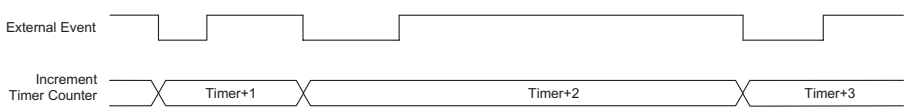
In this mode the internal clock, f_{SYS} or $f_{SYS}/4$ is used as the internal clock for the Timer/Event Counters. After the other bits in the Timer Control Register have been setup, the enable bit T0ON, T1ON or T2ON, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter to run. Each time an internal clock cycle occurs, the Timer/Event Counter increments by one. 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 Interrupt Control Register, INTC, is reset to zero.

Configuring the Event Counter Mode

In this mode, a number of externally changing logic events, occurring on the external timer pin, can be recorded by the Timer/Event Counter. the Operating Mode Select bit pair, T0M1/T0M0, T1M1/T1M0 or T2M1/T2M0, in the Timer Control Register must be set to the correct value as shown.

Control Register Operating Mode Select Bits for the Event Counter Mode	Bit7	Bit6
	0	1

In this mode, the external timer pin, TMR0, TMR1 or TMR2 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 setup, the enable bit T0ON, T1ON or T2ON, 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 T0E, T1E or T2E, 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 Active Edge Select bit 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 Interrupt Control Register, is reset to zero.



Event Counter Mode Timing Chart

It should be noted that in the event counting mode, even if the microcontroller is in the Power Down Mode, the Timer/Event Counter will continue to record externally changing logic events on the timer input pin. As a result when the timer overflows it will generate a timer interrupt and corresponding wake-up source.

Configuring the Pulse Width Measurement 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, T0M1/T0M0 or T1M1/T1M0, in the Timer Control Register must be set to the correct value as shown.

Control Register Operating Mode Select Bits for the Pulse Width Measurement Mode	Bit7	Bit6
	1	1

In this mode the internal clock, f_{SYS} or $f_{SYS}/4$ is used as the internal clock for the Timer/Event Counters. After the other bits in the Timer Control Register have been setup, the enable bit T0ON, T1ON or T2ON, 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 T0E, T1E or T2E, 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, TMR0, TMR1 or TMR2, 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 Measurement 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.

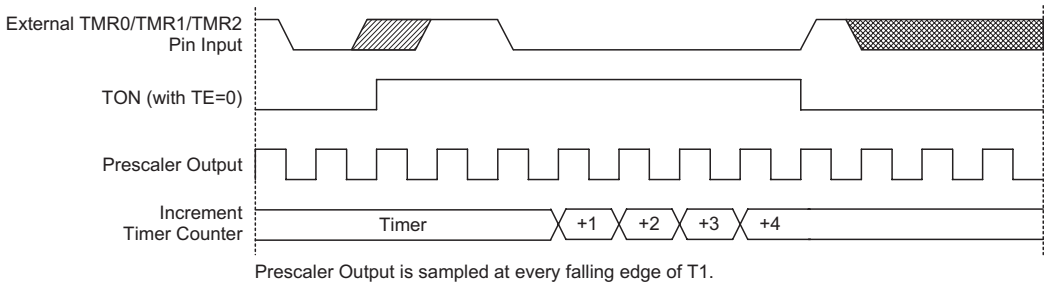
The residual value in the Timer/Event Counter, which can now be read by the program, therefore represents the length of the pulse received on the external timer pin. As the enable bit has now been reset, any further transitions on the external timer pin will be ignored. Not until the enable bit is again set high by the program can the timer begin further pulse width measurements. 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 Interrupt Control Register, is reset to zero.

Programmable Frequency Divider – PFD

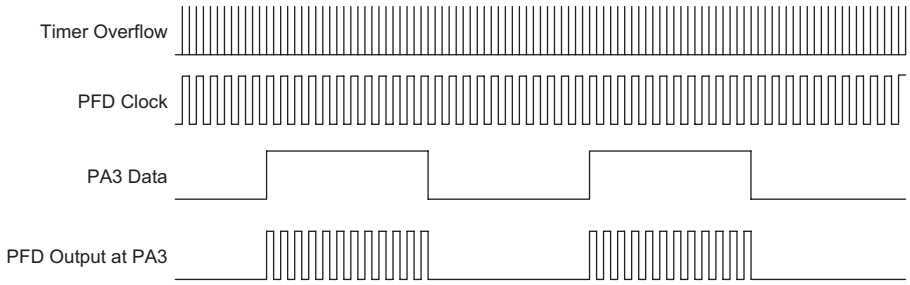
The PFD output is pin-shared with the I/O pin PA3. The PFD function is selected via configuration option, however, if not selected, the pin can operate as a normal I/O pin. The timer overflow signal is the clock source for the PFD circuit. The output frequency is controlled by loading the required values into the timer prescaler 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 the PFD output to change state. The counter will then be automatically reloaded with the preload register value and continue counting-up.

For the PFD output to function, it is essential that the corresponding bit of the Port A control register PAC bit 3 is setup as an output. If setup as an input the PFD output will not function, however, the pin can still be used as a normal input pin. The PFD output will only be activated if bit PA3 is set to ″1″. This output data bit is used as the on/off control bit for the PFD output. Note that the PFD output will be low if the PA3 output data bit is cleared to ″0″.



Prescaler Output is sampled at every falling edge of T1.

Pulse Width Measure Mode Timing Chart



PFD Output Control

Using this method of frequency generation, and if a crystal oscillator is used for the system clock, very precise values of frequency can be generated.

Prescaler

Timer/Event Counter 0 and 2 each possess a prescaler which divides the input clock source to give the Timer/Event counter a higher range. Bits 0~2 of their associated timer control register, defines the division ratio of the internal clock source. Note that the prescaler has no effect when the Timer/Event Counter is in the Event Counter Mode.

I/O Interfacing

The Timer/Event Counter, when configured to run in the event counter or pulse width measurement mode, require the use of the external pins for correct operation. As these pins are shared pins they must be configured correctly to ensure they are setup for use as a Timer/Event Counter inputs and not as normal I/O pins. This is implemented by ensuring that the mode select bits in the Timer/Event Counter control register, select either the event counter or pulse width measurement mode. Additionally the Port Control Register bits must be set high to ensure that the pins are setup as inputs. Any pull-high resistor configuration option on these pins will remain valid even if the pins are used as Timer/Event Counter inputs.

Programming Considerations

When configured to run in the timer mode, the internal system clock is used as the timer clock source and is therefore synchronized 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 measurement 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 synchronized 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. Note that setting the timer enable bit high to turn the timer on, should only be executed after the timer mode bits have been properly setup. Setting the timer enable bit high together with a mode bit modification, may lead to improper timer operation if executed as a single timer control register byte write instruction.

When the Timer/Event counter overflows, its corresponding interrupt request flag in the interrupt control register will be set. If the timer 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 device is 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 device 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 Power Down Mode.

Pulse Width Modulator

The device contains four Pulse Width Modulation, PWM, outputs. Useful for such applications such as motor speed control, the PWM function provides outputs with a fixed frequency but with a duty cycle that can be varied by setting particular values into the corresponding PWM register.

For devices with one PWM output, a single register, located in the Data Memory is assigned to the Pulse Width Modulator and is known as the PWM register. It is in these registers, that the 8-bit value, which represents the overall duty cycle of one modulation cycle of the output waveform, should be placed. To increase the PWM modulation frequency, each modulation cycle is modulated into four/two individual modulation sub-sections, known as the 6+2/7+1 mode. Note that it is only necessary to write the required modulation value into the corresponding PWM register as the subdivision of the waveform into its sub-modulation cycles is implemented automatically within the microcontroller hardware. For all devices, the PWM clock source is the system clock f_{SYS}.

This method of dividing the original modulation cycle into a further 2/4 sub-cycles enables the generation of higher PWM frequencies, which allow a wider range of applications to be served. As long as the periods of the generated PWM pulses are less than the time constants of the load, the PWM output will be suitable as such long time constant loads will average out the pulses of the PWM output. The difference between what is known as the PWM cycle frequency and the PWM modulation frequency as following table.

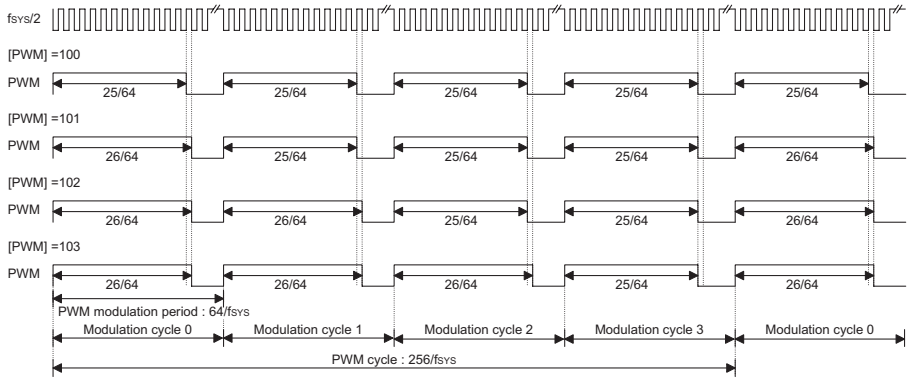
PWM Modulation Frequency	PWM Cycle Freq.	PWM Cycle Duty
$f_{SYS}/64$ for (6+2) bits mode $f_{SYS}/128$ for (7+1) bits mode	$f_{SYS}/256$	[PWM]/256

6+2 PWM Mode

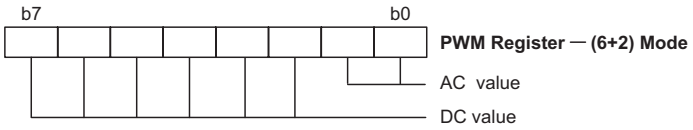
Each full PWM cycle, as it is controlled by an 8-bit PWM register, has 256 clock periods. However, in the 6+2 PWM Mode, each PWM cycle is subdivided into four individual sub-cycles known as modulation cycle 0~modulation cycle 3, denoted as "i" in the table. Each one of these four sub-cycles contains 64 clock cycles. In this mode, a modulation frequency increase by a factor of four is achieved. The 8-bit PWM, PWM0 or PWM1 register value, which represents the overall duty cycle of the PWM waveform, is divided into two groups. The first group which consists of bit2~bit7 is denoted here as the DC value. The second group which consists of bit0~bit1 is known as the AC value. In the 6+2 PWM mode, the duty cycle value of each of the four modulation sub-cycles is shown in the following table.

Parameter	AC (0~3)	DC (Duty Cycle)
Modulation cycle i (i=0~3)	i<AC	$\frac{DC+1}{64}$
	i≥AC	$\frac{DC}{64}$

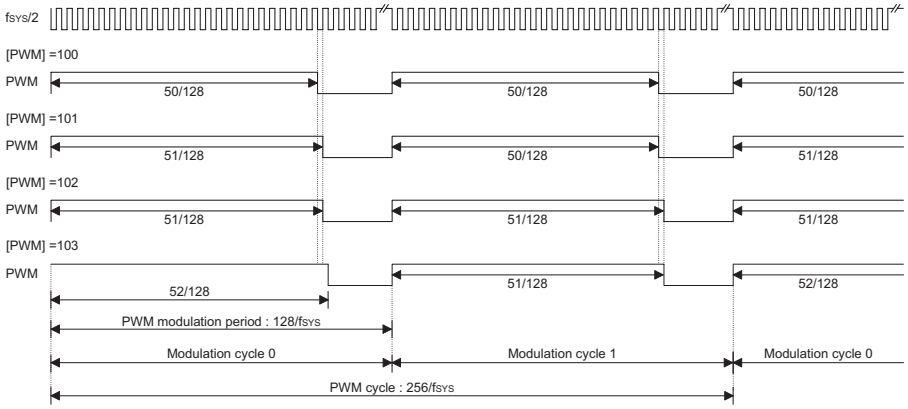
6+2 Mode Modulation Cycle Values



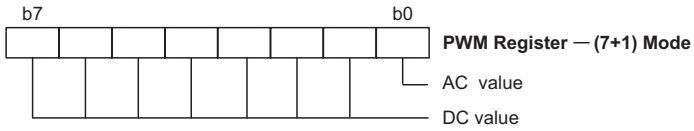
6+2 PWM Mode



PWM Registers for 6+2 Mode



7+1 PWM Mode



PWM Registers for 7+1 Mode

The following diagram illustrates the waveforms associated with the 6+2 mode of PWM operation. It is important to note how the single PWM cycle is subdivided into 4 individual modulation cycles, numbered from 0~3 and how the AC value is related to the PWM value.

7+1 PWM Mode

Each full PWM cycle, as it is controlled by an 8-bit PWM register, has 256 clock periods. However, in the 7+1 PWM mode, each PWM cycle is subdivided into two individual sub-cycles known as modulation cycle 0 ~ modulation cycle 1, denoted as "i" in the table. Each one of these two sub-cycles contains 128 clock cycles. In this mode, a modulation frequency increase of two is achieved. The 8-bit PWM register value, which represents the overall duty cycle of the PWM waveform, is divided into two groups. The first group which consists of bit1~bit7 is denoted here as the DC value. The second group which consists of bit0 is known as the AC value. In the 7+1 PWM mode, the duty cycle value of each of the two modulation sub-cycles is shown in the following table.

Parameter	AC (0~1)	DC (Duty Cycle)
Modulation cycle i (i=0~1)	i<AC	$\frac{DC+1}{128}$
	i≥AC	$\frac{DC}{128}$

7+1 Mode Modulation Cycle Values

The following diagram illustrates the waveforms associated with the 7+1 mode of PWM operation. It is important to note how the single PWM cycle is subdivided into 2 individual modulation cycles, numbered 0 and 1 and how the AC value is related to the PWM value.

PWM Output Control

On all devices, the PWM outputs are pin-shared with pins PD0~PD3. To operate as PWM outputs and not as I/O pins, the correct PWM configuration options must be selected. A "0" must also be written to the corresponding bits in the I/O port control register PDC to ensure that the required PWM output pin is setup as an output. After these two initial steps have been carried out, and of course after the required PWM value has been written into the PWM register, writing a "1" to the corresponding bit in the PD output data register will enable the PWM data to appear on the pin. Writing a "0" to the corresponding bit in the PD output data register will disable the PWM output function and force the output low. In this way, the Port D data output register can be used as an on/off control for the PWM function. Note that if the configuration options have selected the PWM function, but a "1" has been written to its corresponding bit in the PDC control register to configure the pin as an input, then the pin can still function as a normal input line, with pull-high resistor options.

PWM Programming Example

The following sample program shows how the PWM outputs are setup and controlled. Before use the corresponding PWM output configuration options must first be selected.

```
mov a,64h       ; setup PWM value of 100 decimal which is 64H
mov pwm0,a
clr pdc.0       ; setup pin PD0 as an output
set pd.0        ; PD.0=1; enable the PWM output
 :   :
 :   :
clr pd.0        ; disable the PWM output – PD0 will remain low
```

Analog to Digital Converter

The need to interface to real world analog signals is a common requirement for many electronic systems. However, to properly process these signals by a microcontroller, they must first be converted into digital signals by A/D converters. By integrating the A/D conversion electronic circuitry into the microcontroller, the need for external components is reduced significantly with the corresponding follow-on benefits of lower costs and reduced component space requirements.

A/D Overview

Each of the devices contains a 8-channel analog to digital converter which can directly interface to external analog signals, such as that from sensors or other control signals and convert these signals directly into 12-bit digital value.

The following diagram shows the overall internal structure of the A/D converter, together with its associated registers.

A/D Converter Data Registers – ADRL, ADRH

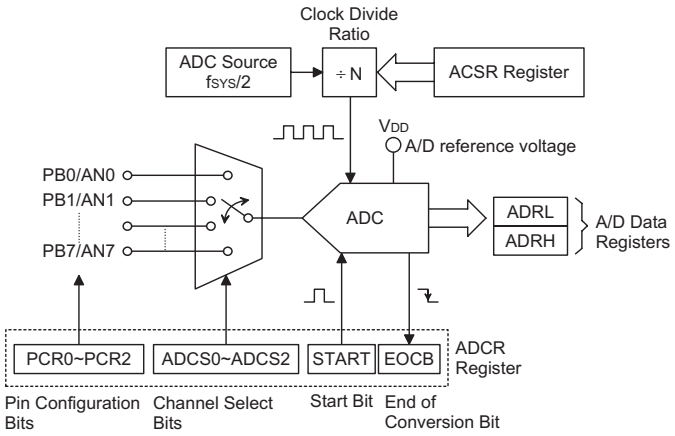
After the conversion process takes place, these registers can be directly read by the microcontroller to obtain the digitised conversion value. For devices which use two A/D Converter Data Registers, note that only the high byte register ADRH utilises its full 8-bit contents. The low byte register utilises only 4 bit of its 8-bit contents as it contains only the lowest bit of the 9-bit converted value.

