

Technical Document

- [Tools Information](#)
- [Application Note](#)
 - [HA0075E MCU Reset and Oscillator Circuits Application Note](#)

Features

- Operating voltage:
 - $f_{SYS}=32768\text{Hz}$: 2.2V~5.5V
 - $f_{SYS}=4\text{MHz}$: 2.2V~5.5V
 - $f_{SYS}=8\text{MHz}$: 3.0V~5.5V
 - $f_{SYS}=12\text{MHz}$: 4.5V~5.5V
- Operating current:
 - $f_{SYS}=32\text{kHz}$ at 3.0V: 5 μA (typ.)
 - $f_{SYS}=1\text{MHz}$ at 3.0V: 140 μA (typ.)
- OTP Program Memory: 4K \times 15
- RAM Data Memory: 192 \times 8
- 24 bidirectional I/O lines
- TinyPower technology for low power operation
- Three pin-shared external interrupts lines and segment
- Single 8-bit programmable Timer/Event Counter with overflow interrupt and 7-stage prescaler
- Single 16-bit programmable Timer/Event Counter with overflow interrupt
- External Crystal, RC, RTC oscillator
- Fully integrated 32kHz oscillator
- Externally supplied system clock option
- Watchdog Timer function
- PFD/Buzzer for audio frequency generation
- Dual Serial Interfaces: SPI and I²C
- LCD driver: 33 \times 2, 33 \times 3 or 32 \times 4
- 4 operating modes: normal, slow, idle and sleep
- 8-level subroutine nesting
- 8-channel 12-bit resolution A/D converter
- 4-channel 12-bit PWM output shared with I/O lines
- Low voltage reset function – 2.1V, 3.15V, 4.2V
- Low voltage detect function – 2.2V, 3.3V, 4.4V
- Bit manipulation instruction
- 15-Bit table read instructions
- 63 powerful instructions
- Up to 0.33 μs instruction cycle with 12MHz system clock at $V_{DD}=5\text{V}$
- All instructions executed in one or two machine cycles
- Power down and wake-up functions to reduce power consumption
- 52-pin QFP, 64/100-pin LQFP packages

General Description

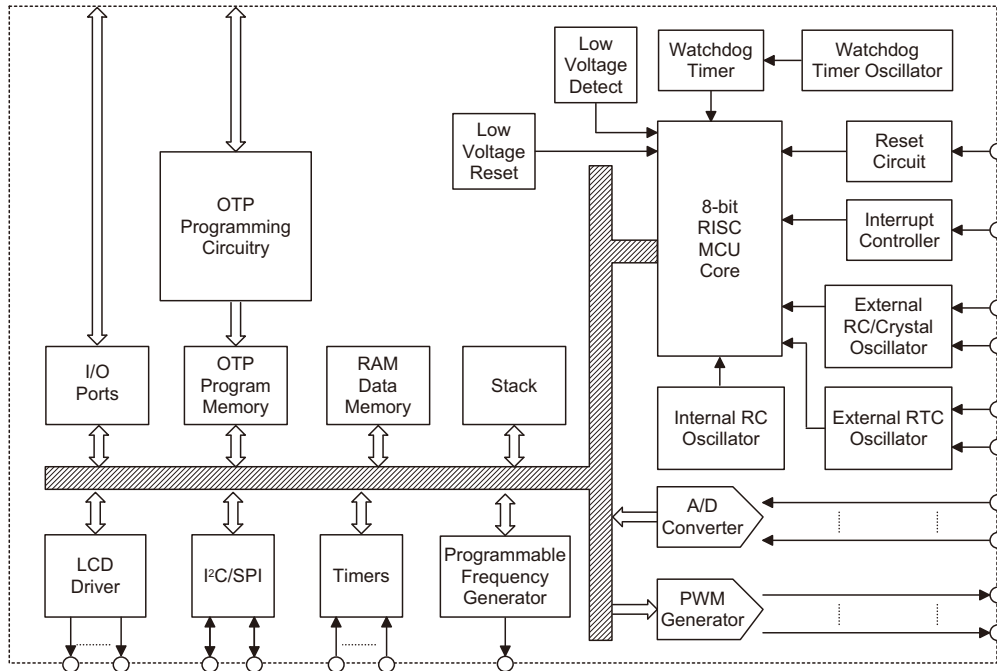
The HT56R64 is a TinyPower™ A/D Type with LCD 8-bit high performance RISC architecture microcontroller, designed especially for applications that interface directly to analog signals and which require an LCD interface. The device includes an integrated multi-channel Analog to Digital Converter, four Pulse Width Modulation outputs and an LCD driver.

With its fully integrated SPI and I²C functions, designers are provided with a means of easy communication with external peripheral hardware. The benefits of integrated A/D, LCD, and PWM functions, in addition to low power consumption, high performance, I/O flexibility and low-cost, provides the device with the versatility for a

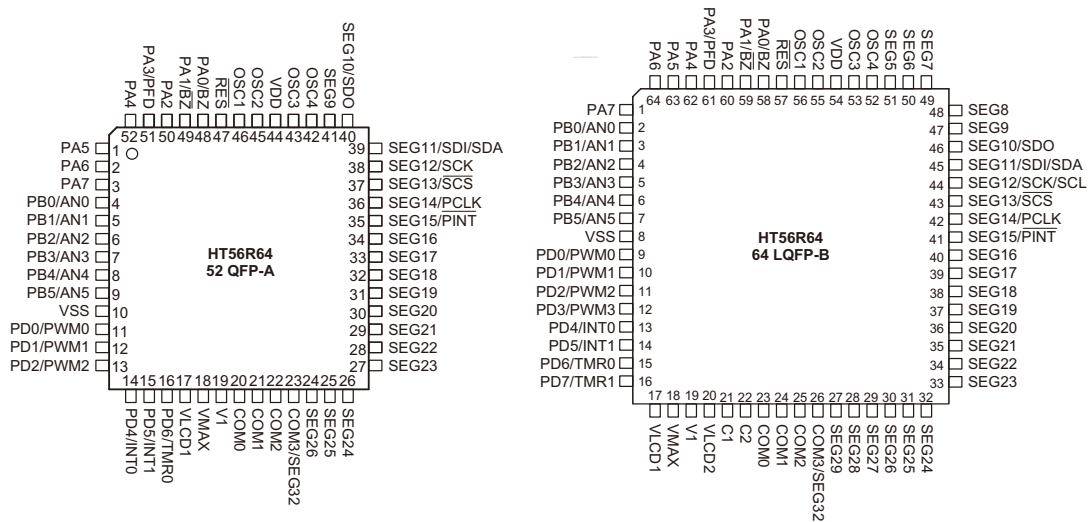
wide range of products in the home appliance and industrial application areas. Some of these products could include electronic metering, environmental monitoring, handheld instruments, electronically controlled tools, motor driving in addition to many others.

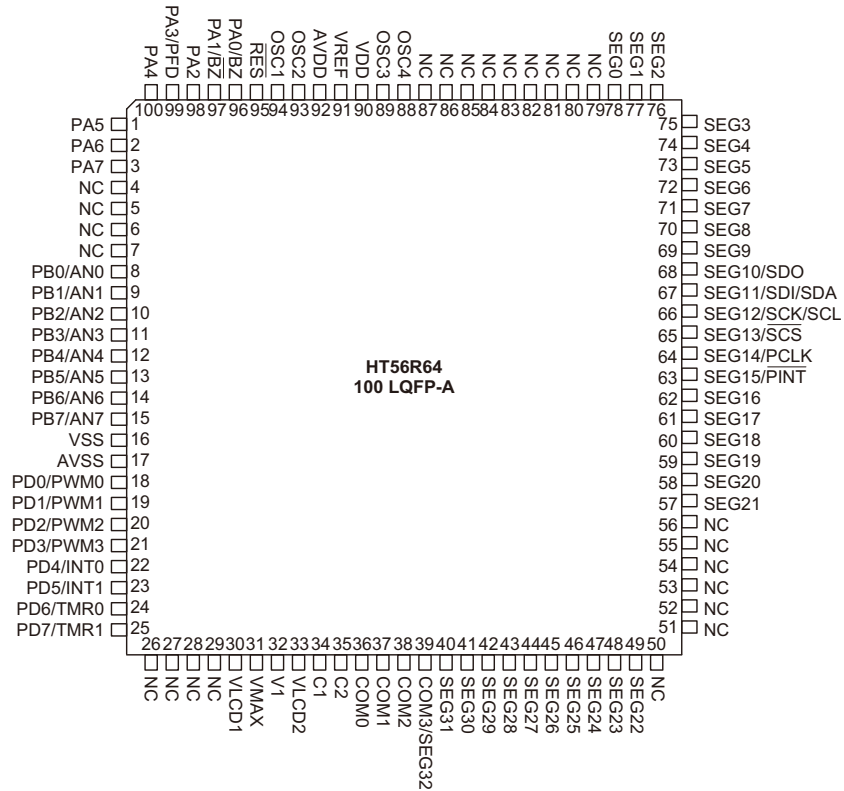
The unique Holtek TinyPower technology also gives the device extremely low current consumption characteristics, an extremely important consideration in the present trend for low power battery powered applications. The usual Holtek MCU features such as power down and wake-up functions, oscillator options, programmable frequency divider, etc. combine to ensure user applications require a minimum of external components.

Block Diagram



Pin Assignment




Pin Description

| Pin Name | I/O | Configuration Option | Description |
|---|-----|----------------------|--|
| PA0/BZ PA1/BZ PA2 PA3/PFD PA4~PA7 | I/O | BZ/BZ PFD | Bidirectional 8-bit input/output port. Each individual bit on this port can be configured as a wake-up input using the PAWU register. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. A pull-high resistor can be connected to each pin using the PAPU register. Pins PA0, PA1 and PA3 are shared with BZ, BZ and PFD respectively, the function of which is chosen via configuration option. Pins PA0~PA3 can also be setup as open drain pins using the MISC register. |
| PB0/AN0~ PB7/AN7 | I/O | — | Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. A pull-high resistor can be connected to each pin using the PBPU register. PB is pin-shared with the A/D input pins. The A/D inputs are selected via software instructions. Once selected as an A/D input, the I/O function and pull-high resistor selections are disabled automatically. |
| PD0/PWM0~ PD3/PWM3 PD4/INT0 PD5/INT1 PD6/TMR0 PD7/TMR1 | I/O | — | Bidirectional 8-bit input/output port. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. A pull-high resistor can be connected to each pin using the PDPU register. The PWM outputs, PWM0~PWM3, are pin shared with pins PD0~PD3, the function of which is chosen using the PWM registers. Pins PD4~PD7 are pin-shared with INT0, INT1, TMR0 and TMR1 respectively. |
| COM0~COM2 COM3/SEG32 | O | — | COM2~COM0 are the LCD common outputs. A bit in the LCD Control Register determines if pin COM3/SEG32 is configured as a segment driver or as a common output driver. |
| SEG31~SEG24 | O | — | LCD driver outputs for LCD panel segments. |

| Pin Name | I/O | Configuration Option | Description |
|---|---|---|---|
| SEG23~SEG16 | O | — | A bit in the LCD Control Register determines if the pins are to be used as segment drivers or as CMOS outputs. |
| SEG8~SEG9 SEG10/SDO SEG11/SDI/SDA SEG12/SCK/SCL SEG13/SCS SEG14/PCLK SEG15/PINT | O O I/O I/O I/O O I/O | I ² C SPI PCLK PINT | SEG8 and SEG9 are LCD driver outputs. SEG10 is pin-shared with the Serial Interface Output line, SDO. SEG11 is pin-shared with the SPI Bus data line, SDI and the I ² C Bus data line SDA. SEG12, is pin-shared with the SPI Bus clock line, SCK, and the I ² C Bus clock line SCL. SEG13 is pin-shared with the Serial Interface Select line, $\overline{\text{SCS}}$. SEG14 is pin-shared with the Peripheral Clock line, PCLK. SEG15 is pin-shared with the Peripheral Interrupt line, $\overline{\text{PINT}}$. All of the SEG8~SEG15 lines can be chosen to be either segment drivers or as logical outputs using LCD control bits. |
| SEG0~SEG7 | O | — | LCD driver outputs for LCD panel segments. SEG0~SEG7 can be chosen to be either segment drivers or as logical outputs using LCD control bits. |
| OSC1 OSC2 | I O | Crystal or RC or EC | 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. EC is external clock mode, we can input clock source directly to OSC1 pin. |
| OSC3 OSC4 | I O | RTC | OSC3 and OSC4 are connected to a 32768Hz crystal oscillator to form a real time clock for f_{SUB} or f_{SL} . |
| $\overline{\text{RES}}$ | I | — | Schmitt Trigger reset input. Active low. |
| VLCD1 | — | — | LCD power supply |
| VREF | I | — | Reference voltage input pin. |
| VMAX | I | — | IC maximum voltage, connect to V_{DD} , V_{LCD} or V1 |
| V1, VLCD2, C1, C2 | I | — | LCD voltage pump |
| VDD | — | — | Positive power supply |
| AVDD | — | — | Analog positive power supply. |
| VSS | — | — | Negative power supply, ground |
| AVSS | — | — | Analog negative power supply, ground |

Note: The Pin Description table represents the largest package available, therefore some of the pins and functions may not be available on smaller package types.

Absolute Maximum Ratings

| | | | |
|-------------------------------|--|-----------------------------|--|
| Supply Voltage | $V_{\text{SS}}-0.3\text{V}$ to $V_{\text{SS}}+6.0\text{V}$ | Storage Temperature | -50°C to 125°C |
| Input Voltage | $V_{\text{SS}}-0.3\text{V}$ to $V_{\text{DD}}+0.3\text{V}$ | Operating Temperature | -40°C to 85°C |
| I_{OL} Total | 150mA | I_{OH} Total | -100mA |
| Total Power Dissipation | 500mW | | |

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 | 3.0 | — | 5.5 | V |
| | | | f _{SYS} =12MHz | 4.5 | — | 5.5 | V |
| AV _{DD} | Analog Operating Voltage | — | V _{REF} =AV _{DD} | 3.0 | — | 5.0 | V |
| I _{DD1} | Operating Current (Crystal OSC, RC OSC) | 3V | No load, f _{SYS} =f _M =1MHz ADC off | — | 140 | 210 | μA |
| | | 5V | | — | 320 | 480 | μA |
| I _{DD2} | Operating Current (Crystal OSC, RC OSC) | 3V | No load, f _{SYS} =f _M =2MHz ADC off | — | 200 | 300 | μA |
| | | 5V | | — | 440 | 660 | μA |
| I _{DD3} | Operating Current (Crystal OSC, RC OSC) | 3V | No load, f _{SYS} =f _M =4MHz ADC off | — | 400 | 600 | μA |
| | | 5V | | — | 800 | 1200 | μA |
| I _{DD4} | Operating Current (EC Mode, Filter On) | 3V | No load, f _{SYS} =f _M =4MHz ADC off | — | 320 | 480 | μA |
| | | 5V | | — | 550 | 820 | μA |
| I _{DD5} | Operating Current (EC Mode, Filter Off) | 3V | No load, f _{SYS} =f _M =4MHz ADC off | — | 300 | 450 | μA |
| | | 5V | | — | 530 | 800 | μA |
| I _{DD6} | Operating Current (Crystal OSC, RC OSC) | 5V | No load, f _{SYS} =f _M =8MHz ADC off | — | 1.5 | 3.0 | mA |
| I _{DD7} | Operating Current (Crystal OSC, RC OSC) | 5V | No load, f _{SYS} =f _M =12MHz ADC off | — | 2.0 | 4.0 | mA |
| I _{DD8} | Operating Current (Slow Mode, f _M =4MHz) (Crystal OSC, RC OSC) | 3V | No load, f _{SYS} =f _{SLOW} =500kHz ADC off | — | 130 | 200 | μA |
| | | 5V | | — | 300 | 450 | μA |
| I _{DD9} | Operating Current (Slow Mode, f _M =4MHz) (Crystal OSC, RC OSC) | 3V | No load, f _{SYS} =f _{SLOW} =1MHz ADC off | — | 170 | 260 | μA |
| | | 5V | | — | 370 | 560 | μA |
| I _{DD10} | Operating Current (Slow Mode, f _M =4MHz) (Crystal OSC, RC OSC) | 3V | No load, f _{SYS} =f _{SLOW} =2MHz ADC off | — | 250 | 380 | μA |
| | | 5V | | — | 520 | 780 | μA |
| I _{DD11} | Operating Current (Slow Mode, f _M =8MHz) (Crystal OSC, RC OSC) | 3V | No load, f _{SYS} =f _{SLOW} =1MHz ADC off | — | 220 | 330 | μA |
| | | 5V | | — | 480 | 720 | μA |
| I _{DD12} | Operating Current (Slow Mode, f _M =8MHz) (Crystal OSC, RC OSC) | 3V | No load, f _{SYS} =f _{SLOW} =2MHz ADC off | — | 300 | 450 | μA |
| | | 5V | | — | 630 | 950 | μA |
| I _{DD13} | Operating Current (Slow Mode, f _M =8MHz) (Crystal OSC, RC OSC) | 3V | No load, f _{SYS} =f _{SLOW} =4MHz ADC off | — | 460 | 690 | μA |
| | | 5V | | — | 920 | 1380 | μA |
| I _{DD14} | Operating Current (f _{SYS} =32768Hz (note 1) or 32K_INT internal RC OSC) | 3V | No load, WDT off, ADC off, LCD on (note 2), R type, V _{LCD} =V _{DD} , 1/2 bias (R _{BIAS} =400kΩ) | — | 12 | 18 | μA |
| | | 5V | | — | 18 | 24 | μA |
| I _{DD15} | Operating Current (f _{SYS} =32768Hz (note 1) or 32K_INT internal RC OSC) | 3V | No load, WDT off, ADC off, LCD on (note 2), R type, V _{LCD} =V _{DD} , 1/3 bias (R _{BIAS} =600kΩ) | — | 10 | 15 | μA |
| | | 5V | | — | 15 | 22 | μA |

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|-------------------|--|-----------------|---|--------------------|------|--------------------|------|
| | | V _{DD} | Conditions | | | | |
| I _{DD16} | Operating Current (f _{SYS} =32768Hz (note 1) or 32K_INT internal RC OSC) | 3V | No load, WDT off, ADC off, LCD on (note 2), C type | — | 6 | 9 | μA |
| | | 5V | 1/3 bias, V _{LCD} =3V | — | 9 | 13 | μA |
| I _{DD17} | Operating Current (f _{SYS} =32768Hz (note 1) or 32K_INT internal RC OSC) | 3V | No load, LCD off, WDT off, ADC off | — | 5 | 8 | μA |
| | | 5V | | — | 8 | 12 | μA |
| I _{STB1} | Standby Current (Sleep) (f _{SYS} , f _{SUB} , f _S , f _{LCD} , f _{WDT} =off) | 3V | No load, system HALT, WD off | — | 0.1 | 1.0 | μA |
| | | 5V | | — | 0.2 | 2.0 | μA |
| I _{STB2} | Standby Current (Sleep) (f _{SYS} , f _{LCD} =off; f _{LCD} , f _{WDT} =f _{SUB} =32768Hz (note 1) or 32K_INT RC OSC) | 3V | No load, system HALT, WDT on | — | 1 | 2 | μA |
| | | 5V | | — | 3 | 5 | μA |
| I _{STB3} | Standby Current (Idle) (f _{SYS} , f _{WDT} =off; f _S (note 3)=f _{SUB} =32768Hz (note 1) or 32K_INT RC OSC) | 3V | No load, system HALT, WDT off, LCD on (note 2), | — | 1 | 2 | μA |
| | | 5V | 1/2 bias, C type, V _{LCD} =V _{DD} | — | 3 | 5 | μA |
| I _{STB4} | Standby Current (Idle) (f _{SYS} , f _{WDT} =off; f _S (note 3)=f _{SUB} =32768Hz (note 1) or 32K_INT RC OSC) | 3V | No load, system HALT, WDT off, LCD on (note 2), | — | 1 | 2 | μA |
| | | 5V | 1/3 bias, C type, V _{LCD} =3V | — | 3 | 5 | μA |
| I _{STB5} | Standby Current (Idle) (f _{SYS} , f _{WDT} =off; f _S (note 3)=f _{SUB} =32768Hz (note 1) or 32K_INT RC OSC) | 3V | No load, system HALT, WDT off, LCD on (note 2), | — | 10 | 15 | μA |
| | | 5V | R type, V _{LCD} =V _{DD} , 1/2 bias (R _{BIAS} =400kΩ) | — | 15 | 22 | μA |
| I _{STB6} | Standby Current (Idle) (f _{SYS} , f _{WDT} =off; f _S (note 3)=f _{SUB} =32768Hz (note 1) or 32K_INT RC OSC) | 3V | No load, system HALT, WDT off, LCD on (note 2), | — | 6 | 9 | μA |
| | | 5V | R type, V _{LCD} =V _{DD} , 1/3 bias (R _{BIAS} =600kΩ) | — | 10 | 15 | μA |
| I _{STB7} | Standby Current (Idle) (f _{SYS} =on, f _{SYS} =f _M =4MHz, f _{WDT} , f _{LCD} =off, f _S (note 3)=f _{SUB} =32768Hz (note 1) or 32K_INT RC OSC) | 3V | No load, system HALT, WDT off, LCD off, SPI or I ² C on, PCLK on, PCLK=f _{SYS} /8 | — | 150 | 220 | μA |
| | | 5V | | — | 350 | 530 | μA |
| V _{IL1} | Input Low Voltage for I/O Ports, TMR and INT | — | — | 0 | — | 0.3V _{DD} | V |
| V _{IH1} | Input High Voltage for I/O Ports, TMR and 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 Voltage | — | 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 |
| V _{LVD} | Low Voltage Detector Voltage | — | Configuration option: 2.2V | 2.08 | 2.20 | 2.32 | V |
| | | — | Configuration option: 3.3V | 3.12 | 3.30 | 3.50 | V |
| | | — | Configuration option: 4.4V | 4.12 | 4.40 | 4.70 | V |
| I _{OL1} | I/O Port Sink Current | 3V | V _{OL} =0.1V _{DD} | 6 | 12 | — | mA |
| | | 5V | | 10 | 25 | — | mA |

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|------------------|------------------------------------|-----------------|-------------------------------------|------|------|------|------|
| | | V _{DD} | Conditions | | | | |
| I _{OH1} | I/O Port Source Current | 3V | V _{OH} =0.9V _{DD} | -2 | -4 | — | mA |
| | | 5V | | -5 | -8 | — | mA |
| I _{OL2} | LCD Common and Segment Current | 3V | V _{OL} =0.1V _{DD} | 210 | 420 | — | μA |
| | | 5V | | 350 | 700 | — | μA |
| I _{OH2} | LCD Common and Segment Current | 3V | V _{OH} =0.9V _{DD} | -80 | -160 | — | μA |
| | | 5V | | -180 | -360 | — | μA |
| R _{PH} | Pull-high Resistance for I/O Ports | 3V | — | 20 | 60 | 100 | kΩ |
| | | 5V | | 10 | 30 | 50 | kΩ |

- Note:
- 32768Hz is in slow start mode (RTCC.4=1) for the D.C. current measurement.
 - LCD waveform is in Type A condition.
 - f_S is the internal clock for Buzzer, RTC, Time base and WDT.
 - Both Timer/Event Counters are off. Timer filter is disabled for all test conditions.

A.C. Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---------------------|---|-----------------|---------------------------------|------|-------|-------|--------------------|
| | | V _{DD} | Conditions | | | | |
| f _{SYS1} | System Clock (Crystal OSC, RC OSC) | — | 2.2V~5.5V | 32 | — | 4000 | kHz |
| | | | 3.0V~5.5V | 32 | — | 8000 | kHz |
| | | | 4.5V~5.5V | 32 | — | 12000 | kHz |
| f _{SYS2} | System Clock (RTC Crystal OSC) | — | 2.2V~5.5V | — | 32768 | — | Hz |
| f _{RTCOSC} | RTC Frequency | — | — — | — | 32768 | — | Hz |
| f _{TIMER} | Timer I/P Frequency (TMR0/TMR1) | — | 2.2V~5.5V | 0 | — | 4000 | kHz |
| | | | 3.3V~5.5V | 0 | — | 8000 | kHz |
| | | | 4.5V~5.5V | 0 | — | 12000 | kHz |
| f _{RC32K} | 32K RC Oscillator | — | 2.2V~5.5V, After Trim | 28.8 | 32.0 | 35.2 | kHz |
| t _{RES} | External Reset Low Pulse Width | — | — | 1 | — | — | μs |
| t _{LVR} | Low Voltage Reset Time | — | — | 0.1 | 0.4 | 0.6 | ms |
| t _{LVDO} | Time for LVDO Become Stable, LVDC is Enabled | — | — | — | — | 100 | μs |
| t _{SST1} | System Start-up Timer Period | — | Power-on | — | 1024 | — | t _{sys} * |
| t _{SST2} | System Start-up Timer Period for XTAL or RTC Oscillator | — | Wake-up from Power Down Mode | — | 1024 | — | t _{sys} * |
| t _{SST3} | System Start-up Timer Period for External RC or External Clock | — | Wake-up from Power Down Mode | — | 1 | 2 | t _{sys} * |
| t _{INT} | Interrupt Pulse Width | — | — | 1 | — | — | μs |

Note: *t_{sys}=1/f_{sys1} or 1/f_{sys2}

ADC Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|------------------|---|-----------------|--|------|------|-----------------------|-----------------|
| | | V _{DD} | Conditions | | | | |
| V _{AD} | A/D Input Voltage | — | 52QFP, 64LQFP | 0 | — | A _{VDD} | V |
| | | | 100LQFP | 0 | — | V _{REF} | V |
| V _{REF} | A/D Input Reference Voltage Range | — | A _{VDD} =5V | 1.6 | — | A _{VDD} +0.1 | V |
| DNL | A/C Differential Non-Linearity | — | A _{VDD} =5V, V _{REF} =A _{VDD} , t _{AD} =0.5μs | -2 | — | 2 | LSB |
| INL | ADC Integral Non-Linearity | — | A _{VDD} =5V, V _{REF} =A _{VDD} , t _{AD} =0.5μs | -4 | — | 4 | LSB |
| I _{ADC} | Additional Power Consumption if A/D Converter is Used | 3V | — | — | 0.50 | 0.75 | mA |
| | | 5V | | — | 1.00 | 1.50 | mA |
| t _{AD} | A/D Clock Period | — | — | 0.5 | — | — | μs |
| t _{ADC} | A/D Conversion Time | — | — | — | 16 | — | t _{AD} |

Power-on Reset Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|-------------------|---|-----------------|------------|------|------|------|------|
| | | V _{DD} | Conditions | | | | |
| V _{POR} | VDD Start Voltage to Ensure Power-on Reset | — | — | — | — | 0 | mV |
| RR _{VDD} | VDD raising rate to Ensure Power-on Reset | — | — | 0.05 | — | — | V/ms |
| t _{POR} | Minimum Time for VDD Stays at V _{POR} to Ensure Power-on Reset | — | — | 200 | — | — | ms |

System Architecture

A key factor in the high-performance features of the Holtek range of microcontrollers is attributed to their 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 instruction set operations, which 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 the device suitable for low-cost, high-volume production for controller applications.

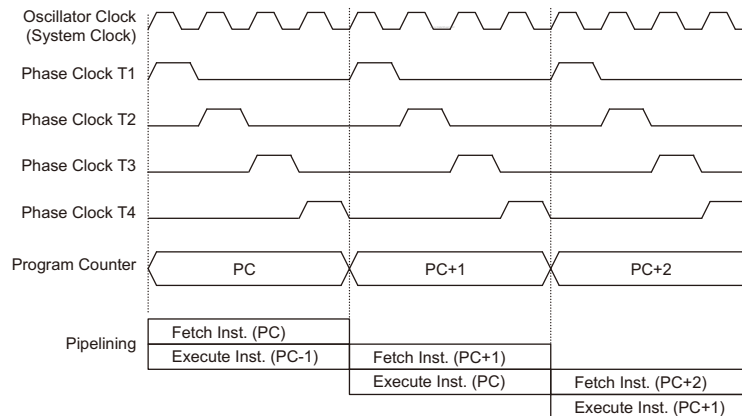
Clocking and Pipelining

The main system clock, derived from either a Crystal/Resonator or RC oscillator is subdivided into four in-

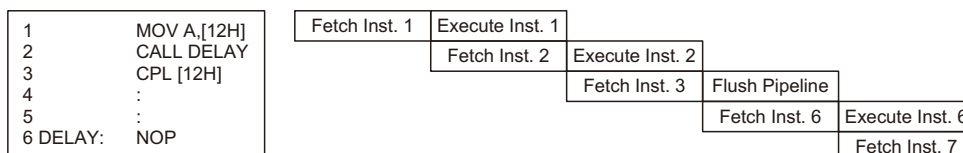
ternally generated non-overlapping clocks, T1~T4. The Program Counter is incremented at the beginning of the T1 clock during which time a new instruction is fetched. The remaining T2~T4 clocks carry out the decoding and execution functions. In this way, one T1~T4 clock cycle forms one instruction cycle. 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 free 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



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. It must be noted that only the lower 8 bits, known as the Program Counter Low Register, are directly addressable.

