TinyPower™ A/D Type
e-Banking ASSP OTP MCU with LCD

# HT45R4U

Revision: V1.10 Date: March 12, 2015

www.holtek.com

# Table of Contents

## Features

- Operating voltage:
  $f_{SYS}$=32768Hz: 2.2V~3.6V
  $f_{SYS}$=4MHz: 2.2V~3.6V
- OTP Program Memory: 16K×16
- RAM Data Memory: 1152×8
- Up to 34 bidirectional I/O lines
- TinyPower technology for low power operation
- Two external pin-shared interrupts lines
- Multiple programmable Timer/Event Counters
  with overflow interrupt and 7-stage prescaler
- External Crystal, Internal RC and 32.768kHz oscillators
- Externally supplied system clock option
- Watchdog Timer function
- PFD/Buzzer for audio frequency generation
- LCD driver function
- 4 operating modes: normal, slow, idle and sleep
- 8-channel 12-bit resolution A/D converter
- Low voltage detect function
- Bit manipulation instruction
- Table read instructions
- 63 powerful instructions
- Up to 1μs instruction cycle with 4MHz system clock at $V_{DD}$=3V
- 8 levels subroutine nesting
- All instructions executed in one or two machine cycles
- Power down and wake-up functions to reduce power consumption
- Package Type: 64-pin LQFP

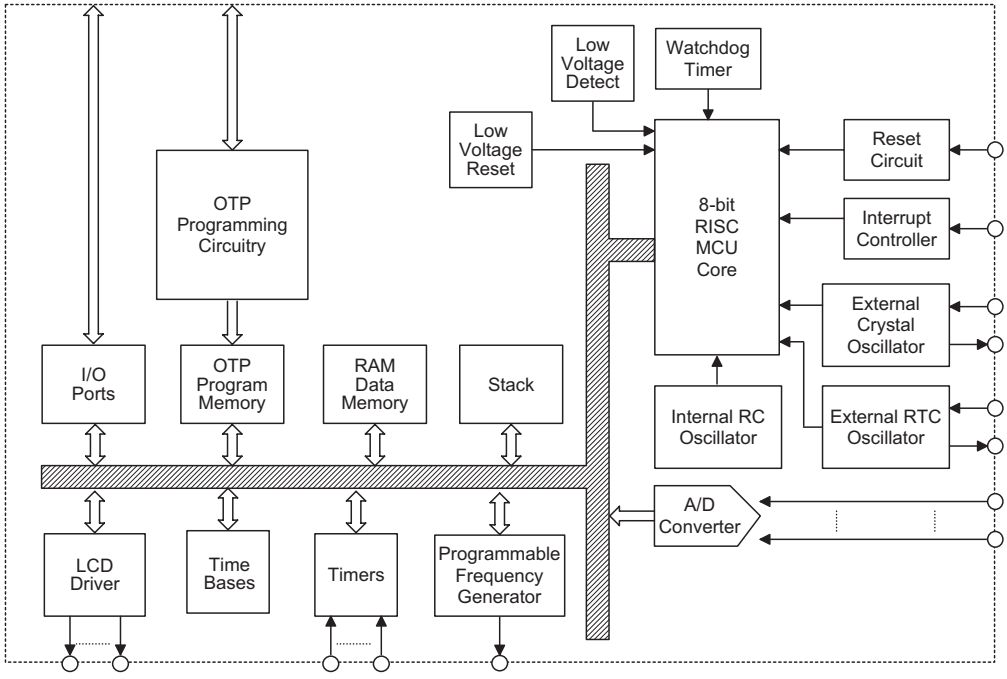
## General Description

The TinyPower™ A/D Type with LCD 8-bit high performance RISC architecture microcontroller is specifically, designed 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 and an LCD driver.

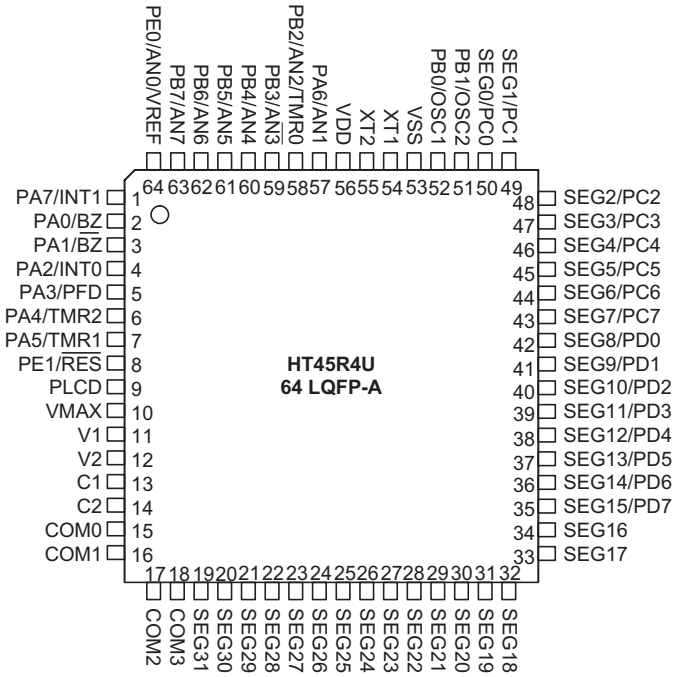
The benefits of integrated A/D and LCD functions, in addition to low power consumption, high performance, I/O flexibility and low-cost, provides the device with the versatility for a wide range of products in the home appliance and industrial application areas. Some of these products could include electronic 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

The following table depicts the pins common to all devices.

| Pin/Pad Name | I/O | Options | Description |
|---|---|---|---|
| PA0/BZ<br>PA1/BZ<br>PA2/INT0<br>PA3/PFD<br>PA4/TMR2<br>PA5/TMR1<br>PA6/AN1<br>PA7/INT1 | I/O | — | Bidirectional 8-bit input/output port. Each individual bit on this port can be configured as a wake-up input by the PAWU register control bit. Software instructions determine if the pin is a CMOS output or Schmitt trigger input. A pull-high resistor can be connected to each pin determined by the PAPU register.<br>PA6 is pin-shared with the A/D input pin. The A/D inputs are selected via software instructions. Once an I/O line is selected as an A/D input, the I/O function and pull-high resistor are disabled automatically.<br>The BZ, BZ, INT0, PFD, TMR2, TMR1 and INT1 are pin-shared with PA0~PA 5 and PA7 respectively. |
| PB0/OSC1<br>PB1/OSC2<br>PB2/AN2/TMR0<br>PB3/AN3~PB7/<br>AN7 | I/O | HXT or<br>HIRC or<br>EC | 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 by the PBPU register.<br>The OSC1 and OSC2 pins are connected to an external crystal or used as I/O pins, determined by configuration options, for the internal system clock. The internal system can also come from the internal high speed RC oscillator, HIRC, which is selected by configuration options. When the HIRC is selected as the system oscillator, the OSC1 and OSC2 pins can be used as normal I/O pins. The abbreviation EC stands for External Clock mode. In the EC mode, an external clock source is provided on OSC1 as the system clock.<br>The AN2/TMR0 and AN3~AN7 pins are pin-shared with PB2~PB7 respectively and controlled by software control registers |
| PC0/SEG0~PC7/<br>SEG7 | 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 determined by the PCPU register.<br>The SEG0~SEG7 pins are LCD driver outputs for LCD panel segments and are pin-shared with PC0~PC7 pins respectively. |

| Pin/Pad Name | I/O | Options | Description |
|---|---|---|---|
| PD0/SEG8~PD7/ SEG15 | 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 determined by the PDPU register.<br>The SEG8~SEG15 pins are LCD driver outputs for LCD panel segments and are pin-shared with PD0~PD7 pins respectively. |
| PE0/AN0/VREF<br>PE1/$\overline{RES}$ | I/O | $\overline{RES}$ | Bidirectional 2-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 PE0 pin determined by the relevant control bit in the PEPU register.<br>The AN0 pin is the A/D converter analog input pin and pin-shared with PE0. The A/D input channel AN0 is selected via software instructions. Once an I/O line is selected as an A/D input, the I/O function together with the pull-high resistor is disabled automatically.<br>The VREF pin is the A/D Converter reference voltage input pin. The "VREFS" bit in the ACSR register is used to select either VREF or AVDD as the ADC reference voltage.<br>The $\overline{RES}$ pin is the Schmitt Trigger reset input pin, active low, and pin-shared with PE1 whose function is determined by a configuration option. When PE1 is configured as an I/O pin, software instructions determine if this pin is an open drain output or a Schmitt Trigger input without pull-high resistor. |
| SEG16~SEG31 | AO | — | The SEG16~SEG31 pins are LCD driver outputs for LCD panel segments. |
| COM0~COM3 | AO | — | The COM0~COM3 pins are LCD driver outputs for LCD panel commons. |
| XT1<br>XT2 | AIO | LXT | The XT1 and XT2 are connected to a 32.768kHz crystal oscillator to form a clock source for $f_{SUB}$. |
| VDD | P | — | Positive power supply |
| VSS | P | — | Negative power supply, ground |
| VMAX | P | — | Device maximum voltage, connected to VDD, PLCD or V1. |
| PLCD | AIO | — | LCD voltage pump |
| C1 | AIO | — | LCD voltage pump |
| C2 | AIO | — | LCD voltage pump |
| V1 | AIO | — | LCD voltage pump |
| V2 | AIO | — | LCD voltage pump |

Legend: I/O: Input/Output type

AO: Analog output

AIO: Analog iput/output

P: Power pin

LXT: High frequency crystal oscillator

## Absolute Maximum Ratings

Supply Voltage ..... $V_{SS}$−0.3V to $V_{SS}$+6.0V
Input Voltage ..... $V_{SS}$−0.3V to $V_{DD}$+0.3V
Storage Temperature..... -50˚C to 125˚C
Operating Temperature..... -40˚C to 85˚C
$I_{OL}$ Total ..... 80mA
$I_{OH}$ Total..... -80mA
Total Power Dissipation ..... 500mW