A/D Converter Control Register – ADCR

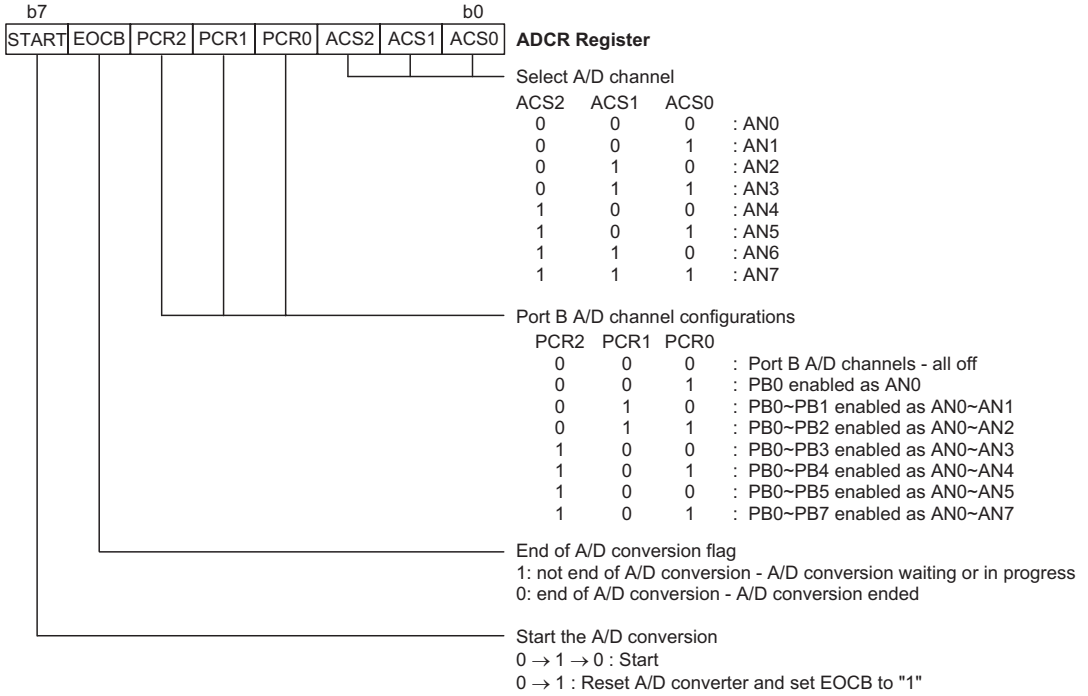
To control the function and operation of the A/D converter, a control register known as ADCR is provided. This 8-bit register defines functions such as the selection of which analog channel is connected to the internal A/D converter, which pins are used as analog inputs and which are used as normal I/Os as well as controlling the start function and monitoring the A/D converter end of conversion status.

One section of this register contains the bits ACS2~ACS0 which define the channel number. As each of the devices contains only one actual analog to digital converter circuit, each of the individual 4 analog inputs must be routed to the converter. It is the function of the ACS2~ACS0 bits in the ADCR register to determine which analog channel is actually connected to the internal A/D converter. Note that the ACS2 bit must always be assigned a zero value.

The ADCR control register also contains the PCR2~PCR0 bits which determine which pins on Port B are used as analog inputs for the A/D converter and which pins are to be used as normal I/O pins. Note that if



A/D Converter Structure



A/D Converter Control Register

the PCR2~PCR0 bits are all set to zero, then all the Port B pins will be setup as normal I/Os and the internal A/D converter circuitry will be powered off to reduce the power consumption.

The START bit in the ADCR register is used to start and reset the A/D converter. When the microcontroller sets this bit from low to high and then low again, an analog to digital conversion cycle will be initiated. When the START bit is brought from low to high but not low again, the EOCB bit in the ADCR register will be set to a ″1″ and the analog to digital converter will be reset. It is the START bit that is used to control the overall on/off operation of the internal analog to digital converter.

The EOCB bit in the ADCR register is used to indicate when the analog to digital conversion process is complete. This bit will be automatically set to ″0″ by the microcontroller after a conversion cycle has ended. In addition, the corresponding A/D interrupt request flag will be set in the interrupt control register, and if the interrupts are enabled, an appropriate internal interrupt signal will be generated. This A/D internal interrupt signal will direct the program flow to the associated A/D internal interrupt address for processing. If the A/D internal interrupt is disabled, the microcontroller can be used to poll the EOCB bit in the ADCR register to check whether it has been cleared as an alternative method of detecting the end of an A/D conversion cycle.

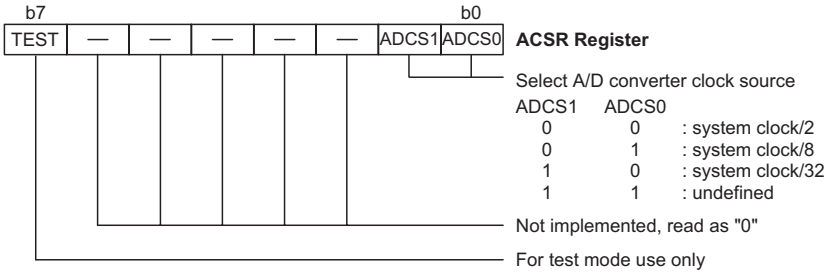
A/D Converter Clock Source Register – ACSR

The clock source for the A/D converter, which originates from the system clock f_{SYS}, is first divided by a division ratio, the value of which is determined by the ADCS1 and ADCS0 bits in the ACSR register.

Although the A/D clock source is determined by the system clock f_{SYS}, and by bits ADCS1 and ADCS0, there are some limitations on the maximum A/D clock source speed that can be selected.

Register	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
ADRL	D3	D2	D1	D0	—	—	—	—
ADRH	D11	D10	D9	D8	D7	D6	D5	D4

A/D Data Register



A/D Converter Clock Source Register

f_{SYS}	A/D Clock Period (t_{AD})			
	ADCS1, ADCS0=00 (f_{SYS}/2)	ADCS1, ADCS0=01 (f_{SYS}/8)	ADCS1, ADCS0=10 (f_{SYS}/32)	ADCS1, ADCS0=11
1MHz	2μs	8μs	32μs	Undefined
2MHz	1μs	4μs	16μs	Undefined
4MHz	500ns*	2μs	8μs	Undefined
8MHz	250ns*	1μs	4μs	Undefined

A/D Clock Period Examples

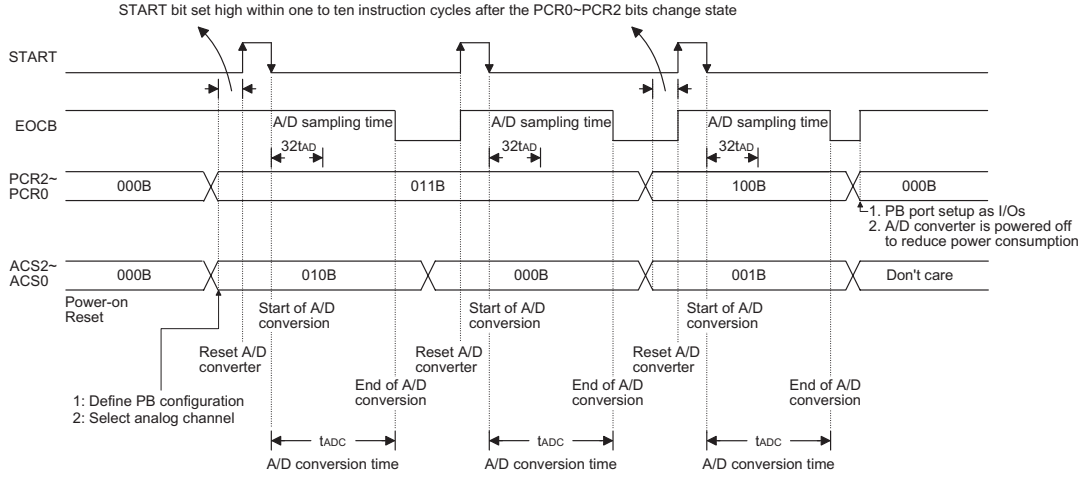
A/D Input Pins

All of the A/D analog input pins are pin-shared with the I/O pins on Port B. Bits PCR2~PCR0 in the ADCR register, not configuration options, determine whether the input pins are setup as normal Port B input/output pins or whether they are setup as analog inputs. In this way, pins can be changed under program control to change their function from normal I/O operation to analog inputs and vice versa. Pull-high resistors, which are setup through configuration options, apply to the input pins only when they are used as normal I/O pins, if setup as A/D inputs the pull-high resistors will be automatically disconnected. Note that it is not necessary to first setup the A/D pin as an input in the PBC port control register to enable the A/D input, when the PCR2~PCR0 bits enable an A/D input, the status of the port control register will be overridden.

The VDD power supply pin is used as the A/D converter reference voltage, and as such analog inputs must not be allowed to exceed this value. Appropriate measures should also be taken to ensure that the VDD pin remains as stable and noise free as possible.

Initialising the A/D Converter

The internal A/D converter must be in a special way. Each time the Port B A/D channel selection bits are modified by the program, the A/D converter must be re-initialised. If the A/D converter is not initialised after the channel selection bits are changed, the EOCB flag may have an undefined value, which may produce a false end of conversion signal. To initialise the A/D converter after the channel selection bits have changed, then, within a time frame of one to ten instruction cycles, the START bit



Note: A/D clock must be f_{SYS}/2, f_{SYS}/8 or f_{SYS}/32

A/D Conversion Timing

in the ADCR register must first be set high and then immediately cleared to zero. This will ensure that the EOCB flag is correctly set to a high condition.

Summary of A/D Conversion Steps

The following summarizes the individual steps that should be executed in order to implement an A/D conversion process.

- Step 1
  Select the required A/D conversion clock by correctly programming bits ADCS1 and ADCS0 in the ACSR register.
- Step 2
  Select which channel is to be connected to the internal A/D converter by correctly programming the ACS2~ACS0 bits which are also contained in the ADCR register.
- Step 3
  Select which pins on Port B are to be used as A/D inputs and configure them as A/D input pins by correctly programming the PCR2~PCR0 bits in the ADCR register. Note that this step can be combined with Step 2 into a single ADCR register programming operation.
- Step 4
  If the interrupts are to be used, the interrupt control registers must be correctly configured to ensure the A/D converter interrupt function is active. The master interrupt control bit, EMI, in the INTC interrupt control register must be set to ″1″ and the A/D converter interrupt bit, EADI, in the INTC register must also be set to ″1″.
- Step 5
  The analog to digital conversion process can now be initialised by setting the START bit in the ADCR register from ″0″ to ″1″ and then to ″0″ again. Note that this bit should have been originally set to ″0″.
- Step 6
  To check when the analog to digital conversion process is complete, the EOCB bit in the ADCR register can be polled. The conversion process is complete when this bit goes low. When this occurs the A/D data registers ADRL and ADRH can be read to obtain the conversion value. As an alternative method if the interrupts are enabled and the stack is not full, the program can wait for an A/D interrupt to occur.

Note: When checking for the end of the conversion process, if the method of polling the EOCB bit in the ADCR register is used, the interrupt enable step above can be omitted.

The following timing diagram shows graphically the various stages involved in an analog to digital conversion process and its associated timing.

The setting up and operation of the A/D converter function is fully under the control of the application program as there are no configuration options associated with the A/D converter. After an A/D conversion process has been initiated by the application program, the microcontroller internal hardware will begin to carry out the conversion, during which time the program can continue with other functions.

Programming Considerations

When programming, special attention must be given to the A/D channel selection bits in the ADCR register. If these bits are all cleared to zero no external pins will be selected for use as A/D input pins allowing the pins to be used as normal I/O pins. When this happens the power supplied to the internal A/D circuitry will be reduced resulting in a reduction of supply current. This ability to reduce power by turning off the internal A/D function by clearing the A/D channel selection bits may be an important consideration in battery powered applications.

Another important programming consideration is that when the A/D channel selection bits change value the A/D converter must be re-initialised. This is achieved by pulsing the START bit in the ADCR register immediately after the channel selection bits have changed state. The exception to this is where the channel selection bits are all cleared, in which case the A/D converter is not required to be re-initialised.

A/D Programming Example

The following two programming examples illustrate how to setup and implement an A/D conversion. In the first example, the method of polling the EOCB bit in the ADCR register is used to detect when the conversion cycle is complete, whereas in the second example, the A/D interrupt is used to determine when the conversion is complete.

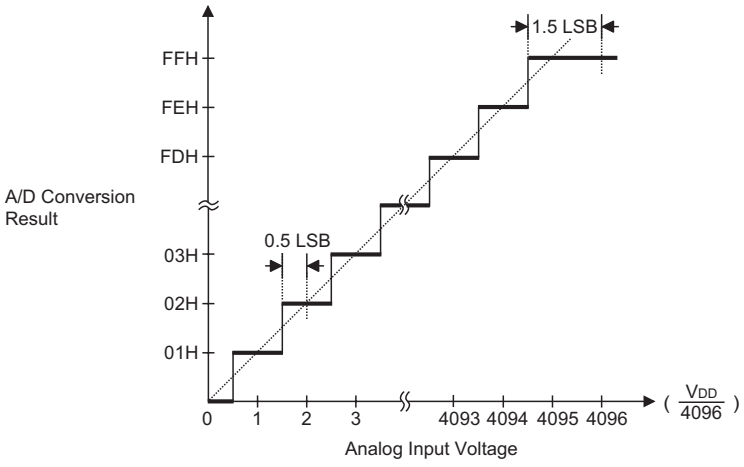
Example: using an EOCB polling method to detect the end of conversion

```
    clr  EADI                  ; disable ADC interrupt
    mov  a,00000001B
    mov  ACSR,a                ; setup the ACSR register to select fSYS/8 as the A/D clock
    mov  a,00100000B           ; setup ADCR register to configure Port as A/D inputs
    mov  ADCR,a                ; and select AN0 to be connected to the A/D converter
         :
         :                     ; As the Port B channel bits have changed the following START
                               ; signal (0-1-0) must be issued within 10 instruction cycles
         :
Start_conversion:
    clr  START
    set  START                 ; reset A/D
    clr  START                 ; start A/D
Polling_EOC:
    sz   EOCB                  ; poll the ADCR register EOCB bit to detect end
                               ; of A/D conversion
    jmp  polling_EOC           ; continue polling
    mov  a,ADRL                ; read conversion result value from the ADRL register
    mov  adrl_buffer,a         ; save result to user defined memory
    mov  a,ADRH                ; read conversion result value from the ADRH register
    Mov  adrh_buffer,a         ; save result to user defined memory
         :
         :
    jmp  start_conversion      ; start next A/D conversion
```

A/D Transfer Function

As the device contain an 12-bit A/D converter, their full-scale converted digitized value is equal to FFFH. Since the full-scale analog input value is equal to the voltage, this gives a single bit analog input value of $V_{DD}/4096$. The following graphs show the ideal transfer function between the analog input value and the digitised output value for the A/D converters.

Note that to reduce the quantisation error, a 0.5 LSB offset is added to the A/D Converter input. Except for the digitised zero value, the subsequent digitised values will change at a point 0.5 LSB below where they would change without the offset, and the last full scale digitised value will change at a point 1.5 LSB below the V_{DD} level.



Ideal A/D Transfer Function

I²C Bus Serial Interface

The I²C bus is a bidirectional 2-wire communication interface originally developed by Philips Semiconductors. The possibility of transmitting and receiving data on only 2 lines offers many new application possibilities for microcontroller based applications and for this reason, an I²C bus is implemented in this device. The I²C bus function is selectable via a configuration option.

There are two lines associated with the I²C bus, the first is known as SDA and is the Serial Data line, the second is known as SCL line and is the Serial Clock line. 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. Four registers exist to control the I²C bus and its associated data transfer, HADR, HCR, HSR and HDR. Communication on the I²C bus requires four steps, a START signal, a slave address transmission, a data transmission and finally a STOP signal.

I²C Bus Slave Address Register – HADR

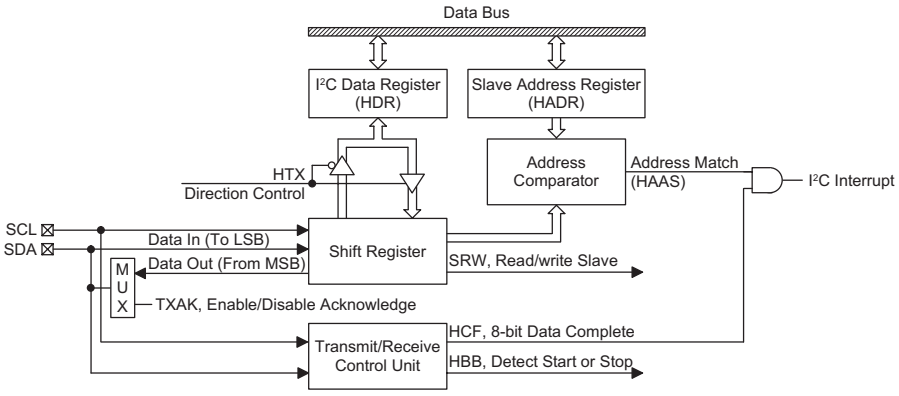
The HADR register is the location where the slave address of the microcontroller is stored. Bits 1~7 of the HADR register define the microcontroller slave address. Bit 0 is not implemented. When a master device, which is connected to the I²C bus, sends out an address, which matches the slave address in the HADR register, the microcontroller slave device will be selected.