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 writable 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.

The lower byte of the Program Counter is fully accessible under program control. Manipulating the PCL might cause program branching, so an extra cycle is needed to pre-fetch. Further information on the PCL register can be found in the Special Function Register section.

| Mode | Program Counter Bits | | | | | | | | | | | |
|--------------------------------|----------------------|------|-----|-----|----|----|----|----|----|----|----|----|
| | 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 |
| External Interrupt 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| External Interrupt 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| Timer/Event Counter 0 Overflow | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| Timer/Event Counter 1 Overflow | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| SPI/I ² C Interrupt | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| Multi-Function Interrupt | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| Skip | Program Counter + 2 | | | | | | | | | | | |
| Loading PCL | PC11 | PC10 | PC9 | PC8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| Jump, Call Branch | #11 | #10 | #9 | #8 | #7 | #6 | #5 | #4 | #3 | #2 | #1 | #0 |
| Return from Subroutine | S11 | S10 | S9 | S8 | S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 |

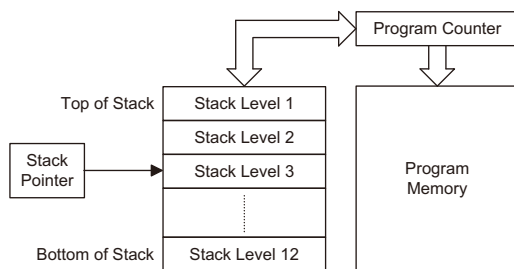
Program Counter

Note: PC11~PC8: Current Program Counter bits
 #11~#0: Instruction code address bits

@7~@0: PCL bits
 S11~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 12 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. For this device the Program Memory is an OTP type, which means it can be programmed only one time. By using the appropriate programming tools, this OTP memory device offer users the flexibility to conveniently debug and develop their applications while also offering a means of field programming.

The program memory is used to store the program instructions, which are to be executed. It also contains data, table, and interrupt entries, and is organized into 4096×15 bits format which are addressed by the program counter and table pointer. The JMP and CALL instructions provide only 11 bits of address to allow branching within any 4K program memory. When doing a JMP or CALL instruction.

Structure

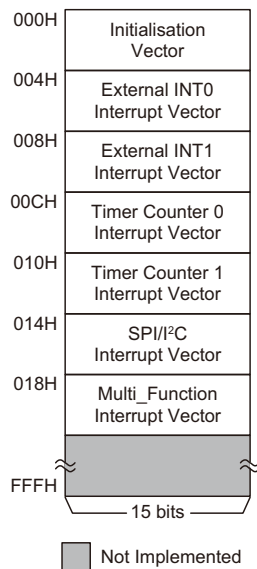
The Program Memory has a capacity of 4K by 15 bits. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries. Table data, which can be setup in any location within the Program Memory, is addressed by a separate table pointer register.

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 0. If the external interrupt pin receives an active edge, the program will jump to this location and begin execution if the external interrupt is enabled and the stack is not full.
- Location 008H
This vector is used by the external interrupt 1. If the external interrupt pin receives an active edge, the program will jump to this location and begin execution if the external interrupt is enabled and the stack is not full.
- Location 00CH
This internal vector is used by the 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 interrupt is enabled and the stack is not full.

- Location 010H
This internal vector is used by the 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 interrupt is enabled and the stack is not full.
- Location 014H
This internal vector is used by the SPI/I²C interrupt. When either an SPI or I²C bus, dependent upon which one is selected, requires data transfer, the program will jump to this location and begin execution if the SPI/I²C interrupt is enabled and the stack is not full.
- Location 018H
This internal vector is used by the Multi-function Interrupt. When the Time Base overflows, the Real Time Clock overflows, the A/D converter completes its conversion process, or an active edge appears on the External Peripheral interrupt pin, the program will jump to this location and begin execution if the relevant interrupt is enabled and the stack is not full.



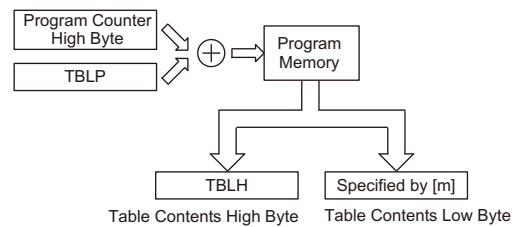
Program Memory Structure

Look-up Table

Any location within the Program Memory can be defined as a look-up table where programmers can store fixed data. To use the look-up table, the table pointer must first be setup by placing the lower order address of the look up data to be retrieved in the table pointer register, TBLP. This register defines the lower 8-bit address of the look-up table.

After setting up the table pointer, 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:



| Instruction | Table Location Bits | | | | | | | | | | | |
|-------------|---------------------|------|-----|-----|----|----|----|----|----|----|----|----|
| | b11 | b10 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| TABRDC [m] | PC11 | PC10 | PC9 | PC8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| TABRDL [m] | 1 | 1 | 1 | 1 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |

Table Location

Note: PC11~PC8: Current program counter bits

@7~@0: Table Pointer TBLP bits

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 "700H" which refers to the start address of the last page within the 2K Program Memory of the device. The table pointer is setup here to have an initial value of "06H". This will ensure that the first data read from the data table will be at the Program Memory address "706H" or 6 locations after the start of the last page. Note that the value for the table pointer is referenced to the first address of the present page if the "TABRDC [m]" instruction is being used. The high byte of the table data which in this case is equal to zero will be transferred to the TBLH register automatically when the "TABRDL [m]" instruction is executed.

Because the TBLH register is a read-only register and cannot be restored, care should be taken to ensure its protection if both the main routine and Interrupt Service Routine use 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.

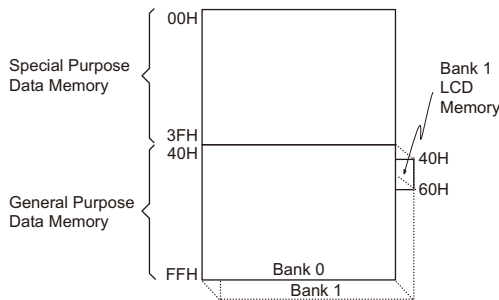
```

Tempreg1      db  ?   ; temporary register #1
Tempreg2      db  ?   ; temporary register #2
:
:
mov          a,06h      ; initialise table pointer - note that this address
                    ; is referenced
mov          tblp,a     ; to the last page or present page
:
:
tabrdl      tempreg1   ; transfers value in table referenced by table pointer
                    ; to tempreg1
                    ; data at prog. memory address "706H" transferred to
                    ; tempreg1 and TBLH
dec         tblp       ; reduce value of table pointer by one
tabrdl      tempreg2   ; transfers value in table referenced by table pointer
                    ; to tempreg2
                    ; data at prog.memory address "705H" transferred to
                    ; tempreg2 and TBLH
                    ; in this example the data "1AH" is transferred to
                    ; tempreg1 and data "0FH" to register tempreg2
:
:
org         700h       ; sets initial address of last page
dc          00Ah, 00Bh, 00Ch, 00Dh, 00Eh, 00Fh, 01Ah, 01Bh
:
:

```

Data Memory

The Data Memory is a volatile area of 8-bit wide RAM internal memory and is the location where temporary information is stored. Divided into three sections, the first of these is an area of RAM where special function registers are located. These registers have fixed locations and are necessary for correct operation of the device. Many of these registers can be read from and written to directly under program control, however, some remain protected from user manipulation. The second area of Data Memory is reserved for general purpose use. All locations within this area are read and write accessible under program control. The third area is reserved for the



Bank 0 RAM 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 pointer registers MP0 and MP1.

LCD Memory. This special area of Data Memory is mapped directly to the LCD display so data written into this memory area will directly affect the displayed data. The addresses of the LCD Memory area overlap those in the General Purpose Data Memory area, switching between the two areas is achieved by setting the Bank Pointer to the correct value.

Structure

The Data Memory is subdivided into 2 banks, known as Bank 0 and Bank 1, all of which are implemented in 8-bit wide RAM. RAM Data Memory located in Bank 0 is subdivided into two sections, the Special Purpose Data Memory and the General Purpose Data Memory. The start address of the RAM Data Memory for all devices is the address "00H". The last Data Memory address is "FFH".

Bank 1 is for LCD display memory which occupy 128x8 location.


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.

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 read and write type but some are protected and are read 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 | 20H | PWM2L |
| 01H | MP0 | 21H | PWM2H |
| 02H | IAR1 | 22H | PWM3L |
| 03H | MP1 | 23H | PWM3H |
| 04H | BP | 24H | ADRL |
| 05H | ACC | 25H | ADRH |
| 06H | PCL | 26H | ADCR |
| 07H | TBLP | 27H | ACSR |
| 08H | TBLH | 28H | CLKMOD |
| 09H | RTCC | 29H | PAWU |
| 0AH | STATUS | 2AH | PAPU |
| 0BH | INTC0 | 2BH | PBPU |
| 0CH | | 2CH | |
| 0DH | TMR0 | 2DH | PDPU |
| 0EH | TMR0C | 2EH | INTEDGE |
| 0FH | TMR1H | 2FH | |
| 10H | TMR1L | 30H | LCDCTRL |
| 11H | TMR1C | 31H | LCDOUT1 |
| 12H | PA | 32H | LCDOUT2 |
| 13H | PAC | 33H | MISC |
| 14H | PB | 34H | MFIC |
| 15H | PBC | 35H | |
| 16H | | 36H | SIMCON0 |
| 17H | | 37H | SIMCON1 |
| 18H | PD | 38H | SIMDR |
| 19H | PDC | 39H | SIMAR/SIMCON2 |
| 1AH | PWM0L | 3AH | |
| 1BH | PWM0H | 3BH | |
| 1CH | PWM1L | 3CH | |
| 1DH | PWM1H | 3DH | |
| 1EH | INTC1 | 3EH | |
| 1FH | | 3FH | |

 : Unused Read as "00"

Special Purpose RAM Data Memory

LCD Memory

The data to be displayed on the LCD is also stored in an area of fully accessible Data Memory. By writing to this area of RAM, the LCD display output can be directly controlled by the application program. As the LCD Memory exists in Bank 1, but have addresses which map into the General Purpose Data Memory, it is necessary to first ensure that the Bank Pointer is set to the value 01H before accessing the LCD Memory. The LCD Memory can only be accessed indirectly using the Memory Pointer MP1 and the indirect addressing register IAR1. When the Bank Pointer is set to Bank 1 to access the LCD Data Memory, if any addresses with a value less than 40H are read, the Special Purpose Memory in Bank 0 will be accessed. Also, if the Bank Pointer is set to Bank 1, if any addresses higher than the last address in Bank 1 are read, then a value of 00H will be returned.

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 for future expansion purposes, attempting to read data from these locations will return a value of 00H.

Indirect Addressing Registers – IAR0, IAR1

The Indirect Addressing Registers, IAR0 and IAR1, although having their locations in normal RAM register space, do not actually physically exist as normal registers. The method of indirect addressing for RAM data 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 access data Bank 0 while the IAR1 and MP1 register pair can access data from Bank 0 and Bank 1. As the Indirect Addressing Registers are not physically implemented, reading the Indirect Addressing Registers indirectly will return a result of "00H" and writing to the registers indirectly will result in no operation.

Memory Pointers – MP0, MP1

For all devices, two Memory Pointers, known as MP0 and MP1 are provided. These Memory Pointers are physically implemented in the Data Memory and can be manipulated in the same way as normal registers providing a convenient way with which to 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, while MP1 and IAR1 are used to access data from Bank 0 and Bank 1.

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 RAM addresses.

Bank Pointer – BP

The Data Memory is divided into 2 Banks, known as Bank 0 and Bank 1. Selecting the required Data Memory area is achieved using the Bank Pointer. If data in Bank 0 is to be accessed, then the BP register must be loaded with the value 00H, while if data in Bank 1 is to be accessed, then the BP register must be loaded with the value 01H.

The Data Memory is initialised to Bank 0 after a reset, except for the WDT time-out reset in the Power Down Mode, in which case, the Data Memory bank remains unaffected. 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 either Bank 0 or Bank 1. Directly addressing the Data Memory will always result in Bank 0 being accessed irrespective of the value of the Bank Pointer.

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.

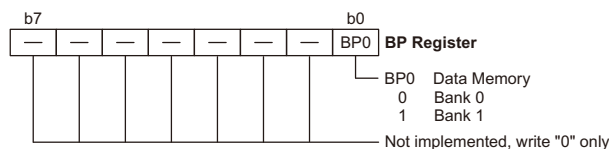
Look-up Table Registers – TBLP, TBLH

These two special function registers are used to control operation of the look-up table which is stored in the Program Memory. TBLP is the table pointer and indicates the location where the table data is located. Its value must be setup before any table read commands are executed. Its value 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.

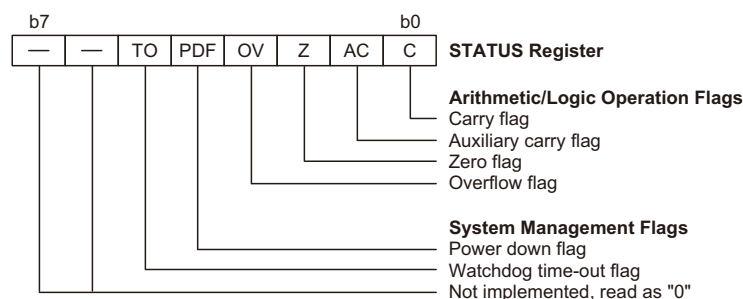
Status Register – STATUS

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



Bank Pointer



Status Register

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 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.

Interrupt Control Register – INTC0, INTC1, MFIC, INTEDGE

These 8-bit registers, control the operation of the device interrupt functions. By setting various bits within these registers 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. The INTEDGE register is used to select the active edges for the two external interrupt pins INT0 and INT1.

Timer/Event Counter Registers – TMR0, TMR1L/ TMR1H, TMR0C, TMR1C

The device contains one internal 8-bit Timer/Event Counter and one 16-bit Timer/Event Counter. The

registers TMR0 and TMR1L/TMR1H are the locations where the timer values are located. These register can also be preloaded with fixed data to allow different time intervals to be setup. Two associated control registers, TMR0C and TMR1C, contains the setup information for these timers, which determines in what mode the timer 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 and PD. 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 and PDC, 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 initialization, 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

The device contains four Pulse Width Modulator function with their own related independent control register, known as PWM0L, PWM0H, PWM1L, PWM1H, PWM2L, PWM2H, PWM3L and PWM3H. The 12-bit contents of each register pair, defines the duty cycle value for the modulation cycle of the 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 and two control registers. The two data registers, a high byte data register known as ADRH, and a low byte data register known as ADRL, are the register locations where the digital value is placed after the completion of an analog to digital conversion cycle. Functions such as the A/D enable/disable, A/D channel selection and A/D clock frequency are determined using the two control registers, ADCR and ACSR.

Serial Interface Registers

The device contains two serial interfaces, an SPI and an I²C interface. The SIMCON0, SIMCON1, SIMAR/SIMCON2 are the control registers for the Serial Interface function while the SIMDR is the data register for the Serial Interface Data.

Port A Wake-up Register – PAWU

All pins on Port A have a wake-up function enable a low going edge on these pins to wake-up the device when it is in a power down mode. The pins on Port A that are used to have a wake-up function are selected using this register.

Pull-High Resistors – PAPU, PBPU, PDPU

All I/O pins on Ports PA, PB and PD, if setup as inputs, can be connected to an internal pull-high resistor. The pins which require a pull-high resistor to be connected are selected using these registers.

Register – CLKMOD

The device operates using a dual clock system whose mode is controlled using this register. The register controls functions such as the clock source, the idle mode enable and the division ratio for the slow clock.

LCD Registers – LCDCTRL, LCDOUT1, LCDOUT2

The device contains a fully integrated LCD Driver function which can be setup in various configurations allowing it to control a wide range of external LCD panels. Most of these options are controlled using the LCDCTRL register. As some of the LCD segment driving pins can also be setup to be used as CMOS outputs, two registers, LCDOUT1 and LCDOUT2, are used to select the required function.

Miscellaneous Register – MISC

The miscellaneous register is used to control two functions. The four lower bits are used for the Watchdog Timer control, while the highest four bits are used to select open drain outputs for pins PA0~PA3.

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 selections for all ports and wake-up selections on certain pins, the user is provided with an I/O structure to meet the needs of a wide range of application possibilities.

The device provides 24 bidirectional input/output lines labeled with port names PA, PB and PD. 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 selected using registers PAPU, PBPU and PDPU and are implemented using weak PMOS transistors.

Port A Wake-up

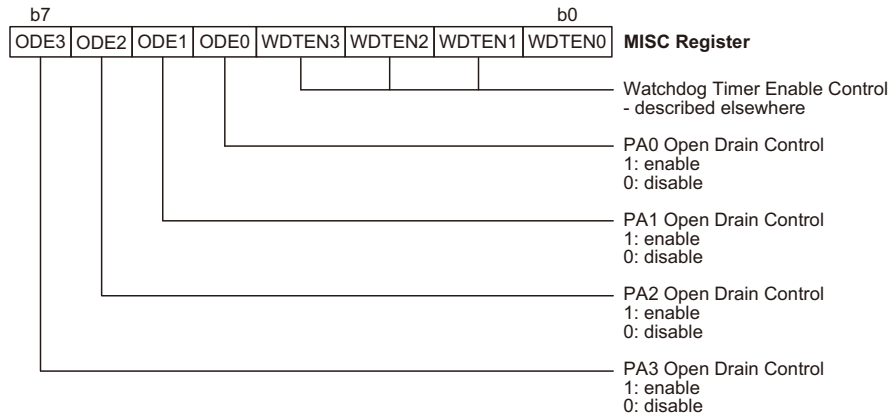
The HALT instruction forces the microcontroller into a Power Down condition which preserves power, a feature that is important for battery and other low-power applications. Various methods exist to wake-up the microcontroller, one of which is to change the logic condition on one of the Port A pins from high to low. After a HALT instruction forces the microcontroller into entering a Power Down condition, the processor will remain in a low-power state until the logic condition of the selected wake-up pin on Port A changes from high to low. This function is especially suitable for applications that can be woken up via external switches. Each pin on Port A can be selected individually to have this wake-up feature using the PAWU register.

Port A Open Drain Function

All I/O pins in the device have CMOS structures, however Port A pins PA0~PA3 can also be setup as open drain structures. This is implemented using the ODE0~ODE3 bits in the MISC register.

I/O Port Control Registers

Each I/O port has its own control register known as PAC, PBC, and PDC, to control the input/output configuration. With this control register, each CMOS output or input with or without pull-high resistor structures can be re-configured 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 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.



PA0~PA3 Open Drain Control – MISC

Pin-shared Functions

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

- External Interrupt Inputs
The external interrupt pins INT0, INT1 are pin-shared with the I/O pins PD4, PD5. For applications not requiring an external interrupt input, the pin-shared external interrupt pin can be used as a normal I/O pin, however to do this, the external interrupt enable bits in the INTC0 register must be disabled.
- External Timer Clock Input
The external timer pins TMR0, TMR1 are pin-shared with the I/O pin PD6, PD7. To configure it to operate as a timer input, the corresponding control bits in the timer control register must be correctly set and the pin must also be setup as an input. Note that the original I/O function will remain even if the pin is setup to be used as an external timer input.
- PFD Output
The device contains a 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 pin will function as a normal logic input with the usual pull-high selection, even if the PFD configuration option has been selected.
- PWM Outputs
The device contains four PWM outputs shared with pin PD0~PD3. The PWM output functions are chosen via registers. Note that the corresponding bit 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 selection, even if the PWM registers have been selected.

- A/D Inputs
The device has eight 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 register remain, however if used as A/D inputs then any pull-high resistor selections associated with these pins will be automatically disconnected.

I/O Pin Structures

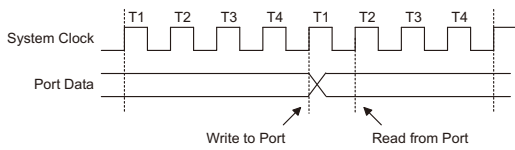
The accompanying diagrams illustrate the I/O pin internal structures. As the exact logical construction of the I/O pin may differ from these drawings, they are supplied as a guide only to assist with the functional understanding of the I/O pins.

Programming Considerations

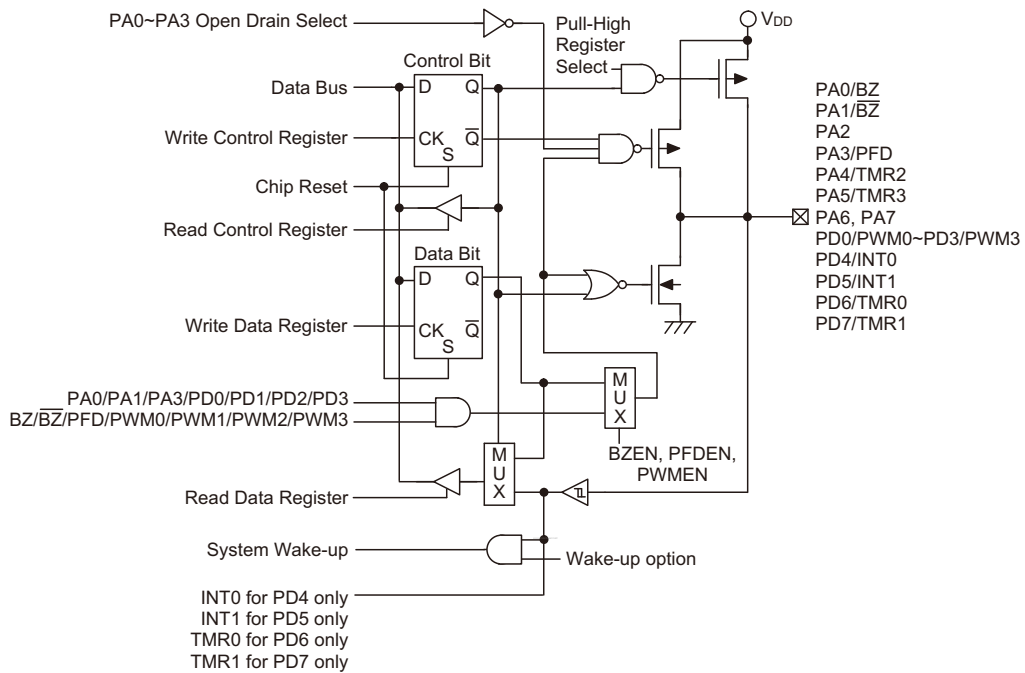
Within the user program, one of the first things to consider is port initialisation. After a reset, all of the I/O data and port control registers will be set high. This means that all I/O pins will default to an input state, the level of which depends on the other connected circuitry and whether pull-high selections have been chosen. If the port control registers, PAC, PBC and PDC, 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 and PD, 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 re-write this data back to the output ports.

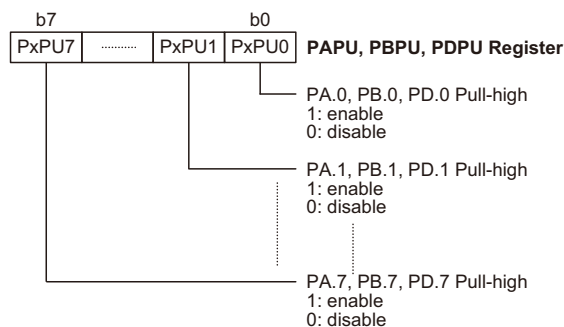
Port A has the additional capability of providing wake-up functions. When the device is in the Power Down Mode, various methods are available to wake the device up. One of these is a high to low transition of any of the Port A pins. Single or multiple pins on Port A can be setup to have this function.



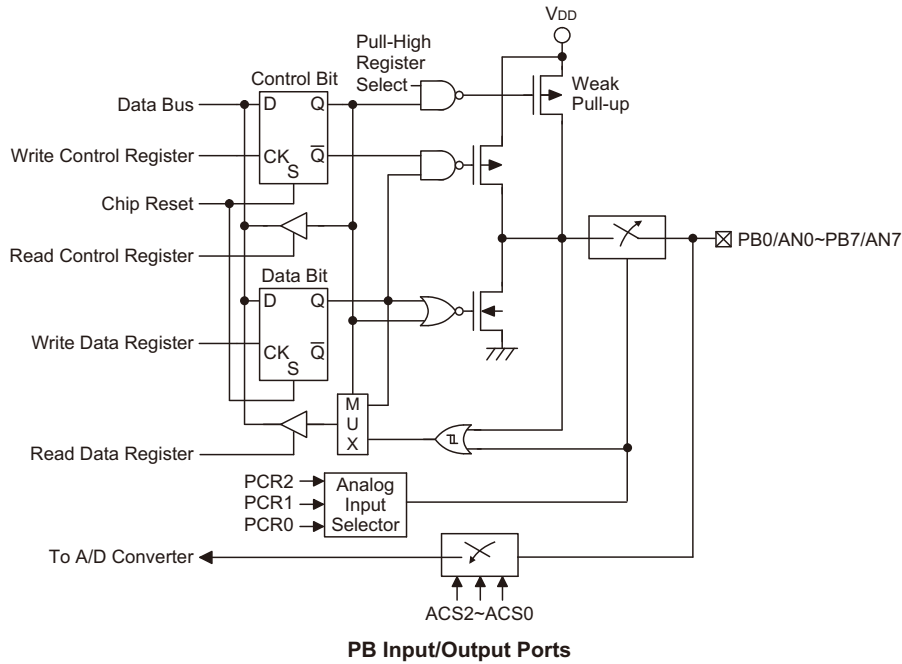
Read/Write Timing



PA, PD Input/Output Ports



Pull-High Resistor Register – PAPU, PBPU, PDPU



Liquid Crystal Display (LCD) Driver

For large volume applications, which incorporate an LCD in their design, the use of a custom display rather than a more expensive character based display reduces costs significantly. However, the corresponding signals required, which vary in both amplitude and time, to drive such a custom display require many special considerations for proper LCD operation to occur. The Holtek LCD Driver function, with its internal LCD signal generating circuitry and various options, will automatically generate these time and amplitude varying signals to provide a means of direct driving and easy interfacing to a range of custom LCDs.

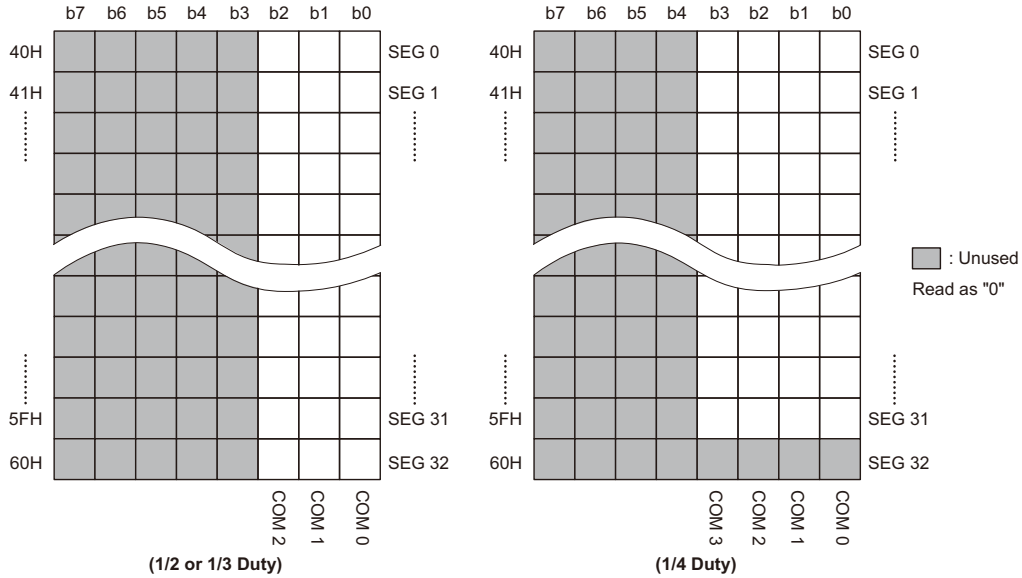
LCD Memory

An area of Data Memory is especially reserved for use by the LCD data. This data area is known as the LCD Memory. Any data written here will be automatically read by the internal LCD driver circuits, which will in turn automatically generate the necessary LCD driving signals. Therefore any data written into the LCD Memory will be immediately reflected into the actual LCD display connected to the microcontroller. The start address of the LCD Memory is 40H; the end address of the LCD Memory is 60H.

As the LCD Data Memory addresses overlap those of the General Purpose Data Memory, the LCD Data Memory is stored in its own memory data bank, which is different from that of the General Purpose Data Memory.