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

## D.C. Characteristics

Ta= 25˚C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $V_{DD}$ | Operating Voltage | — | $f_{SYS}$=4MHz | 2.2 | — | 3.6 | V |
| $I_{DD1}$ | Operating Current (Crystal OSC, HIRC OSC) | 3V | No load, $f_{SYS}=f_M$=4MHz ADC off | — | 440 | 660 | μA |
| $I_{DD2}$ | Operating Current ($f_{SYS}$=32768Hz, See Note 1) | 3V | No load, ADC off, LCD on (note 2), C type, 1/3 bias, $V_{PLCD}$=3V | — | 5 | 10 | μA |
| $I_{DD3}$ | Operating Current ($f_{SYS}$=32768Hz, See Note 1) | 3V | No load, ADC off, LCD off | — | 4 | 8 | μA |
| $I_{STB}$ | Standby Current (Sleep) ($f_{SYS}$, $f_{SUB}$=32768Hz, See Note 1) | 3V | No load, System HALT, WDT on, LCD off | — | 0.9 | 1.5 | μA |
| $V_{IL1}$ | Input Low Voltage (I/O), TMRn and INT | 3V | — | 0 | — | 0.2$V_{DD}$ | V |
| $V_{IH1}$ | Input High Voltage (I/O), TMRn and INT | 3V | — | 0.8$V_{DD}$ | — | $V_{DD}$ | V |
| $V_{IL2}$ | Input Low Voltage ($\overline{RES}$) | — | — | 0 | — | 0.4$V_{DD}$ | V |
| $V_{IH2}$ | Input High Voltage ($\overline{RES}$) | — | — | 0.9$V_{DD}$ | — | $V_{DD}$ | V |
| $I_{OL1}$ | I/O Port Sink Current | 3V | $V_{OL}$=0.1$V_{DD}$ | 4 | 8 | — | mA |
| $I_{OH1}$ | I/O Port Source Current | 3V | $V_{OH}$=0.9$V_{DD}$ | -2 | -4 | — | mA |
| $I_{OL2}$ | PE1/$\overline{RES}$ Sink Current | 3V | $V_{OL}$=0.1$V_{DD}$ | 1 | 1.8 | — | mA |
| $R_{PH}$ | Pull-high Resistance (I/O) | 3V | — | 20 | 60 | 100 | kΩ |
| $V_{REF}$ | A/D Input Reference Voltage Range | — | $AV_{DD}$=3V | 1.6 | — | $AV_{DD}$ +0.1 | V |
| DNL | ADC Differential Non-Linearity | — | $AV_{DD}$=3V, $V_{REF}$=$AV_{DD}$, $t_{AD}$=0.5μs | -2 | — | 2 | LSB |
| INL | ADC Integral Non-Linearity | — | $AV_{DD}$=3V, $V_{REF}$=$AV_{DD}$, $t_{AD}$=0.5μs | -4 | — | 4 | LSB |
| $I_{ADC}$ | Additional Power Consumption if A/D Converter is Used | 3V | — | — | 0.5 | 1 | mA |

Note: 1. 32768Hz is slow start mode (TBC.4=1) in D.C. current measurement.
2. LCD waveform is in Type A condition.
3. $f_{SUB}$ is internal clock for Buzzer, Time base interrupts and WDT.
4. Timer0/1/2 off. Timer filter disable in all test condition.

## A.C. Characteristics

Ta= 25°C

| Symbol | Parameter | $V_{DD}$ | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| $f_{SYS1}$ | System Clock (Crystal OSC, HIRC OSC) | 2.2V~3.6V | — | 400 | — | 4000 | kHz |
| $f_{SYS2}$ | System Clock (LXT) | 2.2V~3.6V | — | — | 32768 | — | Hz |
| $f_{LXT}$ | 32768Hz Crystal Frequency | — | — | — | 32768 | — | Hz |
| $f_{TIMER}$ | Timer I/P Frequency (TMR0~2) | 2.2V~3.6V | — | 0 | — | 4000 | kHz |
| $t_{TMR}$ | TMRn Input Pin Minimum Pulse Width | — | — | 0.3 | — | — | μs |
| $t_{RES}$ | External Reset Minimum Low Pulse Width | — | — | 10 | — | — | μs |
| $t_{INT}$ | Interrupt Pulse Width | — | — | 10 | — | — | μs |
| $t_{AD}$ | A/D Clock Period | — | — | 0.5 | — | — | μs |
| $t_{ADC}$ | A/D Conversion Time | — | — | — | 16 | — | $t_{AD}$ |
| $t_{ON2ST}$ | ADC on to ADC Start | 2.2V~3.6V | — | 2000 | — | — | ns |
| $t_{SST}$ | System Start-up Timer Period (Wake-up from HALT, $f_{SYS}$ off at HALT state, Slow Mode → Normal Mode, Normal Mode → Slow Mode) | — | $f_{SYS}$ =HXT (Slow Mode → Normal Mode(HXT)) | 1024 | — | — | $t_{SYS}$ |
| | | — | $f_{SYS}$ =LXT (Normal Mode → Slow Mode(LXT)) | — | 1024 | — | $t_{SYS}$ |
| | | — | $f_{SYS}$= HXT or LXT (Wake-up from HALT, $f_{SYS}$ off at HALT state) | 1024 | — | — | $t_{SYS}$ |
| | | — | $f_{SYS}$= HIRC | 16 | — | — | $t_{SYS}$ |
| | | — | $f_{SYS}$= EC | 2 | — | — | $t_{SYS}$ |
| | System Start-up Timer Period (Wake-up from HALT, $f_{SYS}$ on at HALT State) | — | — | 2 | — | — | $t_{SYS}$ |
| $t_{RSTD}$ | System Reset Delay Time (Power On Reset, LVR Reset, LVR S/W Reset(LVRC), WDT S/W Reset(WDTC)) | — | — | 25 | 50 | 100 | ms |
| | System Reset Delay Time (Any Reset except Power On Reset) (RES Reset, WDT Normal Reset) | — | — | 8.3 | 16.7 | 33.3 | ms |

Note: 1. $t_{SYS}$= 1/$f_{SYS1}$ or 1/$f_{SYS2}$

2. ADC conversion time($t_{AD}$)= n(bits ADC) + 4(sampling time), The conversion for each bit needs one ADC clock($t_{AD}$).

## HIRC Oscillator Voltage/Temperature vs. Frequency Curves

### Frequency Accurary (Trim 4MHz at $V_{DD}$=3V)

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Condition | | | | |
| $f_{HIRC}$ | System Clock (HIRC) | 3V | Ta=25°C | -2% | 4 | 2% | MHz |
| | | 2.9V~3.1V | Ta=0~70°C | -5% | 4 | 5% | MHz |
| | | 2.2V~3.6V | Ta=0~70°C | -7% | 4 | 7% | MHz |
| | | 2.2V~3.6V | Ta=-40~85°C | -10% | 4 | 10% | MHz |

## LVD/LVR Characteristics

Ta= 25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $V_{LVR}$ | Low Voltage Reset Voltage | — | LVR Enable, 2.1V option | -5%×Typ. | 2.1 | +5%×Typ. | V |
| $V_{LVD1}$ | Low Voltage Detector Voltage | — | LVDEN = 1, $V_{LVD}$ = 2.0V | -5%×Typ. | 2.0 | +5%×Typ. | V |
| $V_{LVD2}$ | | | LVDEN = 1, $V_{LVD}$= 2.2V | | 2.2 | | V |
| $V_{LVD3}$ | | | LVDEN = 1, $V_{LVD}$ = 2.4V | | 2.4 | | V |
| $V_{LVD4}$ | | | LVDEN = 1, $V_{LVD}$ = 2.7V | | 2.7 | | V |
| $V_{LVD5}$ | | | LVDEN = 1, $V_{LVD}$ = 3.0V | | 3.0 | | V |
| $V_{LVD6}$ | | | LVDEN = 1, $V_{LVD}$ = 3.3V | | 3.3 | | V |
| $V_{LVD7}$ | | | LVDEN = 1, $V_{LVD}$ = 3.6V | | 3.6 | | V |
| $V_{LVD8}$ | | | LVDEN = 1, $V_{LVD}$ = 4.0V | | 4.0 | | V |
| $I_{LVR}$ | Additional Power Consumption if LVR is Used | 3V | LVR disable → LVR enable | — | 30 | 45 | μA |
| $I_{LVD}$ | Additional Power Consumption if LVD is Used | 3V | LVD disable → LVD enable (LVR disable) | — | 40 | 60 | μA |
| | | 3V | LVD disable → LVD enable (LVR enable) | — | 30 | 45 | μA |
| $t_{LVR}$ | Low Voltage Width to Reset | — | — | 120 | 240 | 480 | μs |
| $t_{LVD}$ | Low Voltage Width to Interrupt | — | — | 20 | 45 | 90 | μs |

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

## LCD D.C. Characteristics

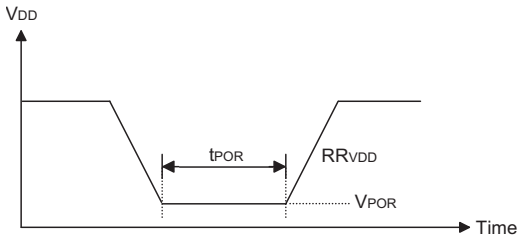
Ta= 25°C

| Sym. | Parameter | $V_{DD}$ | Condition | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| $I_{STB}$ | Standby Current (Idle) ($f_{SYS}$, $f_{WDT}$ off, *fs= $f_{SUB}$ = %32768Hz) | 3V | No load, system HALT, LCD on, WDT off, C type, $V_{PLCD}$= $V_{DD}$, 1/3 Bias | — | 1.3 | 3.0 | μA |
| $I_{OL\_LCD}$ | LCD Common and Segment Sink Current | 3V | $V_{OL}$=0.1$V_{PLCD}$ | 210 | 420 | — | μA |
| $I_{OH\_LCD}$ | LCD Common and Segment Source Current | 3V | $V_{OH}$=0.9$V_{PLCD}$ | -80 | -160 | — | μA |