I²C Bus Input/Output Data Register – HDR

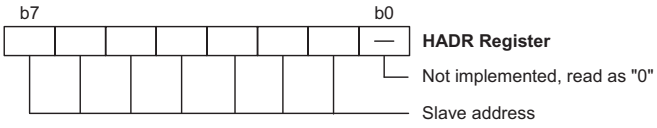
The HDR register is the I²C bus input/output data register. Before the microcontroller writes data to the I²C bus, the actual data to be transmitted must be placed in the HDR register. After the data is received from the I²C bus, the microcontroller can read it from the HDR register. Any transmission of data to the I²C bus or reception of data from the I²C bus must be made via the HDR register.

I²C Bus Control Register – HCR

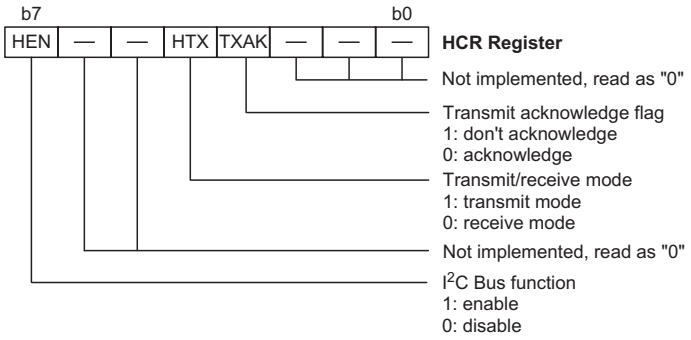
The I²C bus control register HCR contains three bits. Bit 7, known as the HEN bit, determines if the I²C bus function is enabled or disabled, this bit must be set if the I²C bus requires data transfer. Bit 4, known as the HTX bit, determines whether the device is in the transmit mode or receive mode, and must be set high if the device is to be setup as a transmitter. Bit 3, known as the TXAK bit, is the transmit acknowledge bit. After the receipt of 8 bits of data, this bit will be transmitted to the I²C bus on the 9th clock. To continue receiving more data, this bit has to be reset to "0" before more data is received.



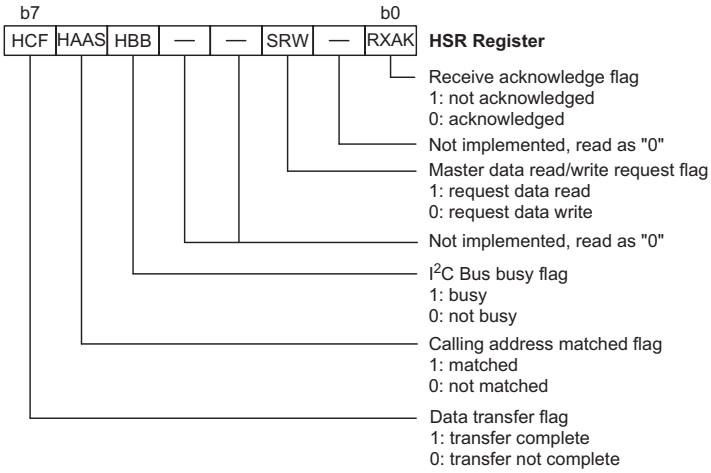
I²C Bus Serial Interface Block Diagram



I²C Bus Slave Address Register



I²C Bus Control Register



I²C Bus Status Register

I²C Bus Status Register – HSR

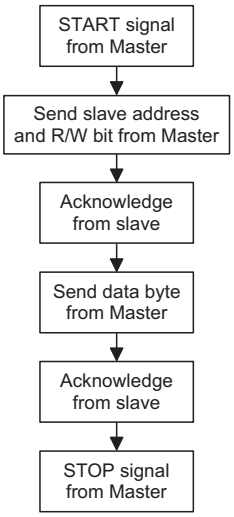
The I²C bus register HSR is an 8-bit status register in which five bits are utilised. Bit 7, known as HCF, is set to ″0″ when a data byte is being transferred, after completion of the data transfer the bit will be set to ″1″. The HAAS bit, which is bit 6, will be set to ″1″ if the transmitted address and the slave address of the device match and if the I²C interrupt request flag is set to ″1″. If the interrupts are enabled and the stack is not full, a subroutine call to 14H will occur. Writing data to the I²C bus will clear the HAAS bit. Also, if the transmitted address on the bus and the slave address of the device do not match, then the HAAS bit will be reset to ″0″.

Bit 5, known as HBB, will be set to ″1″ if the I²C bus is busy, which will occur when a START signal is detected. The HBB bit will be cleared to ″0″ if the bus is free which will occur when a STOP signal is detected. Bit 2, which is the SRW or Slave Read/Write bit, determines whether the master device wishes to transmit or receive data from the I²C bus. When the transmitted address and slave address match, that is when the HAAS bit is set to ″1″, the device will check the SRW bit to determine whether it should be in transmit mode or receive mode. If the SRW bit is equal to ″1″ the master is requesting to read data from the bus, so the device should be in transmit mode. When the SRW bit is equal to ″0″, the master will write data to the bus, therefore the device should be in receive mode to read this data.

Bit 0, is the Receive Acknowledge bit and known as RXAK. When the RXAK bit has been reset to ″0″ it means that a correct acknowledge signal has been received at the 9th clock, after 8 bits of data have been transmitted. When in the transmit mode, the transmitter checks the RXAK bit to determine if the receiver wishes to receive the next byte. The transmitter will therefore continue sending out data until the RXAK bit is set to ″1″. When this occurs, the transmitter will release the SDA line to allow the master to send a STOP signal to release the bus.

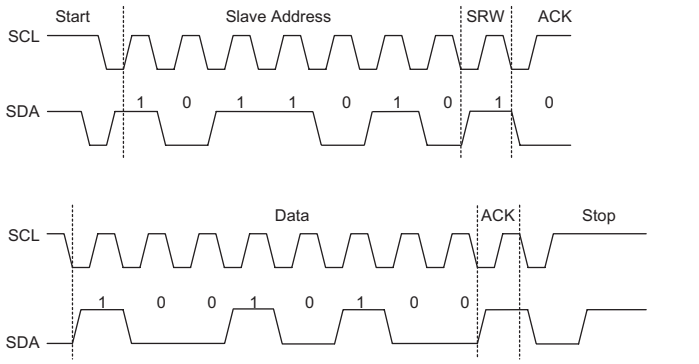
I²C Bus Communication

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 microcontroller matches that of the transmitted address, the HAAS bit in the HSR register will be set and an I^2C interrupt will be generated. After entering the interrupt service routine, the microcontroller 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 microcontroller to determine whether to go into transmit or receive mode. Before any transfer of data to or from the I^2C bus, the microcontroller must initialise the bus, the following are steps to achieve this:

- Step 1
  Write the slave address of the microcontroller to the I^2C bus address register HADR.
- Step 2
  Set the HEN bit in the I^2C bus control register to "1" to enable the I^2C bus.
- Step 3
  Set the EHI bit of the interrupt control register to enable the I^2C bus interrupt.
- **Start Signal**
  The START signal can only be generated by the master device connected to the I^2C bus and not by the microcontroller, which is only a slave device. This START signal will be detected by all devices connected to the I^2C bus. When detected, this indicates that the I^2C 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 I^2C 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 I^2C 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 HSR register. The device will then transmit an acknowledge bit, which is a low level, as the 9th bit. The microcontroller slave device will also set the status flag HAAS when the addresses match. As an I^2C 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 HDR register, or in the receive mode where it must implement a dummy read from the HDR register to release the SCL line.
- **SRW Bit**
  The SRW bit in the HSR register defines whether the microcontroller slave device wishes to read data from the I^2C bus or write data to the I^2C bus. The microcontroller should examine this bit to determine if it is to be a transmitter or a receiver. If the SRW bit is set to "1" then this indicates that the master wishes to read data from the I^2C bus, therefore the microcontroller slave device must be setup to send data to the I^2C bus as a transmitter. If the SRW bit is "0" then this indicates that the master wishes to send data to the I^2C bus, therefore the microcontroller slave device must be setup to read data from the I^2C bus as a receiver.



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

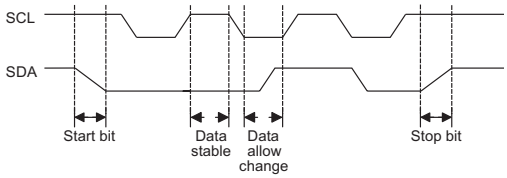
I²C Communication Timing Diagram

- **Acknowledge Bit**

  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. This 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 bit is high, the addresses have matched and the microcontroller slave device must check the SRW bit to determine if it is to be a transmitter or a receiver. If the SRW bit is high, the microcontroller slave device should be setup to be a transmitter so the HTX bit in the HCR register should be set to "1", if the SRW bit is low then the microcontroller slave device should be setup as a receiver and the HTX bit in the HCR register should be set to "0".

- **Data Byte**

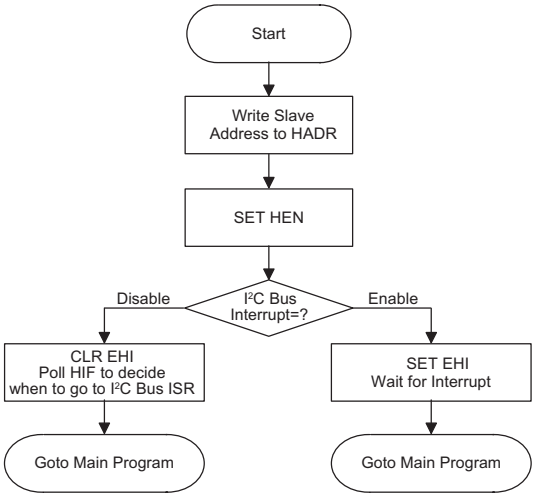
  The transmitted data is 8-bits wide and is transmitted after the slave device has acknowledged receipt of its slave address. The order of serial bit transmission is the MSB first and the LSB last. After receipt of 8-bits of data, the receiver must transmit an acknowledge signal, level "0", before it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



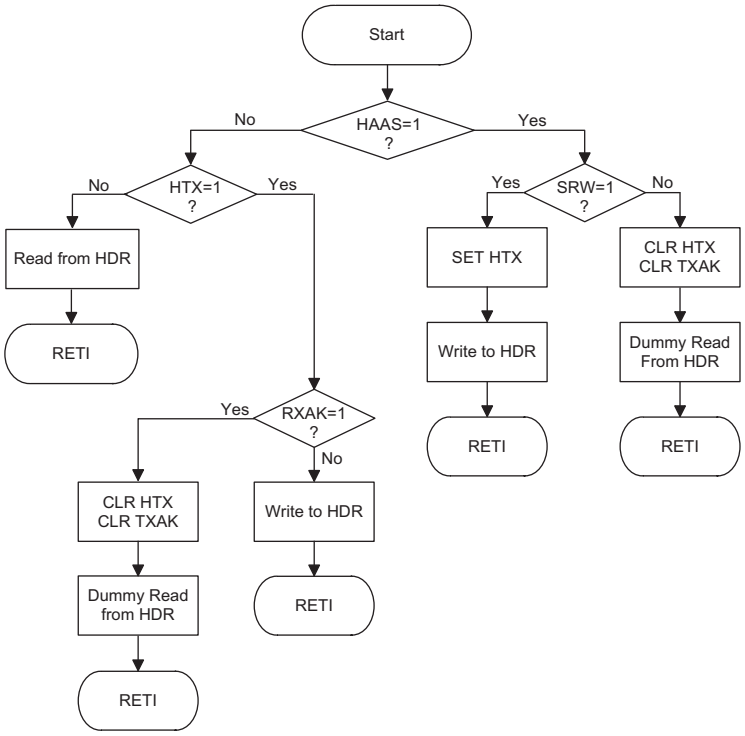
Data Timing Diagram

- **Receive Acknowledge Bit**

  When the receiver wishes to continue to receive the next data byte, it must generate an acknowledge bit, known as TXAK, on the 9th clock. The microcontroller slave device, which is setup as a transmitter will check the RXAK bit in the HSR 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.



I²C Bus Initialization Flow Chart



I²C Bus ISR Flow Chart

SPI Serial Interface

The device includes a single SPI Serial Interface. The SPI interface is a full duplex serial data link, originally designed by Motorola, which allows multiple devices connected to the same SPI bus to communicate with each other. The devices communicate using a master/slave technique where only the single master device can initiate a data transfer. A simple four line signal bus is used for all communication and these pins are shared with normal I/O pins. The SPI function is selected via a configuration option.

SPI Interface Communication

Four lines are used for SPI communication known as SDI - Serial Data Input, SDO - Serial Data Output, SCK - Serial Clock and SCS - Slave Select. Note that the condition of the Slave Select line is conditioned by the CSEN bit in the SBCR control register. If the CSEN bit is high then the SCS line is active while if the bit is low then the SCS line will be in a floating condition. The following timing diagram depicts the basic timing protocol of the SPI bus.

SPI Registers

There are two registers associated with the SPI Interface. These are the SBCR register which is the control register and the SBDR which is the data register. The SBCR register is used to setup the required setup parameters for the SPI bus and also used to store associated operating flags, while the SBDR register is used for data storage.

After Power on, the contents of the SBDR register will be in an unknown condition while the SBCR register will default to the condition below:

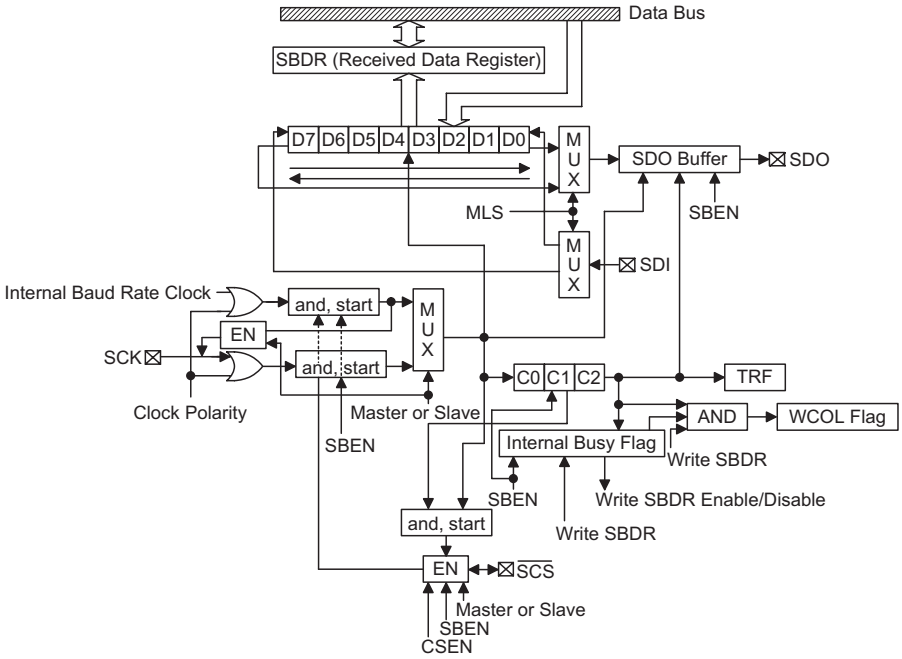
CKS	M1	M0	SBEN	MLS	CSEN	WCOL	TRF
0	1	1	0	0	0	0	0

Note that data written to the SBDR register will only be written to the TXRX buffer, whereas data read from the SBDR register will actual be read from the register.