The LCD Data Memory is stored in Bank 1. The Data Memory Bank is chosen by using the Bank Pointer, which is a special function register in the Data Memory, with the name, BP. When the lowest bit of the Bank Pointer has the binary value "0", only the General Purpose Data Memory will be accessed, no read or write actions to the LCD Memory will take place. To access the LCD Memory therefore requires first that Bank 1 is selected by setting the lowest bit of the Bank Pointer to the binary value "1". After this, the LCD Memory can then be accessed by using indirect addressing through the use of Memory Pointer MP1. With Bank 1 selected, then using MP1 to read or write to the memory area, 40H-60H, will result in operations to the LCD Memory. Directly addressing the LCD Memory is not applicable and will result in a data access to the Bank 0 General Purpose Data Memory.

The diagrams below are based on 33x2, 33x3 or 32x4 format pixel drive capability LCD panels. The 4-COM format will be automatically setup when the 1/4 duty control bit is selected while the 3-COM format will be automatically setup if the 1/3 duty control bit is selected.

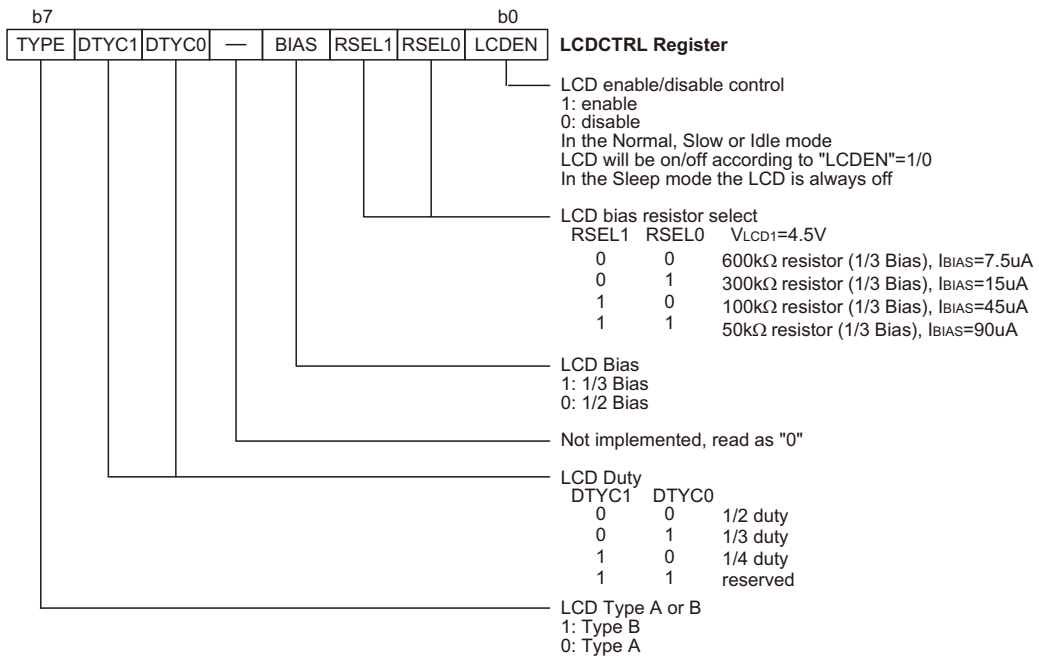
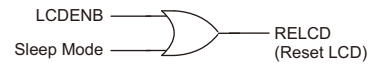


LCD Memory Map

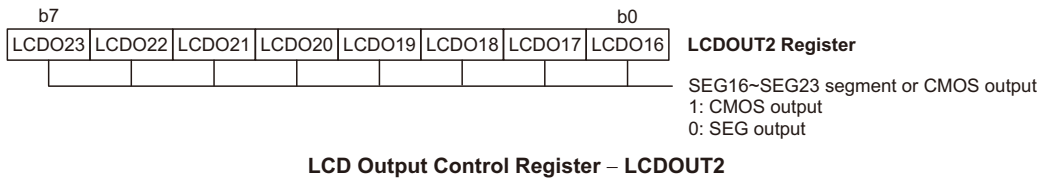
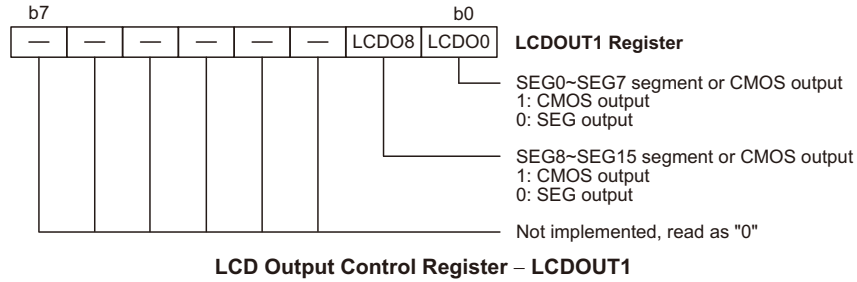
LCD Registers

A single LCD Control Register in the Data Memory, known as LCDCTRL, is used to control the various setup features of the LCD Driver. Various bits in this register control functions such as duty type, bias type, bias resistor selection as well as overall LCD enable and disable. The LCDEN bit in the LCDCTRL register will only be effective when the device is in the Normal, Slow or Idle Mode. If the device is in the Sleep Mode then the

LCD will always be disabled. Bits RSEL0 and RSEL1 select the internal bias resistors to supply the LCD panel with the correct bias voltages. A choice to best match the LCD panel used in the application can be selected also to minimise bias current. The TYPE bit is used to select whether Type A or Type B LCD control signals are used.



LCD Control Register – LCDCTRL



Two registers, LCDOUT1 and LCDOUT2 are used to determine if the output function of LCD pins SEG0~SEG23 are used as LCD segment drivers or CMOS outputs. If used as CMOS outputs then the LCD Data Memory is used to determine the logic level of the CMOS output pins. Note that as only two bits are used to determine the output function of the SEG0~SEG7 and SEG8~SEG15 pins, individual pins from these two groups of pins cannot be chosen to have either a segment or CMOS output function. The output function of pins SEG16~SEG23 can be chosen individually to be either an LCD segment driver or a CMOS input.

LDC Reset Function

The LCD has an internal reset function that is an OR function of the inverted LCDEN bit in the LCDCTRL register and the Sleep function. The LCD reset signal is active high. The LCDENB signal is the inverse of the LCDEN bit in the LCDCTRL register.

RELCD= (Sleep and IDLEN=0) or LCDENB.

LEDSEL=0 & LCDEN=1 must be enabled to activate the LCDCTRL register function.

LCD Clock

The LCD clock source is the internal clock signal, f_{SUB} , divided by 8, using an internal divider circuit. The f_{SUB} internal clock is supplied by either the internal 32K_INT oscillator or the external RTC oscillator, the choice of which is determined by a configuration option. For proper LCD operation, this arrangement is provided to generate an ideal LCD clock source frequency of 4kHz.

| f_{SUB} Clock Source | LCD Clock Frequency |
|------------------------|---------------------|
| Internal 32K_INT Osc. | 4KHz |
| External RTC Osc. | 4KHz |

LCD Clock Source

LCD Driver Output

The number of COM and SEG outputs supplied by the LCD driver, as well as its biasing and duty selections, are dependent upon the LCD control bits selected. The accompanying table lists the various selections. The Bias Type, whether C or R type is selected using a configuration option.

| Duty | Driver Number | Bias | Bias Type | Waveform Type |
|------|---------------|------------|-------------|---------------|
| 1/2 | 33×2 | 1/2 or 1/3 | C or R type | A or B |
| 1/3 | 33×3 | | | |
| 1/4 | 32×4 | | | |

LCD Selections

If the C-type of bias is used then an internal charge pump will be enabled. This charge pump has two voltage multiplier options selected using a configuration option. Note that the C-type bias is not available on the 52-pin QFP package type.

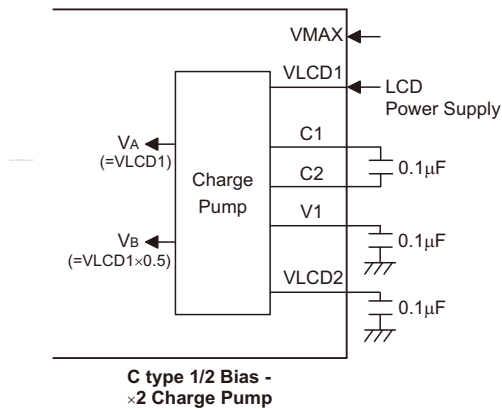
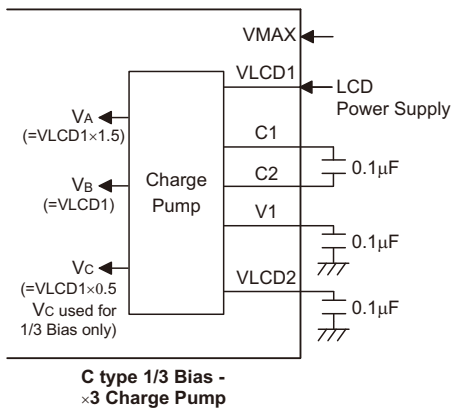
The nature of Liquid Crystal Displays require that only AC voltages can be applied to their pixels as the application of DC voltages to LCD pixels may cause permanent damage. For this reason the relative contrast of an LCD display is controlled by the actual RMS voltage applied to each pixel, which is equal to the RMS value of the voltage on the COM pin minus the voltage applied to the SEG pin. This differential RMS voltage must be greater than the LCD saturation voltage for the pixel to be on and less than the threshold voltage for the pixel to be off. The requirement to limit the DC voltage to zero and to control as many pixels as possible with a minimum number of connections, requires that both a time and amplitude signal is generated and applied to the application LCD. These time and amplitude varying signals are automatically generated by the LCD driver circuits in the

microcontroller. What is known as the duty determines the number of common lines used, which are also known as backplanes or COMs. The duty, which is chosen by a control bit to have a value of 1/2, 1/3 or 1/4 and which equates to a COM number of 2, 3 and 4 respectively, therefore defines the number of time divisions within each LCD signal frame. Two types of signal generation are also provided, known as Type A and Type B, the required type is selected via the TYPE bit in the LCDCTRL register. Type B offers lower frequency signals, however lower frequencies may introduce flickering and influence display clarity. The accompanying timing diagrams depict the LCD signals generated by the microcontroller for various values of duty and bias.

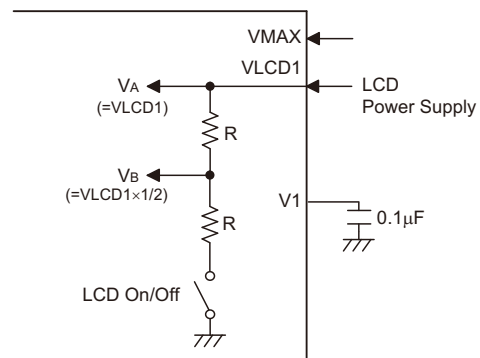
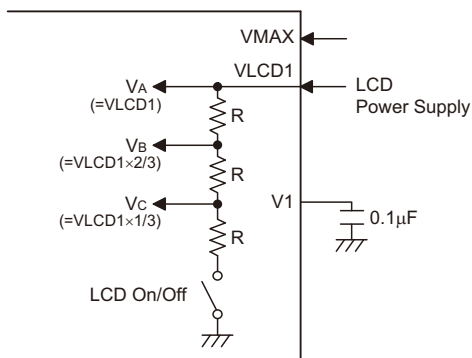
LCD Voltage Source and Biasing

The time and amplitude varying signals generated by the LCD Driver function require the generation of several voltage levels for their operation. The number of voltage levels used by the signal depends upon the value of the BIAS bit in the LCDCTRL register. The device can have either R type or C type biasing selected via a configuration option. Selecting the C type biasing will enable an internal charge pump whose multiplying ratio can be selected using an additional configuration option.

For R type biasing an external LCD voltage source must be supplied on pin VLCD1 to generate the internal biasing voltages. This could be the microcontroller power supply or some other voltage source. For the R type 1/2 bias selection, three voltage levels VSS, VA and VB are utilised. The voltage VA is equal to the externally supplied voltage source applied to pin VLCD1. VB is generated internally by the microcontroller and will have a value equal to VLCD1/2. For the R type 1/3 bias selection, four voltage levels VSS, VA, VB and VC are utilised. The voltage VA is equal to VLCD1, VB is equal to VLCD1×2/3 while VC is equal to VLCD1×1/3. In addition to selecting 1/2 or 1/3 bias, several values of bias resistor can be chosen using bits in the LCDCTRL register. Different values of internal bias resistors can be selected using the RSEL0 and RESEL1 bits in the LCDCTRL register. This along with the voltage on pin VLCD1 will determine the bias current. The connection to the VMAX pin depends upon the voltage that is applied to VLCD1. If the VDD voltage is greater than the voltage applied to the VLCD1 pin then the VMAX pin should be connected to VDD, otherwise the VMAX pin should be connected to pin VLCD1. Note that no external capacitors or resistors are required to be connected if R type biasing is used.



C Type Bias Voltage Levels



R Type Bias Voltage Levels

| Condition | VMAX connection |
|-------------|-----------------------|
| VDD > VLCD1 | Connect VMAX to VDD |
| Otherwise | Connect VMAX to VLCD1 |

R Type Bias Current VMAX Connection

For C type biasing an external LCD voltage source must also be supplied on pin VLCD1 to generate the internal biasing voltages. The C type biasing scheme uses an internal charge pump circuit, which in the case of the 1/3 bias selection can generate voltages higher than what is supplied on VLCD1. This feature is useful in applications where the microcontroller supply voltage is less than the supply voltage required by the LCD. The external LCD power supply should be connected to pin VLCD1 and a filter capacitor connected to pin VLCD2. An additional charge pump capacitor must also be connected between pins C1 and C2 to generate the necessary voltage levels.

For the C type 1/2 bias selection, three voltage levels VSS, VA and VB are utilised. The voltage VA is generated internally and has a value of VLCD1. VB will have a value equal to VA×0.5. For the C type 1/2 bias configuration VC is not used.

For the C type 1/3 bias selection, four voltage levels VSS, VA, VB and VC are utilised. The voltage VA is generated internally and has a value of VLCD1×1.5. VB will have a value equal to VA × 2/3 and VC will have a value equal to VA × 1/3. The connection to the VMAX pin depends upon the bias and the voltage that is applied to VLCD1, the details are shown in the table. Note that C type biasing is not available on the 52-pin QFP package

device types. On these package types, pins C1, C2 and VLCD2 are not provided. It is recommended that a 0.1µF capacitor is connected between the V1 pin and ground on the 52-pin QFP package types.

It is extremely important to ensure that these charge pump generated internal voltages do not exceed the maximum VDD voltage of 5.5V. Note that the C-type bias type is not available on the 52-pin QFP package type.

| Biasing Type | | VMAX Connection |
|--------------|---------------|-----------------------|
| 1/3 Bias | VDD>VLCD1×1.5 | Connect VMAX to VDD |
| | Otherwise | Connect VMAX to V1 |
| 1/2 Bias | VDD>VLCD1 | Connect VMAX to VDD |
| | Otherwise | Connect VMAX to VLCD1 |

C Type Biasing VMAX Connection

Programming Considerations

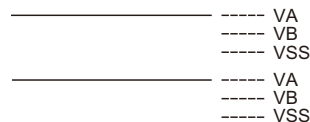
Certain precautions must be taken when programming the LCD. One of these is to ensure that the LCD memory is properly initialised after the microcontroller is powered on. Like the General Purpose Data Memory, the contents of the LCD memory are in an unknown condition after power-on. As the contents of the LCD memory will be mapped into the actual LCD, it is important to initialise this memory area into a known condition soon after applying power to obtain a proper display pattern.

Consideration must also be given to the capacitive load of the actual LCD used in the application. As the load presented to the microcontroller by LCD pixels can be generally modeled as mainly capacitive in nature, it is

During Reset or in HALT Mode

COM0, COM1

All segment outputs



Normal Operation Mode

COM0

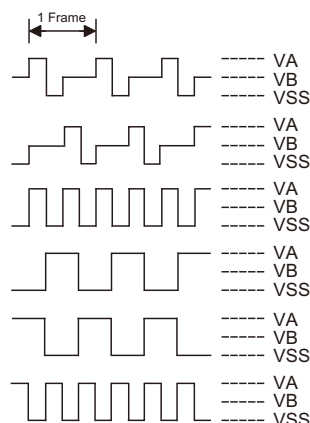
COM1

All segments OFF

COM0 segments ON

COM1 segments ON

All segments ON

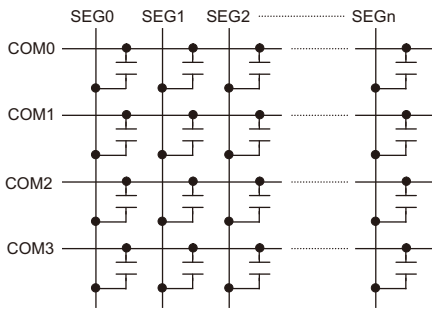


LCD Driver Output – Type A - 1/2 Duty, 1/2 Bias

Note For 1/2 Bias, VA=VLCD1, VB=VLCD1×1/2 for both R and C type.

important that this is not excessive, a point that is particularly true in the case of the COM lines which may be connected to many LCD pixels. The accompanying diagram depicts the equivalent circuit of the LCD.

With such a frequency chosen, the microcontroller internal LCD driver circuits will ensure that the appropriate LCD driving signals are generated to obtain a suitable LCD frame frequency.



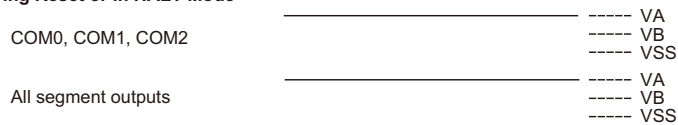
LCD Panel Equivalent Circuit

One additional consideration that must be taken into account is what happens when the microcontroller enters a HALT condition. The "LCDEN" control bit in the LCDCTRL register permits the LCD to be powered off to reduce power consumption. If "LCDEN"=0 is selected, the driving signals to the LCD will cease, producing a blank display pattern but reducing any power consumption associated with the LCD.

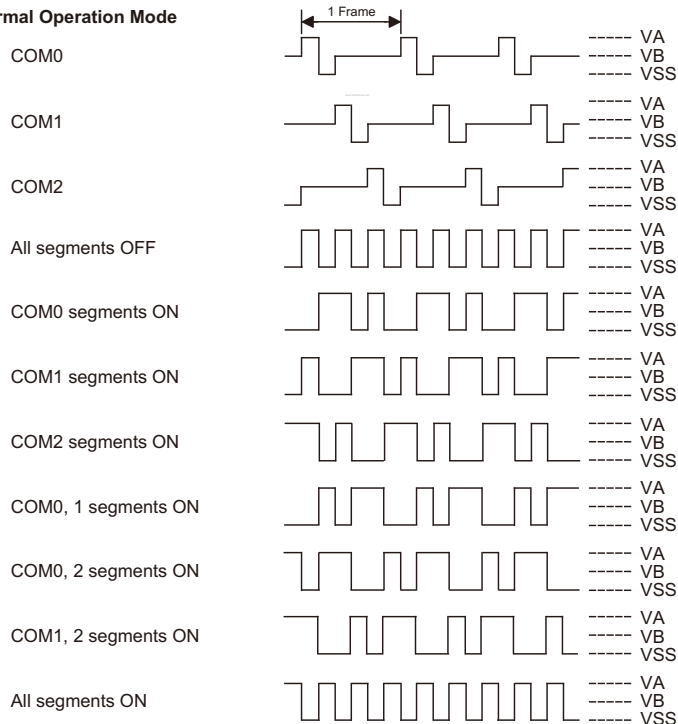
After Power-on, note that as the LCDEN bit in the LCDCTRL register will be cleared to zero, the LCD function will be disabled.

The following timing diagrams depict the LCD signals generated by the microcontroller for various values of duty and bias.

During Reset or in HALT Mode

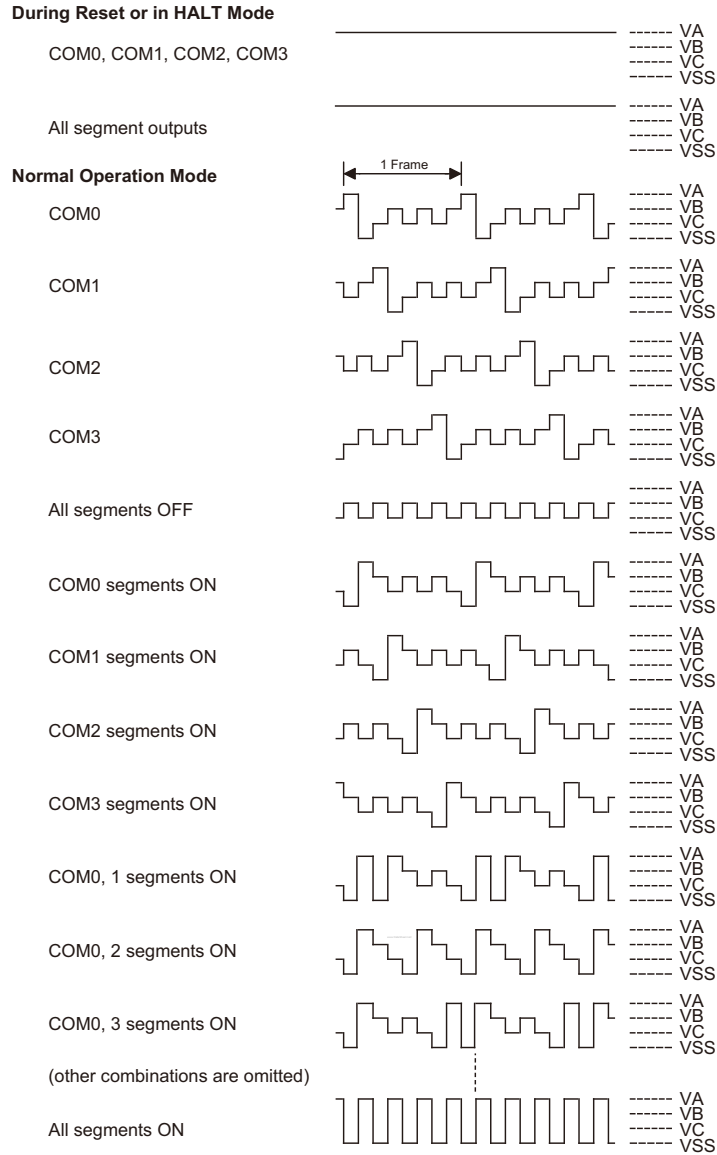


Normal Operation Mode



LCD Driver Output – Type A- 1/3 Duty, 1/2 Bias

Note: For 1/2 Bias, the VA=VLCD1, VB=VLCD1×1/2 for both R and C type.



LCD Driver Output – Type A - 1/4 Duty, 1/3 Bias

Note: For 1/3 R type bias, the $VA=VLCD1$, $VB=VLCD1 \times 2/3$ and $VC=VLCD1 \times 1/3$.
For 1/3 C type bias, the $VA=VLCD1 \times 1.5$, $VB=VLCD1$ and $VC=VLCD1 \times 1/2$.

During Reset or in HALT Mode

COM0, COM1, COM2

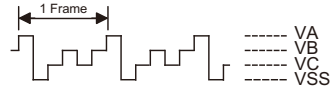


All segment outputs



Normal Operation Mode

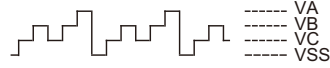
COM0



COM1



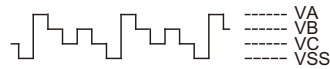
COM2



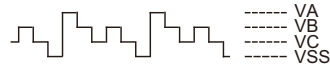
All segments OFF



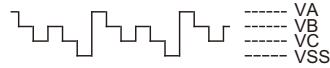
COM0 segments ON



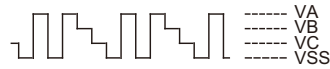
COM1 segments ON



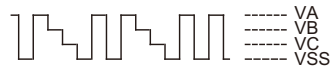
COM2 segments ON



COM0, 1 segments ON



COM0, 2 segments ON



COM1, 2 segments ON

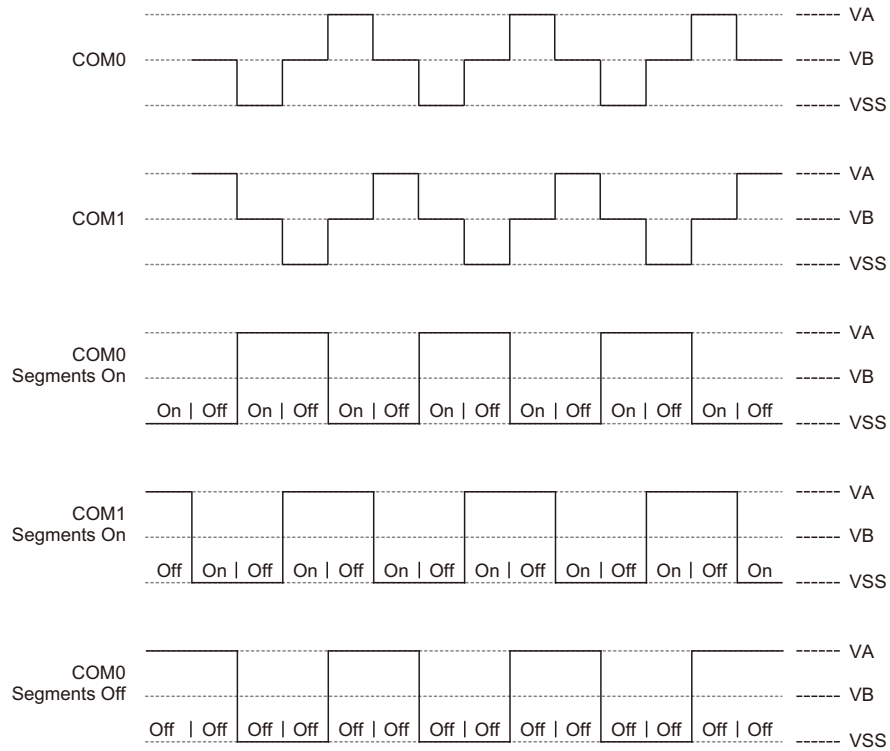


All segments ON



LCD Driver Output – Type A - 1/3 Duty, 1/3 Bias

Note: For 1/3 R type bias, the $VA=VLCD1$, $VB=VLCD1 \times 2/3$ and $VC=VLCD1 \times 1/3$.
For 1/3 C type bias, the $VA=VLCD1 \times 1.5$, $VB=VLCD1$ and $VC=VLCD1 \times 1/2$.



LCD Driver Output – Type B – 1/2 Duty, 1/2 Bias

Note: For 1/2 bias, the $VA=VLCD$, $VB=VLCD \times 1/2$ for both R and C type.

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 devices contain one 8-bit and one 16-bit count-up timer. As each timer has three different operating modes, they can be configured to operate as a general timer, an external event counter or as a pulse width measurement device. The provision of a prescaler to the clock circuitry of the 8-bit Timer/Event Counter also gives added range to this timer.

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

Configuring the Timer/Event Counter Input Clock Source

The internal timer's clock can originate from various sources. The system clock source is used when the Timer/Event Counter is in the timer mode or in the pulse width measurement mode. For Timer/Event Counter 0 this internal clock source is f_{SYS} which is also divided by a prescaler, the division ratio of which is conditioned by the Timer Control Register, TMR0C, bits T0PSC0~T0PSC2. For Timer/Event Counter 1 this internal clock source can be chosen from a combination of internal clocks using a configuration option and the T1S bit in the TMR1C register.

An external clock source is used when the timer is in the event counting mode, the clock source being provided on an external timer pin TMR0 or TMR1, depending upon which timer is used. Depending upon the condition

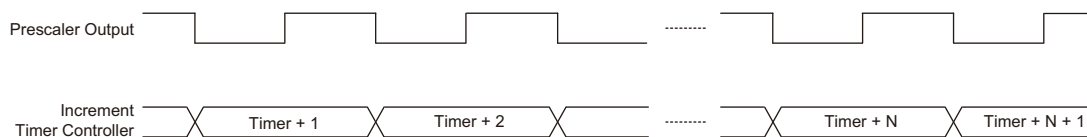
of the T0E or T1E bit, each high to low, or low to high transition on the external timer pin will increment the counter by one.

Timer Registers – TMR0, TMR1L, TMR1H

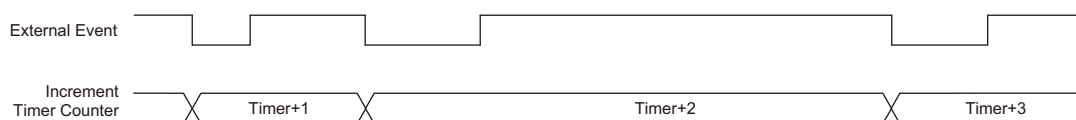
The timer registers are special function registers located in the Special Purpose Data Memory and is the place where the actual timer value is stored. For the 8-bit Timer/Event Counter 0, this register is known as TMR0. For 16-bit Timer/Event Counter 1, the timer registers are known as TMR1L and TMR1H. 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 of FFH for the 8-bit timer or FFFFH for the 16-bit timer at which point the timer overflows and an internal interrupt signal is generated. The timer value will then be reset with the initial preload register value and continue counting.