## Power-on Reset Characteristics

Ta= 25°C

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



## System Architecture

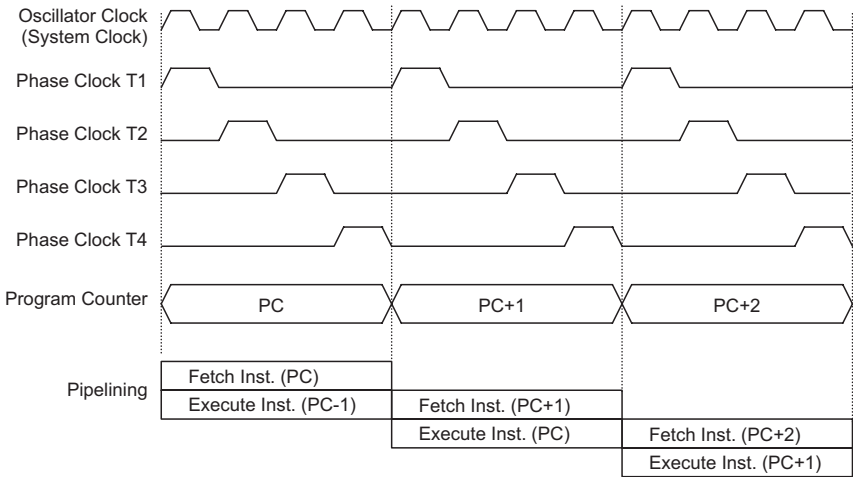
A key factor in the high-performance features of the Holtek range of microcontroller is attributed to their internal system architecture. The range of devices take advantage of the usual features found within RISC microcontroller providing increased speed of operation and enhanced performance. The pipelining 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 HIRC oscillator is subdivided into four internally generated non-overlapping clocks, T1~T4. The Program Counter is incremented at the beginning of the T1 clock during which time a new instruction is fetched. The remaining T2~T4 clocks carry out the decoding and execution functions. In this way, one T1~T4 clock cycle forms one instruction cycle. Although the fetching and execution of instructions takes place in consecutive instruction cycles, the pipelining structure of the microcontroller ensures that instructions are effectively executed in one instruction cycle. The exception to this are instructions where the contents of the Program Counter are changed, such as subroutine calls or jumps, in which case the instruction will take one more instruction cycle to execute.

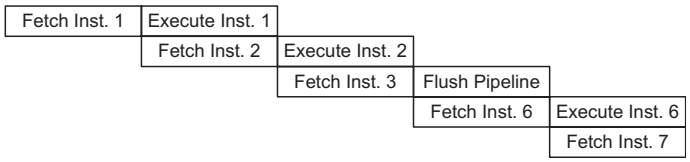
When the HIRC oscillator is used, the OSC1/OSC2 pins as I/O.

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




**System Clocking and Pipelining**

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




**Instruction Fetching**

## Program Counter– PC

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

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 | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | b13 | b12 | b11 | b10 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| Initial Reset | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| External Interrupt 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| External Interrupt 1 | 0 | 0 | 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 | 0 | 0 | 1 | 1 | 0 | 0 |
| Timer/Event Counter 1 Overflow | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| Timer/Event Counter 2 Overflow | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| Multi-Function Interrupt | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| Skip | Program Counter + 2 | | | | | | | | | | | | | |
| Loading PCL | PC13 | PC12 | PC11 | PC10 | PC9 | PC8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| Jump, Call Branch | BP.5 | #12 | #11 | #10 | #9 | #8 | #7 | #6 | #5 | #4 | #3 | #2 | #1 | #0 |
| Return from Subroutine | S13 | S12 | S11 | S10 | S9 | S8 | S7 | S6 | S5 | S4 | S3 | S2 | S1 | S0 |

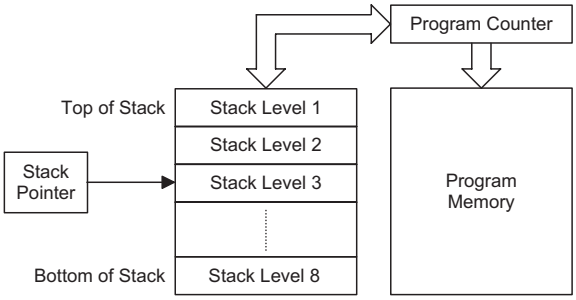
**Program Counter**

Note: PC13~PC8: Current Program Counter bits
@7~@0: PCL bits
#12~#0: Instruction code address bits
S13~S0: Stack register bits
BP.5: Bank pointer 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 multiple levels depending upon the device 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:

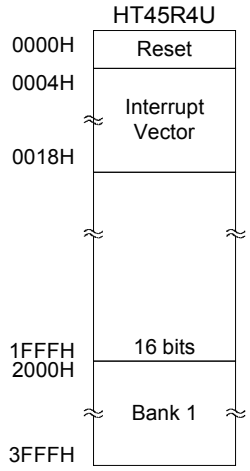
- Arithmetic operations: ADD, ADDM, ADC, ADCM, SUB, SUBM, SBC, SBCM, DAA
- Logic operations: AND, OR, XOR, ANDM, ORM, XORM, CPL, CPLA
- Rotation RRA, RR, RRCA, RRC, RLA, RL, RLCA, RLC
- Increment and Decrement INCA, INC, DECA, DEC
- Branch decision, JMP, SZ, SZA, SNZ, SIZ, SDZ, SIZA, SDZA, CALL, RET,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

## Program Memory

The Program Memory is the location where the user code or program is stored. For these 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.

### Structure

The Program Memory has a capacity of 16K×16 bits. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries. Table data, which can be setup in any location within the Program Memory, is addressed by a separate table pointer register.




**Program Memory Structure**

## Special Vectors

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

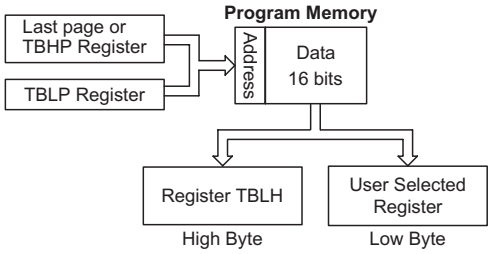
- Location 000H
  This vector is reserved for use by the device reset for program initialisation. After a device reset is initiated, the program will jump to this location and begin execution.
- Location 004H
  This vector is used by the external interrupt 0. If the external interrupt pin receives an 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 Timer/Event Counter 2. If a Timer/Event Counter 2 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 018H
  This internal vector is used by the Multi-function Interrupt. The Multi-function Interrupt vector is shared by several internal functions such as a Time Base overflow or an A/D converter conversion completion. The program will jump to this location and begin execution if the relevant interrupt is enabled and the stack is not full.

## 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 pair must first be setup by placing the address of the look up data to be retrieved in the table pointer register pair, TBLP and TBHP. This register pair defines the address of the look-up table.

After setting up the table pointer pair, the table data can be retrieved from the specific Program Memory page or last Program Memory page using the "TABRD[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.

The following diagram illustrates the addressing/data flow of the look-up table:




| Instruction | Table Location Bits | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | b13 | b12 | b11 | b10 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| TABRD[m] | PC13 | PC12 | PC11 | PC10 | PC9 | PC8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| TABRDL[m] | 1 | 1 | 1 | 1 | 1 | 1 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |

**Table Location**

Note: PC13~PC8:Table pointer TBHP bits
@7~@0: Table Pointer TBLP bits

## Table Program Example

The accompanying example shows how the table pointer and table data is defined and retrieved from the device. This example uses raw table data located in the last page which is stored there using the ORG statement. The value at this ORG statement is "3F00H" which refers to the start address of the last page within the 16K Program Memory of the HT45R4U 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 "3F06H" 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 "TABRD [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 "TABRD [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.

### Table Read Program Example

```
rombank 1 code1

ds .section 'data'
tempreg1 db ?          ; temporary register#1
tempreg2 db ?          ; temporary register#2
:
:
code0 .section  'code'
mov a,06h              ; initialise table pointer - note that this address is referenced
mov tblp,a             ; to the last page or the page that tbhp pointed
mov a,01fh             ; initialise high table pointer
mov tbhp,a             ; it is not necessary to set tbhp if executing tabrdl
:
:
tabrd  tempreg1        ; transfers value in table referenced by table pointer
                       ; to tempregl
                       ; data at prog.memory address 1F06H 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 1F05H transferred   to
                       ; tempreg2 and TBLH
                       ; in this example the data 1AH is transferred to
                       ; tempreg1 and data 0FH to tempreg2
                       ; the value 00H will be transferred to the high byte
                       ; register TBLH
     :
     :
code1 .section  'code'
org 1F00h              ; sets initial address of last page
dc 00Ah,00Bh,00Ch,00Dh,00Eh,00Fh,01Ah,01Bh
```

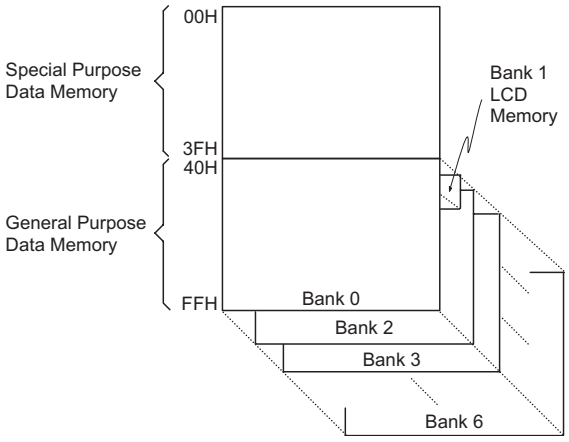
## Data Memory

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 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 different Data Memory banks is achieved by setting the Bank Pointer to the correct value.

### Structure

The Data Memory is subdivided into several banks, all of which are implemented in 8-bit wide RAM. The 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 Data Memory for all devices is the address ″00H″. Registers which are common to all microcontroller, such as ACC, PCL, etc., have the same Data Memory address. The LCD Memory is mapped into Bank 1. Banks 2 to 6 contain only General Purpose Data Memory for those devices with larger Data Memory capacities. As the Special Purpose Data Memory registers are mapped into all bank areas, they can subsequently be accessed from any bank location.