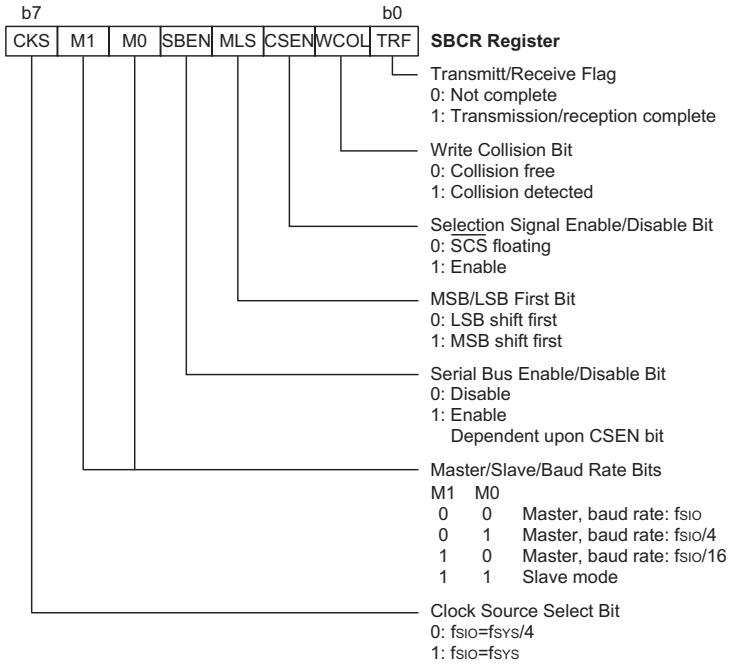
SPI Bus Enable/Disable

To enable the SPI bus, CSEN = 1, $\overline{SCS}$=0, then wait for data to be written to the SBDR (TXRX bufffer) register. For the Master Mode, after data has been written to the SBDR (TXRX buffer) register, then transmission or reception will start automatically. When all the data has been transferred the TRF bit should be set. For the Slave Mode, when clock pulses are received on SCK, data in the TXRX buffer will be shifted out or data on SDI will be shifted in.

To Disable the SPI bus SCK, SDI, SDO, $\overline{SCS}$ should be in a floating condition.

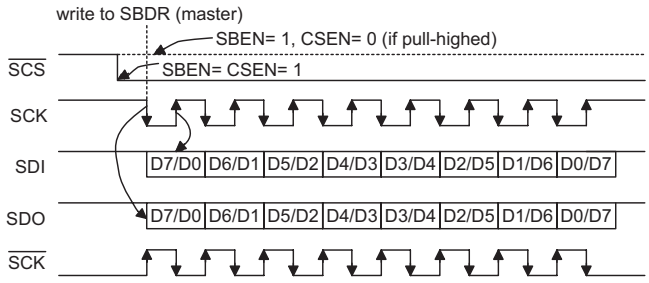


SPI Block Diagram



SPI Interface Control Register

Note: The TRF flag will also generate an SPI interrupt signal, for more information refer to the Interrupt section.



SPI Bus Timing

SPI Operation

All communication is carried out using the 4-line interface for both Master or Slave Mode. The timing diagram shows the basic operation of the bus.

The CSEN bit in the SBCR register controls the overall function of the SPI interface. Setting this bit high, will enable the SPI interface by allowing the SCS line to be active, which can then be used to control the SPI interface. If the CSEN bit is low, the SPI interface will be disabled and the SCS line will be in a floating condition and can therefore not be used for control of the SPI interface. The SBEN bit in the SBCR register must also be high which will place the SDI line in a floating condition and the SDO line high. If in Master Mode the SCK line will be either high or low depending upon the clock polarity configuration option. If in Slave Mode the SCK line will be in a floating condition. If SBEN is low then the bus will be disabled and SCS, SDI, SDO and SCK will all be in a floating condition.

In the Master Mode the Master will always generate the clock signal. The clock and data transmission will be initiated after data has been written to the SBDR register. In the Slave Mode, the clock signal will be received from an external master device for both data transmission or reception. The following sequences show the order to be followed for data transfer in both Master and Slave Mode:

- Master Mode:
  - Step 1
    Select the clock source using the CKS bit in the SBCR control register
  - Step 2
    Setup the M0 and M1 bits in the SBCR control register to select the Master Mode and the required Baud rate. Values of 00, 01 or 10 can be selected.

- Step 3
  Setup the CSEN bit and setup the MLS bit to choose if the data is MSB or LSB first, this must be same as the Slave device.
- Step 4
  Setup the SBEN bit in the SBCR control register to enable the SPI interface.
- Step 5
  For write operations: write the data to the SBDR register, which will actually place the data into the TXRX buffer. Then use the SCK and SCS lines to output the data.
  Goto to step 6.For read operations: the data transferred in on the SDI line will be stored in the TXRX buffer until all the data has been received at which point it will be latched into the SBDR register.
- Step 6
  Check the WCOL bit, if set high then a collision error has occurred so return to step5.
  If equal to zero then go to the following step.
- Step 7
  Check the TRF bit or wait for an SBI serial bus interrupt.
- Step 8
  Read data from the SBDR register.
- Step 9
  Clear TRF.
- Step10
  Goto step 5.

• Slave Mode:

- Step 1
  The CKS bit has a don't care value in the slave mode.
- Step 2
  Setup the M0 and M1 bits to 00 to select the Slave Mode. The CKS bit is don't care.
- Step 3
  Setup the CSEN bit and setup the MLS bit to choose if the data is MSB or LSB first, this must be same as the Master device.
- Step 4
  Setup the SBEN bit in the SBCR control register to enable the SPI interface.
- Step 5
  For write operations: write data to the SBCR register, which will actually place the data into the TXRX register, then wait for the master clock and SCS signal. After this goto step 6.
  For read operations: the data transferred in on the SDI line will be stored in the TXRX buffer until all the data has been received at which point it will be latched into the SBDR register.
- Step 6
  Check the WCOL bit, if set high then a collision error has occurred so return to step5.
  If equal to zero then go to the following step.
- Step 7
  Check the TRF bit or wait for an SBI serial bus interrupt.
- Step 8
  Read data from the SBDR register.
- Step 9
  Clear TRF
- Step10
  Goto step 5

SPI Configuration Options

Several configuration options exist for the SPI Interface function which must be setup during device programming. One option is to enable the operation of the WCOL, write collision bit, in the SBCR register. Another option exists to select the clock polarity of the SCK line. A configuration option also exists to disable or enable the operation of the CSEN bit in the SBCR register. If the configuration option disables the CSEN bit then this bit cannot be used to affect overall control of the SPI Interface.

Error Detection

The WCOL bit in the SBCR register is provided to indicate errors during data transfer. The bit is set by the Serial Interface but must be cleared by the application program. This bit indicates a data collision has occurred which happens if a write to the SBDR register takes place during a data transfer operation and will prevent the write operation from continuing. The bit will be set high by the Serial Interface but has to be cleared by the user application program. The overall function of the WCOL bit can be disabled or enabled by a configuration option.

Programming Considerations

When the device is placed into the Power Down Mode note that data reception and transmission will continue. The TRF bit is used to generate an interrupt when the data has been transferred or received.

UART Bus Serial Interface

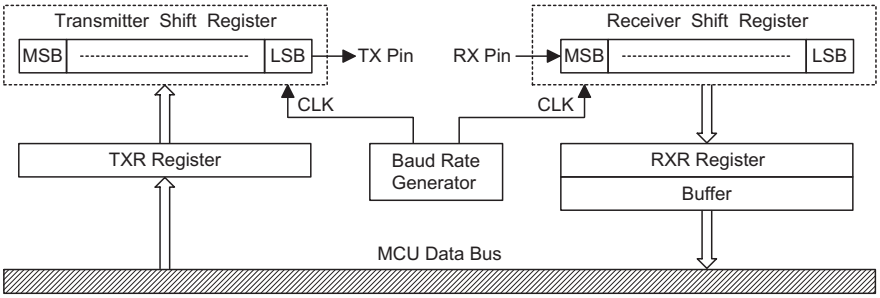
The device contain an integrated full-duplex asynchronous serial communications UART interface that enables communication with external devices that contain a serial interface. The UART function has many features and can transmit and receive data serially by transferring a frame of data with eight or 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.

- UART features
  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 in the UCR2 control register 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 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 status and control registers
  There are five 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.



UART Data Transfer Scheme

- USR register

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

  Further explanation on each of the flags is given below:

  - TXIF

    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 transmit shift registers. When the flag is ″1″ it indicates that the transmit 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 buffer is not yet full.

  - TIDLE

    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 ″1″ the TX pin becomes idle. 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.

  - RXIF

    The RXIF flag is the receive 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.

  - RIDLE

    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 is idle.

  - OERR

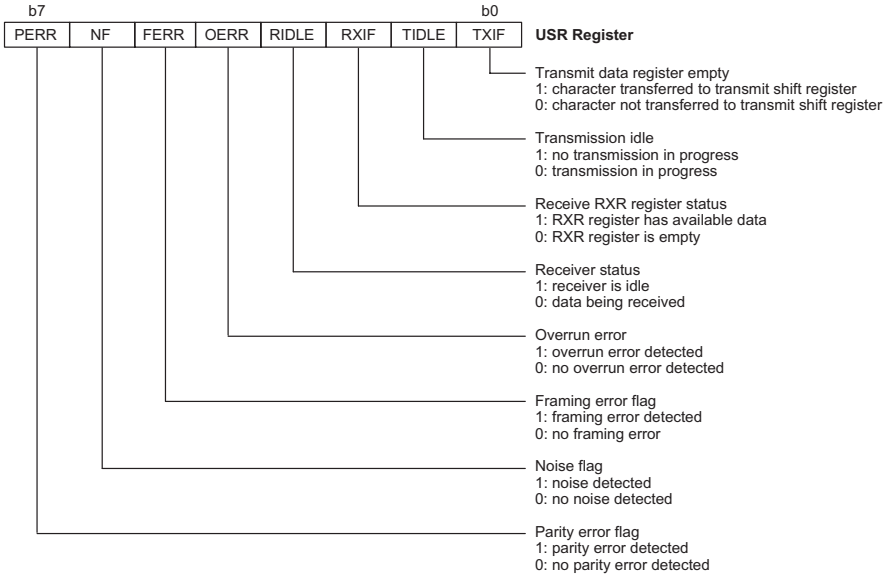
    The OERR flag is the overrun error flag, which indicates when the receiver buffer has overflowed. When this read only flag is ″0″ there is no overrun error. When the flag is ″1″ an overrun error occurs which will inhibit further transfers to the RXR receive data register. The flag is cleared by a software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

  - FERR

    The FERR flag is the framing error flag. When this read only flag is ″0″ it indicates 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 USR status register followed by an access to the RXR data register.

  - NF

    The NF flag is the noise flag. When this read only flag is ″0″ it indicates a 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 an overrun. The NF flag can be cleared by a software sequence which will involve a read to the USR status register, followed by an access to the RXR data register.



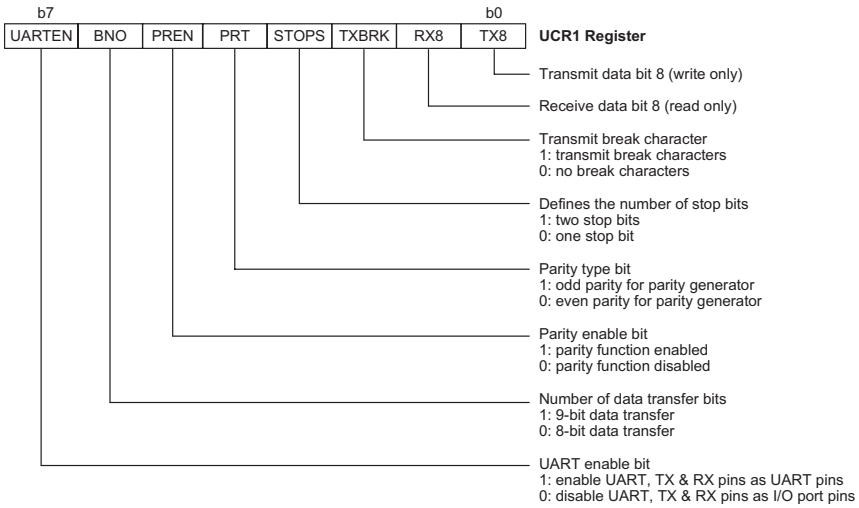
- PERR
  The PERR flag is the parity error flag. When this read only flag is ″0″ it indicates that a parity error has not been detected. When the flag is ″1″ it indicates that the parity of the received word is incorrect. This error flag is applicable only if Parity mode (odd or even) is selected. The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

• 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:

- TX8
  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.
- RX8
  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.
- TXBRK
  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 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.
- STOPS
  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 the bit is equal to ″0″ then only one stop bit is used.
- PRT
  This is the parity type selection bit. When this bit is equal to ″1″ odd parity will be selected, if the bit is equal to ″0″ then even parity will be selected.
- PREN
  This is 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.
- BNO
  This bit is used to select the data length format, which can have a choice of either 8-bits or 9-bits. If this bit is equal to ″1″ then a 9-bit data length will be selected, if the bit is equal to ″0″ then an 8-bit data length will be selected. If 9-bit data length is selected then bits RX8 and TX8 will be used to store the 9th bit of the received and transmitted data respectively.
- UARTEN
  The UARTEN bit is the UART enable bit. When the bit is ″0″ the UART will be disabled and the RX and TX pins will function as General Purpose I/O pins. When the bit is ″1″ the UART will be enabled and the TX and RX pins will function as defined by the TXEN and RXEN 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 baud rate counter value will be reset. When the UART is disabled, all error and status flags will be reset. 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.



- 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 interrupt sources. 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:

  - TEIE

    This bit enables or disables the transmitter empty interrupt. If this bit is equal to ″1″ when the transmitter empty TXIF flag 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.

  - TIIE

    This bit enables or disables the transmitter idle interrupt. If this bit is equal to ″1″ when the transmitter idle TIDLE flag 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 TIDLE flag.

  - RIE

    This bit enables or disables the receiver interrupt. If this bit is equal to ″1″ when the receiver overrun OERR flag or receive data available RXIF flag is set, the UART interrupt request flag will be set. If this bit is equal to ″0″ the UART interrupt will not be influenced by the condition of the OERR or RXIF flags.

  - WAKE

    This bit enables or disables the receiver wake-up function. If this bit is equal to ″1″ and if the MCU is in the Power Down Mode, a low going edge on the RX input pin will wake-up the device. If this bit is equal to ″0″ and if the MCU is in the Power Down Mode, any edge transitions on the RX pin will not wake-up the device.

  - ADDEN

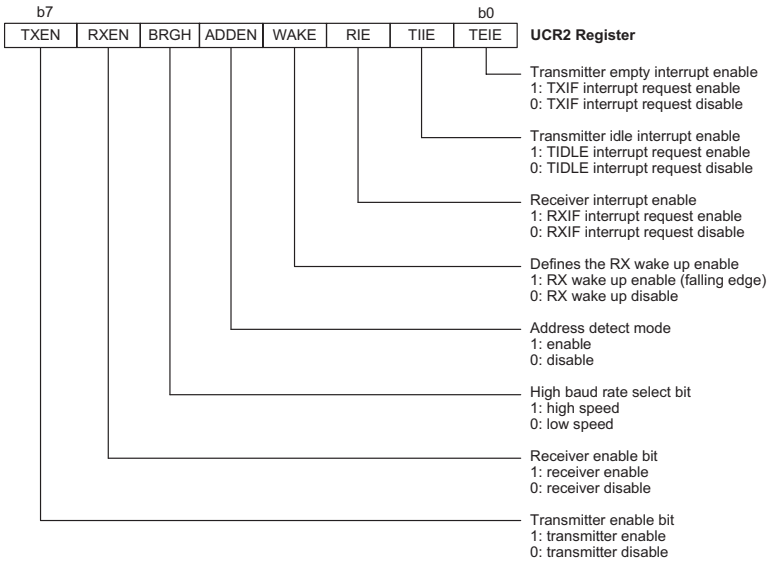
    The ADDEN bit is the address detect mode bit. When this bit is ″1″ the address detect mode is enabled. When this occurs, if the 8th bit, which corresponds to RX7 if BNO=0, or the 9th bit, which corresponds to RX8 if BNO=1, has a value of ″1″ then the received word will be identified as an address, rather 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 or 9 bit depending on the value of BNO. If the address bit is ″0″ an interrupt will not be generated, and the received data will be discarded.

  - BRGH

    The BRGH bit selects the high or low speed mode of the Baud Rate Generator. This bit, together with the value placed in the BRG register, 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.

  - RXEN

    The RXEN bit 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 buffer will be reset. In this situation the RX pin can be used as a general purpose I/O pin. If the RXEN bit is equal to ″1″ the receiver will be enabled and if the UARTEN bit is equal to ″1″ the RX pin will be controlled by the UART. Clearing the RXEN bit during a transmission will cause the data reception to be aborted and will reset the receiver. If this occurs, the RX pin can be used as a general purpose I/O pin.



- TXEN
  The TXEN bit is the Transmitter Enable Bit. When this bit is equal to ″0″ the transmitter will be disabled with any pending transmissions being aborted. In addition the buffer will be reset. In this situation the TX pin can be used as a general purpose I/O pin. If the TXEN bit is equal to ″1″ the transmitter will be enabled and if the UARTEN bit is equal to ″1″ the TX pin will be controlled by the UART. Clearing the TXEN bit during a transmission will cause the transmission to be aborted and will reset the transmitter. If this occurs, the TX pin can be used as a general purpose I/O pin.