To achieve a maximum full range count of FFH for the 8-bit timer or FFFFH for the 16-bit timer, 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 switched off and data is written to its preload registers, this data will be immediately written into the actual timer registers. However, if the Timer/Event Counter is enabled and counting, any new data written into the preload data registers during this period will remain in the preload registers and will only be written into the timer registers the next time an overflow occurs.

For the 16-bit Timer/Event Counter which has 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 register, namely 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



Timer Mode Timing Chart



Event Counter Mode Timing Chart

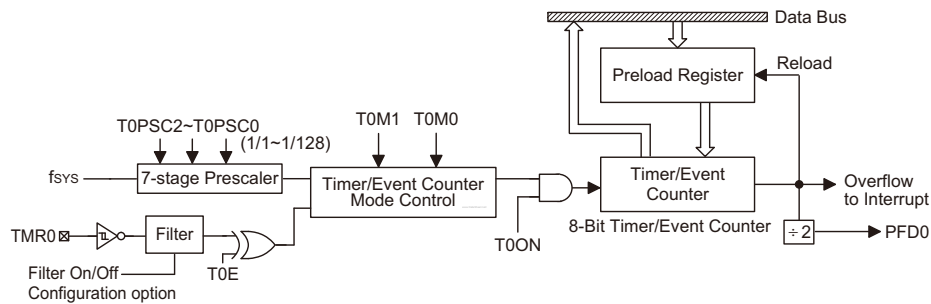
the low byte timer register is only carried out when a write to its associated high byte timer register, namely TMR1H, is executed. On the other hand, using instructions to preload data into the high byte timer register will result in the data being directly written to the high byte timer register. At the same time the data in the low byte timer register 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 Registers – TMR0C, TMR1C

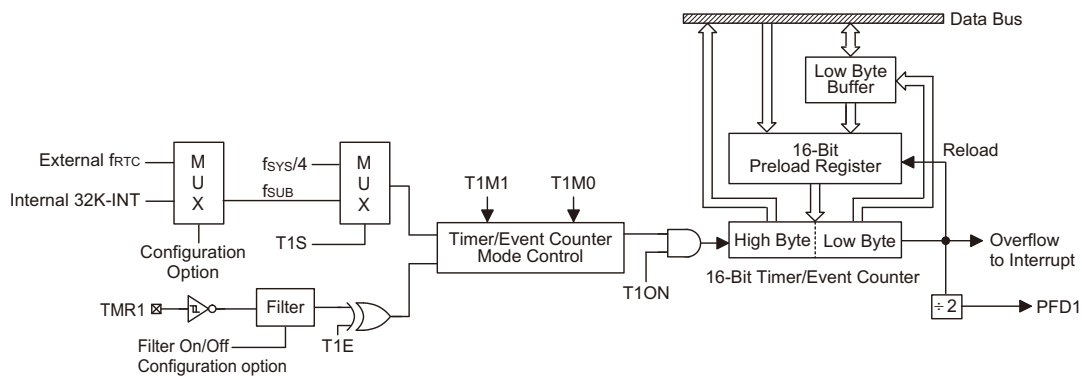
The flexible features of the Holtek microcontroller Timer/Event Counters enable them to operate in three different modes, the options of which are determined by the contents of their respective control register.

It is the Timer Control Register together with its corresponding timer registers that control the full operation of the Timer/Event Counters. Before the timers can be used, it is essential that the appropriate 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.

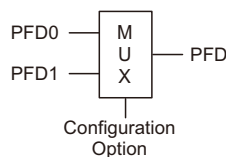
To choose which of the three modes the timer is to operate in, either in the timer mode, the event counting mode or the pulse width measurement mode, bits 7 and 6 of the Timer Control Register, which are known as the bit pair TOM1/TOM0 or T1M1/T1M0 respectively, depending upon which timer is used, must be set to the required logic levels. The timer-on bit, which is bit 4 of the Timer Control Register and known as T0ON or T1ON, depending upon which timer is used, provides the basic on/off control of the respective timer. Setting the bit high allows the counter to run, clearing the bit stops the counter. For timers that have prescalers, bits 0~2 of the Timer Control Register determine the division ratio of the input clock prescaler. The prescaler bit settings have no effect if an external clock source is used. If the timer is in the event count or pulse width measurement mode, the active transition edge level type is selected by the logic

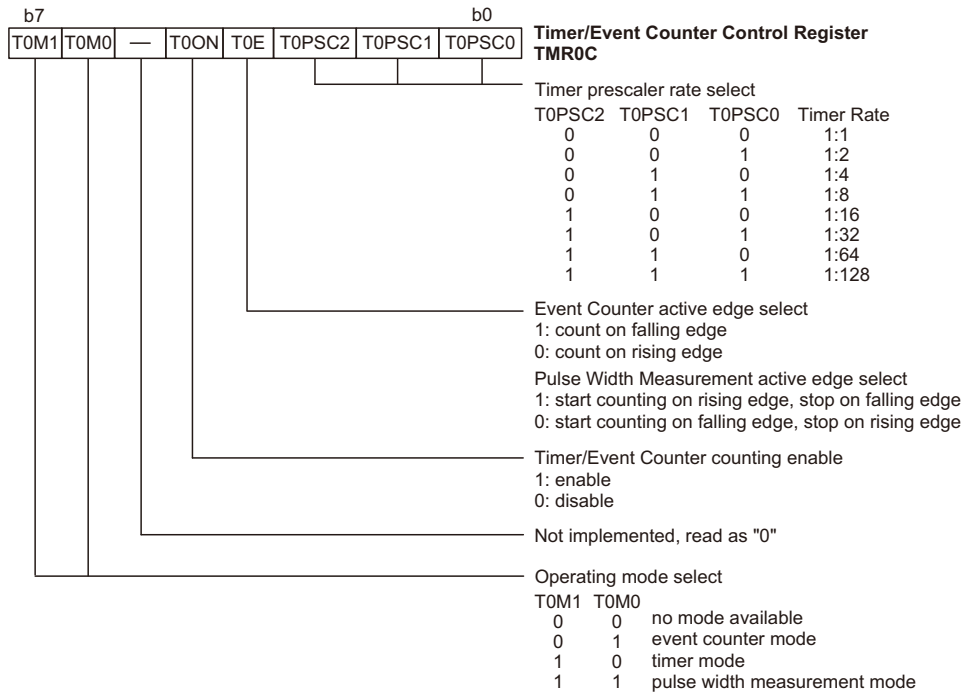


Timer/Event Counter 0 Structure

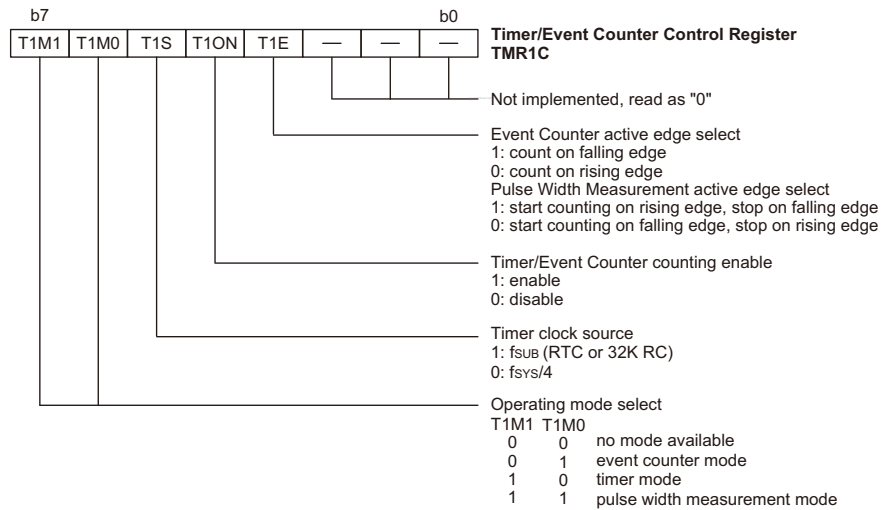


Timer/Event Counter 1 Structure





Timer/Event Counter Control Register TMR0C



Timer/Event Counter 1 Control Register TMR1C

level of bit 3 of the Timer Control Register which is known as T0E or T1E depending upon which timer is used. An additional T1S bit in the TMR1C register is used to determine the clock source for Timer/Event Counter 1.

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 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 Timer Mode

| Bit7 | Bit6 |
|------|------|
| 1 | 0 |

In this mode the internal clock, f_{SYS} , is used as the internal clock for 8-bit Timer/Event Counter 0 and f_{SUB} or $f_{SYS}/4$ is used as the internal clock for 16-bit Timer/Event Counter 1. However, the clock source, f_{SYS} , for 8-bit timer is further divided by a prescaler, the value of which is determined by the Prescaler Rate Select bits T0PSC2~T0PSC0, which are bits 2~0 in the Timer Control Register. After the other bits in the Timer Control Register have been setup, the enable bit T0ON or T1ON, 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, INTC0, is reset to zero.

Configuring the Event Counter Mode

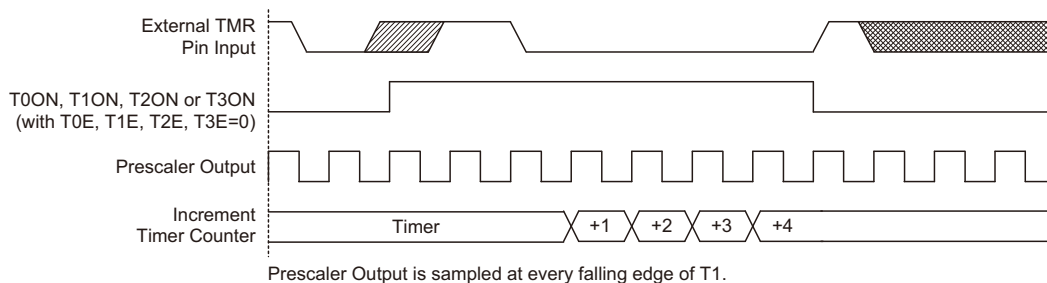
In this mode, a number of externally changing logic events, occurring on the external timer pin, can be recorded by the Timer/Event Counter. To operate in this mode, the Operating Mode Select bit pair, T0M1/T0M0 or T1M1/T1M0, in the Timer Control Register must be set to the correct value as shown.

Control Register Operating Mode
Select Bits for the Event Counter Mode

| Bit7 | Bit6 |
|------|------|
| 0 | 1 |

In this mode, the external timer pin, TMR0 or TMR1, 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 or T1ON, 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 or T1E, 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, INTC0, is reset to zero.

As the external timer pin is shared with an I/O pin, to ensure that the pin is configured to operate as an event counter input pin, two things have to happen. The first is to ensure that the Operating Mode Select bits in the Timer Control Register place the Timer/Event Counter in the Event Counting Mode, the second is to ensure that the port control register configures the pin as an input. It should be noted that in the event counting mode, even if the microcontroller is in the 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.



Pulse Width Measure Mode Timing Chart

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, TOM1/TOM0 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} , is used as the internal clock for 8-bit Timer/Event Counter 0 and f_{SUB} or $f_{SYS}/4$ is used as the internal clock for 16-bit Timer/Event Counter 1. However, the clock source, f_{SYS} , for 8-bit timer is further divided by a prescaler, the value of which is determined by the Prescaler Rate Select bits T0PSC2~T0PSC0, which are bits 2~0 in the Timer Control Register. After the other bits in the Timer Control Register have been setup, the enable bit T0ON or T1ON, 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 or T1E, 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 or TMR1, 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, INTC, is reset to zero.

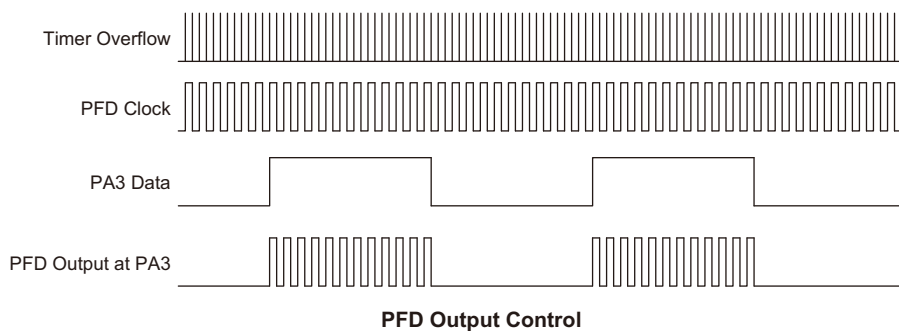
As the external timer pin is shared with an I/O pin, to ensure that the pin is configured to operate as a pulse width measurement pin, two things have to happen. The first is to ensure that the Operating Mode Select bits in the Timer Control Register place the Timer/Event Counter in the Pulse Width Measurement Mode, the second is to ensure that the port control register configures the pin as an input.

Programmable Frequency Divider – PFD

The Programmable Frequency Divider provides a means of producing a variable frequency output suitable for applications requiring a precise frequency generator.

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 clock source for the PFD circuit can originate from either the timer 0 or timer 1 overflow signal selected via configuration option. The output frequency is controlled by loading the required values into the timer registers and prescaler registers to give the required division ratio. The timer 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 timer 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".

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

Bits T0PSC0~T0PSC2 of the TMR0C register can be used to define the pre-scaling stages of the internal clock sources of the Timer/Event Counter 0. The Timer/Event Counter overflow signal can be used to generate signals for the PFD and Timer Interrupt.

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 pin for correct operation. As this pin is a shared pin it must be configured correctly to ensure it is setup for use as a Timer/Event Counter input and not as a normal I/O pin. 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 must be set high to ensure that the pin is setup as an input. Any pull-high resistor on this pin will remain valid even if the pin is used as a Timer/Event Counter input.

Timer/Event Counter Pins Internal Filter

The external Timer/Event Counter pins are connected to an internal filter to reduce the possibility of unwanted event counting events or inaccurate pulse width measurements due to adverse noise or spikes on the external Timer/Event Counter input signal. As this internal filter circuit will consume a limited amount of power, a configuration option is provided to switch off the filter function, an option which may be beneficial in power sensitive applications, but in which the integrity of the input signal is high. Care must be taken when using the filter on/off configuration option as it will be applied not only to both external Timer/Event Counter pins but also to the external interrupt input pins. Individual Timer/Event Counter or external interrupt pins cannot be selected to have a filter on/off function.

Programming Considerations

When configured to run in the timer mode, the internal system clock is used as the timer clock source and is therefore synchronised with the overall operation of the microcontroller. In this mode when the appropriate timer

register is full, the microcontroller will generate an internal interrupt signal directing the program flow to the respective internal interrupt vector. For the pulse width 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 synchronised with the internal system or timer clock.

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

Timer Program Example

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

```
org 04h          ; external interrupt vector
reti
org 08h          ; Timer/Event Counter 0 interrupt vector
jmp tmrint      ; jump here when the Timer/Event Counter 0 overflows
:
org 20h          ; main program
; internal Timer/Event Counter 0 interrupt routine
tmrint:
:
; Timer/Event Counter 0 main program placed here
:
reti
:
:

begin:
; setup Timer 0 registers
mov a,09bh      ; setup Timer 0 preload value
mov tmr0,a;
mov a,081h      ; setup Timer 0 control register
mov tmr0c,a     ; timer mode and prescaler set to /2
; setup interrupt register
mov a,009h      ; enable master interrupt and timer interrupt
mov int0c,a
set tmr0c.4     ; start Timer/Event Counter 0 - note mode bits must be previously setup
```

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 an output with a fixed frequency but with a duty cycle that can be varied by setting particular values into the corresponding PWM register.

| Channel | PWM Mode | Output Pin | Register Names |
|---------|----------|------------|----------------|
| 1 | 8+4 | PD0 | PWM0L PWM0H |
| 2 | 8+4 | PD1 | PWM1L PWM1H |
| 3 | 8+4 | PD2 | PWM2L PWM2H |
| 4 | 8+4 | PD3 | PWM3L PWM3H |

PWM Overview

Four register pairs, located in the Data Memory are assigned to the Pulse Width Modulator and are known as the PWM registers. It is in each register pair that the 12-bit value, which represents the overall duty cycle of one modulation cycle of the output waveform, should be placed. The PWM registers also contain the enable/disable control bit for the PWM outputs. To increase the PWM modulation frequency, each modulation cycle is modulated into sixteen individual modulation sub-sections, known as the 8+4 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. The PWM clock source is the system clock f_{SYS} .

This method of dividing the original modulation cycle into a further 16 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 should be understood. As the PWM clock is the system clock, f_{SYS} , and as the PWM value is 12-bits wide, the overall PWM cycle frequency is $f_{SYS}/4096$. However, when in the 8+4 mode of operation, the PWM modulation frequency will be $f_{SYS}/256$.

8+4 PWM Mode Modulation

Each full PWM cycle, as it is 12-bits wide, has 4096 clock periods. However, in the 8+4 PWM mode, each PWM cycle is subdivided into sixteen individual sub-cycles known

as modulation cycle 0 ~ modulation cycle 15, denoted as "i" in the table. Each one of these sixteen sub-cycles contains 256 clock cycles. In this mode, a modulation frequency increase of sixteen is achieved. The 12-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 bit4~bit11 is denoted here as the DC value. The second group which consists of bit0~bit3 is known as the AC value. In the 8+4 PWM mode, the duty cycle value of each of the two modulation sub-cycles is shown in the following table.

| Parameter | AC (0~15) | DC (Duty Cycle) |
|--------------------------------|-------------|--------------------|
| Modulation cycle i (i=0~15) | $i < AC$ | $\frac{DC+1}{256}$ |
| | $i \geq AC$ | $\frac{DC}{256}$ |

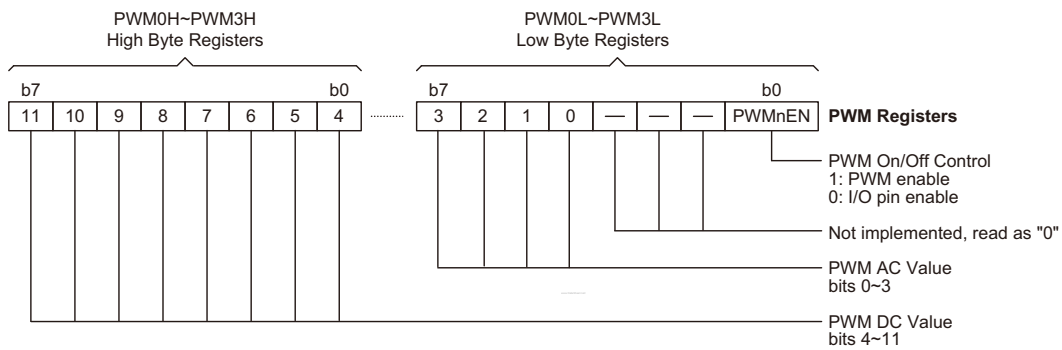
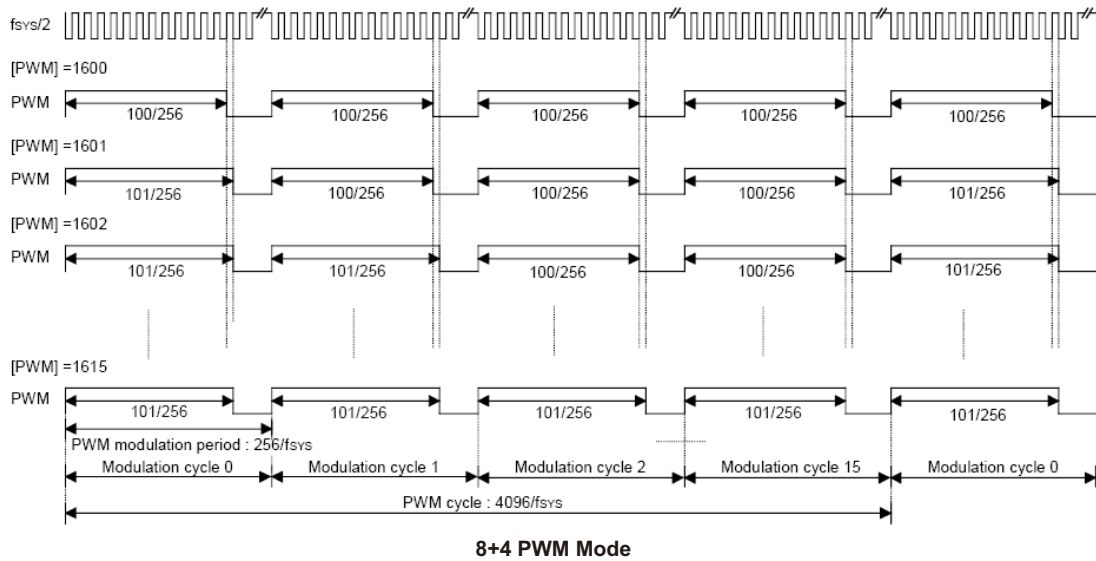
8+4 Mode Modulation Cycle Values

The accompanying diagram illustrates the waveforms associated with the 8+4 mode of PWM operation. It is important to note how the single PWM cycle is subdivided into 16 individual modulation cycles, numbered 0~15 and how the AC value is related to the PWM value.

PWM Output Control

| PWM Modulation Frequency | PWM Cycle Frequency | PWM Cycle Duty |
|--------------------------|---------------------|---------------------------|
| $f_{SYS}/256$ | $f_{SYS}/4096$ | (PWM register value)/4096 |

The four PWM0~PWM3 outputs are shared with pins PD0~PD3. To operate as a PWM output and not as an I/O pin, bit 0 of the relevant PWM register bit must be set high. A zero must also be written to the corresponding bit in the PDC port control register, to ensure that the PWM0 output pin is setup as an output. After these two initial steps have been carried out, and of course after the required PWM 12-bit value has been written into the PWM register pair register, writing a "1" to the corresponding PD data register will enable the PWM data to appear on the pin. Writing a "0" to the bit will disable the PWM output function and force the output low. In this way, the Port D data output register bits, can also be used as an on/off control for the PWM function. Note that if the enable bit in the PWM register is set high to enable 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 selections.



PWM Programming Example

The following sample program shows how the PWM output is setup and controlled.

```

mov a, 64h          ; setup PWM0 value to 1600 decimal which is 640H
mov pwm0h, a       ; setup PWM0H register value
clr pwm0l          ; setup PWM0L register value
clr pdc.0          ; setup pin PD0 as an output
set pwm0en         ; set the PWM0 enable bit
set pd.0           ; Enable the PWM0 output
: :
: :
clr pd.0           ; PWM0 output disabled - 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

The device contains an 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 either a 12-bit digital value.

| Input Channels | Conversion Bits | Input Pins |
|----------------|-----------------|------------|
| 8 | 12 | PB0~PB7 |

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

A/D Converter Data Registers – ADRL, ADRH

The device, which has an internal 12-bit A/D converter, requires two data registers, a high byte register, known as ADRH, and a low byte register, known as ADRL. After the conversion process takes place, these registers can be directly read by the microcontroller to obtain the digitised conversion value. 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 bits of the 12-bit converted value.

In the following table, D0~D11 is the A/D conversion data result bits.

| Register | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|----------|-------|-------|-------|-------|-------|-------|-------|-------|
| ADRL | D3 | D2 | D1 | D0 | — | — | — | — |
| ADRH | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 |

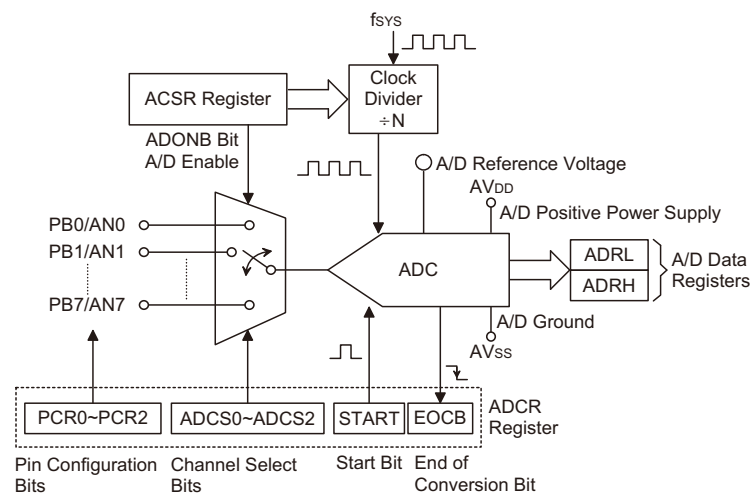
A/D Data Registers

A/D Converter Control Registers – ADCR, ACSR

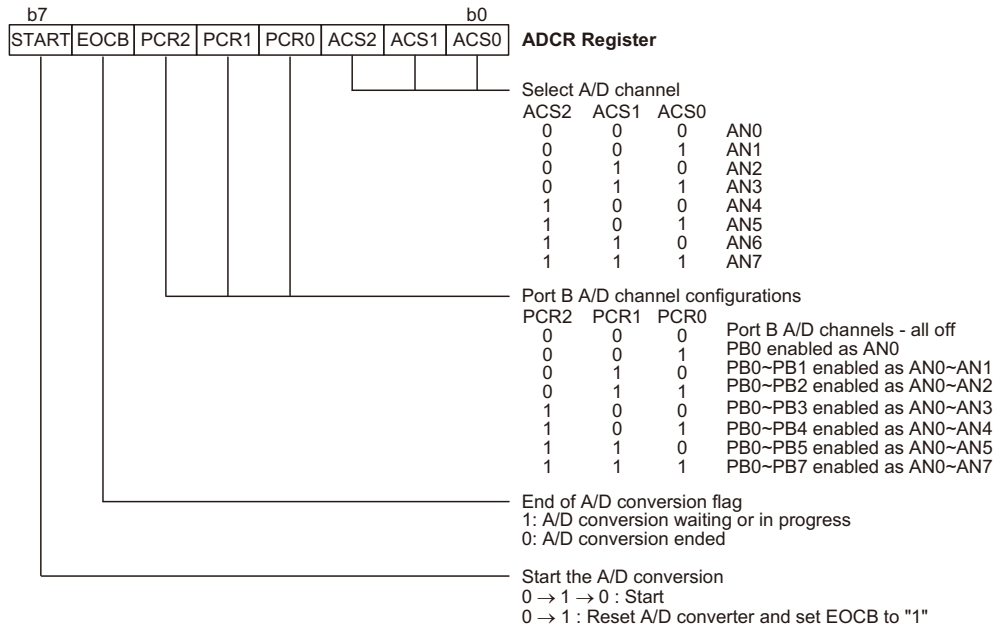
To control the function and operation of the A/D converter, two control registers known as ADCR and ACSR are provided. These 8-bit registers define functions such as the selection of which analog channel is connected to the internal A/D converter, which pins are used as analog inputs and which are used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

The ACS2~ACS0 bits in the ADCR register define the channel number. As the device contains only one actual analog to digital converter circuit, each of the individual 8 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.

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. If the 3-bit address on PCR2~PCR0 has a value of "111", then all eight pins, namely AN0~AN7 will all be set as analog inputs. Note that if 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.



A/D Converter Structure



A/D Converter Control Register – ADCR

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.

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 ADCS2, ADCS1 and ADCS0 bits in the ACSR register.

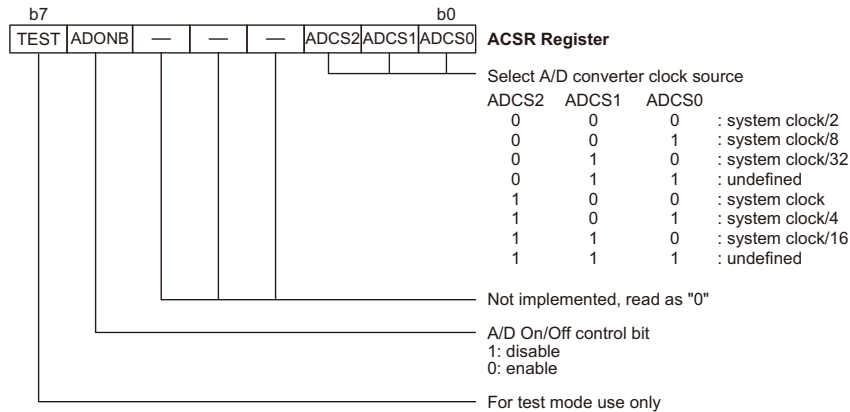
Controlling the on/off function of the A/D converter circuitry is implemented using the ADONB bit in the ACSR register and the value of the PCR bits in the ADCR register. Both the ADONB bit must be cleared to "0" and the value of the PCR bits must have a non-zero value for the A/D converter to be enabled.

| PCR | ADONB | A/D |
|-----|-------|-----|
| 0 | x | Off |
| > 0 | 0 | On |
| > 0 | 1 | Off |

Although the A/D clock source is determined by the system clock f_{SYS} , and by bits ADCS2, ADCS1 and ADCS0, there are some limitations on the maximum A/D clock source speed that can be selected. As the minimum value of permissible A/D clock period, t_{AD} , is 0.5 μ s, care must be taken for system clock speeds in excess of 4MHz. For system clock speeds in excess of 4MHz, the ADCS2, ADCS1 and ADCS0 bits should not be set to "000". Doing so will give A/D clock periods that are less than the minimum A/D clock period which may result in inaccurate A/D conversion values. Refer to the following table for examples, where values marked with an asterisk * show where, depending upon the device, special care must be taken, as the values may be less than the specified minimum A/D Clock Period.

| f _{sys} | A/D Clock Period (t _{AD}) | | | |
|------------------|---|---|--|-------------------------|
| | ADCS2, ADCS1, ADCS0=000 (f _{sys} /2) | ADCS2, ADCS1, ADCS0=001 (f _{sys} /8) | ADCS2, ADCS1, ADCS0=010 (f _{sys} /32) | ADCS2, ADCS1, ADCS0=011 |
| 1MHz | 2μs | 8μs | 32μs | Undefined |
| 2MHz | 1μs | 4μs | 16μs | Undefined |
| 4MHz | 500ns* | 2μs | 8μs | Undefined |
| 8MHz | 250ns* | 1μs | 4μs | Undefined |
| 12MHz | 167ns* | 667ns* | 2.67μs | Undefined |

A/D Clock Period Examples



A/D Converter Control Register – ACSR

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, 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 register programming, 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 as when the PCR2~PCR0 bits enable an A/D input, the status of the port control register will be overridden. The A/D converter has its own power supply pins AVDD and AVSS and a VREF reference pin. The analog input values must not be allowed to exceed the value of VREF.

Initialising the A/D Converter

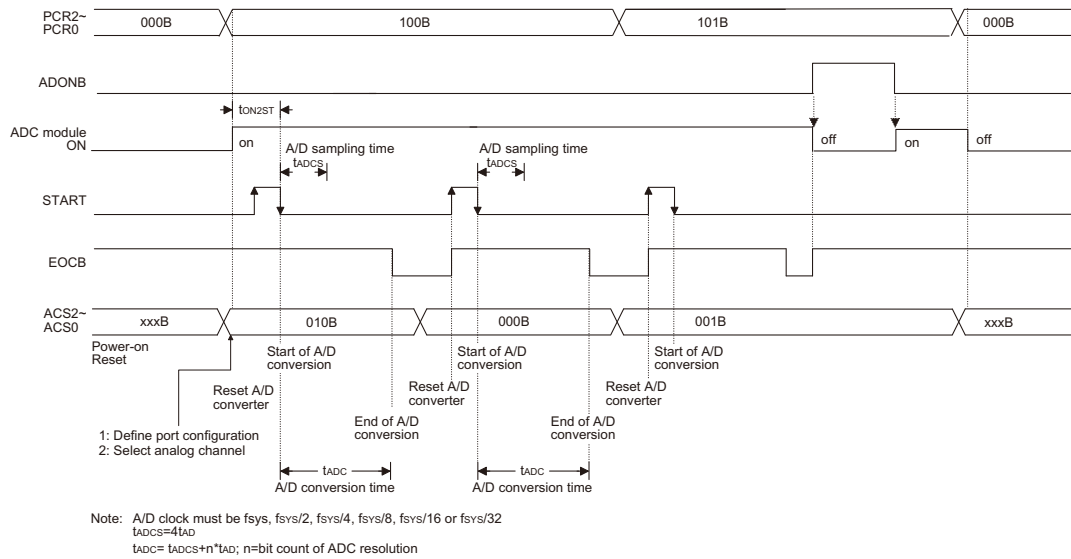
The internal A/D converter must be initialised in a special way. Each time the Port B A/D channel selection bits are modified by the program, the A/D converter must be re-initialised. If the A/D converter is not initialised after the channel selection bits are changed, the EOCB flag may have an undefined 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 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 summarises 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 ADCS2, ADCS1 and ADCS0 in the ACSR register.
- Step 2
Enable the A/D by clearing the ADONB in the ACSR register to zero.
- Step 3
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 4
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.



A/D Conversion Timing

- Step 5
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 INTC0 interrupt control register must be set to "1", the multi-function interrupt enable bit, EMFI, in the INTC1 register and the A/D converter interrupt bit, EADI, in the INTC1 register must also be set to "1".
- Step 6
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 7
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 accompanying 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. The time taken for the A/D conversion is $16t_{AD}$ where t_{AD} is equal to the A/D clock period.

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. The ADONB bit in the ACSR register can also be used to power down the A/D function.

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              ; select fsys/8 as A/D clock and turn on ADONB bit
mov  a,00100000B        ; setup ADCR register to configure Port PB0~PB3
                                ; as A/D inputs
                                ; and select AN0 to be connected to the A/D converter
mov  ADCR,a
:
:                          ; As the Port B channel bits have changed the
                                ; following START
                                ; signal (0-1-0) must be issued
                                ; 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 low byte conversion result value
mov  adrl_buffer,a        ; save result to user defined register
mov  a,ADRH              ; read high byte conversion result value
mov  adrh_buffer,a        ; save result to user defined register
:
jmp  start_conversion     ; start next A/D conversion