**Data Memory Structure**

| Data Memory | Capacity | Banks |
|---|---|---|
| Special Purpose | 64×8 | Common to Bank 0 ~ 6: 00H ~ 3FH |
| General Purpose | 1152×8 | 0: 40H ~ FFH<br>1: 40H ~ 5FH (for LCD only)<br>2: 40H ~ FFH<br>:<br>:<br>6: 40H ~ FFH |

**Data Memory Content**

### 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 programing for both reading and writing operations. By using the bit operation 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. For the device with larger Data Memory capacities, the General Purpose Data Memory, in addition to being located in Bank 0, is also stored in Banks 2 to Bank 6.

### 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″. The Special Function registers are mapped into all banks and can therefore be accessed from any bank location.

### Display Memory

The data to be displayed on the LCD display is stored in an area of fully accessible Data Memory. By writing to this area of RAM, the display output can be directly controlled by the application program. As this 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 Display Memory. The Display 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 Display 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.

| | | | |
|---|---|---|---|
| 00H | IAR0 | 20H | Unused |
| 01H | MP0 | 21H | Unused |
| 02H | IAR1 | 22H | ADCR0 |
| 03H | MP1 | 23H | ADCR1 |
| 04H | BP | 24H | ADRL |
| 05H | ACC | 25H | ADRH |
| 06H | PCL | 26H | ACER |
| 07H | TBLP | 27H | SMOD |
| 08H | TBLH | 28H | SMOD1 |
| 09H | TBHP | 29H | PAWU |
| 0AH | STATUS | 2AH | PAPU |
| 0BH | INTC0 | 2BH | PBPU |
| 0CH | LVDC | 2CH | PCPU |
| 0DH | TMR0 | 2DH | PDPU |
| 0EH | TMR0C | 2EH | INTEDGE |
| 0FH | BZC | 2FH | Unused |
| 10H | TBC | 30H | LCDCTRL |
| 11H | Unused | 31H | Unused |
| 12H | PA | 32H | Unused |
| 13H | PAC | 33H | WDTC |
| 14H | PB | 34H | MFIC |
| 15H | PBC | 35H | Unused |
| 16H | PC | 36H | Unused |
| 17H | PCC | 37H | Unused |
| 18H | PD | 38H | PCFS |
| 19H | PDC | 39H | PDFS |
| 1AH | PE | 3AH | TMR2 |
| 1BH | PEC | 3BH | TMR2C |
| 1CH | PEPU | 3CH | TMR1 |
| 1DH | Unused | 3DH | TMR1C |
| 1EH | INTC1 | 3EH | Unused |
| 1FH | LVRC | 3FH | Unused |

: Unused, read as "xx"

**HT45R4U Special Purpose Data Memory**

## Special Function Register Description

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 Pointers, MP0 or MP1. Acting as a pair, IAR0 and MP0 can together access data from Bank 0 while the IAR1 and MP1 register pair can access data from any bank. As the Indirect Addressing Registers are not physically implemented, reading the Indirect Addressing Registers indirectly will return a result of ″00H″ and writing to the registers indirectly will result in no operation.

### Memory Pointers – MP0, MP1

Two Memory Pointers, known as MP0 and MP1 are provided. These Memory Pointers are physically implemented in the Data Memory and can be manipulated in the same way as normal registers providing a convenient way with which to 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 all banks.