- 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 BRG register and the second is the value of the BRGH bit within the UCR2 control register. 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 in the BRG register determines the division factor, N, which is used in the following baud rate calculation formula. 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	$\frac{f_{SYS}}{[64\ (N+1)]}$	$\frac{f_{SYS}}{[16\ (N+1)]}$

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 Register and Error Values

For a clock frequency of 8MHz, 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 9600.

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

Re-arranging this equation gives $N = \frac{f_{SYS}}{(BRx64)} - 1$

Giving a value for $N = \frac{8000000}{(9600x64)} - 1 = 12.0208$

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

$BR = \frac{8000000}{[64(12+1)]} = 9615$

Therefore the error is equal to = 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_{SYS}=8MHz			f_{SYS}=7.159MHz			f_{SYS}=4MHz			f_{SYS}=3.579545MHz		
	BRG	Kbaud	Error	BRG	Kbaud	Error	BRG	Kbaud	Error	BRG	Kbaud	Error
0.3	—	—	—	—	—	—	207	0.300	0.00	185	0.300	0.00
1.2	103	1.202	0.16	92	1.203	0.23	51	1.202	0.16	46	1.19	-0.83
2.4	51	2.404	0.16	46	2.38	-0.83	25	2.404	0.16	22	2.432	1.32
4.8	25	4.807	0.16	22	4.863	1.32	12	4.808	0.16	11	4.661	-2.9
9.6	12	9.615	0.16	11	9.322	-2.9	6	8.929	-6.99	5	9.321	-2.9
19.2	6	17.857	-6.99	5	18.64	-2.9	2	20.83	8.51	2	18.643	-2.9
38.4	2	41.667	8.51	2	37.29	-2.9	1	—	—	1	—	—
57.6	1	62.5	8.51	1	55.93	-2.9	0	62.5	8.51	0	55.93	-2.9
115.2	0	125	8.51	0	111.86	-2.9	—	—	—	—	—	—

Baud Rates and Error Values for BRGH = 0

Baud Rate K/BPS	Baud Rates for BRGH=1											
	f_{SYS}=8MHz			f_{SYS}=7.159MHz			f_{SYS}=4MHz			f_{SYS}=3.579545MHz		
	BRG	Kbaud	Error	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	207	2.404	0.16	185	2.405	0.23	103	2.404	0.16	92	2.406	0.23
4.8	103	4.808	0.16	92	4.811	0.23	51	4.808	0.16	46	4.76	-0.83
9.6	51	9.615	0.16	46	9.520	-0.832	25	9.615	0.16	22	9.727	1.32
19.2	25	19.231	0.16	22	19.454	1.32	12	19.231	0.16	11	18.643	-2.9
38.4	12	38.462	0.16	11	37.287	-2.9	6	35.714	-6.99	5	37.286	-2.9
57.6	8	55.556	-3.55	7	55.93	-2.9	3	62.5	8.51	3	55.930	-2.9
115.2	3	125	8.51	3	111.86	-2.9	1	125	8.51	1	111.86	-2.9
250	1	250	0	—	—	—	0	250	0	—	—	—

Baud Rates and Error Values for BRGH = 1

- Setting up and controlling the UART
  - Introduction

    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′s 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

    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[1]	0	1
Example of 9-bit Data Formats				
1	9	0	0	1
1	8	0	1	1
1	8	1[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.

- 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 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 TX8 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 TX pin is used as a UART transmitter pin and not as an I/O 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:

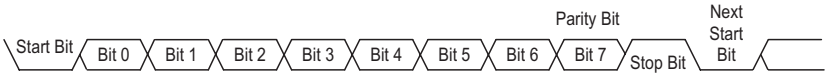
  1. A USR register access
  2. A TXR register write execution

  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.

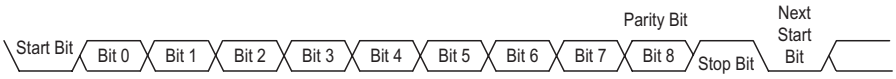
  During a data transmission, a write instruction to the TXR register will place the data into the TXR register, which will be copied to the shift register at the end of the present transmission. When there is no data transmission in progress, a write instruction to the TXR register will place the data 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.



8-Bit Data Format



9-Bit Data Format

- Transmit break

  If the TXBRK bit is set then break characters will be sent on the next transmission. Break character transmission consists of a start bit, followed by 13× N '0' bits and stop 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, 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 highs at the end of the last break character will ensure that the start bit of the next frame is recognized.

- UART receiver
  - Introduction

    The UART is capable of receiving word lengths of either 8 or 9 bits. If the BNO bit is set, the word length will be set to 9 bits with the MSB being stored in the RX8 bit of the UCR1 register. At the receiver core lies the Receive Serial Shift Register, 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, LSB first. In the read mode, the RXR register forms a buffer between the internal bus and the receiver shift register. The RXR register is a two byte deep FIFO data buffer, where two bytes can be held in the FIFO while a 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 this 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 BNO, PRT, PREN and STOPS bits to define the 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 RX pin is used as a UART receiver pin and not as an I/O 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 when RXR register has data available, at least one more character can be read.
    - When the contents of the shift register have been transferred to the RXR register, then if the RIE bit is set, an interrupt will be generated.
    - If during reception, a frame error, noise error, parity error, or an overrun error has been detected, then the error flags can be set.

    The RXIF bit can be cleared using the following software sequence:
    1. A USR register access
    2. An RXR register read execution
  - Receive 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 flag

    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 this 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 Flag

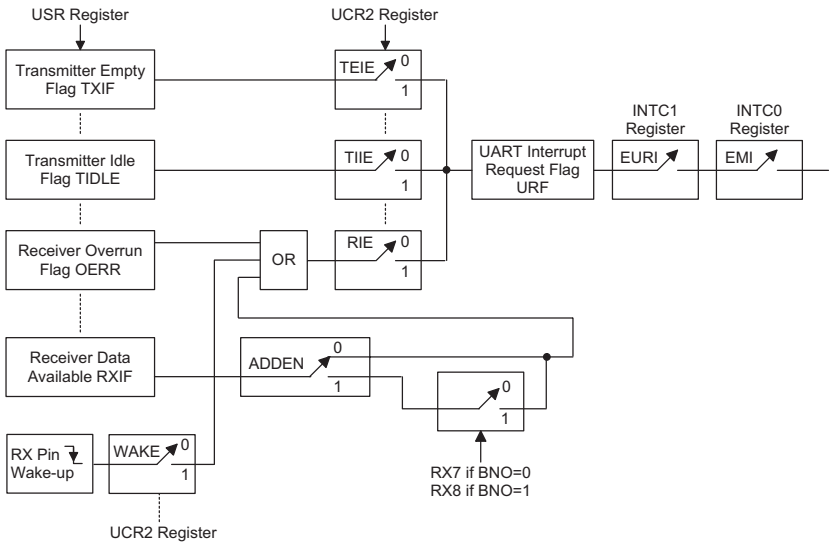
    The read only framing error flag, FERR, in the USR register, is set if a zero is detected instead of stop bits. If two stop bits are selected, both stop bits must be high, otherwise the FERR flag will be set. The FERR flag is buffered along with the received data and is cleared on any reset.

  - Parity Error - PERR Flag

    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 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 scheme

  The UART internal function possesses its own internal interrupt and independent interrupt vector. Several individual UART conditions can generate an internal UART interrupt. 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 the UART interrupt is enabled and the stack is not full, the program will jump to the UART interrupt vector where it can be serviced before returning to the main program. Four of these conditions, have a corresponding USR register flag, which will generate a UART interrupt if its associated interrupt enable flag in



UART Interrupt Scheme

the UCR2 register is set. The two transmitter interrupt conditions have their own corresponding enable bits, while the two receiver interrupt conditions have a shared enable 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 low going 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 delay of 1024 system clock cycles 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 EURI bit in the INTC1 interrupt control register to prevent a UART interrupt from occurring.

- Address detect mode

  Setting the Address Detect Mode bit, ADDEN, in the UCR2 register, enables this special mode. If this bit is enabled 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 enabled, then when data is available, an interrupt will only be generated, if the highest received bit has a high value. Note that the EURI and EMI interrupt enable bits must also be enabled for correct interrupt generation. This highest address bit is the 9th bit if BNO=1 or the 8th bit if BNO=0. If this 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 not enabled, then a Receiver Data Available interrupt will be generated each time the RXIF flag is set, irrespective of the data last bit status. The address detect mode and parity enable are mutually exclusive functions. Therefore if the address detect mode is enabled, then to ensure correct operation, the parity function should be disabled by resetting the parity enable bit to zero.

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

ADDEN Bit Function

- UART operation in power down mode

  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 terminated and the external TX transmit pin will be forced to a logic high level. In a similar way, if the MCU enters the Power Down Mode while receiving data, then the reception of data will likewise be terminated. 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.

  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 1024 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, EURI 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 1024 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, UART, I^2C, SPI Bus, Time-base, real-time clock or an A/D converter 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 external interrupt is controlled by the action of the external $\overline{INT}$ pin.

Interrupt Registers

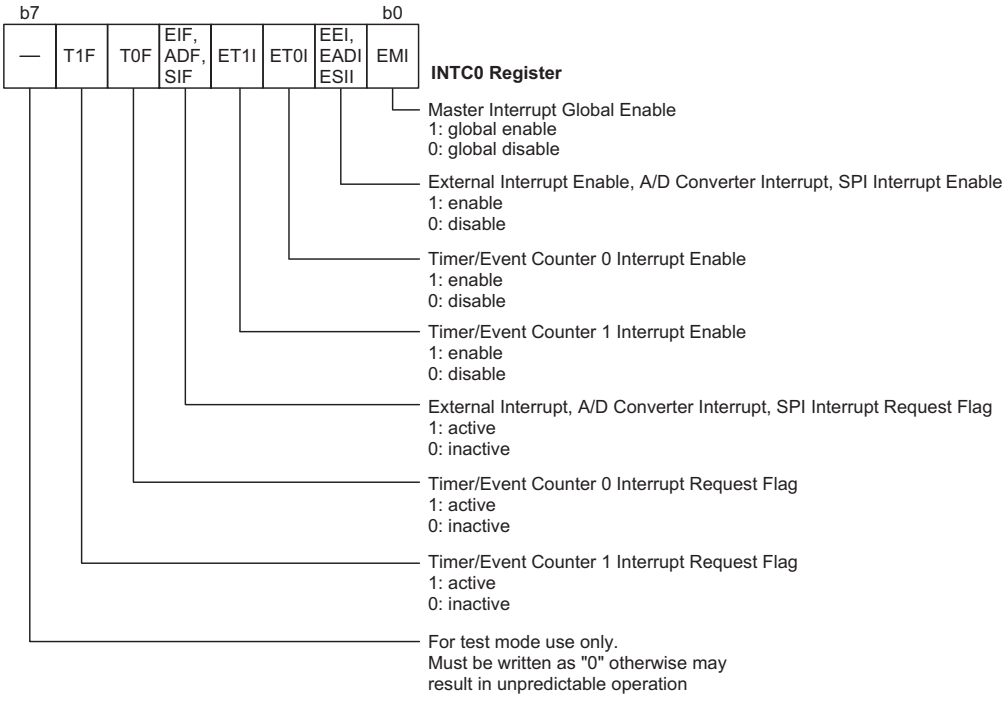
Overall interrupt control, which means interrupt enabling and request flag setting, is controlled by the INTC0, INTC1 and MFIC registers, which are located in the Data Memory. By controlling the appropriate enable bits in these registers each individual interrupt can be enabled or disabled. Also when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

Interrupt Operation

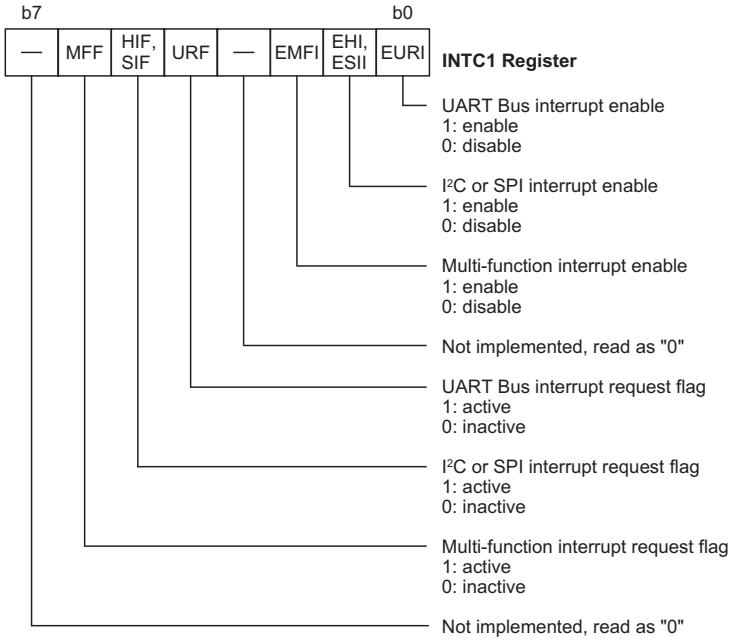
A Timer/Event Counter overflow, an end of A/D conversion, the external interrupt line being pulled low, a UART, SPI, I^2C Bus or multi-function interrupt will all generate an interrupt request by setting their corresponding request flag, if their appropriate interrupt enable bit is set. When this happens, 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 statement, 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.

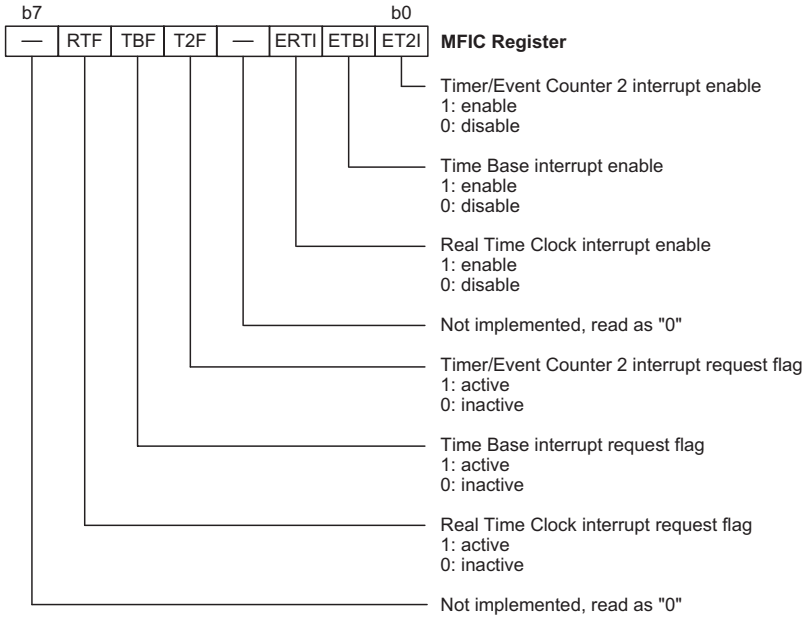
Once an interrupt subroutine is serviced, all the other interrupts will be blocked, as the EMI bit will be cleared automatically. This will prevent any further interrupt nesting from occurring. However, if other interrupt requests occur during this interval, although the interrupt will not be immediately 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.



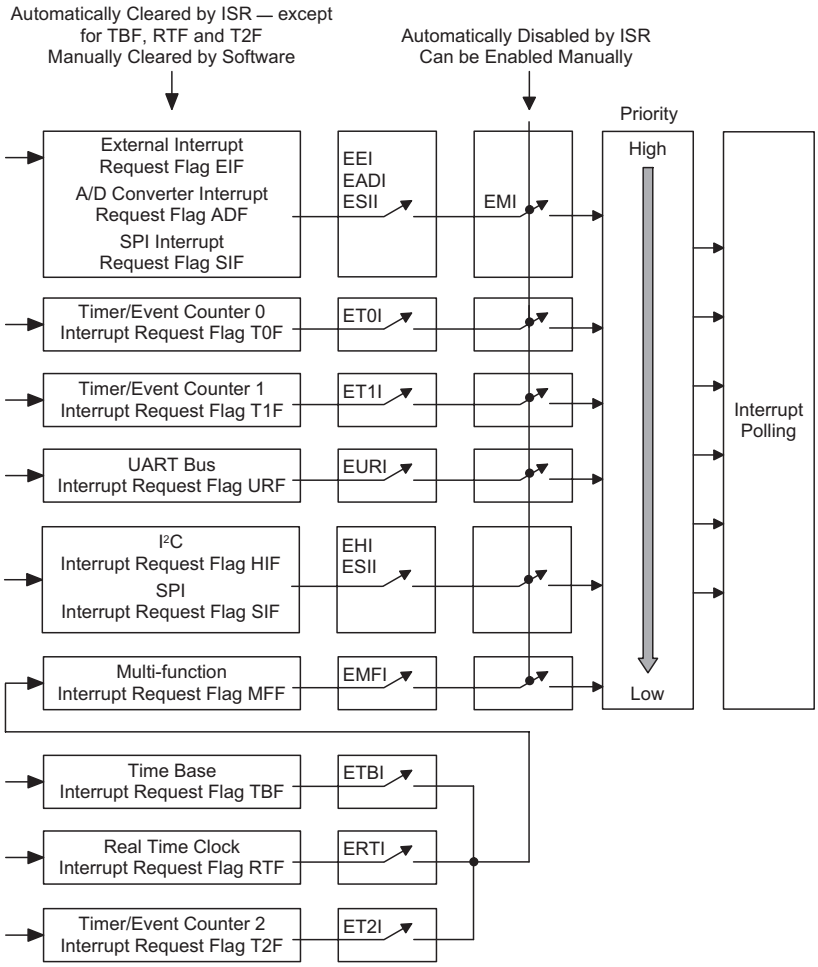
Interrupt Control 0 Register



Interrupt Control 1 Register



Multifunction Interrupt Control Register



Interrupt Scheme

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. In the table, for vector locations 04H and 14H, where more than one interrupt share the same vector, the effective interrupt is chosen via configuration option.