```

Example: using the interrupt method to detect the end of conversion

```

clr  EADI                ; disable ADC interrupt
mov  a,00000001B
mov  ACSR,a              ; select fsys/8 as A/D clock and turn on ADONB bit

mov  a,00100000B        ; setup ADCR register to configure Port PB0~PB3
                                ; as A/D inputs
                                ; and select AN0 to be connected to the A/D
mov  ADCR,a
:
:                          ; As the Port B channel bits have changed the
                                ; following START signal (0-1-0) must be issued
                                ;
:
Start_conversion:
clr  START
set  START                ; reset A/D
clr  START                ; start A/D
clr  ADF                  ; clear ADC interrupt request flag
set  EADI                 ; enable ADC interrupt
set  EMFI                 ; enable multi-function interrupt
set  EMI                 ; enable global interrupt
:
:
:
; ADC interrupt service routine
ADC_:
mov  acc_stack,a         ; save ACC to user defined memory
mov  a,STATUS
mov  status_stack,a     ; save STATUS to user defined memory
:
:
mov  a,ADRL               ; read low byte conversion result value
mov  adrl_buffer,a        ; save result to user defined register
mov  a,ADRH              ; read high byte conversion result value
mov  adrh_buffer,a        ; save result to user defined register
:
:
EXIT_ISR:
mov  a,status_stack
mov  STATUS,a           ; restore STATUS from user defined memory
mov  a,acc_stack
clr  ADF                ; clear ADC interrupt flag
reti

```

A/D Transfer Function

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

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.

Serial Interface

The device contains both SPI and I²C serial interface functions, which allows two methods of easy communication with external peripheral hardware. As the SPI and I²C function share the same external pins and internal registers their function must first be chosen by selecting the correct configuration option.

SPI Interface

The SPI interface is often used to communicate with external peripheral devices such as sensors, Flash or EEPROM memory devices etc. Originally developed by Motorola, the four line SPI interface is a synchronous serial data interface that has a relatively simple communication protocol simplifying the programming requirements when communicating with external hardware devices.

• **SPI Interface Operation**

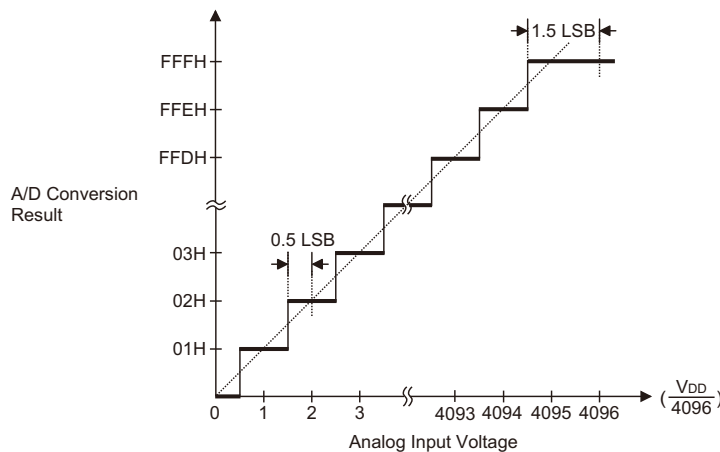
The SPI interface is a full duplex synchronous serial data link. Communication between devices connected to the SPI interface is carried out in a slave/master mode with all data transfer initiations being implemented by the master. Multiple slave devices can be connected to the SPI serial bus with each device controlled using its slave select line. The SPI is a four line interface with pin names SDI, SDO, SCK and SCS. Pins SDI and SDO are the Serial Data Input and Serial Data Output lines, SCK is the Serial Clock line and SCS is the Slave Select line. As the SPI interface pins are pin-shared with segment pins and with the I2C function pins, the SPI interface must first be enabled by selecting the correct configuration option. After the SPI configuration option has been selected it can then also be selected using the SIMEN bit in the SIMCON0 register.

The SPI function in this device offers the following features:

- ♦ Full duplex synchronous data transfer
- ♦ Both Master and Slave modes
- ♦ LSB first or MSB first data transmission modes
- ♦ Transmission complete flag
- ♦ Supports UART interface bridge
- ♦ IDLE mode supported

Several other configuration options also exist to setup various SPI interface options as follows:

- ♦ SPI pin enabled
- ♦ Rising or falling active clock edge
- ♦ WCOL bit enabled or disabled
- ♦ CSEN bit enabled or disabled



The status of the SPI interface pins is determined by a number of factors, whether the device is in master or slave mode and upon the condition of certain control bits such as CSEN and SIMEN.

| | Master/Slave (SIMEN=0) | Master (SIMEN=1) | | Slave (=1) | | |
|-----|------------------------|------------------------|------------------------|------------|---------------------|---------------------|
| | | CSEN=1 | CSEN=0 | CSEN=0 | SCS line=0 (CSEN=1) | SCS line=1 (CSEN=1) |
| SCS | Z | L | Z | Z | I, Z | I, Z |
| SDO | Z | O | O | O | O | Z |
| SDI | Z | I, Z | I, Z | I, Z | I, Z | Z |
| SCK | Z | L(CPOL=1) H(CPOL=0) | L(CPOL=1) H(CPOL=0) | I, Z | I, Z | Z |

"Z" floating, "H" output high, "L" output low, "I" Input, "O" output level, "I,Z" input floating (no pull-high)

SPI Interface Pin Status

• SPI Registers

The SIMDR register is used to store the data being transmitted and received. There are two control registers associated with the SPI interface, SIMCON0 and SIMCON2 and one data register known as SIMDR. The SIMCON1 register is not used by the SPI function. Register SIMCON0 is used to control the enable/disable function, the power down control and to set the data transmission clock frequency. Register SIMCON2 is used for other control functions such as LSB/MSB selection, write collision flag etc. The following gives further explanation of each bit:

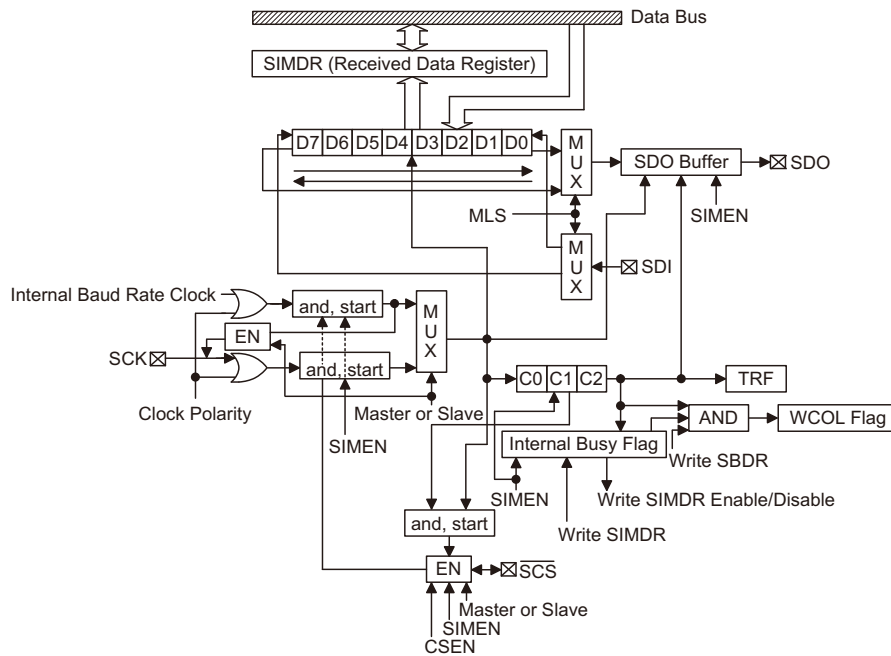
♦ SIMEN

The SIMEN bit is the overall on/off control for the SPI interface. When the SIMEN bit is cleared to zero to disable the SPI interface, the SDI, SCO,

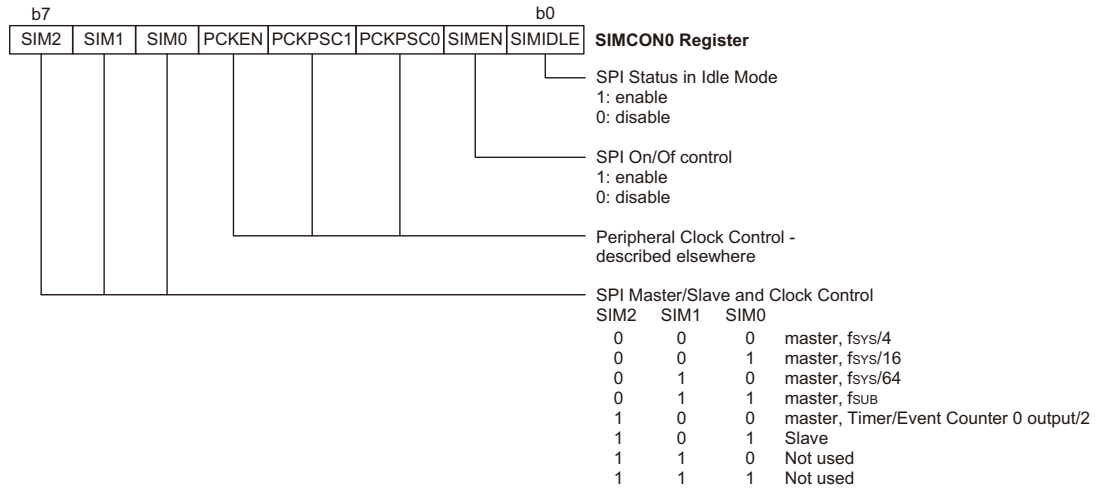
SCK and SCS lines will be in a floating condition and the SPI operating current will be reduced to <0.1µA at 5V. When the bit is high the SPI interface is enabled. Note that when the SIMEN bit changes from low to high the contents of the SPI control registers will be in an unknown condition and should therefore be initialised by the application program.

♦ SIMIDLE

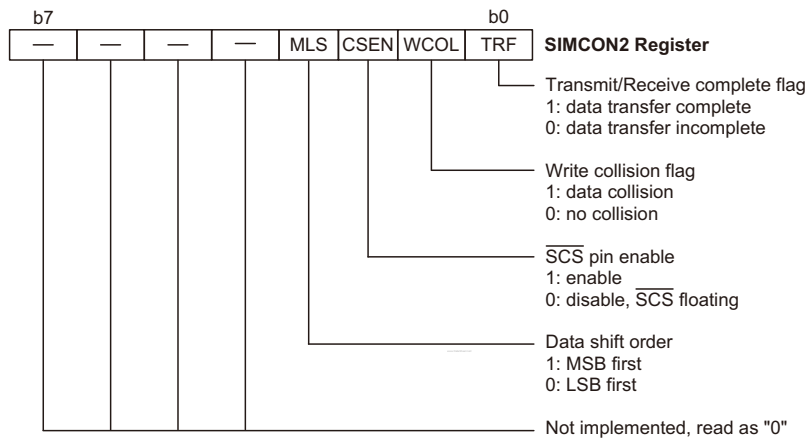
The SIMIDLE bit is used to select if the SPI interface continues running when the device is in the IDLE mode. Setting the bit high allows the SPI clock to keep running and enables the SPI interface to maintain operation when the device is in the Idle mode. Clearing the bit to zero disables any SPI operations when in the Idle mode.



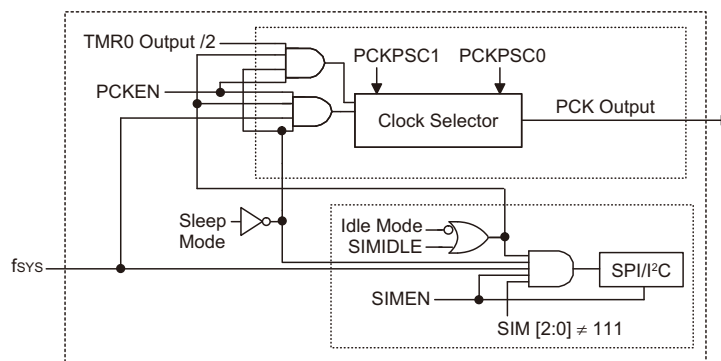
Block Diagram



SPI Control Register – SIMCON0



SPI Control Register – SIMCON2



SIM Module Structure

Note: Sleep Mode = HALT & IDLEN=0
Idle Mode = HALT & IDLEN=1

SPI SIM Module Structure

◆ SIM0~SIM2

These three bits control the Master/Slave selection and also setup the SPI interface clock speed when in the Master Mode. The SPI clock is a function of the system clock whether it be RC type or Crystal type but can also be chosen to be sourced from Timer/Event Counter 0 divided by two. If the Slave Mode is selected then the clock will be supplied by the external Master device.

The following gives further explanation of each bit:

◆ TRF

The TRF bit is the Transmit/Receive Complete flag and is cleared by the application program and can be used to generate an interrupt. When the bit is high the data has been transmitted or received. If the bit is low the data is being transmitted or has not yet been received.

◆ WCOL

The WCOL bit is used to detect if a data collision has occurred. If this bit is high it means that data has been attempted to be written to the SMDR register during a data transfer operation. This writing operation will be ignored if data is being transferred. The bit can be cleared by the application program. Note that using the SCEN bit can be disabled or enabled via configuration option.

◆ CSEN

The CSEN bit is used as an on/off control for the \overline{SCS} pin. If this bit is low then the \overline{SCS} pin will be disabled and placed into a floating condition. If the bit is high the \overline{SCS} pin will be enabled and used as a select pin.

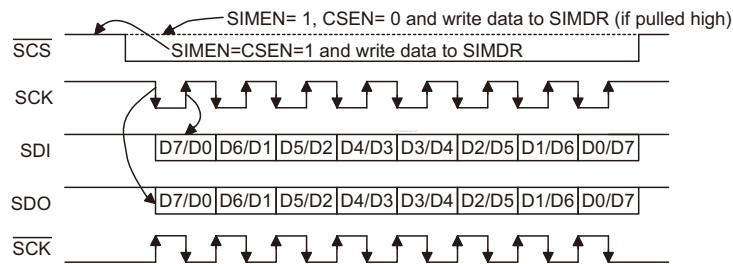
◆ MLS

The MLS is used to select how the data is transferred, either MSB or LSB first. Setting the bit high will select MSB first and low for LSB first.

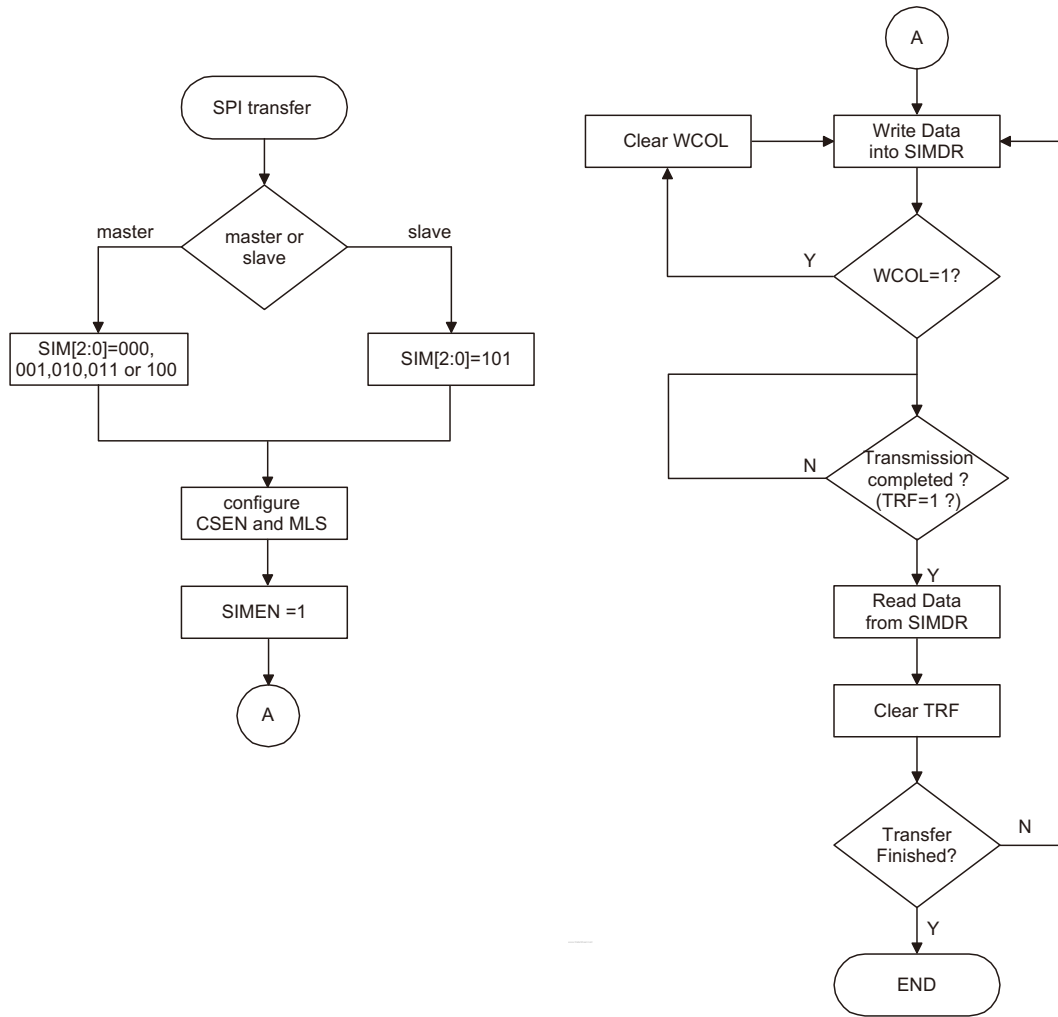
Note that the SIMCON2 register is the same as the SIMAR register used by the I²C interface.

• SPI Communication

After the SPI interface is enabled by setting the SIMEN bit high, then in the Master Mode, when data is written to the SIMDR register, transmission/reception will begin simultaneously. When the data transfer is complete, the TRF flag will be set automatically. In the Slave Mode, when the clock signal from the master has been received, any data in the SIMDR register will be transmitted and any data on the SDI pin will be shifted into the SIMDR register. The master should output an \overline{SCS} signal before a clock signal is provided and slave data transfers should be enabled/disabled before/after an \overline{SCS} signal is received.



SPI Interface Timing



SPI Transfer Control Flowchart

I²C Interface

The I²C bus is a bidirectional 2-line communication interface originally developed by Philips. The possibility of transmitting and receiving data on only 2 lines offers many new application possibilities for microcontroller based applications.

• I²C Interface Operation

As the I²C interface pins are pin-shared with segment pins and with the SPI function pins, the I²C interface must first be enabled by selecting the correct 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.

• I²C Registers

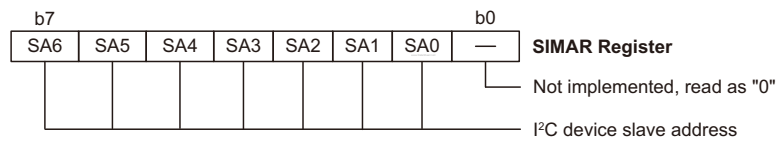
There are three control registers associated with the I²C bus, SIMCON0, SIMCON1 and SIMAR and one data register, SIMDR.

The SIMDR register is used to store the data being transmitted and received on the I²C bus. Before the microcontroller writes data to the I²C bus, the actual data to be transmitted must be placed in the SIMDR register. After the data is received from the I²C bus, the microcontroller can read it from the SIMDR register. Any transmission of data to the I²C bus or reception of data from the I²C bus must be made via the SIMDR register.

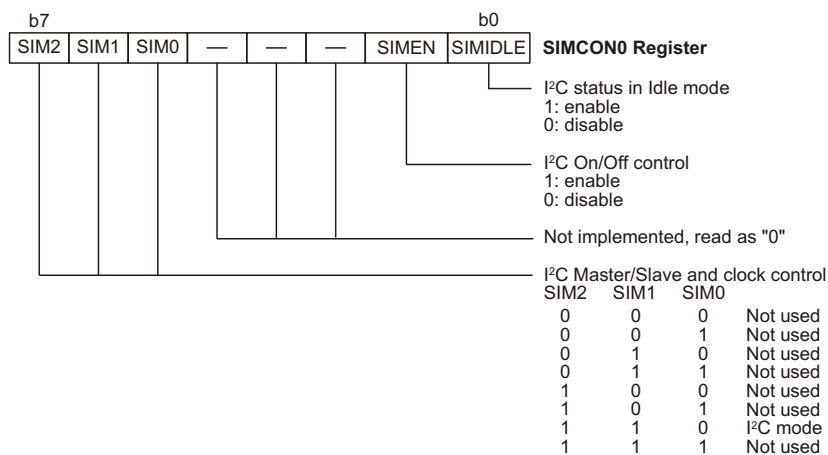
The SIMAR register is the location where the slave address of the microcontroller is stored. Bits 1~7 of the SIMAR register define the microcontroller slave address. Bit 0 is not defined. When a master device, which is connected to the I²C bus, sends out an address, which matches the slave address in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

Note that the SIMAR register is the same register as SIMCON2 which is used by the SPI interface.

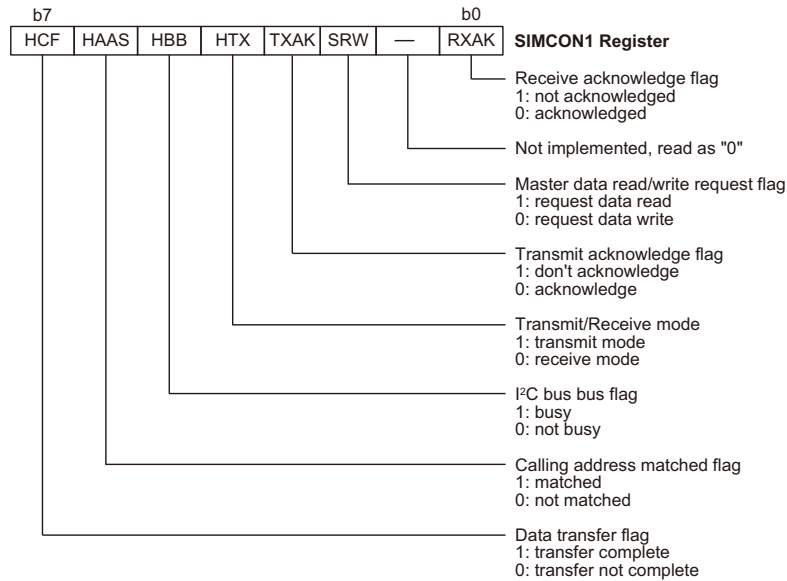
The SIMCON0 register is used for the I²C overall on/off control and to describe if the I²C interface remains active in the Idle Mode.



Slave Address Register – SIMAR



I²C Control Register – SIMCON0



I²C Control Register – SIMCON1

The following gives further explanation of each bit:

♦ SIMEN

The SIMEN bit determines if the I²C bus is enabled or disabled. If data is to be transferred or received on the I²C bus then this bit must be set high.

♦ SIMIDLE

The SIMIDLE bit is used to select if the I²C interface continues running when the device is in the IDLE mode. Setting the bit high allows the I²C interface to maintain operation when the device is in the Idle mode. Clearing the bit to zero disables any I²C operations when in the Idle mode.

The SIMCON1 register is used to control and monitor the status of the I²C bus.

The following gives further explanation of each bit:

♦ HCF

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

♦ HAAS

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

♦ HBB

The HBB flag is the I²C busy flag. This flag will be high when the I²C bus is busy which will occur when a START signal is detected. The flag will be reset to zero when the bus is free which will occur when a STOP signal is detected.