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

#### Indirect Addressing Program Example

```
data .section   'data'
adres1   db ?
adres2   db ?
Adres3   db ?
adres4   db ?
block    db ?
code .section at 0 'code'
org      00h
start:
     mov a,04h               ; 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 several banks, the total number of which depends upon the device chosen. 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, and so on.

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 any bank. Directly addressing the Data Memory will always result in Bank 0 being accessed irrespective of the value of the Bank Pointer. Accessing data from banks other than Bank 0 must be implemented using Indirect addressing.

### BP Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | D7 | D6 | BP5 | D4 | D3 | BP2 | BP1 | BP0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7~6 **D7~D6**: undefined bits
These bits can be read or written by user software program

Bit 5 **BP5**: Program memory bank point
0: Program memory Bank 0
1: Program memory Bank 1
The program Memory has the capacity of 16K words implemented as 8K words x 2 Banks.

Bit 4~3 **D4~D3**: undefined bits
These bits can be read or written by user software program

Bit 2~0 **BP2~PB0**: Data memory bank point
000: General Purpose Data Memory Bank 0
001: Bank 1, LCD display memory
010: General Purpose Data Memory Bank 2
011: General Purpose Data Memory Bank 3
100: General Purpose Data Memory Bank 4
101: General Purpose Data Memory Bank 5
110: General Purpose Data Memory Bank 6
111: Unomplemented

## Accumulator − ACC

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

### Program Counter Low Register − PCL

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

### Look-up Table Registers − TBLP, TBHP, TBLH

These three special function registers are used to control operation of the look-up table which is stored in the Program Memory. The TBLP and TBHP registers are the lower order byte and high order byte table pointers and indicates the location where the table data is located. Their value must be setup before any table read commands are executed. Their 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 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 Registers

These 8-bit registers, INTC0, INTC1, MFIC and INTEDGE, 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 the INTC0 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

The device contains three internal 8-bit Timer/Event Counters. The registers TMR0, TMR1 and TMR2 are the locations where the timer values are located. These registers can also be preloaded with fixed data to allow different time intervals to be setup. The associated control registers, TMR0C, TMR1C and TMR2C, contain 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, PC, PD and PE. These labeled I/O registers are mapped to specific addresses within the Data Memory as shown in the Data Memory table, which are used to transfer the appropriate output or input data on that port. With each I/O port there is an associated control register labeled PAC, PBC, PCC, PDC and PEC 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 this device.

### A/D Converter Registers – ADRL, ADRH, ADCR0, ADCR1, ACER

The device contains a multiple channels 12-bit A/D converter. The correct operation of the A/D requires the use of two data registers and three 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 three control registers, ADCR0, ADCR1 and ACER.

### 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 resister.

### Pull-High Resistors − PAPU, PBPU, PCPU, PDPU, PEPU

All I/O pins on Ports PA, PB, PC, PD and PE 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.

### System Mode control Register – SMOD

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, PCFS, PDFS

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. All of the 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, PCFS and PDFS, are used to select the required function.

## Input/Output Ports

Holtek 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 up to 34 bidirectional input/output lines labeled with port names PA, PB, PC, PD and PE. 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, PCPU, PDPU and PEPU are implemented using weak PMOS transistors.

**PAPU Register**

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

**PBPU Register**

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

### PCPU Register

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

### PDPU Register

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

Bit 7~0 I/O Port bit 7~bit 0 Pull-High Control
0: disable
1: enable

### PEPU Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | — | — | — | — | D0 |
| R/W | — | — | — | — | — | — | — | R/W |
| POR | — | — | — | — | — | — | — | 0 |

Bit 7~1 Unimplemented, read as "0"

Bit 0 Port E bit 0 Pull-High Control
0: disable
1: enable

## 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.

### PAWU Register

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

Bit 7~0 Port A bit 7~bit 0 Wake-up Control
0: disable
1: enable

## I/O Port Control Registers

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

### PAC Register

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

### PBCRegister

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

### PCC Register

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

### PDC Register

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

Bit 7~0 I/O Port bit 7~bit 0 Input/Output Control
0: output
1: input

### PEC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | — | — | — | D1 | D0 |
| R/W | — | — | — | — | — | — | R/W | R/W |
| POR | — | — | — | — | — | — | 1 | 1 |

Bit 7~2 Unimplemented, read as "0"

Bit 1~0 Port E bit 1~bit 0 Input/Output Control
0: output
1: input

## 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 PA2 and PA7 respectively. 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 and TMR2 are pin-shared with I/O pins. To configure them to operate as timer inputs, 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

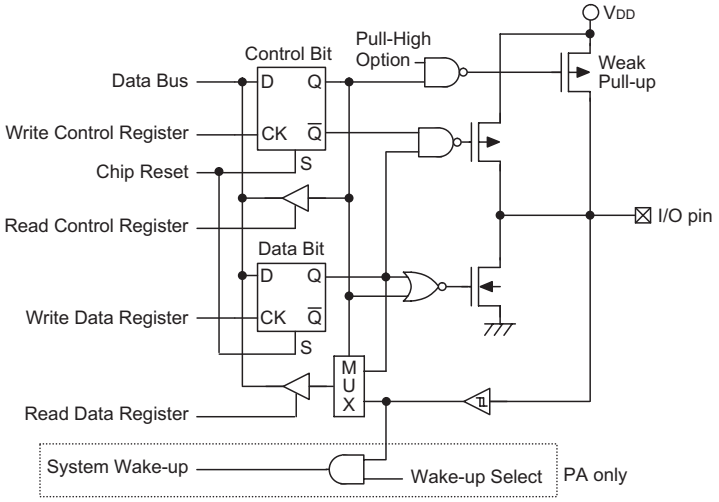
The device contains a PFD function whose single output is pin-shared with I/O pin PA3. The output function of this pin is chosen via a configuration option and remains fixed after the device is programmed. Note that the corresponding bit of the port control register, PAC.3, must setup the pin as an output 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.

### A/D Inputs

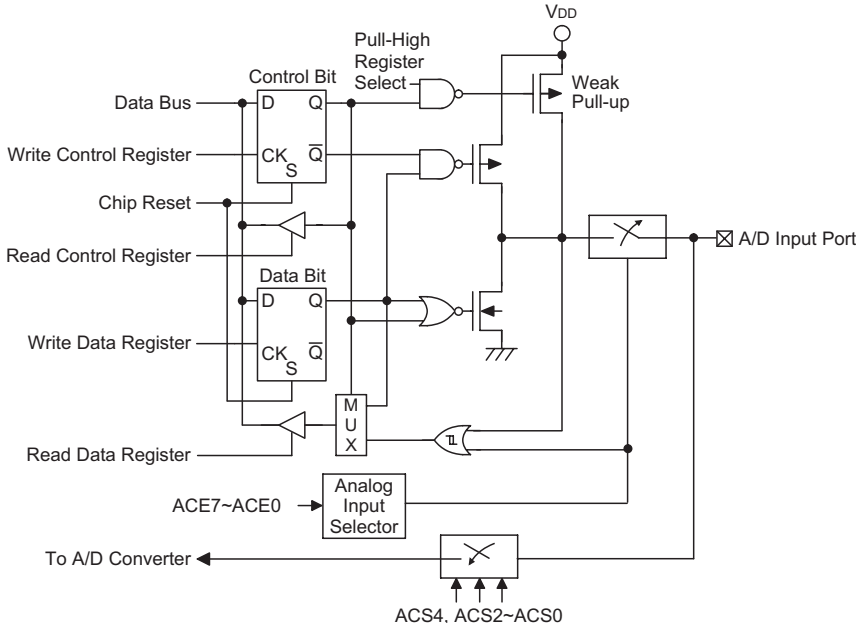
The device contains a multi-channel A/D converter inputs. All of these analog inputs are pin-shared with I/O pins. If these pins are to be used as A/D inputs and not as normal I/O pins then the corresponding bits in the A/D Converter Channel Enable Register, ACER, must be properly set. There are no configuration 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 internal structures of some I/O pin types. As the exact logical construction of the I/O pin will differ from these drawings, they are supplied as a guide only to assist with the functional understanding of the I/O pins. The wide range of pin-shared structures does not permit all types to be shown.




**Generic Input/Output Structure**

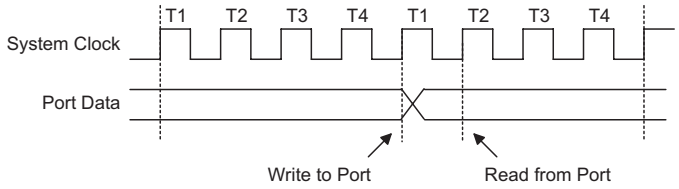



**A/D Input/Output Structure**

## 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, PCC, PDC and PEC, are then programmed to setup some pins as outputs, these output pins will have an initial high output value unless the associated port data registers, PA, PB, PC, PD and PE, are first programmed. Selecting which pins are inputs and which are outputs can be achieved byte-wide by loading the correct values into the appropriate port control register or by programming individual bits in the port control register using the "SET [m].i" and "CLR [m].i" instructions. Note that when using these bit control instructions, a read-modify-write operation takes place. The microcontroller must first read in the data on the entire port, modify it to the required new bit values and then rewrite this data back to the output ports.

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.



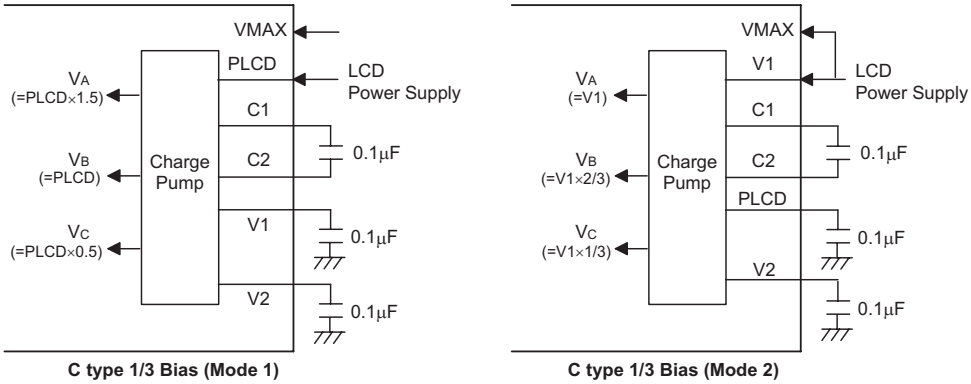

**Read/Write Timing**

## 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 COM and SEG 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 device contains an LCD Driver function, which with their 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.

| Duty | Driver Number | Bias | Bias Type | Wave Type |
|---|---|---|---|---|
| 1/4 | 32x4 | 1/3 | C | A or B |

**LCD Selections**

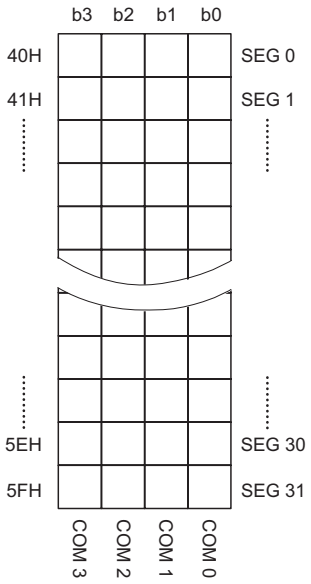



**C Type 1/3 Bias**

### Display Memory

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