Interrupt Source	Priority	Vector
External Interrupt A/D converter interrupt SPI Interrupt	1	04H
Timer/Event Counter 0 Overflow	2	08H
Timer/Event Counter 1 Overflow	3	0CH
UART Bus Interrupt	4	10H
I^2C Bus Interrupt SPI Interrupt	5	14H
Multi-function Interrupt: - Timer/Event Counter 2 Overflow - Real Time Clock Overflow - Time Base Overflow	6	18H

In cases where both external and internal interrupts are enabled and where an external and internal interrupt occurs 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, EEI, must first be set. An actual external interrupt will take place when the external interrupt request flag, EIF, is set, a situation that will occur when a high to low transition appears on the $\overline{INT}$ line. The external interrupt pin is pin-shared with the I/O pin PA5 and can only be configured as an external interrupt pin if the corresponding external interrupt enable bit in the INTC register has been set. The pin must also be setup as an input by setting the corresponding PAC.5 bit in the port control register. When the interrupt is enabled, the stack is not full and a high to low 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, EIF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts. Note that any pull-high resistor configuration options on this pin will remain valid even if the pin is used as an external interrupt input. As this interrupt vector location is shared with other interrupts, to be effective it must be selected via configuration option.

Timer/Event Counter Interrupt

For a Timer/Event Counter generated internal interrupt to occur, the global interrupt enable bit, EMI, and the corresponding internal interrupt enable bit must be first set. For Timer/Event Counter 0 the interrupt enable is bit 2 of the INTC0 register and known as ET0I, for Timer/Event Counter 1 the interrupt enable is bit 3 of the INTC0 register and known as ET1I, while for the Timer/Event Counter 2 the interrupt enable is bit 0 of the MFIC register and is known as ET2I. An actual Timer/Event Counter interrupt will be generated when the Timer/Event Counter interrupt request flag is set, caused by a timer overflow. For Timer/Event Counter 0 this is bit 5 of the INTC0 register and known as T0F, for Timer/Event Counter 1 this is bit 6 of the INTC0 register and is known as T1F, while for Timer/Event Counter 2 this is bit 4 of the MFIC register and is known as T2F. Because the interrupt vector for Timer/Event Counter 2 is contained with the Multi-function interrupt, for an interrupt to be generated by Timer/Event Counter 2, the Multi-function interrupt must also be enabled by setting the EMFI bit in the INTC1 register. When this is done, a Timer/Event Counter 2 overflow will also cause the Multi-function request flag, known as MFF, which is bit 6 of the INTC1 register to be set and in turn generate the interrupt.

When the master interrupt global enable bit is set, the stack is not full and the corresponding internal interrupt enable bit is set, an internal interrupt will be generated when the corresponding timer overflows. This will create a subroutine call to location 008H, 00CH and 018H for Timer/Event Counter 0, 1 and 2 respectively. It should be noted that the Timer/Event Counter 2 interrupt vector is included within the Multi-function interrupt as it is shared with other interrupts. After entering the timer interrupt execution routine, the corresponding interrupt request flags, T0F or T1F will be reset and the EMI bit will be cleared to disable other interrupts. For Timer/Event Counter 2, when its interrupt occurs, the EMI bit will be cleared to disable other interrupts, however only the MFF interrupt request flag will be reset. As the T2F flag will not be automatically reset, it has to be cleared by the application program.

Time Base Interrupt

For a Time Base interrupt to occur the the global interrupt enable bit, EMI, and the corresponding internal interrupt enable bit, which is bit 1 of the MFIC register, known as ETBI, must be first set. An actual Time Base interrupt will be generated when the Time Base interrupt request flag is set which is bit 5 of the MFIC register and known as TBF. This will occur when when a time-out signal is generated from the Time Base. Because the interrupt vector for the Time Base is contained with the Multi-function interrupt, for an interrupt to be generated by the Time Base, the Multi-function interrupt must also

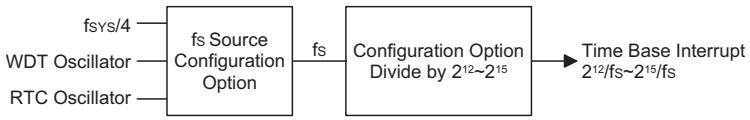
be enabled by setting the EMFI bit in the INTC1 register. When this is done, a Time Base overflow will also cause the Multi-function request flag, known as MFF, which is bit 6 of the INTC1 register to be set and in turn generate the interrupt. When the master interrupt global enable bit is set, the stack is not full and the corresponding Time Base interrupt enable bit is set, an internal Time Base interrupt will be generated when a time-out signal is generated from the Time Base. This will create a subroutine call to location 018H. It should be noted that the Time Base interrupt vector is included within the Multi-function interrupt as it is shared with other interrupts. When a Time Base interrupt occurs, the EMI bit will be cleared to disable other interrupts, however only the MFF interrupt request flag will be reset. As the TBF flag will not be automatically reset, it has to be cleared by the application program. The purpose of the Time Base interrupt is to provide an interrupt signal at fixed time periods. The Time Base interrupt clock source originates from the internal clock source f_S. This f_S input clock first passes through a divider, the division ratio of which is selected by configuration options to provide longer Time Base interrupt periods. The Time Base interrupt time-out period ranges from $2^{12}/f_S$~$2^{15}/f_S$. The clock source that generates f_S, which in turn controls the Time Base interrupt period, can originate from three different sources, the RTC oscillator, Watchdog Timer oscillator or the System oscillator/4, the choice of which is determine by the f_S clock source configuration option. Note that if the RTC oscillator is selected as the system clock, then f_S, and correspondingly the Time Base interrupt, will also have the RTC oscillator as its clock source.

Real Time Clock Interrupt

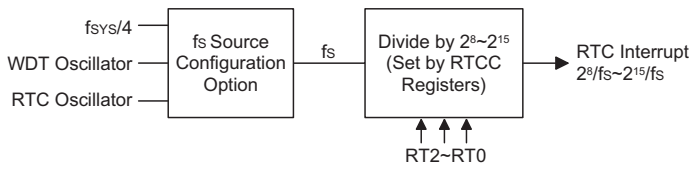
For a Real Time Clock interrupt to occur the global interrupt enable bit, EMI, and the corresponding internal interrupt enable bit, which is bit 2 of the MFIC register, known as ERTI, must be first set. An actual Real Time Clock interrupt will be generated when the Real Time Clock interrupt request flag is set which is bit 6 of the MFIC register and known as RTF. When the master interrupt global enable bit is set, the stack is not full and the corresponding Real Time Clock interrupt enable bit is set, an internal Real Time Clock interrupt will be generated when a time-out signal occurs, a subroutine call to location 018H will be created. Because the interrupt vector for the Real Time Clock is contained with the Multi-function interrupt, for an interrupt to be generated by the Real Time Clock, the Multi-function interrupt must also be enabled by setting the EMFI bit in the INTC1 register. When this is done, a Real Time Clock overflow will also cause the Multi-function request flag, known as MFF, which is bit 6 of the INTC1 register to be set and in turn generate the interrupt. When a Real Time interrupt occurs, the EMI bit will be cleared to disable other interrupts, however only the MFF interrupt request flag will be reset. As the RTF flag will not be automatically reset, it has to be cleared by the application program. It is important not to confuse the RTC interrupt with the RTC oscillator.

Similar in operation to the Time Base interrupt, the purpose of the RTC interrupt is also to provide an interrupt signal at fixed time periods. The RTC interrupt clock source originates from the internal clock source f_S. This f_S input clock first passes through a divider, the division ratio of which is selected by programming the appropriate bits in the RTCC register to obtain longer RTC interrupt periods whose value ranges from $2^8/f_S$~$2^{15}/f_S$. The clock source that generates f_S, which in turn controls the RTC interrupt period, can originate from three different sources, the RTC oscillator, Watchdog Timer oscillator or the System oscillator/4, the choice of which is determine by the f_S clock source configuration option. Note that if the RTC oscillator is selected as the system clock, then f_S, and correspondingly the RTC interrupt, will also have the RTC oscillator as its clock source.

Note that the RTC interrupt period is controlled by both configuration options and an internal register RTCC. A configuration option selects the source clock for the internal clock f_S, and the RTCC register bits RT2, RT1 and RT0 select the division ratio. Note that the actual division ratio can be programmed from 2^8 to 2^{15}. For details of the actual RTC interrupt periods, consult the RTCC register section.



Time Base Interrupt



RTC Interrupt

Note after a wake-up the system requires 1024 clock cycles to resume normal operation. If the 32768Hz RTC oscillator is also selected as the system clock source, then for RTC interrupt applications that are timing sensitive after a wake-up, precautions should be taken when selecting the 2^8, 2^9 and 2^{10} RTC interrupt division. For these division ratios, after a wake-up, some following RTC interrupt events will be missed during this 1024 clock cycle period.

A/D Interrupt

For an A/D interrupt to occur, the global interrupt enable bit, EMI, and the corresponding interrupt enable bit, EADI, must be first set. An actual A/D interrupt will take place when the A/D converter request flag, ADF, is set, a situation that will occur when an A/D conversion process has completed. When the interrupt is enabled, the stack is not full and an A/D conversion process finishes execution, a subroutine call to the A/D interrupt vector at location 04H, will take place. When the interrupt is serviced, the A/D interrupt request flag, ADF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts. As this interrupt vector location is shared with other interrupts, to be effective it must be selected via configuration option.

UART Interrupt

For a UART interrupt to occur, the global interrupt enable bit, EMI, and its corresponding UART interrupt enable bit, EURI, which is bit 0 of the INTC1 register, must first be set. An actual UART interrupt will be generated when the UART interrupt request flag URF is set, which is bit 4 of the INTC1 register. When the master interrupt global bit is set, the stack is not full and the corresponding EURI interrupt enable bit is set, a UART internal interrupt will be generated when a UART interrupt request occurs. This will create a subroutine call to its corresponding vector location 010H. When a UART internal interrupt occurs, the interrupt request flag URF will be reset and the EMI bit cleared to disable other interrupts.

There are various UART conditions, which can generate a UART interrupt, such as transmit data register empty (TXIF), received data available (RXIF), transmission idle (TIDLE), overrun error (OERR) as well as address detected These conditions are reflected by various flags within the UART status register, known as the USR register. Various bits in the UART setup register, UCR2, determine if these flags can generate a UART interrupt signal. More details on these two registers and how they influence the operation of the UART interrupt can be found in the UART section.

I^2C Bus interrupt

For an I^2C interrupt to occur, the global interrupt enable bit, EMI, and the corresponding interrupt enable bit EHI, which is bit 1 of the INTC1 register, must be first set. An actual I^2C interrupt will be generated when the I^2C interrupt request flag, which is bit 5 of the INTC1 register, and known as HIF. When the master interrupt global enable bit is set, the stack is not full and the corresponding I^2C interrupt enable bit is set, an internal interrupt will be generated when a matching I^2C slave address is received or from the completion of an I^2C data byte transfer. This will create a subroutine call to location 14H. When an I^2C interrupt occurs, the interrupt request flag HIF will be reset and the EMI bit will be cleared to disable other interrupts. As this interrupt vector location is shared with other interrupts, to be effective it must be selected via configuration option.

SPI Interrupt

For an SPI interrut to occur, the global interrupt enable bit, EMI, and the corresponding interrupt enable bit ESII, which is bit 1 of the INTC0/INTC1 register, must be first set. An actual SPI interrupt will be generated when the SPI interrupt request flag, which is bit 4/bit 5 of the INTC0/ INTC1 register, and known as SIF. When the master interrupt global enable bit is set, the stack is not full and the corresponding SPI interrupt enable bit is set, an internal interrupt will be generated. This will create a subroutine call to location 04H or 14H. When an SPI interrupt occurs, the interrupt request flag SIF will be reset and the EMI bit will be cleared to disable other interrupts. As this interrupt vector location is shared with other interrupts, to be effective it must be selected via configuration option.

Multi-Function Interrupt

An additional interrupt known as the Multi-function interrupt is provided. Unlike the other interrupts, this interrupt has no independent source, but rather is formed from three other existing interrupt sources, namely the Time Base interrupt, the Real Time Clock interrupt and the Timer/Event Counter 2 interrupt. The Multi-function interrupt is enabled by setting the EMFI bit, which is bit 2 of the INTC1 register. An actual Multi-function interrupt will be generated when the Multi-function interrupt request flag MFF is set, this is bit 6 of the INTC1 register. When the master interrupt global bit is set, the stack is not full and the corresponding EMFI interrupt enable bit is set, a Multi-Function internal interrupt will be generated when either a Time Base overflow, a Real Time Clock overflow or a Timer/Event Counter 2 overflow occurs. This will create a subroutine call to its corresponding vector location 018H. When a Multi-function internal interrupt occurs, the Multi-Function request flag MFF will be reset and the EMI bit will be cleared to disable other interrupts. However, it must be noted that the request flags from the original source of the Multi-function interrupt, namely the Time-Base, Real Time Clock or Timer/Event Counter 2, will not be automatically reset and must be manually reset by the user.

It should also be noted that there is no independent interrupt vectors for the Time Base interrupt, the Real Time Clock interrupt or the Timer/Event Counter 2 interrupt because all three interrupts use the common Multi-function interrupt Vector.

Programming Considerations

By disabling the 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 INTC register until the corresponding interrupt is serviced or until the request flag is cleared by a software instruction.

It is recommended that programs do not use the "CALL subroutine" instruction within the interrupt subroutine. Interrupts often occur in an unpredictable manner or need to be serviced immediately in some applications. If only one stack is left and 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 processor when in the Power Down Mode.

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

Reset and Initialisation

A reset function is a fundamental part of any microcontroller ensuring that the device 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.

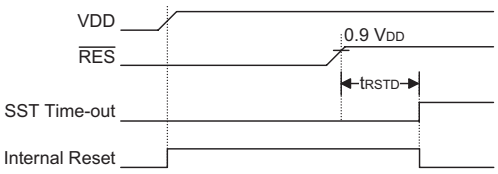
In addition to the power-on reset, situations may arise where it is necessary to forcefully apply a reset condition when the microcontroller is running. One example of this is where after power has been applied and the microcontroller is already running, the $\overline{RES}$ line is forcefully pulled low. In such a case, known as a normal operation reset, some of the microcontroller registers remain unchanged allowing the microcontroller to proceed 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 setup.

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

Reset Functions

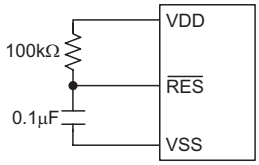
There are five ways in which a microcontroller reset can occur, through events occurring both internally and externally:

- 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.
  Although the microcontroller has an internal RC reset function, if the VDD power supply rise time is not fast enough or does not stabilise 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 $\overline{RES}$ pin, whose additional time delay will ensure that the $\overline{RES}$ pin remains low for an extended period to allow the power supply to stabilise. During this time delay, normal operation of the microcontroller will be inhibited. After the $\overline{RES}$ 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.



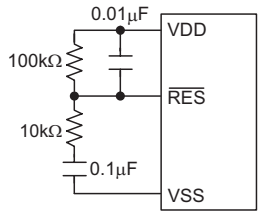
Power-On Reset Timing Chart

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 minimise any stray noise interference.



Basic Reset Circuit

For applications that operate within an environment where more noise is present the Enhanced Reset Circuit shown is recommended.

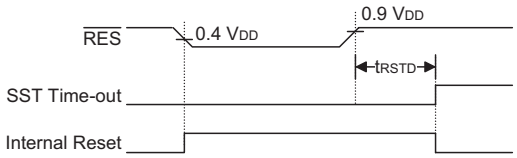


Enhanced Reset Circuit

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

- $\overline{RES}$ Pin Reset

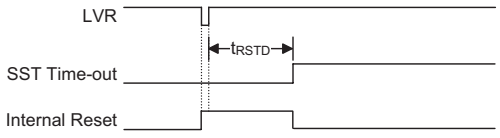
  This type of reset occurs when the microcontroller is already running and the $\overline{RES}$ pin is forcefully pulled low by external hardware such as an external switch. In this case as in the case of other reset, the Program Counter will reset to zero and program execution initiated from this point.