♦ HTX

The HTX flag is the transmit/receive mode bit. This flag should be set high to set the transmit mode and low for the receive mode.

♦ TXAK

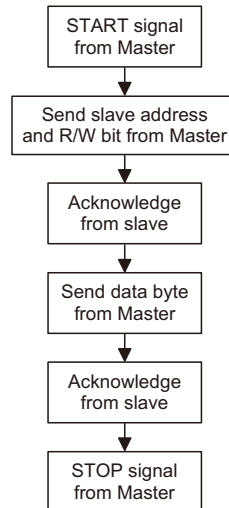
The TXAK flag is the transmit acknowledge flag. After the receipt of 8-bits of data, this bit will be transmitted to the bus on the 9th clock. To continue receiving more data, this bit has to be reset to zero before further data is received.

♦ SRW

The SRW bit is the Slave Read/Write bit. This 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 high, the device will check the SRW bit to determine whether it should be in transmit mode or receive mode. If the SRW bit is high, the master is requesting to read data from the bus, so the device should be in transmit mode. When the SRW bit is zero, the master will write data to the bus, therefore the device should be in receive mode to read this data.

♦ RXAK

The RXAK flag is the receive acknowledge flag. When the RXAK bit has been reset to zero 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 SIMCON1 register will be set and an I²C 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 I²C 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 I²C bus address register SIMAR.

Step 2

Set the SIMEN bit in the SIMCON0 register to "1" to enable the I²C bus.

Step 3

Set the ESIM bit of the interrupt control register to enable the I²C bus interrupt.

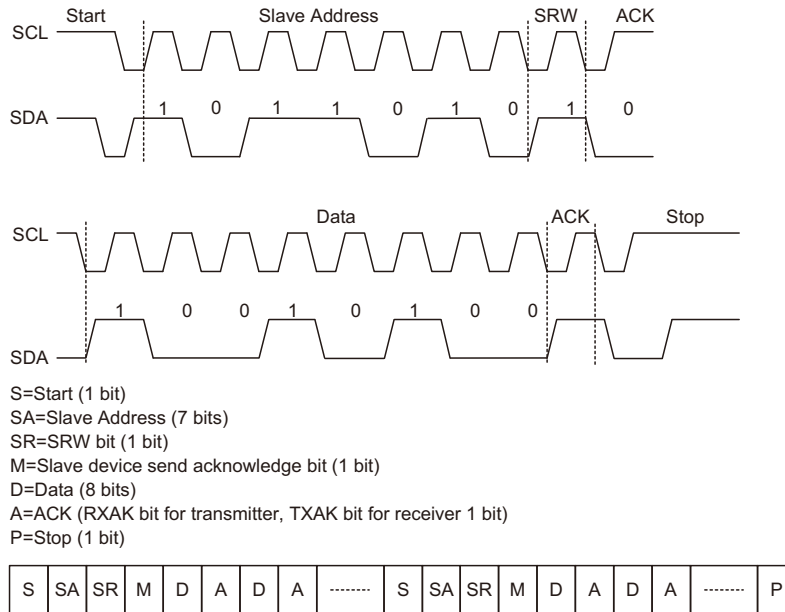
- **Start Signal**

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

- **Slave Address**

The transmission of a START signal by the master will be detected by all devices on the I²C bus. To determine which slave device the master wishes to communicate with, the address of the slave device will be sent out immediately following the START signal. All slave devices, after receiving this 7-bit address data, will compare it with their own 7-bit slave address. If the address sent out by the master matches the internal address of the microcontroller slave device, then an internal I²C bus interrupt signal will be generated. The next bit following the address, which is the 8th bit, defines the read/write status and will be saved to the SRW bit of the SIMCON1 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.

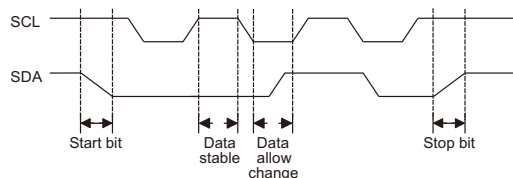
As an I²C bus interrupt can come from two sources, when the program enters the interrupt subroutine, the HAAS bit should be examined to see whether the interrupt source has come from a matching slave address or from the completion of a data byte transfer. When a slave address is matched, the device must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



I²C Communication Timing Diagram

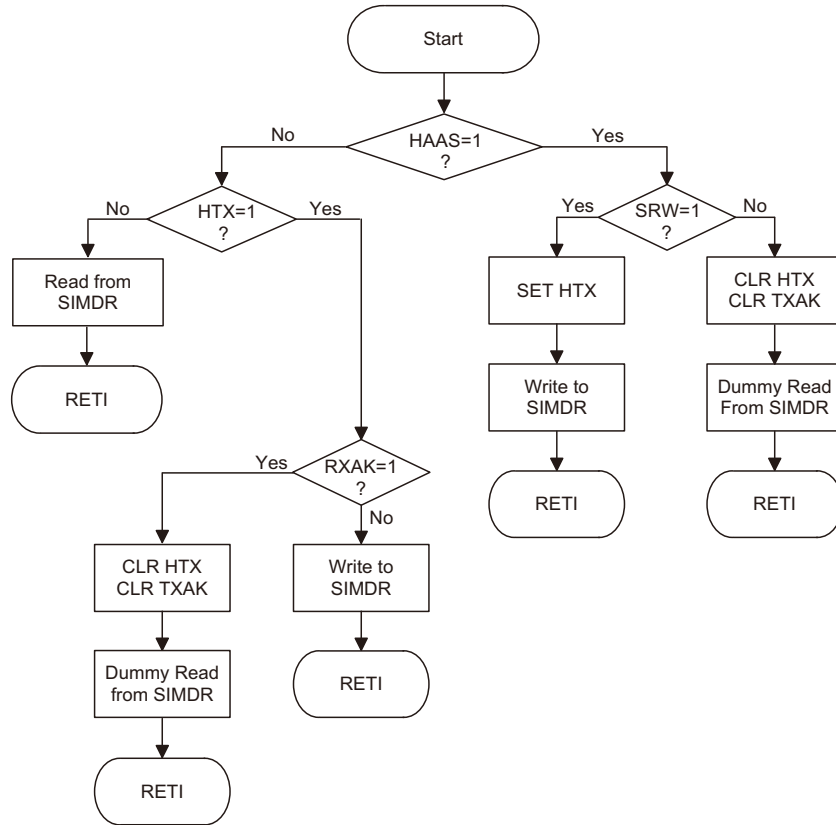
- SRW Bit**
 The SRW bit in the SIMCON1 register defines whether the microcontroller slave device wishes to read data from the I²C bus or write data to the I²C bus. The microcontroller should examine this bit to determine if it is to be a transmitter or a receiver. If the SRW bit is set to "1" then this indicates that the master wishes to read data from the I²C bus, therefore the microcontroller slave device must be setup to send data to the I²C bus as a transmitter. If the SRW bit is "0" then this indicates that the master wishes to send data to the I²C bus, therefore the microcontroller slave device must be setup to read data from the I²C bus as a receiver.
- 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 SIMCON1 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 SIMCON1 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 SIMDR register. If setup as a transmitter, the microcontroller slave device must first write the data to be transmitted into the SIMDR register. If setup as a receiver, the microcontroller slave device must read the transmitted data from the SIMDR 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 SIMCON1 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.

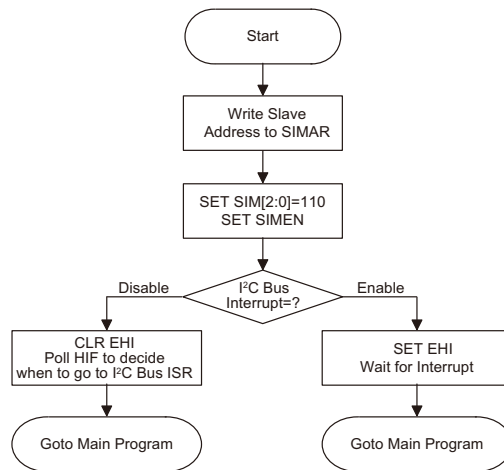


I²C Bus ISR Flow Chart

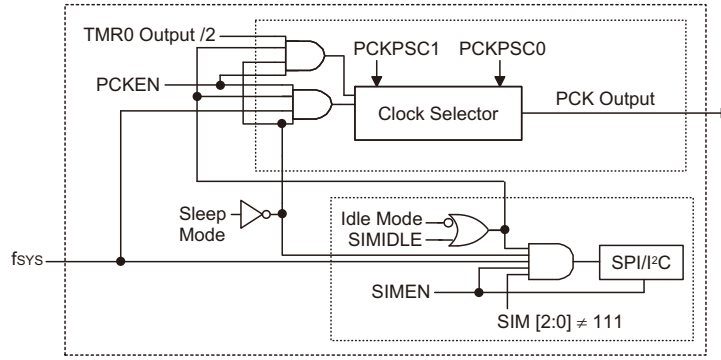
Peripheral Clock Output

The Peripheral Clock Output allows the device to supply external hardware with a clock signal synchronised to the microcontroller clock.

As the peripheral clock output pin, PINT, is shared with the LCD segment line SEG14, the required pin function is chosen via configuration option. The clock source for the Peripheral Clock Output can originate from either the Timer/Event Counter 0 divided by two or a divided ratio of the internal fsys clock. The clock source is selected using the PCKEN bit in the SIMCON0 register. The required division ratio of the system clock is selected using the PCKPSC0 and PSCPSC1 bits in the same register. If the system enters the Power down mode this will also influence the operation of the Peripheral Clock Output as shown in the block diagram.



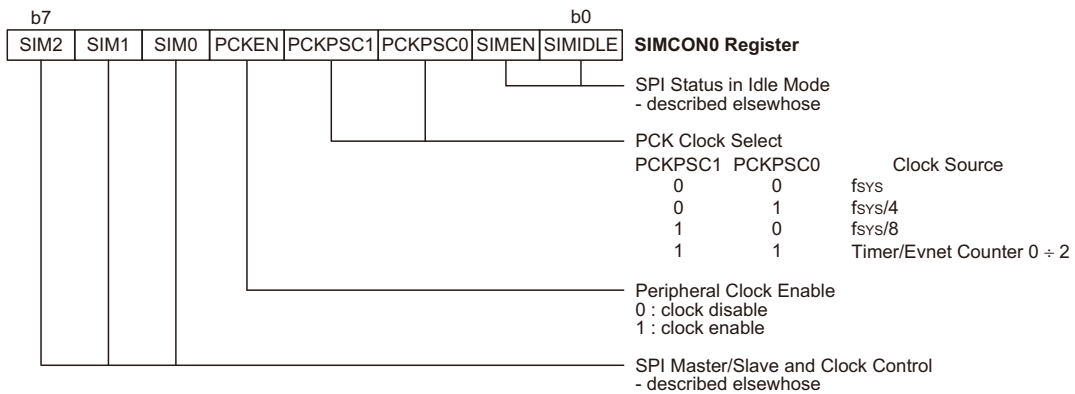
I²C Bus Initialisation Flow Chart



SIM Module Structure

Note: Sleep Mode = HALT & IDLEN=0
Idle Mode = HALT & IDLEN=1

SPI SIM Module Structure



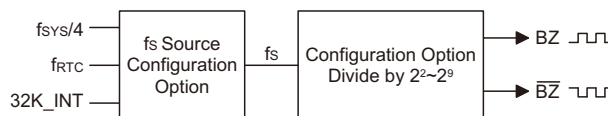
Peripheral Clock Output Control – SIMCON0

Buzzer

Operating in a similar way to the Programmable Frequency Divider, the Buzzer function provides a means of producing a variable frequency output, suitable for applications such as Piezo-buzzer driving or other external circuits that require a precise frequency generator. The BZ and \overline{BZ} pins form a complimentary pair, and are pin-shared with I/O pins, PA0 and PA1. A configuration option is used to select from one of three buzzer options. The first option is for both pins PA0 and PA1 to be used as normal I/Os, the second option is for both pins to be configured as BZ and \overline{BZ} buzzer pins, the third option selects only the PA0 pin to be used as a BZ buzzer pin with the PA1 pin retaining its normal I/O pin function. Note that the BZ pin is the inverse of the \overline{BZ} pin which together generate a differential output which can supply more power to connected interfaces such as buzzers.

The buzzer is driven by the internal clock source, f_S , which then passes through a divider, the division ratio of which is selected by configuration options to provide a range of buzzer frequencies from $f_S/2^2$ to $f_S/2^9$. The clock source that generates f_S , which in turn controls the buzzer frequency, can originate from three different sources, the RTC oscillator, the 32K_INT oscillator or the System oscillator/4, the choice of which is determined by the f_S clock source configuration option. Note that the buzzer frequency is controlled by configuration options, which select both the source clock for the internal clock f_S and the internal division ratio. There are no internal registers associated with the buzzer frequency.

If the configuration options have selected both pins PA0 and PA1 to function as a BZ and \overline{BZ} complementary pair of buzzer outputs, then for correct buzzer operation it is



Buzzer Function

essential that both pins must be setup as outputs by setting bits PAC0 and PAC1 of the PAC port control register to zero. The PA0 data bit in the PA data register must also be set high to enable the buzzer outputs, if set low, both pins PA0 and PA1 will remain low. In this way the

single bit PA0 of the PA register can be used as an on/off control for both the BZ and BZ buzzer pin outputs. Note that the PA1 data bit in the PA register has no control over the BZ buzzer pin PA1.

PA0/PA1 Pin Function Control

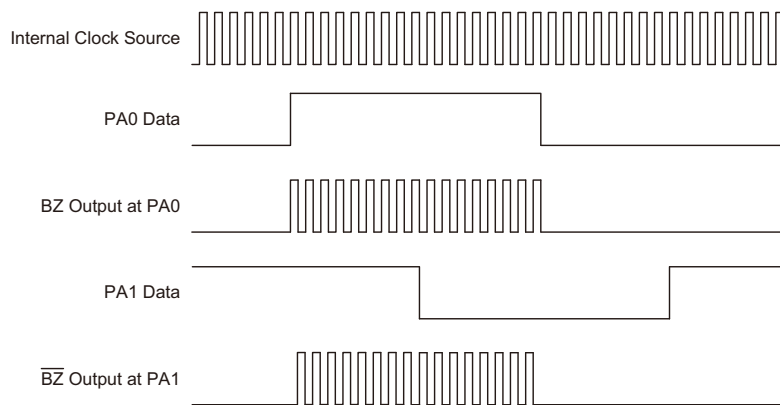
| PAC Register PAC0 | PAC Register PAC1 | PA Data Register PA0 | PA Data Register PA1 | Output Function |
|-------------------|-------------------|----------------------|----------------------|----------------------------------|
| 0 | 0 | 1 | x | PA0=BZ PA1=BZ |
| 0 | 0 | 0 | x | PA0="0" PA1="0" |
| 0 | 1 | 1 | x | PA0=BZ PA1=input line |
| 0 | 1 | 0 | x | PA0="0" PA1=input line |
| 1 | 0 | x | D | PA0=input line PA1=D |
| 1 | 1 | x | x | PA0=input line PA1=input line |

"x" stands for don't care

"D" stands for Data "0" or "1"

If configuration options have selected that only the PA0 pin is to function as a BZ buzzer pin, then the PA1 pin can be used as a normal I/O pin. For the PA0 pin to function as a BZ buzzer pin, PA0 must be setup as an output by setting bit PAC0 of the PAC port control register to zero. The PA0 data bit in the PA data register must also be set high to enable the buzzer output, if set low pin PA0 will remain low. In this way the PA0 bit can be used as an on/off control for the BZ buzzer pin PA0. If the PAC0 bit of the PAC port control register is set high, then pin PA0 can still be used as an input even though the configuration option has configured it as a BZ buzzer output.

Note that no matter what configuration option is chosen for the buzzer, if the port control register has setup the pin to function as an input, then this will override the configuration option selection and force the pin to always behave as an input pin. This arrangement enables the pin to be used as both a buzzer pin and as an input pin, so regardless of the configuration option chosen; the actual function of the pin can be changed dynamically by the application program by programming the appropriate port control register bit.



Buzzer Output Pin Control

Note: The above drawing shows the situation where both pins PA0 and PA1 are selected by configuration option to be BZ and BZ buzzer pin outputs. The Port Control Register of both pins must have already been setup as output. The data setup on pin PA1 has no effect on the buzzer outputs.

Interrupts

Interrupts are an important part of any microcontroller system. When an external event or an internal function such as a Timer/Event Counter 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 device contains several external interrupt and internal interrupts functions. The external interrupts are controlled by the action of the external INT0, INT1 and $\overline{\text{PINT}}$ pins, while the internal interrupts are controlled by the Timer/Event Counter overflows, the Time Base interrupt, the RTC interrupt, the SPI/I²C interrupt and the the A/D converter interrupt.

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, Time Base, RTC overflow, SPI/I²C data transfer complete, an end of A/D conversion or the external interrupt line being triggered 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 accompanying 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 Priority

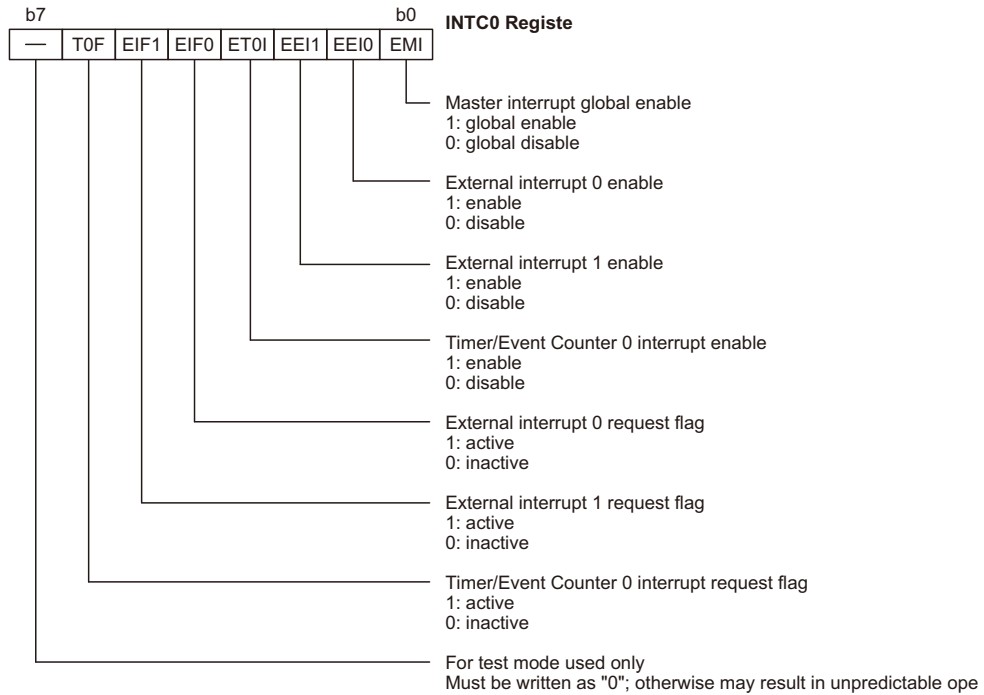
Interrupts, occurring in the interval between the rising edges of two consecutive T2 pulses, will be serviced on the latter of the two T2 pulses, if the corresponding interrupts are enabled. In case of simultaneous requests, the following table shows the priority that is applied.

| Interrupt Source | Priority | Vector |
|--------------------------------|----------|--------|
| External Interrupt 0 | 1 | 04H |
| External Interrupt 1 | 2 | 08H |
| Timer/Event Counter 0 Overflow | 3 | 0CH |
| Timer/Event Counter 1 Overflow | 4 | 10H |
| SPI/I ² C Interrupt | 5 | 14H |
| Multi-function Interrupt | 6 | 18H |

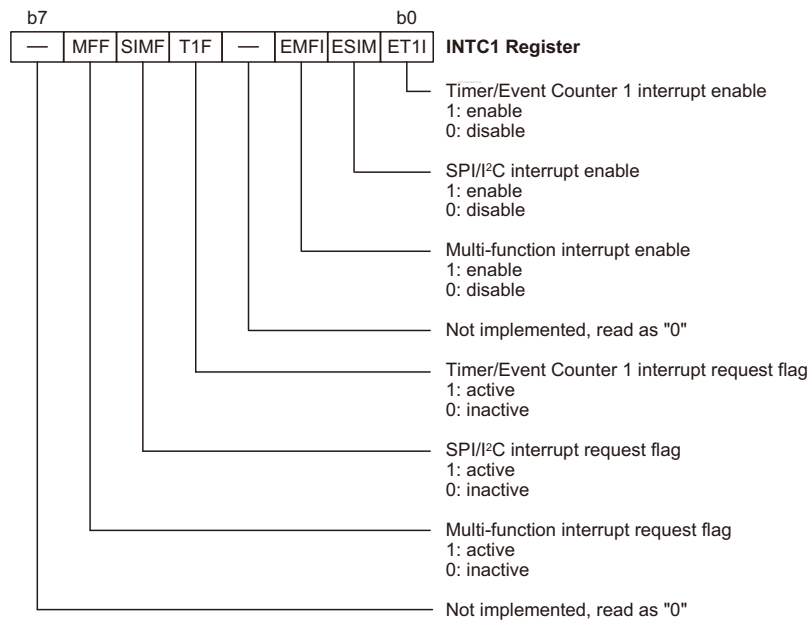
The A/D converter interrupt, Real Time clock interrupt, Time Base interrupt and External Peripheral interrupt all share the same interrupt vector which is 18H. Each of these interrupts have their own individual interrupt flag but also share the same MFF interrupt flag. The MFF flag will be cleared by hardware once the Multi-function interrupt is serviced, however the individual interrupts that have triggered the Multi-function interrupt need to be cleared by the application program.

External Interrupt

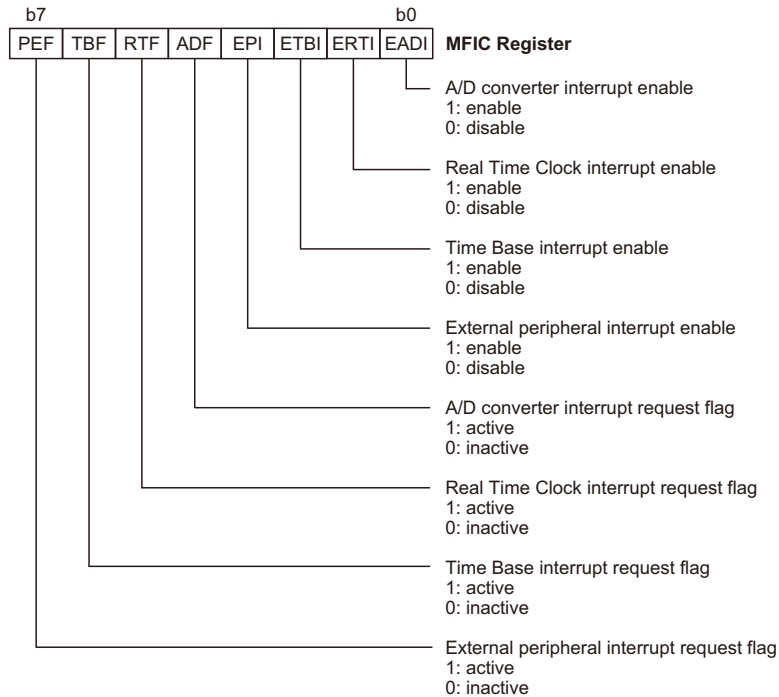
For an external interrupt to occur, the global interrupt enable bit, EMI, and external interrupt enable bits, EEI0 and EEI1, must first be set. Additionally the correct interrupt edge type must be selected using the INTEDGE register to enable the external interrupt function and to choose the trigger edge type. An actual external interrupt will take place when the external interrupt request flag, EIF0 or EIF1, is set, a situation that will occur when a transition, whose type is chosen by the edge select bit, appears on the INT0 or INT1 pin. The external interrupt pins are pin-shared with the I/O pins PD4 and PD5 and can only be configured as external interrupt pins if their corresponding external interrupt enable bit in the INTC0 register has been set. The pin must also be setup as an input by setting the corresponding PDC.4 and PDC.5 bits in the port control register. When the interrupt is enabled, the stack is not full and the correct transition type appears on the



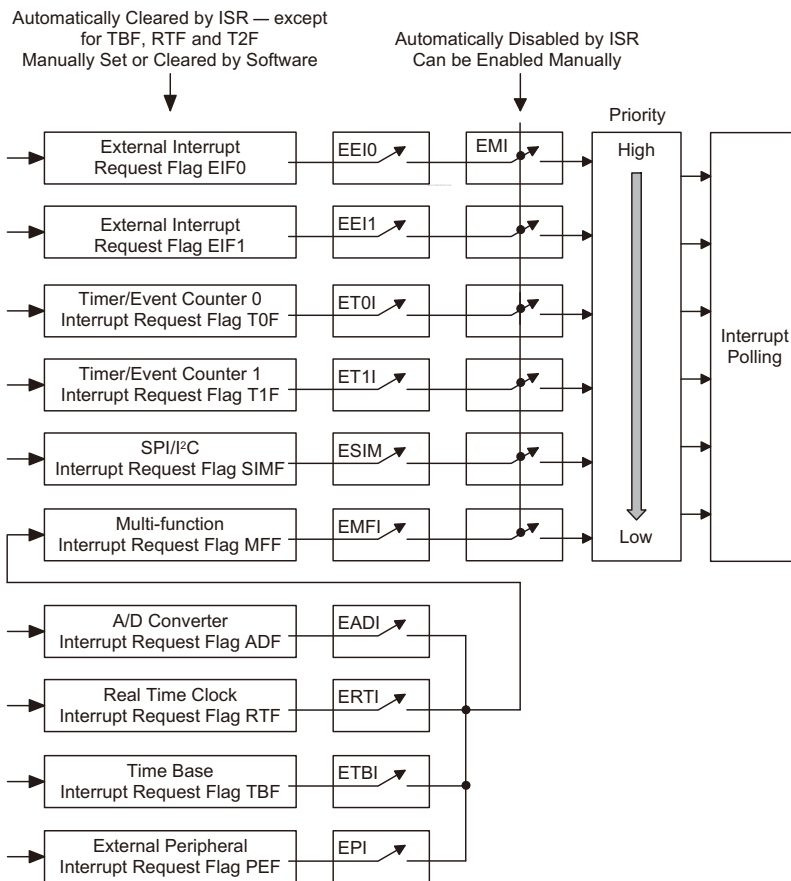
Interrupt Control Register INTC0



Interrupt Control Register INTC1

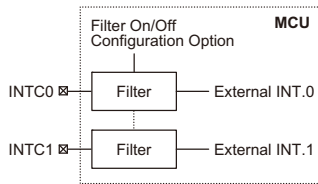


Interrupt Control Register MFIC



external interrupt pin, a subroutine call to the external interrupt vector at location 04H or 08H, will take place. When the interrupt is serviced, the external interrupt request flags, EIF0 or EIF1, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts. Note that any pull-high resistor selections on this pin will remain valid even if the pin is used as an external interrupt input.

The INTEDGE register is used to select the type of active edge that will trigger the external interrupt. A choice of either rising and falling edge types can be chosen along with an option to allow both edge types to trigger an external interrupt. Note that the INTEDGE register can also be used to disable the external interrupt function.



The external interrupt pins are connected to an internal filter to reduce the possibility of unwanted external interrupts due to adverse noise or spikes on the external interrupt input signal. As this internal filter circuit will consume a limited amount of power, a configuration option is provided to switch off the filter function, an option which may be beneficial in power sensitive applications, but in which the integrity of the input signal is high. Care must be taken when using the filter on/off configuration option as it will be applied not only to both the external interrupt pins but also to the Timer/Event Counter external input pins. Individual external interrupt or Timer/Event Counter pins cannot be selected to have a filter on/off function.

External Peripheral Interrupt

The External Peripheral Interrupt operates in a similar way to the external interrupt and is contained within the Multi-function interrupt.

For an external peripheral interrupt to occur, the global interrupt enable bit, EMI, external peripheral interrupt enable bit, EPI, and Multi-function interrupt enable bit, EMFI, must first be set. An actual external peripheral interrupt will take place when the external interrupt request flag, PEF, is set, a situation that will occur when a negative transition, appears on the PINT pin. The external peripheral interrupt pin is pin-shared with the segment pin SEG15, and is configured as a peripheral interrupt pin via a configuration option. When the interrupt is enabled, the stack is not full and a negative transition type appears on the external peripheral interrupt pin, a subroutine call to the Multi-function interrupt vector at location 18H, will take place. When the external peripheral interrupt is serviced, the EMI bit will be cleared to disable other interrupts, however only the MFF interrupt request flag will be reset. As the PEF flag will not be automatically reset, it has to be cleared by the application program.