As the Display Memory addresses overlap those of the General Purpose Data Memory, it s stored in its own independent Bank 1 area. The Data Memory Bank to be used is chosen by using the Bank Pointer, which is a special function register in the Data Memory, with the name, BP. To access the Display Memory therefore requires first that Bank 1 is selected by writing a value of 01H to the BP register. After this, the 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, starting with address 40H, will result in operations to the Display Memory. Directly addressing the Display Memory is not applicable and will result in a data access to the Bank 0 General Purpose Data Memory.




**LCD Display Memory**

## LCD Registers

A control register in the Data memory is used to control the various setup features of the LCD Driver. This control register for the LCD function is named LCDCTRL. Two control bits in the register control functions including waveform type and overall LCD enable and disable. The LCDEN bit in the LCDCTRL register, which provide the overall LCD enable/disable function, 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 display will always be disabled. The TYPE bit in the same register is used to select whether Type A or Type B LCD control signals are used.

Two registers, PCFS and PDFS are used to determine if the output function of display pins SEG0~SEG15 are used as segment drivers or CMOS outputs. If used as CMOS outputs then the Display Memory is used to determine the logic level of the CMOS output pins. The output function of pins SEG0~SEG15 can be chosen individually to be either a segment driver or a CMOS input.

### LCDCTRL Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | TYPE | — | — | — | — | — | — | LCDEN |
| R/W | R/W | — | — | — | — | — | — | R/W |
| POR | 0 | — | — | — | — | — | — | 0 |

Bit 7 **TYPE**: LCD waveform type selection
0: Type A
1: Type B

Bit 6~1 unimplemented, read as "0"

Bit 0 **LCDEN**: LCD function enable control
0: LCD Disabled
1: LCD Enabled
This bit is only available to control the LD function in Normal, Slow and Idle modes. In Sleep mode, the LCD function will be always switched off.

**PCFS Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | PCFS7 | PCFS6 | PCFS5 | PCFS4 | PCFS3 | PCFS2 | PCFS1 | PCFS0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Bit 7~0 Port C bit 7~ bit 0 function selection
0: I/O
1: SEG

**PDFS Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | PDFS7 | PDFS6 | PDFS5 | PDFS4 | PDFS3 | PDFS2 | PDFS1 | PDFS0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Bit 7~0 Port D bit 7~ bit 0 function selection
0: I/O
1: SEG

## LCD 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 $\overline{\text{LCDEN}}$ signal is the inverse of the LCDEN bit in the LCDCTRL register.

Reset LCD = Sleep Mode or $\overline{\text{LCDEN}}$.

LCDEN =0 and LCDEN =1 must be enabled to activate the register function.

| LCDEN | Sleep Mode | Reset LCD |
|---|---|---|
| 0 | Off | √ |
| 0 | On | √ |
| 1 | Off | x |
| 1 | On | √ |

**LCD Reset Function**

## Clock Source

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 the external 32768Hz oscillator. 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 |
|---|---|
| External 32768Hz Osc. | 4kHz |

**LCD Clock Source**

## LCD Driver Output

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 has a value of 1/4 and which equates to a COM number of 4, etc, 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.

## 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 bias voltage type is C type 1/3 bias for LCD display in this device. The C type biasing enables an internal charge pump whose multiplier ratio can be selected using an additional configuration option.

For C type biasing an external LCD voltage source must also be supplied on pin PLCD 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 PLCD pin. This feature is useful in applications where the microcontroller supply voltage is less than the supply voltage required by the LCD. An additional charge pump capacitor must also be connected between pins C1 and C2 to generate the necessary voltage levels.

For the C type mode 1 selection, four voltage levels $V_{SS}$, $V_A$, $V_B$ and $V_C$ are utilised. The voltage $V_A$ is generated internally and has a value of $V_{PLCD} \times 3/2$. The voltage $V_B$ will have a value equal to $V_{PLCD}$ and $V_C$ will have a value equal to $V_{PLCD} \times 1/2$. The connection to the VMAX pin depends upon the voltage that is applied to PLCD and the details are shown in the table

| Voltage Level | VMAX Connection |
|---|---|
| $V_{DD} > V_{PLCD} \times 3/2$ | Connect VMAX to VDD |
| Otherwise | Connect VMAX to V1 |

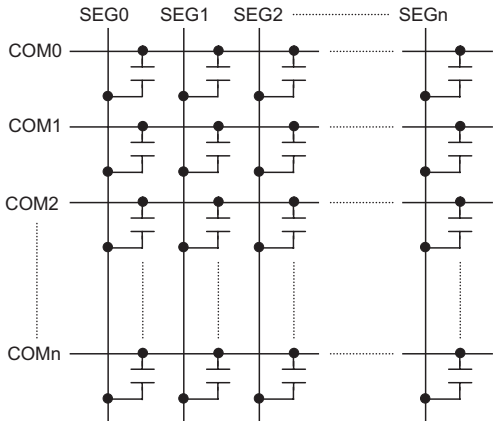
**C Type 1/3 Biasing VMAX Connection (for mode 1)**

For the C type mode 2 selection, four voltage levels $V_{SS}$, $V_A$, $V_B$ and $V_C$ are utilised. The voltage $V_A$ is generated internally and has a value of V1. The voltage $V_B$ will have a value equal to V1 × 2/3 and $V_C$ will have a value equal to V1 × 1/3. The VMAX pin must connected to the V1 pin.

## Programming Considerations

Certain precautions must be taken when programming the LCD. One of these is to ensure that the Display Memory is properly initialised after the microcontroller is powered on. Like the General Purpose Data Memory, the contents of the Display Memory are in an unknown condition after power-on. As the contents of the Display Memory y will be mapped into the actual display, 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 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.

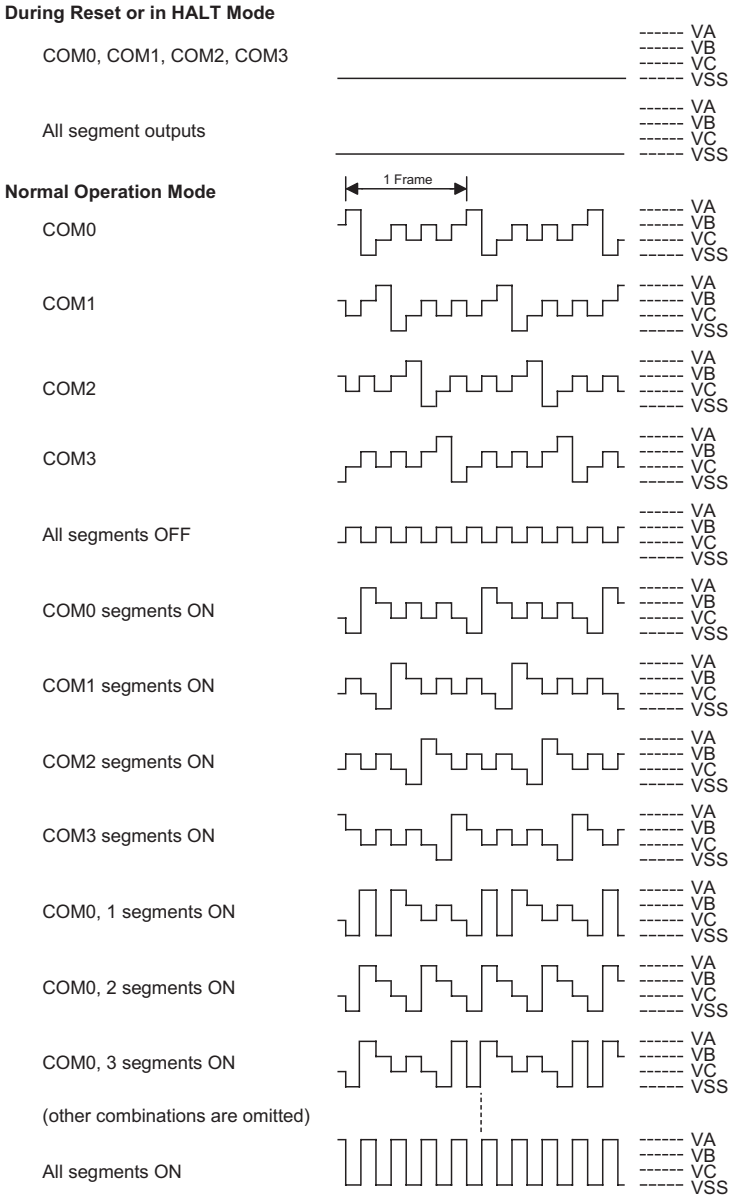



**LCD Panel Equivalent Circuit**

One additional consideration that must be taken into account is what happens when the microcontroller enters a Power Down condition. The LCDEN control bit in the LCDCTRL register permits the display to be powered off to reduce power consumption. If this bit is zero, the driving signals to the display 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 will be cleared to zero, the display function will be disabled.

The accompanying timing diagrams depict the display driver signals generated by the microcontroller for various values of duty and bias. The huge range of various permutations only permits a few types to be displayed here.




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

Note: For 1/3 C type bias mode 1, the $V_A=V_{PLCD} \times 3/2$, $V_B=V_{PLCD}$, $V_C=V_{PLCD} \times 1/2$.
For 1/3 C type bias mode 2, the $V_A=V1$, $V_B=V1 \times 2/3$, $V_C=V1 \times 1/3$.

## Timer/Event Counters

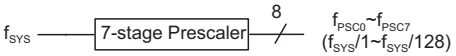
The provision of timers form an important part of any microcontroller, giving the designer a means of carrying out time related functions. The device contains three 8-bit count-up timers. As each timer has three different operating modes, they can be configured to operate as a general timer, an external 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 the 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.

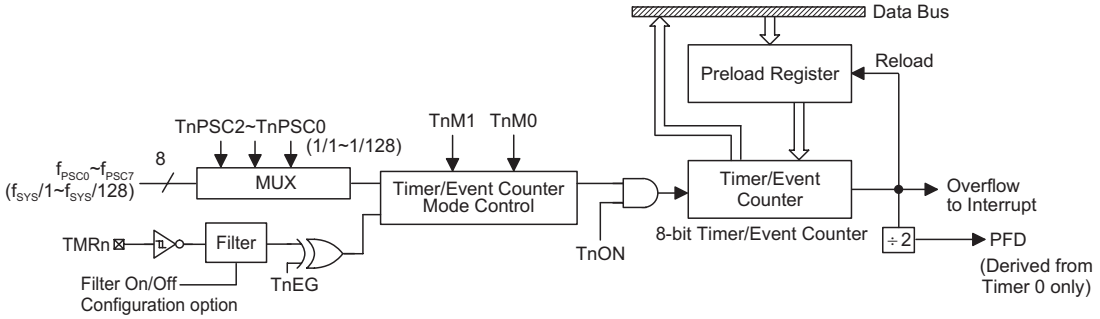
### Configuring the Timer/Event Counter Input Clock Source

The internal timer 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. This internal clock source is fSYS which is also divided by a prescaler, the division ratio of which is conditioned by the Timer Control Register, TMRnC, bits TnPSC0~TnPSC2.

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, TMR1 or TMR2 depending upon which timer is used. Depending upon the condition of the TnEG bit, each high to low, or low to high transition on the external timer pin will increment the counter by one.




**Timer/Event Counters Shared Prescaler**




**8-bit Timer/Event Counter Structure (n=0, 1, 2)**

## Timer Registers − TMR0, TMR1, TMR2

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. These registers are known as TMR0, TMR1 or TMR2. 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 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, 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.

## Timer Control Registers − TMR0C, TMR1C, TMR2C

The flexible features of the Holtek microcontroller Timer/Event Counters enable them to operate in three different modes, the options of which are determined by the contents of 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 corresponding Timer Control Register, which are known as the bit pair TnM1/TnM0, must be set to the required logic levels. The timer-on bit, which is bit 4 of the Timer Control Register and known as TnON, 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 level of bit 3 of the Timer Control Register which is known as TnEG.

### TMRnC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | TnM1 | TnM0 | — | TnON | TnEG | TnPSC2 | TnPSC1 | TnPSC0 |
| R/W | R/W | R/W | — | R/W | R/W | R/W | R/W | R/W |
| POR | 0 | 0 | — | 0 | 1 | 0 | 0 | 0 |

Bit 7~6 **TnM[1:0]:** Operating mode select
00: no mode available
01: Event Counter mode
10: Timer mode
11: Pulse Width Measurement mode

Bit 5 Unimplemented, read as "0"

Bit 4 **TnON:** Timer/Event Counter counting enable
0: enable
1: disable

Bit 3 **TnEG:** Event Counter active edge select
0: count on rising edge
1: count on falling edge
Pulse Width Measurement active edge select
0: start counting on falling edge, stop on rising edge
1: start counting on rising edge, stop on falling edge