$\overline{RES}$ Reset Timing Chart

- Low Voltage Reset – LVR

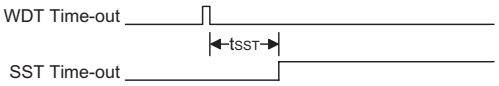
  The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device, which is selected via a configuration option. If the supply voltage of the device drops to within a range of 0.9V~V_{LVR} such as might occur when changing the battery, the LVR will automatically reset the device 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 1ms, the LVR will ignore it and will not perform a reset function.



Low Voltage Reset Timing Chart

- Watchdog Time-out Reset during Normal Operation

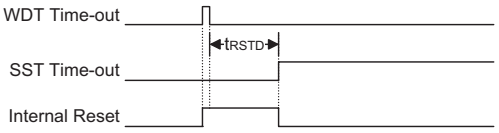
  The Watchdog time-out Reset during normal operation is the same as a hardware $\overline{RES}$ pin reset except that the Watchdog time-out flag TO will be set to "1".



WDT Time-out Reset during Power Down Timing Chart

- Watchdog Time-out Reset during Power Down

  The Watchdog time-out Reset during Power Down 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.



WDT Time-out Reset during Normal Operation 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 Power Down function or Watchdog Timer. The reset flags are shown in the table:

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

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
Prescaler	The Timer Counter Prescaler will be cleared
Input/Output Ports	I/O ports will be setup 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 each of the microcontroller internal registers. Note that where more than one package type exists the table will reflect the situation for the larger package type.

Register	Reset (Power On)	$\overline{\text{RES}}$ or LVR Reset	WDT Time-out (Normal Operation)	WDT Time-out (HALT)*
MP0	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
MP1	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
BP	0000 0000	0000 0000	0000 0000	00u0 00uu
ACC	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
TBLH	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
RTCC	---0 -111	---0 -111	---0 -111	--uu uuuu
STATUS	--00 xxxx	--uu uuuu	--1u uuuu	--11 uuuu
INTC0	-000 0000	-000 0000	-000 0000	-uuu uuuu
TMR0H	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR0L	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR0C	00-0 1000	00-0 1000	00-0 1000	uuuu uuuu
TMR1H	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR1L	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR1C	00-0 1---	00-0 1---	00-0 1---	uu-u u---
PA	1111 1111	1111 1111	1111 1111	uuuu uuuu
PAC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PB	1111 1111	1111 1111	1111 1111	uuuu uuuu
PBC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PCC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PD	1111 1111	1111 1111	1111 1111	uuuu uuuu
PDC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PWM0	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
PWM1	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu

Register	Reset (Power On)	$\overline{RES}$ or LVR Reset	WDT Time-out (Normal Operation)	WDT Time-out (HALT)*
PWM2	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
PWM3	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
INTC1	–000 –000	–000 –000	–000 –000	–uuu –uuu
TBHP	xxxx xxxx	uuuu uuuu	uuuu uuuu	uuuu uuuu
HADR	xxxx xxx–	xxxx xxx–	xxxx xxx–	uuuu uuu–
HCR	0––0 0–––	0––0 0–––	0––0 0–––	u––u u–––
HSR	100– –0–1	100– –0–1	100– –0–1	uuuu uuuu
HDR	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
ADRL	xxxx ————	xxxx ————	xxxx ————	uuuu ————
ADRH	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
ADCR	0100 0000	0100 0000	0100 0000	uuuu uuuu
ACSR	1––– ––00	1––– ––00	1––– ––00	u––– ––uu
PF	1111 1111	1111 1111	1111 1111	uuuu uuuu
PFC	1111 1111	1111 1111	1111 1111	uuuu uuuu
PG	1111 1111	1111 1111	1111 1111	uuuu uuuu
PGC	1111 1111	1111 1111	1111 1111	uuuu uuuu
TMR2	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
TMR2C	00–0 1000	00–0 1000	00–0 1000	uu–u uuuu
MFIC	–000 –000	–000 –000	–000 –000	–uuu –uuu
USR	0000 1011	0000 1011	0000 1011	uuuu uuuu
UCR1	0000 00x0	0000 00x0	0000 00x0	uuuu uuuu
UCR2	0000 0000	0000 0000	0000 0000	uuuu uuuu
TXR/RXR	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu
BRG	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu xxxx
SBCR	0110 0000	0110 0000	0110 0000	uuuu uuuu
SBDR	xxxx xxxx	xxxx xxxx	xxxx xxxx	uuuu uuuu

″u″ stands for unchanged
″x″ stands for unknown
″–″ stands for unimplemented

Oscillator

Various oscillator options offer the user a wide range of functions according to their various application requirements. Two types of system clocks can be selected while various clock source options for the Watchdog Timer are provided for maximum flexibility. All oscillator options are selected through the configuration options.

The two methods of generating the system clock are:

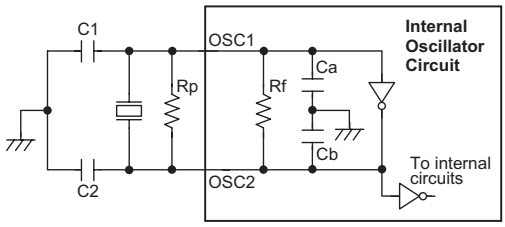
- External crystal/resonator oscillator
- External RC oscillator

One of these two methods must be selected using the configuration options.

More information regarding the oscillator is located in Application Note HA0075E on the Holtek website.

External Crystal/Resonator Oscillator

The simple connection of a crystal across OSC1 and OSC2 will create the necessary phase shift and feedback for oscillation, and will normally not require external capacitors. However, for some crystals and most resonator types, to ensure oscillation and accurate frequency generation, it may be necessary to add two small value external capacitors, C1 and C2. The exact



Note: 1. Rp is normally not required.
2. Although not shown OSC1/OSC2 pins have a parasitic capacitance of around 7pF.

Crystal/Resonator Oscillator

values of C1 and C2 should be selected in consultation with the crystal or resonator manufacturer's specification. The external parallel feedback resistor, Rp, is normally not required but in some cases may be needed to assist with oscillation start up.

Internal Ca, Cb, Rf Typical Values @ 5V, 25°C		
Ca	**Cb**	**Rf**
11~13pF	13~15pF	470kΩ

Oscillator Internal Component Values

Crystal Oscillator C1 and C2 Values			
Crystal Frequency	**C1**	**C2**	**CL**
8MHz	TBD	TBD	TBD
4MHz	TBD	TBD	TBD
1MHz	TBD	TBD	TBD
Note: 1. C1 and C2 values are for guidance only. 2. CL is the crystal manufacturer specified load capacitor value.			

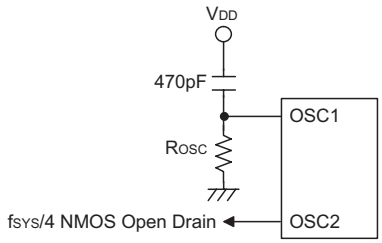
Crystal Recommended Capacitor Values

Resonator C1 and C2 Values		
Resonator Frequency	**C1**	**C2**
3.58MHz	TBD	TBD
1MHz	TBD	TBD
455kHz	TBD	TBD
Note: C1 and C2 values are for guidance only.		

Resonator Recommended Capacitor Values

External RC Oscillator

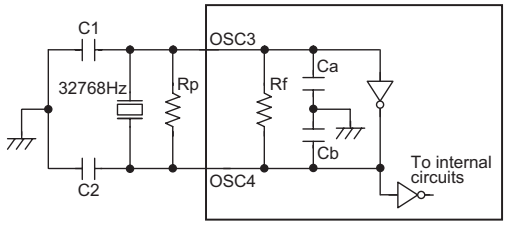
Using the external system RC oscillator requires that a resistor, with a value between 24kΩ and 1MΩ, is connected between OSC1 and ground, and a capacitor is connected to VDD. The generated system clock divided by 4 will be provided on OSC2 as an output which can be used for external synchronization purposes. Note that as the OSC2 output is an NMOS open-drain type, a pull high resistor should be connected if it to be used to monitor the internal frequency. Although this is a cost effective oscillator configuration, the oscillation frequency can vary with VDD, temperature and process variations and is therefore not suitable for applications where timing is critical or where accurate oscillator frequencies are required.For the value of the external resistor Rosc refer to the Holtek website for typical RC Oscillator vs. Temperature and VDD characteristics graphics. Note that it is the only microcontroller internal circuitry together with the external resistor, that determine the frequency of the oscillator. The external capacitor shown on the diagram does not influence the frequency of oscillation.



RC Oscillator

External RTC Oscillator

When the microcontroller enters the Power Down Mode, the system clock is switched off to stop microcontroller activity and to conserve power. However, in many microcontroller applications it may be necessary to keep some internal functions such as timers operational even when the microcontroller is in the Power Down Mode. To do this, a 32768Hz oscillator, also known as the Real Time Clock or RTC oscillator, is provided. To implement this clock, the OSC3 and OSC4 pins should be connected to a 32768Hz crystal. However, for some crystals, to ensure oscillation and accurate frequency



Note: 1. Rp is normally not required.
2. Although not shown OSC3/OSC4 pins have a parasitic capacitance of around 7pF.

Internal RC Oscillator + External RTC Oscillator

generation, it may be necessary to add two small value external capacitors, C1 and C2. The exact values of C1 and C2 should be selected in consultation with the crystal or resonator manufacturer′s specification. The external parallel feedback resistor, Rp, is normally not required but in some cases may be needed to assist with oscillation start up. Using the slower 32768Hz oscillator as the system oscillator will of course use less power and is known as the Slow Mode.

Internal Ca, Cb, Rf Typical Values @ 5V, 25°C		
Ca	**Cb**	**Rf**
TBD	TBD	TBD

RTC Oscillator Internal Component Values

RTC Oscillator C1 and C2 Values			
Crystal Frequency	**C1**	**C2**	**CL**
32768Hz	TBD	TBD	TBD

Note: 1. C1 and C2 values are for guidance only.
2. CL is the crystal manufacturer specified load capacitor value.

32768 Hz Crystal Recommended Capacitor Values

When the system enters the Power Down Mode, the 32768Hz oscillator will keep running and if it is selected as the Timer and Watchdog Timer source clock, will also keep these functions operational.

During power up there is a time delay associated with the RTC oscillator, waiting for it to start up. The QOSC bit in the RTCC register, is provided to give a quick start-up function and can be used to minimise this delay. During a power up condition, this bit will be cleared to 0 which will initiate the RTC oscillator quick start-up function. However, as there is additional power consumption associated with this quick start-up function, to reduce power consumption after start up takes place, it is recommended that the application program should set the QOSC bit high about 2 seconds after power on. It should be noted that, no matter what condition the QOSC bit is set to, the RTC oscillator will always function normally, only there is more power consumption associated with the quick start-up function.

Watchdog Timer Oscillator

The WDT oscillator is a fully self-contained free running on-chip RC oscillator with a typical period of 65μs at 5V requiring no external components. When the device enters the Power Down Mode, the system clock will stop running but the WDT oscillator continues to free-run and to keep the watchdog active. However, to preserve power in certain applications the WDT oscillator can be disabled via a configuration option.

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 device enters this mode, the normal operating current, will be reduced to an extremely low standby current level. This occurs because when the device enters the Power Down Mode, the system oscillator is stopped which reduces the power consumption to extremely low levels, however, as the device maintains its present internal condition, it 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 MCU must have its power supply constantly maintained to keep the device in a known condition but where the power supply capacity is limited such as in battery applications.

Entering the Power Down Mode

There is only one way for the device 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 is selected to come from the WDT oscillator. The WDT will stop if its clock source originates from the system clock.
- 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 Power Down 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 the device. 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. This also applies to devices which have different package types, as there may be undonbed pins, which must either be setup as outputs or if setup as inputs must have pull-high resistors connected. Care must also be taken with the loads, which are connected to I/O pins, which are setup 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. Also note that additional standby current will also be required if the configuration options have enabled the Watchdog Timer internal oscillator.

Wake-up

After the system enters the Power Down 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 device will experience a full system reset, however, if the device is 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.

Each pin on Port A can be setup via an individual configuration option 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 device 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 to "1" before entering the Power Down Mode, the wake-up function of the related interrupt will be disabled.

No matter what the source of the wake-up event is, once a wake-up situation occurs, a time period equal to 1024 system clock periods will be required before normal system operation resumes. However, if the wake-up has originated due to an interrupt, the actual interrupt subroutine execution will be delayed by an additional one or more cycles. If the wake-up results in the execution of the next instruction following the "HALT" instruction, this will be executed immediately after the 1024 system clock period delay has ended.

Watchdog Timer

The Watchdog Timer is provided to prevent program malfunctions or sequences from jumping to unknown locations, due to certain uncontrollable external events such as electrical noise. It operates by providing a device reset when the WDT counter overflows. The WDT clock is supplied by one of two sources selected by configuration option: its own self contained dedicated internal WDT oscillator or $f_{SYS}/4$. Note that if the WDT configuration option has been disabled, then any instruction relating to its operation will result in no operation.

In the A/D Type series of microcontrollers, all Watchdog Timer options, such as enable/disable, WDT clock source and clear instruction type all selected through configuration options. There are no internal registers associated with the WDT in the A/D Type MCU series. One of the WDT clock sources is an internal oscillator which has an approximate period of 65μs at a supply voltage of 5V. However, it should be noted that this specified internal clock period can vary with VDD, temperature and process variations. The other WDT clock source option is the $f_{SYS}/4$ clock. Whether the WDT clock source is its own internal WDT oscillator, RTC oscillator or from $f_{SYS}/4$, it is further divided by 16 via an internal 15-bit counter and a clearable single bit counter to give longer Watchdog time-outs. As the clear instruction only resets the last stage of the divider chain, for this reason the actual division ratio and corresponding Watchdog Timer time-out can vary by a factor of two. The exact division ratio depends upon the residual value in the Watchdog Timer counter before the clear instruction is executed. It is important to realise that as there are no independent internal registers or configuration options associated with the length of the Watchdog Timer time-out, it is completely dependent upon the frequency of $f_{SYS}/4$, RTC oscillator or the internal WDT oscillator.

If the $f_{SYS}/4$ clock is used as the WDT clock source, it should be noted that when the system enters the Power Down Mode, then the instruction clock is stopped and the WDT will lose its protecting purposes. For systems that operate in noisy environments, using the internal WDT oscillator is strongly recommended.

Under normal program operation, a WDT time-out will initialise a device reset and set the status bit TO. However, if the system is in the Power Down Mode, when a WDT time-out occurs, the TO bit in the status register will be set and only the Program Counter and Stack Pointer will be reset. Three methods can be adopted to clear the contents of the WDT. The first is an external hardware reset, which means a low level on the $\overline{RES}$ pin, the second is using the watchdog software instructions and the third is via a ″HALT″ instruction.

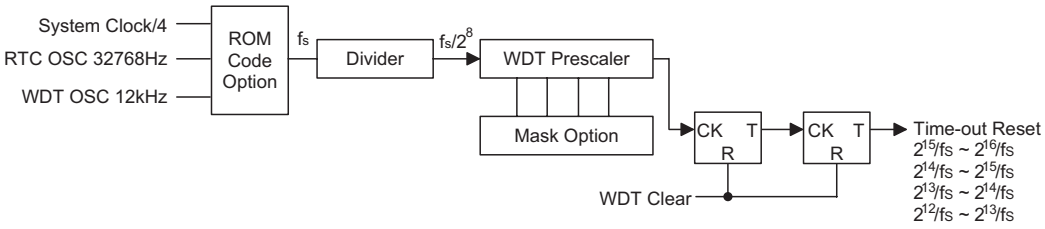
There are two methods of using software instructions to clear the Watchdog Timer, one of which must be chosen by configuration option. The first option is to use the single ″CLR WDT″ instruction while the second is to use the two commands ″CLR WDT1″ and ″CLR WDT2″. For the first option, a simple execution of ″CLR WDT″ will clear the WDT while for the second option, both ″CLR WDT1″ and ″CLR WDT2″ must both be executed to successfully clear the WDT. Note that for this second option, if ″CLR WDT1″ is used to clear the WDT, successive executions of this instruction will have no effect, only the execution of a ″CLR WDT2″ instruction will clear the WDT. Similarly after the ″CLR WDT2″ instruction has been executed, only a successive ″CLR WDT1″ instruction can clear the Watchdog Timer.

Time Base

The internal time base function provides a periodic time-out signal which in turn generates an interrupt. Its time-out period ranges from $2^{12}/f_S$ to $2^{15}/f_S$ the actual value is chosen via configuration option. When a time base time-out occurs, the related interrupt request flags, MFF in INTC1 and TBF in MFIC, are set. If the time base interrupt enable bits, EMFI and ETBI, are enabled, and the stack is not full, a subroutine call to location 18H will occur. Note that as the TBF flag will not be cleared automatically, it must be cleared manually by the application program.