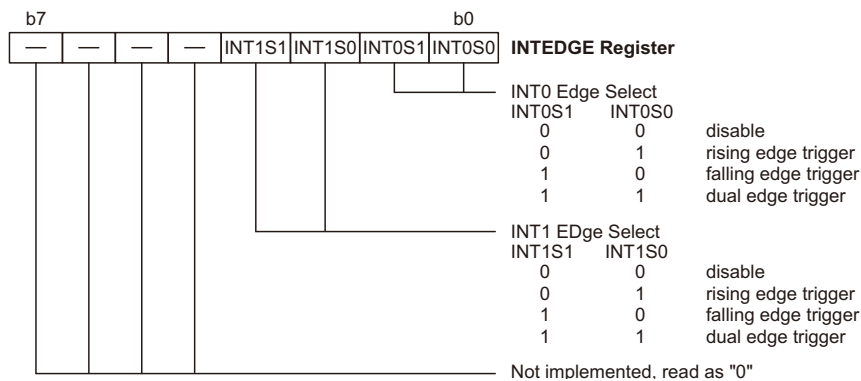
Timer/Event Counter Interrupt

For a Timer/Event Counter interrupt to occur, the global interrupt enable bit, EMI, and the corresponding timer interrupt enable bit, ET0I or ET1I, must first be set. An actual Timer/Event Counter interrupt will take place when the Timer/Event Counter request flag, T0F or T1F, is set, a situation that will occur when the Timer/Event Counter overflows. When the interrupt is enabled, the stack is not full and a Timer/Event Counter overflow occurs, a subroutine call to the timer interrupt vector at location 0CH or 10C, will take place. When the interrupt is serviced, the timer interrupt request flag, T0F or T1F, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

A/D Interrupt

The A/D Interrupt is contained within the Multi-function Interrupt.

For an A/D Interrupt to be generated, the global interrupt enable bit, EMI, A/D Interrupt enable bit, EADI, and Multi-function interrupt enable bit, EMFI, must first be



Interrupt Active Edge Register – INTEDGE

set. An actual A/D Interrupt will take place when the A/D Interrupt request flag, ADF, is set, a situation that will occur when the A/D conversion process has finished. When the interrupt is enabled, the stack is not full and the A/D conversion process has ended, a subroutine call to the Multi-function interrupt vector at location 18H, will take place. When the A/D Interrupt is serviced, the EMI bit will be cleared to disable other interrupts, however only the MFF interrupt request flag will be reset. As the ADF flag will not be automatically reset, it has to be cleared by the application program.

SPI/I²C Interface Interrupt

For an SPI/I²C interrupt to occur, the global interrupt enable bit, EMI, and the corresponding interrupt enable bit, ESIM must be first set. An actual SPI/I²C interrupt will take place when the SPI/I²C interface request flag, SIMF, is set, a situation that will occur when a byte of data has been transmitted or received by the SPI/I²C interface. When the interrupt is enabled, the stack is not full and a byte of data has been transmitted or received by the SPI/I²C interface, a subroutine call to the SPI/I²C interrupt vector at location 14H, will take place. When the interrupt is serviced, the SPI/I²C request flag, SIMF will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

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 four other existing interrupt sources, namely the A/D Converter interrupt, Time Base interrupt, Real Time Clock interrupt and the External Peripheral interrupt.

For a Multi-function interrupt to occur, the global interrupt enable bit, EMI, and the Multi-function interrupt enable bit, EMFI, must first be set. An actual Multi-function interrupt will take place when the Multi-function interrupt request flag, MFF, is set. This will occur when either a Time Base overflow, a Real Time Clock overflow, an A/D conversion completion or an External Peripheral Interrupt is generated. When the interrupt is enabled and the stack is not full, and either one of the interrupts contained within the Multi-function interrupt occurs, a subroutine call to the Multi-function interrupt vector at location 018H will take place. When the interrupt is serviced, the Multi-Function request flag, MFF, will be automatically reset and the EMI

bit will be automatically 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 interrupt, Real Time Clock interrupt, A/D Converter interrupt or External Peripheral interrupt will not be automatically reset and must be manually reset by the application program.

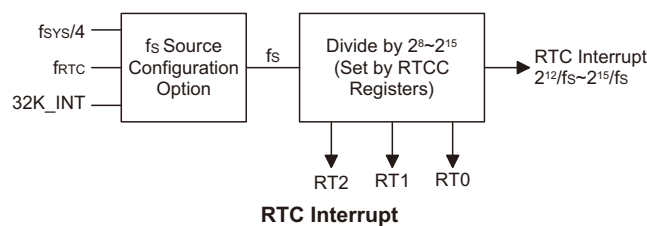
Real Time Clock Interrupt

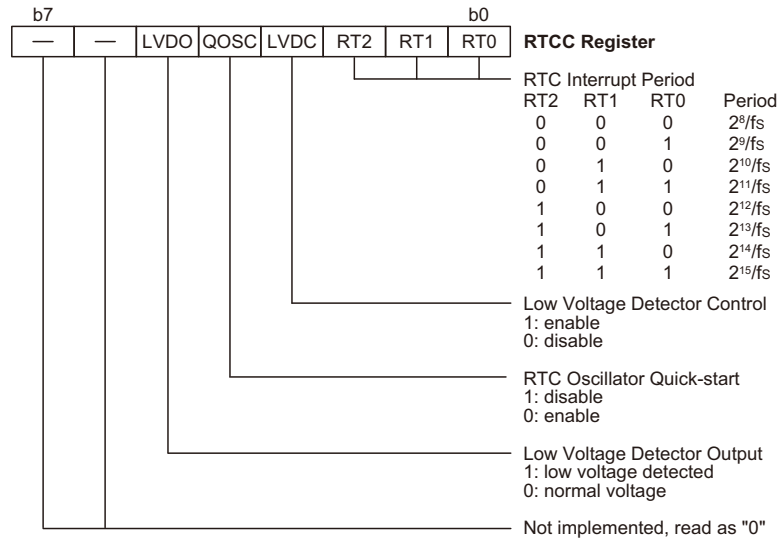
The Real Time Clock Interrupt is contained within the Multi-function Interrupt.

For a Real Time Clock interrupt to be generated, the global interrupt enable bit, EMI, Real Time Clock interrupt enable bit, ERTI, and Multi-function interrupt enable bit, EMFI, must first be set. An actual Real Time Clock interrupt will take place when the Real Time Clock request flag, RTF, is set, a situation that will occur when the Real Time Clock overflows. When the interrupt is enabled, the stack is not full and the Real Time Clock overflows, a subroutine call to the Multi-function interrupt vector at location 18H, will take place. When the Real Time Clock interrupt is serviced, 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.

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 $2^8/f_S \sim 2^{15}/f_S$. The clock source that generates f_S , which in turn controls the RTC interrupt period, can originate from three different sources, the RTC oscillator, 32K_INT oscillator or the System oscillator/4, the choice of which is determined by the f_S clock source configuration option.

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} .





Real Time Clock Control Register – RTCC

Time Base Interrupt

The Time Base Interrupt is contained within the Multi-function Interrupt.

For a Time Base Interrupt to be generated, the global interrupt enable bit, EMI, Time Base Interrupt enable bit, ETBI, and Multi-function interrupt enable bit, EMFI, must first be set. An actual Time Base Interrupt will take place when the Time Base Interrupt request flag, TBF, is set, a situation that will occur when the Time Base overflows. When the interrupt is enabled, the stack is not full and the Time Base overflows, a subroutine call to the Multi-function interrupt vector at location 18H, will take place. When the Time Base Interrupt is serviced, 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 function is to provide an interrupt signal at fixed time periods. The Time Base interrupt clock source originates from 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 2¹²/f_S~2¹⁵/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, the 32K_INT internal oscillator or the System oscillator/4, the choice of which is determine by the f_S clock

source configuration option.

Essentially operating as a programmable timer, when the Time Base overflows it will set a Time Base interrupt flag which will in turn generate an Interrupt request via the Multi-function Interrupt vector.

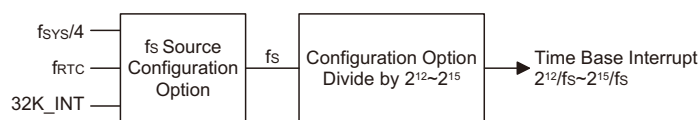
Programming Considerations

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 INTC0, INTC1 and MFIC registers until the corresponding interrupt is serviced or until the request flag is cleared by the application program.

It is recommended that programs do not use the "CALL 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 status or other registers are altered by the interrupt service program, which may corrupt the desired control sequence, then the contents should be saved in advance.



Time Base Interrupt

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 $\overline{\text{RES}}$ line is forcefully pulled low. In such a case, known as a normal operation reset, some of the microcontroller registers remain unchanged allowing the microcontroller to 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.

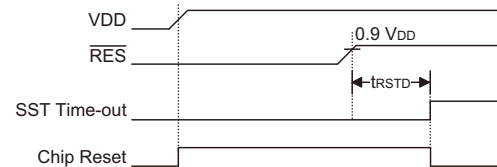
Another reset exists in the form of a Low Voltage Reset, LVR, where a full reset, similar to the $\overline{\text{RES}}$ reset is implemented in situations where the power supply voltage falls below a certain threshold.

Reset Functions

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

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

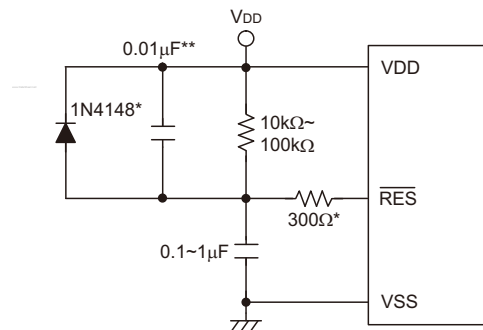
proper reset operation. For this reason it is recommended that an external RC network is connected to the $\overline{\text{RES}}$ pin, whose additional time delay will ensure that the $\overline{\text{RES}}$ pin remains low for an extended period to allow the power supply to stabilise. During this time delay, normal operation of the microcontroller will be inhibited. After the $\overline{\text{RES}}$ line reaches a certain voltage value, the reset delay time t_{RSTD} is invoked to provide an extra delay time after which the microcontroller will begin normal operation. The abbreviation SST in the figures stands for System Start-up Timer.



Power-On Reset Timing Chart

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

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



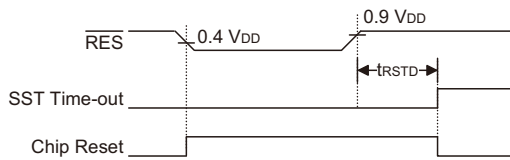
- Note: ******* It is recommended that this component is added for added ESD protection
******** It is recommended that this component is added in environments where power line noise is significant

External $\overline{\text{RES}}$ Circuit

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

• **RES Pin Reset**

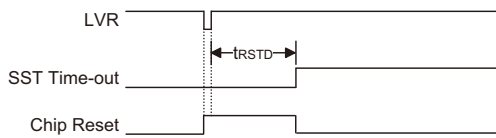
This type of reset occurs when the microcontroller is already running and the RES pin is forcefully pulled low by external hardware such as an external switch. In this case as in the case of other reset, the Program Counter will reset to zero and program execution initiated from this point.



RES Reset Timing Chart

• **Low Voltage Reset – LVR**

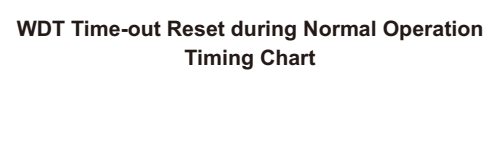
The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device, which is selected via a configuration option. If the supply voltage of the device drops to within a range of $0.9V \sim V_{LVR}$ such as might occur when changing the battery, the LVR will automatically 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 $0.9V \sim V_{LVR}$ 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**

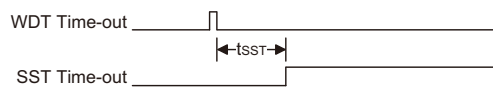
The Watchdog time-out Reset during normal operation is the same as a hardware RES pin reset except that the Watchdog time-out flag TO will be set to "1".



WDT Time-out Reset during Normal Operation 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 Power Down 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 | RES reset during power-on |
| u | u | 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) | RES Reset (Normal Operation) | WDT Time-out (Normal Operation) | WDT Time-out (HALT) |
|----------|------------------|------------------------------|---------------------------------|---------------------|
| MP0 | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| MP1 | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| BP | - - - - - - - 0 | - - - - - - - 0 | - - - - - - - 0 | - - - - - - - u |
| ACC | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| PCL | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 |
| TBLP | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| TBLH | x x x x x x x x | u u u u u u u u | u u u u u u u u | u u u u u u u u |
| RTCC | - - 0 0 0 1 1 1 | - - 0 0 0 1 1 1 | - - 0 0 0 1 1 1 | - - u u u u u u |
| STATUS | - - 0 0 x x x x | - - u u u u u u | - - 1 u u u u u | - - 1 1 u u u u |
| INTC0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | u u u u u u u u |
| TMR0 | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| TMR0C | 0 0 - 0 1 0 0 0 | 0 0 - 0 1 0 0 0 | 0 0 - 0 1 0 0 0 | u u - u u u u u |
| TMR1H | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| TMR1L | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| TMR1C | 0 0 0 0 1 - - - | 0 0 0 0 1 - - - | 0 0 0 0 1 - - - | u u u u u - - - |
| PA | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PAC | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PB | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PBC | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PD | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PDC | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | 1 1 1 1 1 1 1 1 | u u u u u u u u |
| PWM0L | 0 0 0 0 - - - 0 | 0 0 0 0 - - - 0 | 0 0 0 0 - - - 0 | u u u u - - - u |
| PWM0H | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | u u u u u u u u |
| PWM1L | 0 0 0 0 - - - 0 | 0 0 0 0 - - - 0 | 0 0 0 0 - - - 0 | u u u u - - - u |
| PWM1H | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | u u u u u u u u |
| INTC1 | - 0 0 0 - - 0 0 | - 0 0 0 - - 0 0 | - 0 0 0 - - 0 0 | - u u u - - u u |
| TBHP | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| PWM2L | 0 0 0 0 - - - 0 | 0 0 0 0 - - - 0 | 0 0 0 0 - - - 0 | u u u u - - - u |
| PWM2H | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | u u u u u u u u |
| PWM3L | 0 0 0 0 - - - 0 | 0 0 0 0 - - - 0 | 0 0 0 0 - - - 0 | u u u u - - - u |
| PWM3H | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | 0 0 0 0 0 0 0 0 | u u u u u u u u |
| ADRL | x x x x - - - - | x x x x - - - - | x x x x - - - - | u u u u - - - - |
| ADRH | x x x x x x x x | x x x x x x x x | x x x x x x x x | u u u u u u u u |
| ADCR | 0 1 0 0 0 0 0 0 | 0 1 0 0 0 0 0 0 | 0 1 0 0 0 0 0 0 | u u u u u u u u |
| ACSR | 1 0 - - - 0 0 0 | 1 0 - - - 0 0 0 | 1 0 - - - 0 0 0 | u u - - - u u u |

| Register | Reset (Power-on) | RES Reset (Normal Operation) | WDT Time-out (Normal Operation) | WDT Time-out (HALT) |
|---------------|------------------|------------------------------|---------------------------------|---------------------|
| CLKMOD | 000 – 0011 | 000 – 0011 | 000 – 0011 | uuuu – uuu |
| PAWU | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| PAPU | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| PBPU | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| PDPU | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| INTEDGE | ---- 0000 | ---- 0000 | ---- 0000 | ---- uuuu |
| LCDCTRL | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| LCDOUT1 | ---- --00 | ---- --00 | ---- --00 | ---- --uu |
| LCDOUT2 | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| MISC | 0000 1010 | 0000 1010 | 0000 1010 | uuuu uuuu |
| MFIC | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| SIMCON0 | 1110 0000 | 1110 0000 | 1110 0000 | uuuu uuuu |
| SIMCON1 | 1000 00–1 | 1000 00–1 | 1000 00–1 | uuuu uu–u |
| SIMDR | x xxx x xxx | x xxx x xxx | x xxx x xxx | uuuu uuuu |
| SIMAR/SIMCON2 | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |

Note: "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. Five 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.

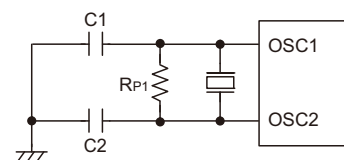
System Clock Configurations

There are five methods of generating the system clock, two high oscillators, two low oscillators and an externally supplied clock. The two high oscillators are the external crystal/ceramic oscillator and the external RC network. The two low oscillators are the fully integrated 32K_INT oscillator and the external RTC oscillator. Selecting whether the low or high oscillator is used as the system oscillator is implemented using the HLCLK bit in the CLKMOD register. The source clock for the high and low oscillators is chosen via configuration options. The frequency of the slow oscillator is also determined using the SLOWC0~SLOWC2 bits in the CLKMOD register.

System Crystal/Ceramic Oscillator

After selecting the correct oscillator configuration option, for most crystal oscillator configurations, the simple connection of a crystal across OSC1 and OSC2 will create the necessary phase shift and feedback for oscillation,

without requiring external capacitors. However, for some crystal types and frequencies, to ensure oscillation, it may be necessary to add two small value capacitors, C1 and C2. Using a ceramic resonator will usually require two small value capacitors, C1 and C2, to be connected as shown for oscillation to occur. The values of C1 and C2 should be selected in consultation with the crystal or resonator manufacturer's specification. In most applications, resistor R_{P1} is not required, however for those applications where the LVR function is not used, R_{P1} may be necessary to ensure the oscillator stops running when VDD falls below its operating range. The internal oscillator circuit contains a filter circuit to reduce the possibility of erratic operation due to noise on the oscillator pins.



Crystal/Ceramic Oscillator

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

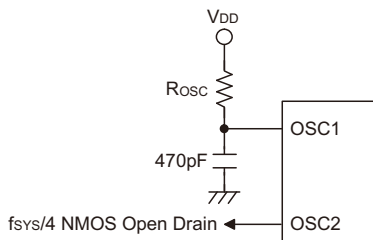
| Crystal Oscillator C1 and C2 Values | | |
|-------------------------------------|------|------|
| Crystal/Resonator Frequency | C1 | C2 |
| 12MHz Crystal | — | — |
| 8MHz Crystal | — | — |
| 4MHz Crystal | — | — |
| 1MHz Crystal | — | — |
| 455kHz Resonator (see Note 2) | 10pF | 10pF |

Note: 1. C1 and C2 values are for guidance only.
2. XTAL mode configuration option: 455kHz.

Crystal Recommended Capacitor Values

External System RC Oscillator

After selecting the correct configuration option, using the external system RC oscillator requires that a resistor, with a value between 47kΩ and 1.5MΩ, is connected between OSC1 and VDD, and a 470pF capacitor is connected to ground. 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 R_{OSC} refer to the Appendix section for typical RC Oscillator vs. Temperature and VDD characteristics graphics.



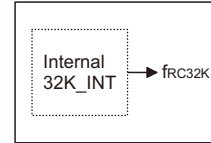
RC Oscillator

Note that an internal capacitor together with the external resistor, R_{OSC}, are the components which determine the frequency of the oscillator. The external capacitor shown on the diagram does not influence the frequency of oscillation. This external capacitor should be added to improve oscillator stability if the open-drain OSC2 output is utilised in the application circuit. The internal oscillator circuit contains a filter circuit to reduce the possibility of erratic operation due to noise on the oscillator pins.

Internal 32K_INT Oscillator

When microcontrollers enter a power down condition, their internal clocks are normally switched off to stop microcontroller activity and to conserve power. However, in many microcontroller applications it may be necessary to keep some internal functions operational, such as timers, even when the microcontroller is in the Power-down mode. To do this, the device has a 32K_INT oscillator, which is a fully integrated free run-

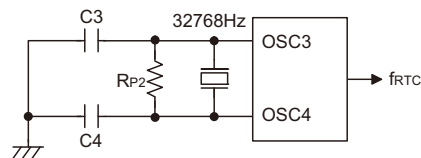
ning RC oscillator with a typical period of 31.2μs at 5V, requiring no external components. It is selected via configuration option. When the device enters the Power Down Mode, the system clock will stop running, however the 32K_INT oscillator will continue to run if selected to keep various internal functions operational.



Internal 32K_INT Oscillator

External RTC Oscillator

With a function similar to the internal 32K-INT 32KHz oscillator, that is to keep some device functions operational during power down, this device also has an external RTC oscillator. This oscillator also remains active at all times, even when the microcontroller is in the Power-down mode. This clock source has a fixed frequency of 32768Hz and requires a 32768Hz crystal to be connected between pins OSC3 and OSC4.

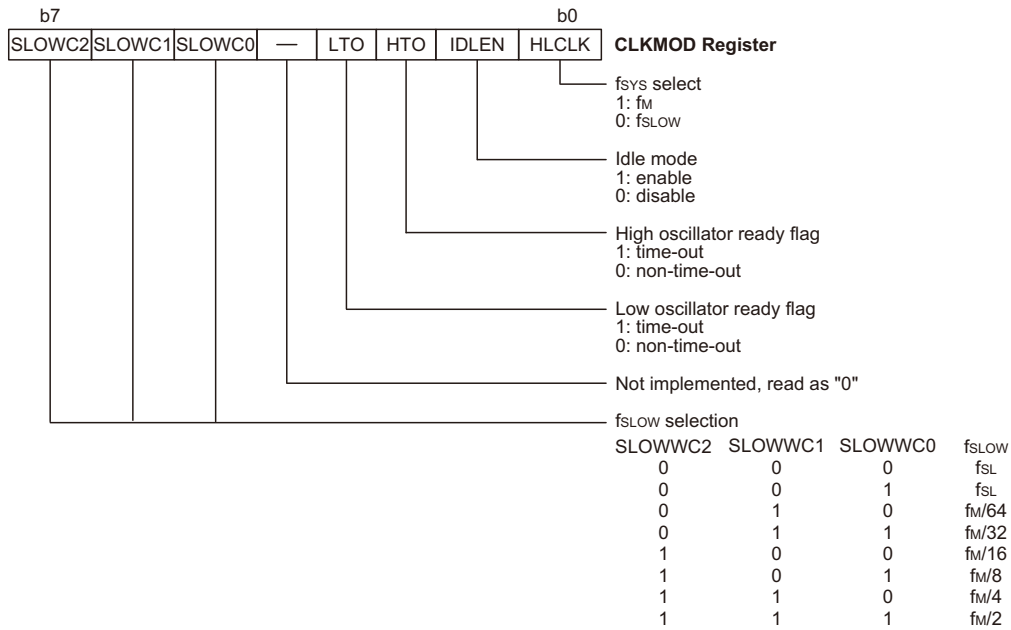


External RTC Oscillator

The external resistor and capacitor components connected to the 32768Hz crystal are not necessary to provide oscillation. For applications where precise RTC frequencies are essential, these components may be required to provide frequency compensation due to different crystal manufacturing tolerances.

A configuration option selects whether the external RTC oscillator or the internal 32K_INT oscillator is selected. Selecting the external RTC oscillator for use as a system oscillator is implemented using bits in the CLKMOD register.

During power-up there is a time delay associated with the RTC oscillator waiting for it to start-up. To minimise this time delay, bit 4 of the RTCC register, known as the QOSC bit, is provided to have a quick start-up function. During a power-up condition, this bit will be cleared to zero 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 for 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.



Clock Control Register – CLKMOD

| 32768Hz Oscillator C1 and C2 Values | | |
|---|-----|------|
| Crystal Frequency | C3 | C4 |
| 32768Hz | 8pF | 10pF |
| Note: 1. C3 and C4 values are for guidance only. 2. R _{P2} =5M~10MΩ is recommended. | | |

32768 Hz Crystal Recommended Capacitor Values

External Oscillator

The system clock can also be supplied by an externally supplied clock giving users a method of synchronising their external hardware to the microcontroller operation. This is selected using a configuration option and supplying the clock on pin OSC1. Pin OSC2 should be left floating if the external oscillator is used. The internal oscillator circuit contains a filter circuit to reduce the possibility of erratic operation due to noise on the oscillator pin, however as the filter circuit consumes a certain amount of power, a configuration option exists to turn this filter off. Not using the internal filter should be considered in power sensitive applications and where the externally supplied clock is of a high integrity and supplied by a low impedance source.

Dual Clock Mode

The device has a dual clock mode for system clock operation, one is known as the high oscillator and the other as the low oscillator. The High system clock source f_M is selected using a configuration option and can be either an external crystal or external RC oscillator.

The low oscillator clock source, also known as the Sub-clock, f_{SUB}, is selected also by configuration option

and can be either the external RTC oscillator or the internal 32K_INT oscillator.

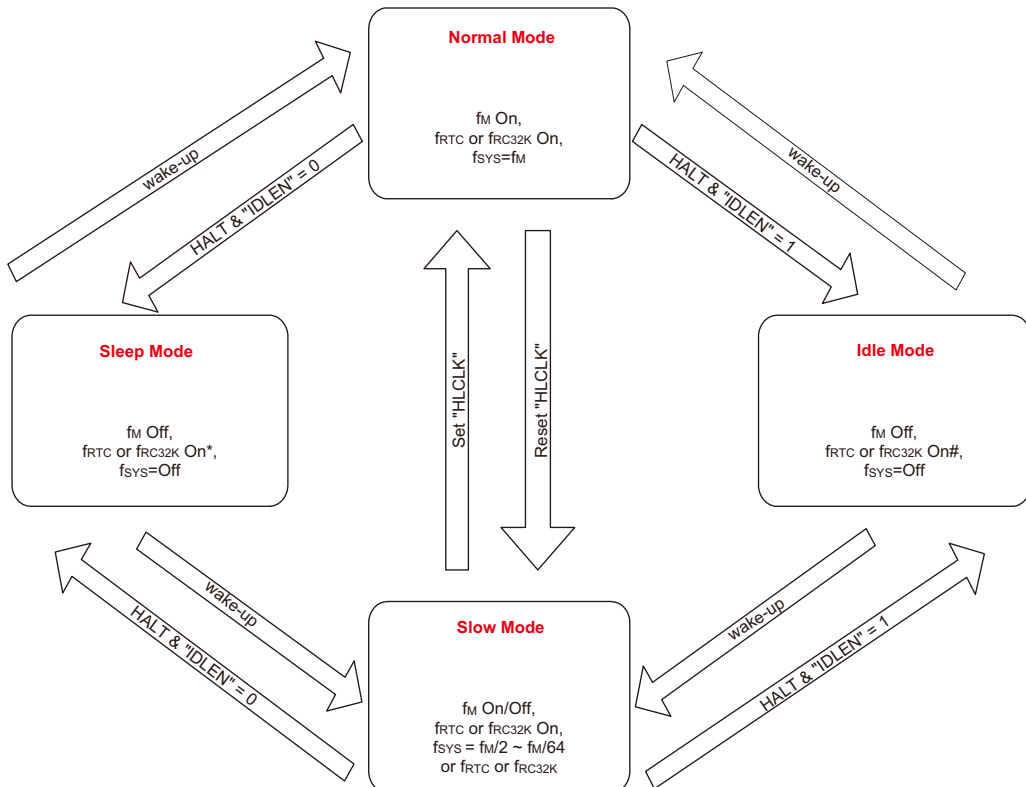
The actual frequency of the slow system clock, f_{SLOW}, is also determined using the SLOWC0~SLOWC2 bits in the CLKMOD register.

The LCD clock source is provided by f_{LCD} which is f_{SUB} divided by 8, giving a frequency of 4kHz.

The f_S clock is an internal clock source for the Buzzer, the RTC oscillator interrupt, the Time Base interrupt and the Watchdog Timer. The source clock for f_S is selected from one of the oscillators, f_{SUB} or f_{SYS}/4, using a configuration option.