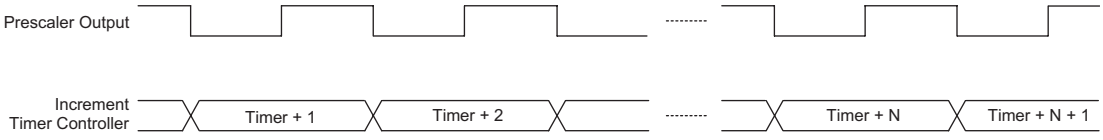
Bit 2~0 **TnPSC[2:0]:** Timer prescaler rate select
000: 1:1
001: 1:2
010: 1:4
011: 1:8
100: 1:16
101: 1:32
110: 1:64
111: 1:128

## 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, TnM1/TnM0, 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 Counters. However, the clock source, $f_{SYS}$, for the 8-bit timer is further divided by a prescaler, the value of which is determined by the Prescaler Rate Select bits TnPSC2~TnPSC0, which are bits 2~0 in the Timer Control Register. After the other bits in the Timer Control Register have been setup, the enable bit TnON or TnON, 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 corresponding Interrupt Control Register, is reset to zero.




**Timer Mode Timing Chart**

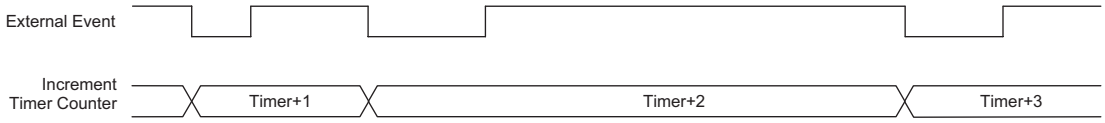
### Configuring the Event Counter Mode

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, TnM1/TnM0, 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, 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 TnON, which is bit 4 of the Timer Control Register, can be set high to enable the Timer/Event Counter to run. If the Active Edge Select bit, TnEG, which is bit 3 of the Timer Control Register, is low, the Timer/Event Counter will increment each time the external timer pin receives a low to high transition. If the 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 corresponding Interrupt Control Register, 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.




**Event Counter 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, TnM1/TnM0, 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 |

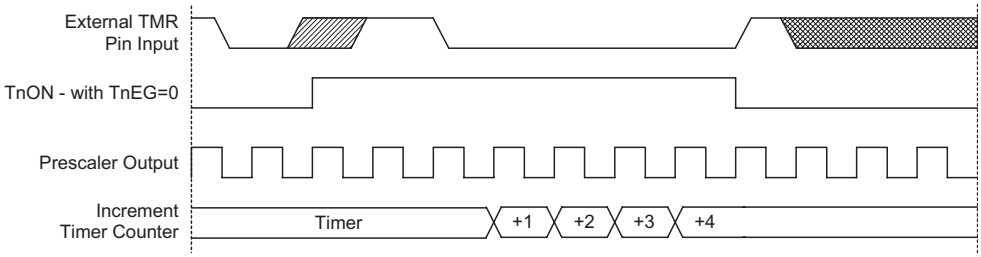
In this mode the internal clock, $f_{SYS}$, is used as the internal clock for the 8-bit Timer/Event Counters. However, the clock source, $f_{SYS}$, for the 8-bit timer is further divided by a prescaler, the value of which is determined by the Prescaler Rate Select bits TnPSC2~TnPSC0, which are bits 2~0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

If the Active Edge Select bit TnEG, which is bit 3 of the Timer Control Register, is low, once a high to low transition has been received on the external timer pin, the Timer/Event Counter will start counting until the external timer pin returns to its original high level. At this point the enable bit will be automatically reset to zero and the Timer/Event Counter will stop counting. If the Active Edge Select bit is high, the Timer/Event Counter will begin counting once a low to high transition has been received on the external timer pin and stop counting when the external timer pin returns to its original low level. As before, the enable bit will be automatically reset to zero and the Timer/Event Counter will stop counting. It is important to note that in the Pulse Width 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 corresponding Interrupt Control Register, 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.




**Pulse Width Capture Mode Timing Chart**

## Programmable Frequency Divider – PFD

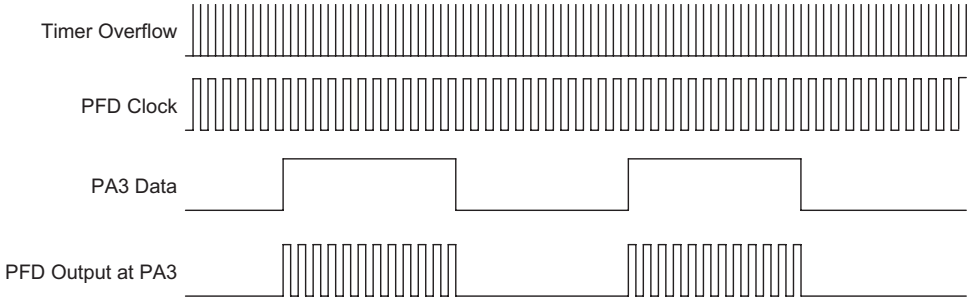
The Programmable Frequency Divider provides a means of producing a variable frequency output suitable for applications requiring a precise frequency generator.

The PFD output is pin-shared with the I/O pin PA3. The PFD function is selected selected via a software control bit, PFDC, however, if not selected, the pin can operate as a normal I/O pin.

The clock source for the PFD circuit originates from Timer/Event Counter 0 overflow signal. 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.




**PFD Output Control**

Bits T0PSC0~T0PSC2 of the control register can be used to define the pre-scaling stages of the internal clock source of the Timer/Event Counter. 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 external pins for correct operation. As these pins are shared pins they must be configured correctly to ensure they are setup for use as Timer/Event Counter inputs and not as a normal I/O pins. This is implemented by ensuring that the mode select bits in the Timer/Event Counter control register, select either the event counter or pulse width measurement mode. Additionally the Port Control Register must be set high to ensure that the pin is setup as an input. Any pull-high resistor on these pins 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 synchronized 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.

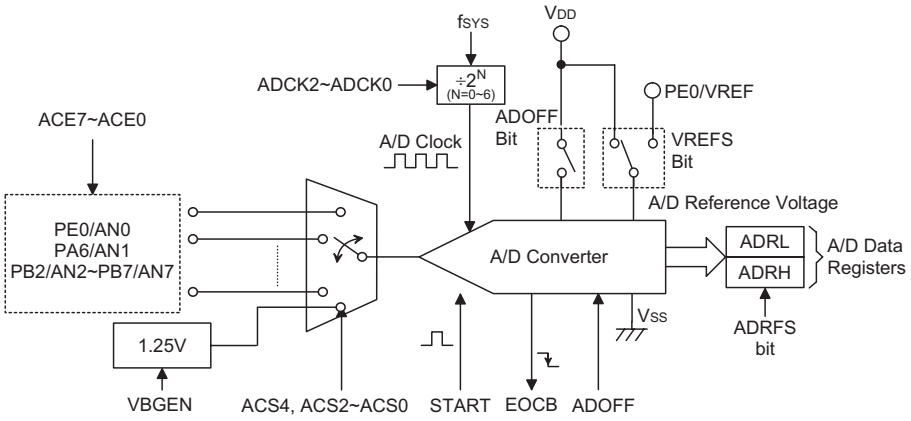
```
org  04h        ; external interrupt 0 vector
reti
org  08h        ; external interrupt 1 vector
reti
org  0ch        ; Timer/Event Counter 0 interrupt vector
jmp  tmr0int    ; jump here when the Timer/Event Counter 0 overflows
:
org  20h        ; main program
tmr0int:        ; internal timer/event counter 0 interrupt routine
:
:               ; timer/event counter 0 interrupt routine 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
```

# 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.




**A/D Converter Structure**

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

As the device contains an internal 12-bit A/D converter, they require two data registers to store the converted value. These are 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. As only 12 bits of the 16-bit register space is utilised, the format in which the data is stored is controlled by the ADRFS bit in the ADCR0 register as shown in the accompanying table. D0~D11 are the A/D conversion result data bits. Any unused bits will be read as zero.

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

**A/D Data Registers**

## A/D Converter Control Register – ADCR0, ADCR1, ACER

To control the function and operation of the A/D converter, two control registers known as ADCR0 and ADCR1 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 ADCR0 register and ACS4 bit is the ADCR1 register define the ADC input channel number. As the device contains only one actual analog to digital converter hardware circuit, each of the individual 8 analog inputs must be routed to the converter. It is the function of the ACS4 and ACS2~ACS0 bits to determine which analog channel input pins or internal 1.25V is actually connected to the internal A/D converter.

The ACER control register contains the ACER7~ACER0 bits which determine which pins on PE0, PA6 and PB2~PB7 are used as analog inputs for the A/D converter input and which pins are not to be used as the A/D converter input. Setting the corresponding bit high will select the A/D input function, clearing the bit to zero will select either the I/O or other pin-shared function. When the pin is selected to be an A/D input, its original function whether it is an I/O or other pin-shared function will be removed. In addition, any internal pull-high resistors connected to these pins will be automatically removed if the pin is selected to be an A/D input.

### ADCR0 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | START | EOCB | ADOFF | ACS4 | — | ACS2 | ACS1 | ACS0 |
| R/W | R/W | R | R/W | R/W | — | R/W | R/W | R/W |
| POR | 0 | 1 | 1 | 0 | — | 0 | 0 | 0 |

Bit 7 **START:** Start the A/D conversion
0→1→0 : start
0→1 : reset the A/D converter and set EOCB to "1"
This bit is used to initiate an A/D conversion process. The bit is normally low but if set high and then cleared low again, the A/D converter will initiate a conversion process. When the bit is set high the A/D converter will be reset.

Bit 6 **EOCB:** End of A/D conversion flag
0: A/D conversion ended
1: A/D conversion in progress
This read only flag is used to indicate when an A/D conversion process has completed. When the conversion process is running the bit will be high.

Bit 5 **ADOFF:** ADC module power on/off control bit
0: ADC module power on
1: ADC module power off
This bit controls the power to the A/D internal function. This bit should be cleared to zero to enable the A/D converter. If the bit is set high then the A/D converter will be switched off reducing the device power consumption. As the A/D converter will consume a limited amount of power, even when not executing a conversion, this may be an important consideration in power sensitive battery powered applications.
Note: 1. it is recommended to set ADOFF=1 before entering IDLE/SLEEP Mode for saving power.
2. ADOFF=1 will power down the ADC module.

Bit 4 **ACS4:** Select Internal 1.25V as ADC input Control
0: disable
1: enable
This bit enables 1.25V to be connected to the A/D converter. The V125EN bit must first have been set to enable the bandgap circuit 1.25V voltage to be used by the A/D converter. When the ACS4 bit is set high, the bandgap 1.25V voltage will be routed to the A/D converter and the other A/D input channels disconnected.

Bit 3 unimplemented, read as "0"

Bit 2~0 **ACS2, ACS1, ACS0:** Select A/D channel (when ACS4 is "0")
000: AN0
001: AN1
010: AN2
011: AN3
100: AN4
101: AN5
110: AN6
111: AN7
These are the A/D channel select control bits. As there is only one internal hardware A/D converter each of the eight A/D inputs must be routed to the internal converter using these bits. If bit ACS4 in the ADCR1 register is set high then the internal 1.25V will be routed to the A/D Converter.