Real Time Clock – RTC

The real time clock operates in a similar way to the time base in that it is used to generate a regular interrupt signal. Its time-out period ranges from $f_S/2^8$ to $f_S/2^{15}$ the actual value is chosen by programming the RT0~RT2 bits in the RTCC register. When an RTC time-out occurs, the related interrupt request flags, MFF in INTC1 and RTF in MFIC, are set. If the interrupt enable bits, EMFI and ERTI, are enabled, and the stack is not full, a subroutine call to location 18H occurs. Note that as the RTF flag will not be cleared automatically, it must be cleared manually by the application program.



Watchdog Timer

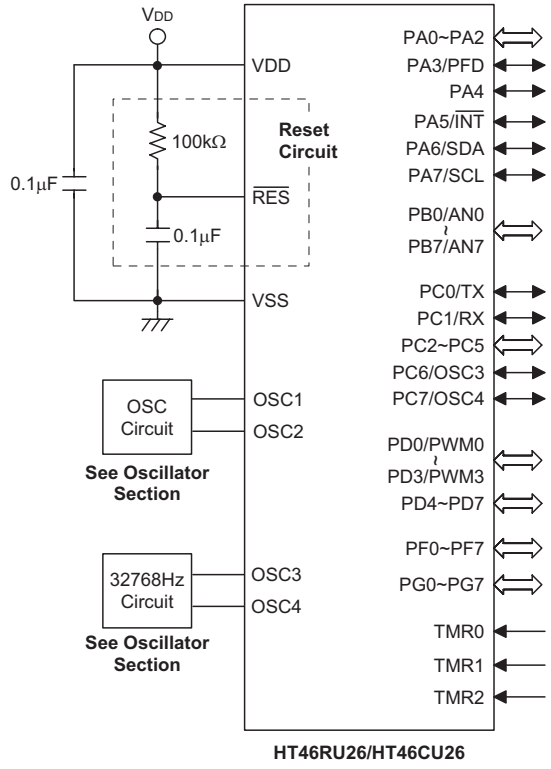
Configuration Options

Configuration options refer to certain options within the MCU that are programmed into the device during the programming process. During the development process, these options are selected using the HT-IDE software development tools. As these options are programmed into the device using the hardware programming tools, once they are selected they cannot be changed later as the application software has no control over the configuration options. All options must be defined for proper system function, the details of which are shown in the table.

No.	Options
I/O Options	
1	PA0~PA7: wake-up enable or disable - bit option
2	PA0~PA7: pull-high enable or disable
3	PB0~PB7: pull-high enable or disable
4	PC0~PC7: pull-high enable or disable
5	PD0~PD4: pull-high enable or disable
6	PF0~PF4: pull-high enable or disable
7	PG0~PG7: pull-high enable or disable
Oscillator Options	
8	OSC type selection: RC or crystal
9	f_{SYS} clock source: OSC or RTC oscillator
10	f_S internal clock source: RTC oscillator, WDT oscillator or $f_{SYS}/4$
Time Base	
11	Time base time-out period: $2^{12}/f_S$, $2^{13}/f_S$, $2^{14}/f_S$, $2^{15}/f_S$
PFD Options	
12	PA3: normal I/O or PFD output
13	PFD clock selection: Timer/Event Counter 0 or Timer/Event Counter 1
PWM Options	
14	PD0~PD3: PWM0~PWM3 function selection
15	PWM mode: 6+2 or 7+1 mode selection
WDT Options	
16	WDT: enable or disable
17	CLRWDT instructions: 1 or 2 instructions
18	WDT time-out period selection. WDT time-out period: $2^{12}/f_S$~$2^{13}/f_S$, $2^{13}/f_S$~$2^{14}/f_S$, $2^{14}/f_S$~$2^{15}/f_S$, $2^{15}/f_S$~$2^{16}/f_S$.
Interrupt Options	
19	Interrupt vector selection. 04H: $\overline{INT}$, 14H: I^2C 04H: $\overline{INT}$, 14H: SPI 04H: A/D, 14H: I^2C 04H: A/D, 14H: SPI 04H: SPI, 14H: I^2C
I^2C Bus Options	
20	I^2C Bus function: enable or disable

No.	Options
SPI Options	
21	SPI function: enable or disable
22	SPI WCOL function: enable or disable
23	SPI CSEN function: enable or disable
24	SPI CPOL function: enable or disable
LVR Options	
25	LVR function: enable or disable

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 microcontrollers, a comprehensive and flexible set of over 60 instructions is provided to enable programmers to implement their application with the minimum of programming overheads.

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

Instruction Timing

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

Moving and Transferring Data

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

Arithmetic Operations

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

Logical and Rotate Operations

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

Branches and Control Transfer

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

Bit Operations

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

Table Read Operations

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

Other Operations

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

Instruction Set Summary

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

Table conventions:

x: Bits immediate data

m: Data Memory address

A: Accumulator

i: 0~7 number of bits

addr: Program memory address

Mnemonic	Description	Cycles	Flag Affected
Arithmetic			
ADD A,[m]	Add Data Memory to ACC	1	Z, C, AC, OV
ADDM A,[m]	Add ACC to Data Memory	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

Mnemonic	Description	Cycles	Flag Affected
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
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			
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 bitwise logical AND operation. The result is stored in the Accumulator.

Operation ACC ← ACC "AND" x

Affected flag(s) Z

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

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

Operation [m] ← ACC "AND" [m]

Affected flag(s) Z

CALL addr	Subroutine call
Description	Unconditionally calls a subroutine at the specified address. The Program Counter then increments by 1 to obtain the address of the next instruction which is then pushed onto the stack. The specified address is then loaded and the program continues execution from this new address. As this instruction requires an additional operation, it is a two cycle instruction.
Operation	Stack ← Program Counter + 1 Program Counter ← addr
Affected flag(s)	None
CLR [m]	Clear Data Memory
Description	Each bit of the specified Data Memory is cleared to 0.
Operation	[m] ← 00H
Affected flag(s)	None
CLR [m].i	Clear bit of Data Memory
Description	Bit i of the specified Data Memory is cleared to 0.
Operation	[m].i ← 0
Affected flag(s)	None
CLR WDT	Clear Watchdog Timer
Description	The TO, PDF flags and the WDT are all cleared.
Operation	WDT cleared TO ← 0 PDF ← 0
Affected flag(s)	TO, PDF
CLR WDT1	Pre-clear Watchdog Timer
Description	The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT2 and must be executed alternately with CLR WDT2 to have effect. Repetitively executing this instruction without alternately executing CLR WDT2 will have no effect.
Operation	WDT cleared TO ← 0 PDF ← 0
Affected flag(s)	TO, PDF
CLR WDT2	Pre-clear Watchdog Timer
Description	The TO, PDF flags and the WDT are all cleared. Note that this instruction works in conjunction with CLR WDT1 and must be executed alternately with CLR WDT1 to have effect. Repetitively executing this instruction without alternately executing CLR WDT1 will have no effect.
Operation	WDT cleared TO ← 0 PDF ← 0
Affected flag(s)	TO, PDF

CPL [m] Complement Data Memory

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

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

Affected flag(s) Z

CPLA [m] Complement Data Memory with result in ACC

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

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

Affected flag(s) Z

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

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

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

Affected flag(s) C

DEC [m] Decrement Data Memory

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

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

Affected flag(s) Z

DECA [m] Decrement Data Memory with result in ACC

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

Operation $ACC \leftarrow [m] - 1$

Affected flag(s) Z

HALT Enter power down mode

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

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

Affected flag(s) TO, PDF

INC [m]	Increment Data Memory
Description	Data in the specified Data Memory is incremented by 1.
Operation	[m] ← [m] + 1
Affected flag(s)	Z
INCA [m]	Increment Data Memory with result in ACC
Description	Data in the specified Data Memory is incremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged.
Operation	ACC ← [m] + 1
Affected flag(s)	Z
JMP addr	Jump unconditionally
Description	The contents of the Program Counter are replaced with the specified address. Program execution then continues from this new address. As this requires the insertion of a dummy instruction while the new address is loaded, it is a two cycle instruction.
Operation	Program Counter ← addr
Affected flag(s)	None
MOV A,[m]	Move Data Memory to ACC
Description	The contents of the specified Data Memory are copied to the Accumulator.
Operation	ACC ← [m]
Affected flag(s)	None
MOV A,x	Move immediate data to ACC
Description	The immediate data specified is loaded into the Accumulator.
Operation	ACC ← x
Affected flag(s)	None
MOV [m],A	Move ACC to Data Memory
Description	The contents of the Accumulator are copied to the specified Data Memory.
Operation	[m] ← ACC
Affected flag(s)	None
NOP	No operation
Description	No operation is performed. Execution continues with the next instruction.
Operation	No operation
Affected flag(s)	None
OR A,[m]	Logical OR Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical OR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″OR″ [m]
Affected flag(s)	Z

OR A,x Logical OR immediate data to ACC

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

Operation: ACC ← ACC "OR" x

Affected flag(s): Z

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

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

Operation: [m] ← ACC "OR" [m]

Affected flag(s): Z

RET Return from subroutine

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

Operation: Program Counter ← Stack

Affected flag(s): None

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

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

Operation: Program Counter ← Stack
ACC ← x

Affected flag(s): None

RETI Return from interrupt

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

Operation: Program Counter ← Stack
EMI ← 1

Affected flag(s): None

RL [m] Rotate Data Memory left

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

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

Affected flag(s): None

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

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

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

Affected flag(s): None

RLC [m] Rotate Data Memory left through Carry

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

Operation
[m].(i+1) ← [m].i; (i = 0~6)
[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 $ACC \leftarrow ACC - [m] - \overline{C}$

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

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

Description The contents of the specified Data Memory and the complement of the carry flag are 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 $[m] \leftarrow ACC - [m] - \overline{C}$

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

SDZ [m] Skip if decrement Data Memory is 0

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

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

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 $[m].i \leftarrow 1$

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 $\neq$ 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

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

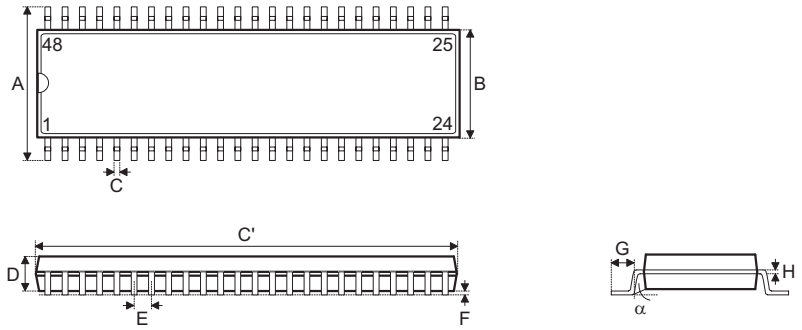
XOR A,[m]	Logical XOR Data Memory to ACC
Description	Data in the Accumulator and the specified Data Memory perform a bitwise logical XOR operation. The result is stored in the Accumulator.
Operation	ACC ← ACC ″XOR″ [m]
Affected flag(s)	Z

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

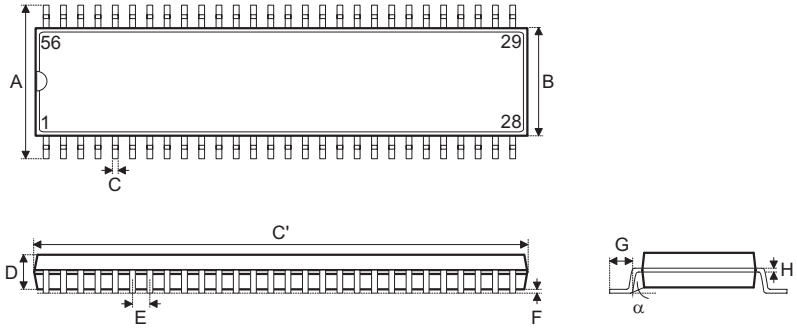
Package Information

48-pin SSOP (300mil) Outline Dimensions



Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	395	—	420
B	291	—	299
C	8	—	12
C′	613	—	637
D	85	—	99
E	—	25	—
F	4	—	10
G	25	—	35
H	4	—	12
α	0°	—	8°

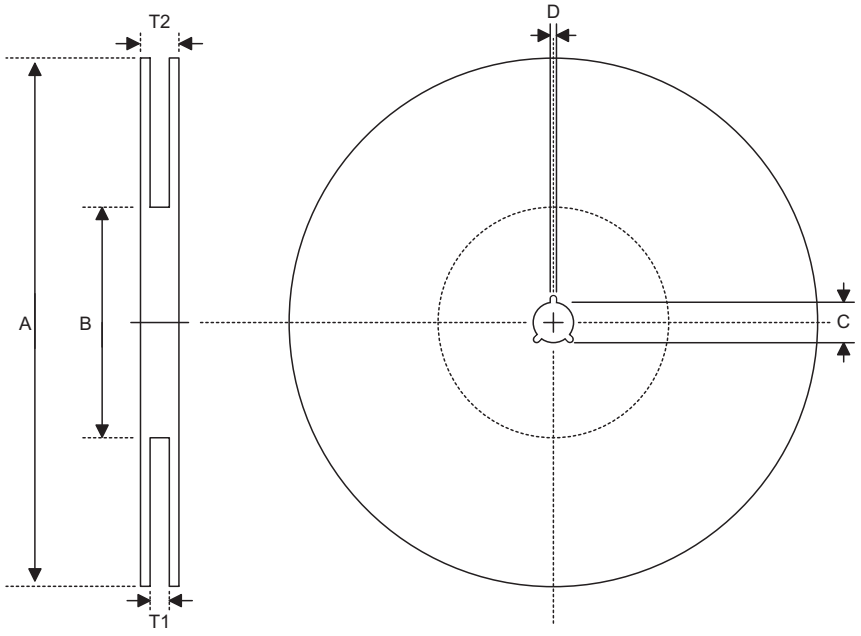
56-pin SSOP (300mil) Outline Dimensions



Symbol	Dimensions in mil		
	Min.	Nom.	Max.
A	395	—	420
B	291	—	299
C	8	—	12
C′	720	—	730
D	89	—	99
E	—	25	—
F	4	—	10
G	25	—	35
H	4	—	12
α	0°	—	8°

Product Tape and Reel Specifications

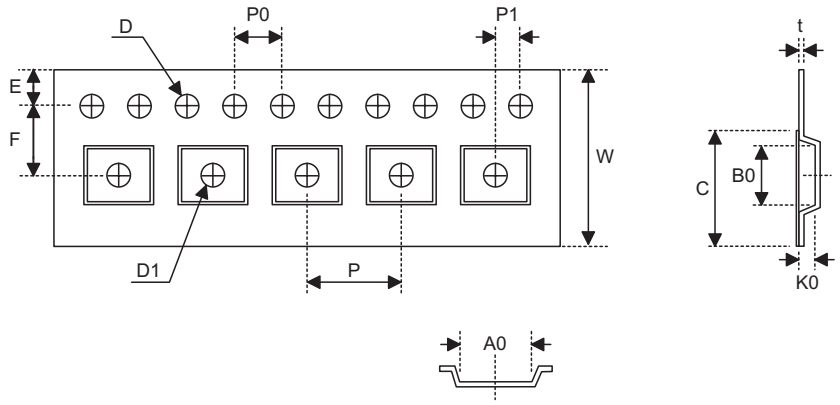
Reel Dimensions



SSOP 48W

Symbol	Description	Dimensions in mm
A	Reel Outer Diameter	330±1
B	Reel Inner Diameter	100±0.1
C	Spindle Hole Diameter	13+0.5 −0.2
D	Key Slit Width	2±0.5
T1	Space Between Flange	32.2+0.3 −0.2
T2	Reel Thickness	38.2±0.2

Carrier Tape Dimensions



SSOP 48W

Symbol	Description	Dimensions in mm
W	Carrier Tape Width	32±0.3
P	Cavity Pitch	16±0.1
E	Perforation Position	1.75±0.1
F	Cavity to Perforation (Width Direction)	14.2±0.1
D	Perforation Diameter	2 Min.
D1	Cavity Hole Diameter	1.5+0.25
P0	Perforation Pitch	4±0.1
P1	Cavity to Perforation (Length Direction)	2±0.1
A0	Cavity Length	12±0.1
B0	Cavity Width	16.2±0.1
K1	Cavity Depth	2.4±0.1
K2	Cavity Depth	3.2±0.1
t	Carrier Tape Thickness	0.35±0.05
C	Cover Tape Width	25.5

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

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

Holtek Semiconductor (China) Inc. (Dongguan Sales Office)
Building No. 10, Xinzhu Court, (No. 1 Headquarters), 4 Cuizhu Road, Songshan Lake, Dongguan, China 523808
Tel: 86-769-2626-1300
Fax: 86-769-2626-1311

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