The Dual Clock Mode can operate in four states as follows:

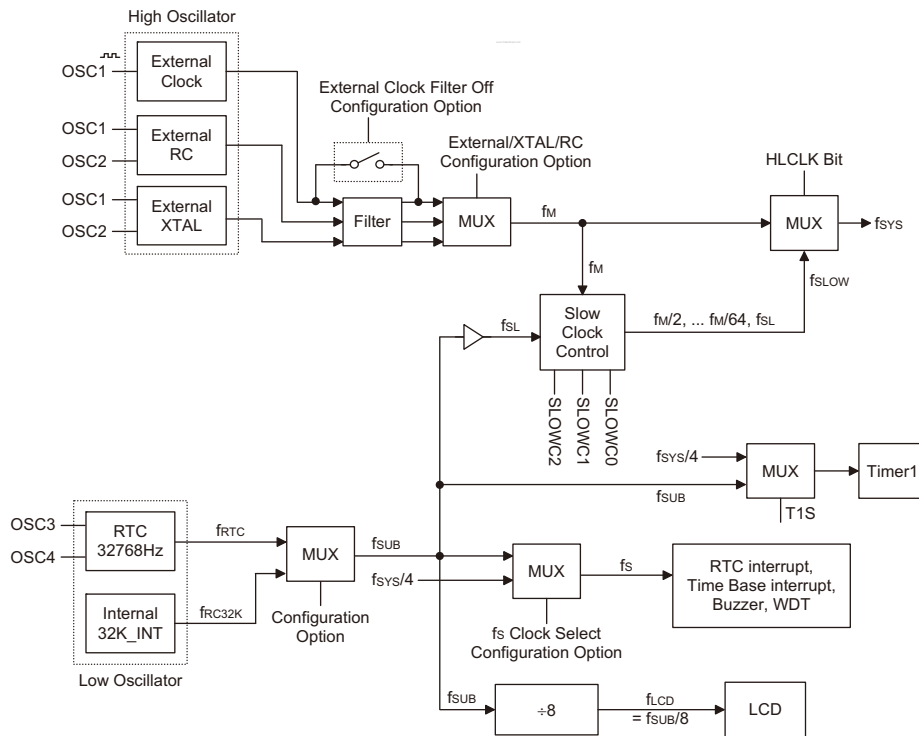
- Normal mode: f_M on, f_{SLOW} on, f_{SYS}=f_M, CPU on, f_S on, f_{LCD} on/off (using the LCDEN bit), f_{WDT} on/off (using a configuration option or WDT control register).
- Slow mode0: f_M off, f_{SLOW}=32K_INT oscillator or RTC oscillator, f_{SYS}=f_{SLOW}, CPU on, f_S on, f_{LCD} on/off (using the LCDEN bit), f_{WDT} on/off (using a configuration option or WDT control register).
- Slow mode1: f_M on, f_{SLOW}=f_M/2~f_M/64, f_{SYS}=f_{SLOW}, CPU on, f_S on, f_{LCD} on/off (using the LCDEN bit), f_{WDT} on/off (using a configuration option or WDT control register).
- Idle mode: f_M, f_{SLOW}, f_{SYS} off, CPU off; f_{SUB} on, f_S on/off (by selecting f_{SUB} or f_{SYS}/4), f_{LCD} on/off (using the LCDEN bit), f_{WDT} on/off (using a configuration option or WDT control register).
- Sleep mode: f_M, f_{SLOW}, f_{SYS}, f_S, f_{LCD} off, CPU off; f_{SUB}, f_{WDT} on/off (using a configuration option or WDT control register).



* Depending on WDT enable/disable condition.

Either fRTC or frc32k be on.

Dual Clock Mode Operation



Dual Clock Mode Structure

Power Down Mode and Wake-up

Power Down Mode

All of the Holtek microcontrollers have the ability to enter a Power Down 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 re-

quired 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.

Low Voltage Detector – LVD

This Low Voltage Detect internal function provides a means for the user to monitor when the power supply voltage falls below a certain fixed level as specified in the DC characteristics. Bits 3 and 5 of the RTCC register are used to control the overall function of the LVD. Bit 3 is the enable/disable control bit and is known as LVDC, when set low the overall function of the LVD will be disabled. Bit 5 is the LVD detector output bit and is known as LVDO. Under normal operation, and when the power supply voltage is above the specified VLVD value in the DC characteristic section, the LVDO bit will remain at a zero value. If the power supply voltage should fall below this VLVD value then the LVDO bit will change to a high value indicating a low voltage condition. Note that the LVDO bit is a read-only bit. By polling the LVDO bit in the RTCC register, the application program can therefore determine the presence of a low voltage condition.

After power-on, or after a reset, the LVD will be switched off by clearing the LVDC bit in the RTCC register to zero. Note that if the LVD is enabled there will be some power consumption associated with its internal circuitry, however, by clearing the LVDC bit to zero the power can be minimised. It is important not to confuse the LVD with the LVR function. In the LVR function an automatic reset will be generated by the microcontroller, whereas in the LVD function only the LVDO bit will be affected with no influence on other microcontroller functions.

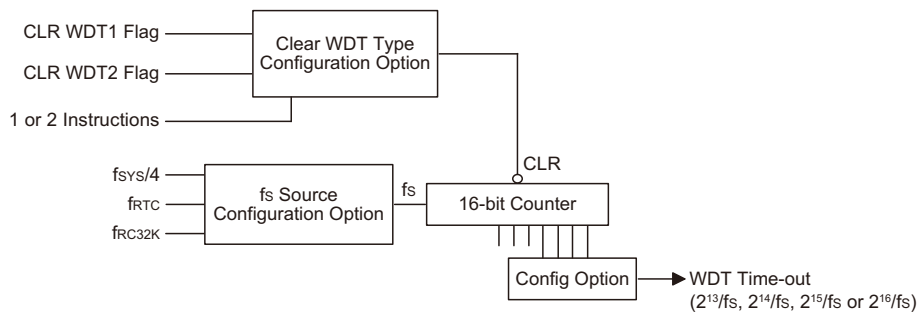
There are a range of voltage values, selected using a configuration option, which can be chosen to activate the LVD.

Watchdog Timer

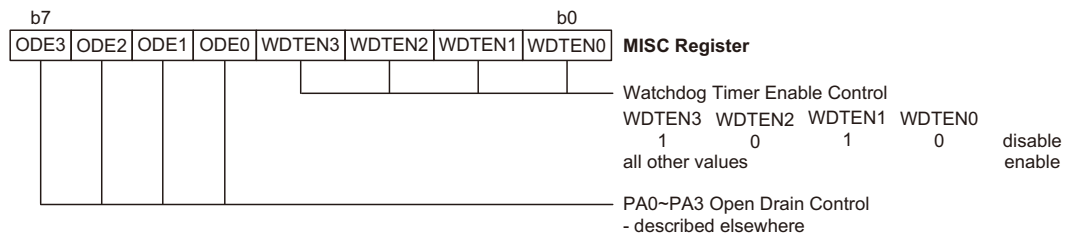
The Watchdog Timer is provided to prevent program malfunctions or sequences from jumping to unknown locations, due to certain uncontrollable external events such as electrical noise. It operates by providing a device reset when the Watchdog Timer counter overflows. The Watchdog Timer clock source is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

Most of the Watchdog Timer options, such as enable/disable, Watchdog Timer clock source and clear instruction type are selected using configuration options. In addition to a configuration option to enable the Watchdog Timer, there are four bits, WDTEN3~WDTEN0, in the MISC register to offer an additional enable control of the Watchdog Timer. These bits must be set to a specific value of 1010 to disable the Watchdog Timer. Any other values for these bits will keep the Watchdog Timer enabled. After power on these bits will have the disabled value of 1010.

One of the WDT clock sources is the internal f_{SUB} , which can be sourced from either the 32K_INT internal oscillator or the RTC oscillator. The 32K_INT internal oscillator has an approximate period of 31.2 μ s 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 RTC oscillator is supplied by an external 32768Hz crystal. The other Watchdog Timer



Watchdog Timer



Watchdog Timer Software Control – MISC

clock source option is the $f_{SYS}/4$ clock. Whether the Watchdog Timer clock source is its own internal 32K_INT, the RTC oscillator or $f_{SYS}/4$, it is divided by $2^{13}\sim 2^{16}$, using configuration option to obtain the required Watchdog Timer time-out period. The max time out period is when the 2^{16} option is selected. This time-out period may vary with temperature, VDD and process variations. 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.

If the $f_{SYS}/4$ clock is used as the Watchdog Timer clock source, it should be noted that when the system enters the Power Down Mode, then the instruction clock is stopped and the Watchdog Timer will lose its protecting purposes. For systems that operate in noisy environments, using the 32K_INT RC oscillator is strongly recommended.

Under normal program operation, a Watchdog Timer 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 Watchdog Timer 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 Watchdog Timer. 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 Watchdog Timer. Note that for this second option, if "CLR WDT1" is used to clear the Watchdog Timer, successive executions of this instruction will have no effect, only the execution of a "CLR WDT2" instruction will clear the Watchdog Timer. Similarly after the "CLR WDT2" instruction has been executed, only a successive "CLR WDT1" instruction can clear the Watchdog Timer.

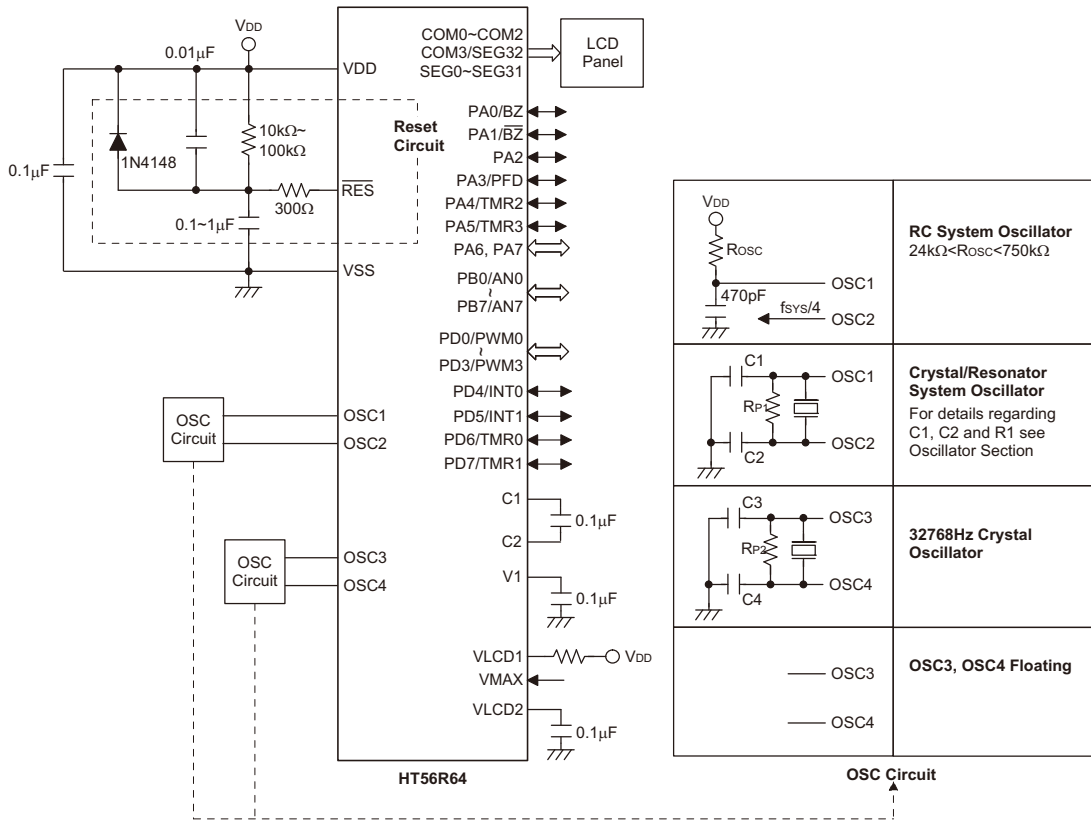
Configuration Options

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 |
|---------------------------|--|
| Oscillator Options | |
| 1 | Oscillator type selection: External Crystal Oscillator External RC Oscillator Externally supplied clock - internal filter on Externally supplied clock - internal filter off |
| 2 | f_{SUB} clock selection: RTC or 32K_INT |
| 3 | f_S clock selection: f_{SUB} or $f_{SYS}/4$ |
| 4 | XTAL mode selection: 455KHz or 1M~12MHz |
| 5 | 32768 XTAL: enable or disable |
| PFD Options | |
| 6 | PA3: normal I/O or PFD output |
| 7 | PFD clock selection: Timer/Event Counter 0 or Timer/Event Counter 1 |
| Buzzer Options | |
| 8 | PA0/PA1: normal I/O or $\overline{BZ/BZ}$ or PA0=BZ and PA1 as normal I/O |
| 9 | Buzzer frequency: $f_S/2^2, f_S/2^3, f_S/2^4, f_S/2^5, f_S/2^6, f_S/2^7, f_S/2^8, f_S/2^9$ |

| No. | Options |
|--|--|
| Time Base Options | |
| 10 | Time base time-out period: $2^{12}/f_S$, $2^{13}/f_S$, $2^{14}/f_S$, $2^{15}/f_S$, |
| LCD Options | |
| 11 | LCD type: R or C |
| Watchdog Options | |
| 12 | Watchdog Timer function: enable or disable |
| 13 | CLRWDT instructions: 1 or 2 instructions |
| 14 | WDT time-out period: $2^{12}/f_S \sim 2^{13}/f_S$, $2^{13}/f_S \sim 2^{14}/f_S$, $2^{14}/f_S \sim 2^{15}/f_S$, $2^{15}/f_S \sim 2^{16}/f_S$ |
| LVD/LVR Options | |
| 15 | LVD function: enable or disable |
| 16 | LVR function: enable or disable |
| 17 | LVR/LVD voltage: 2.1V/2.2V or 3.15V/3.3V or 4.2V/4.4V |
| SPI | |
| 18 | SPI pin enable/disable |
| 19 | SPI_CPOL : clock polarity is rising or falling edge |
| 20 | SPI_WCOL : enable/disable |
| 21 | SPI_CSEN : enable/disable, used to enable/disable (1/0) software CSEN function |
| I²C | |
| 22 | I ² C pin enable (if SPI & I2C both function pin enabled. SPI pin has higher priority) |
| PCLK function | |
| 23 | Peripheral Clock Output - PCLK or Segment pin - SEG14 |
| PINTB function | |
| 24 | External peripheral interrupt or Segment function |
| Timer/Event Counter and External Interrupt pins filter function | |
| 25 | Interrupt and Timer/Event Counter input pins internal filter On/Off control – applies to all pins |
| Lock Options | |
| 26 | Lock All |
| 27 | Partial Lock |

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 0.5 μ s and branch or call instructions would be implemented within 1 μ s. 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 en-

sure correct handling of carry and borrow data when results exceed 255 for addition and less than 0 for subtraction. The increment and decrement instructions INC, INCA, DEC and DECA provide a simple means of increasing or decreasing by a value of one of the values in the destination specified.

Logical and Rotate Operations

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

Branches and Control Transfer

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

Bit Operations

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

Table Read Operations

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

Other Operations

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

Instruction Set Summary

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

Table conventions:

x: Bits immediate data

m: Data Memory address

A: Accumulator

i: 0-7 number of bits

addr: Program memory address

| Mnemonic | Description | Cycles | Flag Affected |
|----------------------------------|---|-------------------|---------------|
| Arithmetic | | | |
| ADD A,[m] | Add Data Memory to ACC | 1 | Z, C, AC, OV |
| ADDM A,[m] | Add ACC to Data Memory | 1 ^{Note} | Z, C, AC, OV |
| ADD A,x | Add immediate data to ACC | 1 | Z, C, AC, OV |
| ADC A,[m] | Add Data Memory to ACC with Carry | 1 | Z, C, AC, OV |
| ADCM A,[m] | Add ACC to Data memory with Carry | 1 ^{Note} | Z, C, AC, OV |
| SUB A,x | Subtract immediate data from the ACC | 1 | Z, C, AC, OV |
| SUB A,[m] | Subtract Data Memory from ACC | 1 | Z, C, AC, OV |
| SUBM A,[m] | Subtract Data Memory from ACC with result in Data Memory | 1 ^{Note} | Z, C, AC, OV |
| SBC A,[m] | Subtract Data Memory from ACC with Carry | 1 | Z, C, AC, OV |
| SBCM A,[m] | Subtract Data Memory from ACC with Carry, result in Data Memory | 1 ^{Note} | Z, C, AC, OV |
| DAA [m] | Decimal adjust ACC for Addition with result in Data Memory | 1 ^{Note} | C |
| Logic Operation | | | |
| AND A,[m] | Logical AND Data Memory to ACC | 1 | Z |
| OR A,[m] | Logical OR Data Memory to ACC | 1 | Z |
| XOR A,[m] | Logical XOR Data Memory to ACC | 1 | Z |
| ANDM A,[m] | Logical AND ACC to Data Memory | 1 ^{Note} | Z |
| ORM A,[m] | Logical OR ACC to Data Memory | 1 ^{Note} | Z |
| XORM A,[m] | Logical XOR ACC to Data Memory | 1 ^{Note} | Z |
| AND A,x | Logical AND immediate Data to ACC | 1 | Z |
| OR A,x | Logical OR immediate Data to ACC | 1 | Z |
| XOR A,x | Logical XOR immediate Data to ACC | 1 | Z |
| CPL [m] | Complement Data Memory | 1 ^{Note} | Z |
| CPLA [m] | Complement Data Memory with result in ACC | 1 | Z |
| Increment & Decrement | | | |
| INCA [m] | Increment Data Memory with result in ACC | 1 | Z |
| INC [m] | Increment Data Memory | 1 ^{Note} | Z |
| DECA [m] | Decrement Data Memory with result in ACC | 1 | Z |
| DEC [m] | Decrement Data Memory | 1 ^{Note} | Z |

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

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

Instruction Definition

| | |
|-------------------|--|
| ADC A,[m] | Add Data Memory to ACC with Carry |
| Description | The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the Accumulator. |
| Operation | $ACC \leftarrow ACC + [m] + C$ |
| 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 | $[m] \leftarrow ACC + [m] + C$ |
| 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 | $ACC \leftarrow ACC + [m]$ |
| 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 | $ACC \leftarrow ACC + x$ |
| 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 | $[m] \leftarrow ACC + [m]$ |
| 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 | $ACC \leftarrow ACC \text{ "AND" } [m]$ |
| 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 | $ACC \leftarrow ACC \text{ "AND" } x$ |
| 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 | $[m] \leftarrow ACC \text{ "AND" } [m]$ |
| 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 \leftarrow Program Counter + 1 Program Counter \leftarrow addr |
| Affected flag(s) | None |
| CLR [m] | Clear Data Memory |
| Description | Each bit of the specified Data Memory is cleared to 0. |
| Operation | [m] \leftarrow 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 \leftarrow 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 \leftarrow 0 PDF \leftarrow 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. Repeatedly executing this instruction without alternately executing CLR WDT2 will have no effect. |
| Operation | WDT cleared TO \leftarrow 0 PDF \leftarrow 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. Repeatedly executing this instruction without alternately executing CLR WDT1 will have no effect. |
| Operation | WDT cleared TO \leftarrow 0 PDF \leftarrow 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 | $[m] \leftarrow [m] + 1$ |
| 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 | $ACC \leftarrow [m] + 1$ |
| 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 | $Program\ Counter \leftarrow addr$ |
| 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 | $ACC \leftarrow [m]$ |
| Affected flag(s) | None |
| MOV A,x | Move immediate data to ACC |
| Description | The immediate data specified is loaded into the Accumulator. |
| Operation | $ACC \leftarrow x$ |
| 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 | $[m] \leftarrow ACC$ |
| 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 | $ACC \leftarrow ACC \text{ "OR" } [m]$ |
| 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 | $[m].(i+1) \leftarrow [m].i; (i = 0\sim 6)$ $[m].0 \leftarrow C$ $C \leftarrow [m].7$ |
| 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 | $ACC.(i+1) \leftarrow [m].i; (i = 0\sim 6)$ $ACC.0 \leftarrow C$ $C \leftarrow [m].7$ |
| 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 | $[m].i \leftarrow [m].(i+1); (i = 0\sim 6)$ $[m].7 \leftarrow [m].0$ |
| 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 | $ACC.i \leftarrow [m].(i+1); (i = 0\sim 6)$ $ACC.7 \leftarrow [m].0$ |
| 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 | $[m].i \leftarrow [m].(i+1); (i = 0\sim 6)$ $[m].7 \leftarrow C$ $C \leftarrow [m].0$ |
| 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 | $ACC.i \leftarrow [m].(i+1); (i = 0\sim 6)$ $ACC.7 \leftarrow C$ $C \leftarrow [m].0$ |
| 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 | $ACC \leftarrow ACC - [m] - \bar{C}$ |
| 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 | $[m] \leftarrow ACC - [m] - \bar{C}$ |
| 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 |

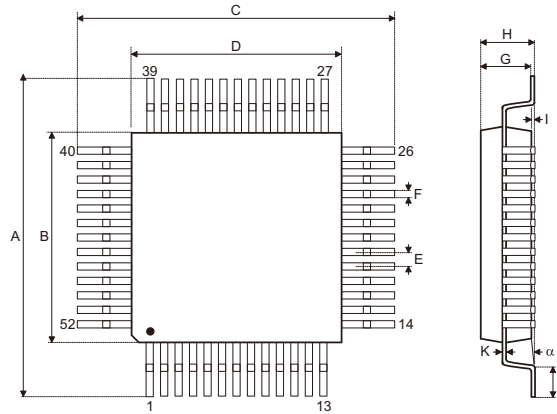
| | |
|-------------------|--|
| SIZ [m] | Skip if increment Data Memory is 0 |
| Description | The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | $[m] \leftarrow [m] + 1$ 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 | $ACC \leftarrow [m] + 1$ 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 | $ACC \leftarrow ACC - [m]$ |
| 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 | $[m] \leftarrow ACC - [m]$ |
| 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 | $ACC \leftarrow ACC - x$ |
| 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 | $[m].3 \sim [m].0 \leftrightarrow [m].7 \sim [m].4$ |
| 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 | $ACC.3 \sim ACC.0 \leftarrow [m].7 \sim [m].4$ $ACC.7 \sim ACC.4 \leftarrow [m].3 \sim [m].0$ |
| 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 | $ACC \leftarrow [m]$ 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 | $[m] \leftarrow$ program code (low byte) $TBLH \leftarrow$ 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 | $[m] \leftarrow$ program code (low byte) $TBLH \leftarrow$ program code (high byte) |
| Affected flag(s) | None |

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

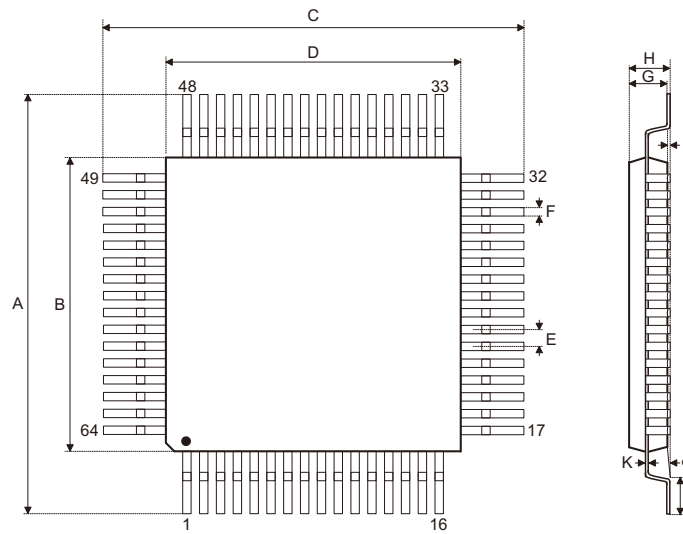
Package Information

Note that the package information provided here is for consultation purposes only. As this information may be updated at regular intervals users are reminded to consult the Holtek website (<http://www.holtek.com.tw/english/literature/package.pdf>) for the latest version of the package information.

52-pin QFP (14mm×14mm) Outline Dimensions


| Symbol | Dimensions in inch | | |
|----------|--------------------|-------|-------|
| | Min. | Nom. | Max. |
| A | 0.681 | — | 0.689 |
| B | 0.547 | — | 0.555 |
| C | 0.681 | — | 0.689 |
| D | 0.547 | — | 0.555 |
| E | — | 0.039 | — |
| F | — | 0.016 | — |
| G | 0.098 | — | 0.122 |
| H | — | — | 0.134 |
| I | — | 0.004 | — |
| J | 0.029 | — | 0.041 |
| K | 0.004 | — | 0.008 |
| α | 0° | — | 7° |

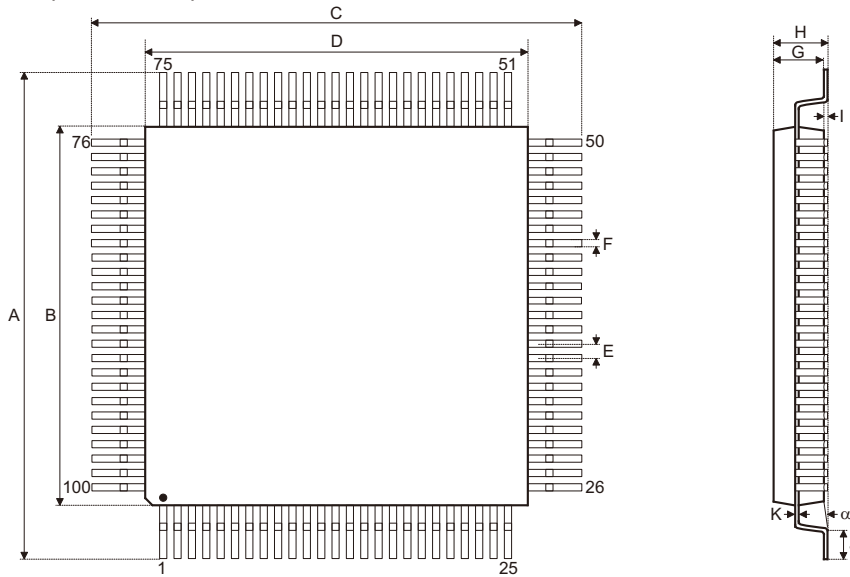
| Symbol | Dimensions in mm | | |
|----------|------------------|------|-------|
| | Min. | Nom. | Max. |
| A | 17.30 | — | 17.50 |
| B | 13.90 | — | 14.10 |
| C | 17.30 | — | 17.50 |
| D | 13.90 | — | 14.10 |
| E | — | 1.00 | — |
| F | — | 0.40 | — |
| G | 2.50 | — | 3.10 |
| H | — | — | 3.40 |
| I | — | 0.10 | — |
| J | 0.73 | — | 1.03 |
| K | 0.10 | — | 0.20 |
| α | 0° | — | 7° |

64-pin LQFP (7mm×7mm) Outline Dimensions


| Symbol | Dimensions in inch | | |
|----------|--------------------|-------|-------|
| | Min. | Nom. | Max. |
| A | 0.350 | — | 0.358 |
| B | 0.272 | — | 0.280 |
| C | 0.350 | — | 0.358 |
| D | 0.272 | — | 0.280 |
| E | — | 0.016 | — |
| F | 0.005 | — | 0.009 |
| G | 0.053 | — | 0.057 |
| H | — | — | 0.063 |
| I | 0.002 | — | 0.006 |
| J | 0.018 | — | 0.030 |
| K | 0.004 | — | 0.008 |
| α | 0° | — | 7° |

| Symbol | Dimensions in mm | | |
|----------|------------------|------|------|
| | Min. | Nom. | Max. |
| A | 8.90 | — | 9.10 |
| B | 6.90 | — | 7.10 |
| C | 8.90 | — | 9.10 |
| D | 6.90 | — | 7.10 |
| E | — | 0.40 | — |
| F | 0.13 | — | 0.23 |
| G | 1.35 | — | 1.45 |
| H | — | — | 1.60 |
| I | 0.05 | — | 0.15 |
| J | 0.45 | — | 0.75 |
| K | 0.09 | — | 0.20 |
| α | 0° | — | 7° |

100-pin LQFP (14mm×14mm) Outline Dimensions



| Symbol | Dimensions in inch | | |
|--------|--------------------|-------|-------|
| | Min. | Nom. | Max. |
| A | 0.626 | — | 0.634 |
| B | 0.547 | — | 0.555 |
| C | 0.626 | — | 0.634 |
| D | 0.547 | — | 0.555 |
| E | — | 0.020 | — |
| F | — | 0.008 | — |
| G | 0.053 | — | 0.057 |
| H | — | — | 0.063 |
| I | — | 0.004 | — |
| J | 0.018 | — | 0.030 |
| K | 0.004 | — | 0.008 |
| α | 0° | — | 7° |

| Symbol | Dimensions in mm | | |
|--------|------------------|------|-------|
| | Min. | Nom. | Max. |
| A | 15.90 | — | 16.10 |
| B | 13.90 | — | 14.10 |
| C | 15.90 | — | 16.10 |
| D | 13.90 | — | 14.10 |
| E | — | 0.50 | — |
| F | — | 0.20 | — |
| G | 1.35 | — | 1.45 |
| H | — | — | 1.60 |
| I | — | 0.10 | — |
| J | 0.45 | — | 0.75 |
| K | 0.10 | — | 0.20 |
| α | 0° | — | 7° |

Holtek Semiconductor Inc. (Headquarters)

No.3, Creation Rd. II, Science Park, Hsinchu, Taiwan
Tel: 886-3-563-1999
Fax: 886-3-563-1189
<http://www.holtek.com.tw>

Holtek Semiconductor Inc. (Taipei Sales Office)

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

Holtek Semiconductor Inc. (Shenzhen Sales Office)

5F, Unit A, Productivity Building, No.5 Gaoxin M 2nd Road, Nanshan District, Shenzhen, China 518057
Tel: 86-755-8616-9908, 86-755-8616-9308
Fax: 86-755-8616-9722

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>

Copyright © 2012 by HOLTEK SEMICONDUCTOR INC.

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