### ADCR1 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | VBGEN | ADRFS | VREFS | — | ADCK2 | ADCK1 | ADCK0 |
| R/W | — | R/W | R/W | R/W | — | R/W | R/W | R/W |
| POR | — | 0 | 0 | 0 | — | 0 | 0 | 0 |

Bit 7 unimplemented, read as "0"

Bit 6 **VBGEN:** Internal 1.25V Control
0: disable
1: enable
This bit controls the internal Bandgap circuit on/off function to the A/D converter. When the bit is set high the bandgap voltage 1.25V can be used by the A/D converter. If 1.25V is not used by the A/D converter and the LVR/LVD function is diabled then the bandgap reference circuit will be automatically switched off to conserve power. When 1.25V is switched on for use by the A/D converter, a time $t_{BG}$ should be allowed for the bandgap circuit to stabilise before implementing an A/D conversion.

Bit 5 **ADRFS:** ADC Data Format Control
0: ADC Data MSB is ADRH bit 7, LSB is ADRL bit 4
1: ADC Data MSB is ADRH bit 3, LSB is ADRL bit 0
This bit controls the format of the 12-bit converted A/D value in the two A/D data registers. Details are provided in the A/D data registers section.

Bit 4 **VREFS:** Selecte ADC reference voltage
0: Internal ADC power
1: VREF pin
This bit is used to select the reference voltage for the A/D converter. If the bit is high, then the A/D converter reference voltage is supplied on the external VREF pin. If the pin is low, then the internal reference is used which is taken from the power supply pin VDD.

Bit 3 Unimplemented, read as "0"

Bit 2~0 **ADCK2~ADCK0:** Select ADC converter clock source
000: $f_{SYS}$
001: $f_{SYS}/2$
010: $f_{SYS}/4$
011: $f_{SYS}/8$
100: $f_{SYS}/16$
101: $f_{SYS}/32$
110: $f_{SYS}/64$
111: undefined
These three bits are used to select the clock source for the A/D converter.

**ACER Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | ACE7 | ACE6 | ACE5 | PCR4 | ACE3 | ACE2 | ACE1 | ACE0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7 **ACE7**: Define PB7 is A/D input or not
0: Not A/D input
1: A/D input, AN7

Bit 6 **ACE6**: Define PB6 is A/D input or not
0: Not A/D input
1: A/D input, AN6

Bit 5 **ACE5**: Define PB5 is A/D input or not
0: Not A/D input
1: A/D input, AN5

Bit 4 **ACE4**: Define PB4 is A/D input or not
0: Not A/D input
1: A/D input, AN4

Bit 3 **ACE3**: Define PB3 is A/D input or not
0: Not A/D input
1: A/D input, AN3

Bit 2 **ACE2**: Define PB2 is A/D input or not
0: Not A/D input
1: A/D input, AN2

Bit 1 **ACE1**: Define PA6 is A/D input or not
0: Not A/D input
1: A/D input, AN1

Bit 0 **ACE0**: Define PE0 is A/D input or not
0: Not A/D input
1: A/D input, AN0

## A/D Operation

The START bit in the ADCR0 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 ADCR0 register will be set high and the analog to digital converter will be reset. It is the START bit that is used to control the overall start operation of the internal analog to digital converter.

The EOCB bit in the ADCR0 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 ADCR0 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 $f_{SYS}$, can be chosen to be either $f_{SYS}$ or a subdivided version of $f_{SYS}$. The division ratio value is determined by the ADCK2~ADCK0 bits in the ADCR1 register.

Although the A/D clock source is determined by the system clock $f_{SYS}$, and by bits ADCK2~ADCK0, 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 0.5μs, care must be taken for system clock frequencies equal to or greater than 4MHz. For example, if the system clock operates at a frequency of 4MHz, the ADCK2~ADCK0 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 ($t_{ADCK}$) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | ADCK2, ADCK1, ADCK0 =000 ($f_{SYS}$) | ADCK2, ADCK1, ADCK0 =001 ($f_{SYS}$/2) | ADCK2, ADCK1, ADCK0 =010 ($f_{SYS}$/4) | ADCK2, ADCK1, ADCK0 =011 ($f_{SYS}$/8) | ADCK2, ADCK1, ADCK0 =100 ($f_{SYS}$/16) | ADCK2, ADCK1, ADCK0 =101 ($f_{SYS}$/32) | ADCK2, ADCK1, ADCK0 =110 ($f_{SYS}$/64) | ADCK2, ADCK1, ADCK0 =111 |
| 1MHz | 1μs | 2μs | 4μs | 8μs | 16μs* | 32μs* | 64μs* | Undefined |
| 2MHz | 500ns | 1μs | 2μs | 4μs | 8μs | 16μs* | 32μs* | Undefined |
| 4MHz | 250ns* | 500ns | 1μs | 2μs | 4μs | 8μs | 16μs* | Undefined |

**A/D Clock Period Examples**

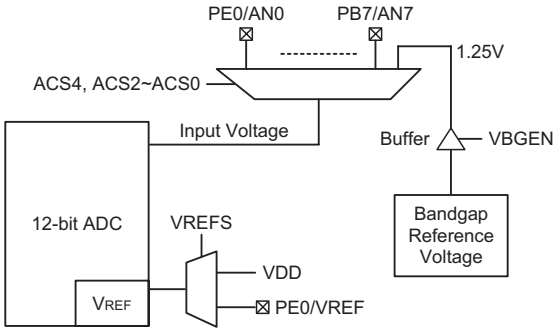
Controlling the power on/off function of the A/D converter circuitry is implemented using the ADOFF bit in the ADCR0 register. This bit must be zero to power on the A/D converter. When the ADOFF bit is cleared to zero to power on the A/D converter internal circuitry a certain delay, as indicated in the timing diagram, must be allowed before an A/D conversion is initiated. Even if no pins are selected for use as A/D inputs by clearing the ACE7~ACE0 bits in the ACER register, if the ADOFF bit is zero then some power will still be consumed. In power conscious applications it is therefore recommended that the ADOFF is set high to reduce power consumption when the A/D converter function is not being used.

The reference voltage supply to the A/D Converter can be supplied from either the positive power supply pin, VDD, or from an external reference sources supplied on pin VREF. The desired selection is made using the VREFS bit. As the VREF pin is pin-shared with other functions, when the VREFS bit is set high, the VREF pin function will be selected and the other pin functions will be disabled automatically.

## A/D Input Pins

All of the A/D analog input pins are pin-shared with the I/O pins on PE0, PA6 and PB2~PB7 as well as other functions. The ACE7~ACE0 bits in the ACER register, determine whether the input pins are setup as A/D converter analog inputs or whether they have other functions. If the ACE7~ACE0 bits for its corresponding pin is set high then the pin will be setup to be an A/D converter input and the original pin functions disabled. In this way, pins can be changed under program control to change their function between A/D inputs and other functions. All pull-high resistors, which are setup through register programming, will be automatically disconnected if the pins are setup as A/D inputs. Note that it is not necessary to first setup the A/D pin as an input in the PAC, PEC or PBC port control register to enable the A/D input as when the ACE7~ ACE0 bits enable an A/D input, the status of the port control register will be overridden.

The A/D converter has its own reference voltage pin, VREF, however the reference voltage can also be supplied from the power supply pin, a choice which is made through the VREFS bit in the ADCR1 register. The analog input values must not be allowed to exceed the value of $V_{REF}$.




**A/D Input Structure**

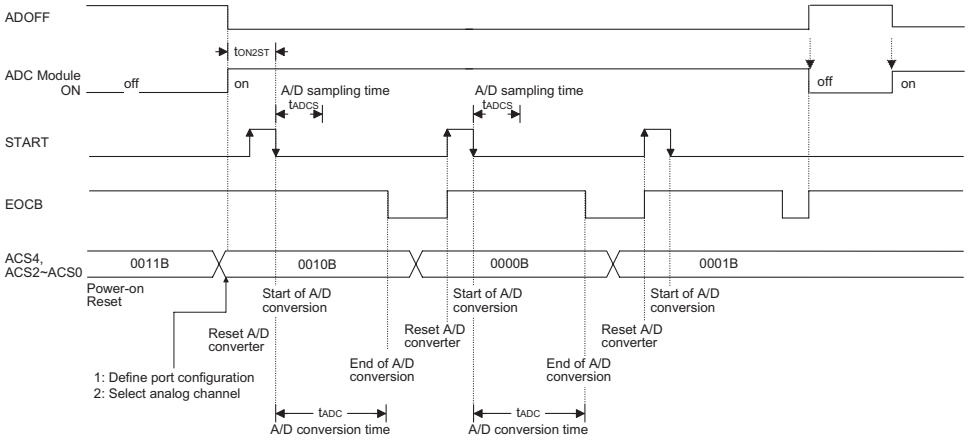
## Summary of A/D Conversion Steps

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 ADCK2~ADCK0 in the ADCR1 register.
- Step 2
  Enable the A/D by clearing the ADOFF bit in the ADCR0 register to zero.
- Step 3
  Select which channel is to be connected to the internal A/D converter by correctly programming the ACS4 and ACS2~ACS0 bits which are also contained in the ADCR0 register.
- Step 4
  Select which pins are to be used as A/D inputs and configure them by correctly programming the ACE7~ACE0 bits in the ACER register.
- 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, and the A/D converter interrupt bit, ADE, must both be set high to do this.
- Step 6
  The analog to digital conversion process can now be initialised by setting the START bit in the ADCR register from low to high and then low again. Note that this bit should have been originally cleared to zero.
- Step 7
  To check when the analog to digital conversion process is complete, the EOCB bit in the ADCR0 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 ADCR0 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. 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, $t_{ADC}$, taken for the A/D conversion is 16 $t_{AD}$ where $t_{AD}$ is equal to the A/D clock period.




**A/D Conversion Timing**

## A/D Transfer Function

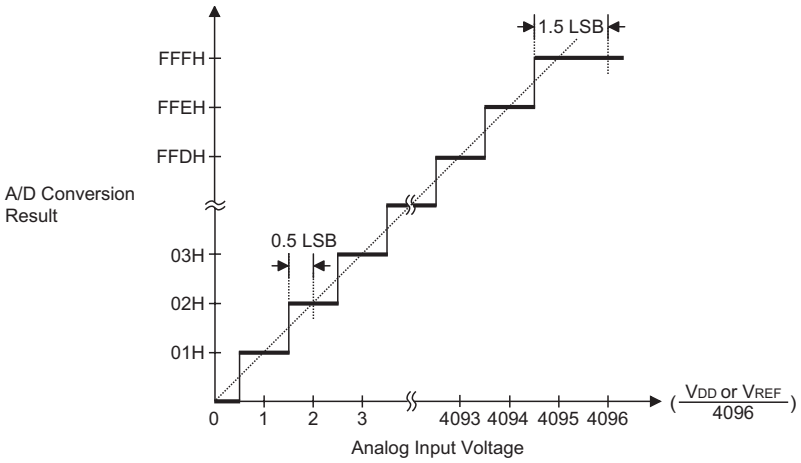
As the devices contain a 12-bit A/D converter, its full-scale converted digitised value is equal to FFFH. Since the full-scale analog input value is equal to the $V_{DD}$ or $V_{REF}$ voltage, this gives a single bit analog input value of $V_{DD}$ or $V_{REF}$ divided by 4096.

$$1\ LSB = (V_{DD} \text{ or } V_{REF}) \div 4096$$

The A/D Converter input voltage value can be calculated using the following equation:

$$\text{A/D input voltage} = \text{A/D output digital value} \times (V_{DD} \text{ or } V_{REF}) \div 4096$$

The diagram shows the ideal transfer function between the analog input value and the digitised output value for the A/D converter. 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 $V_{DD}$ or $V_{REF}$ level.




**Ideal A/D Transfer Function**

## Programming Considerations

During microcontroller operations where the A/D converter is not being used, the A/D internal circuitry can be switched off to reduce power consumption, by setting bit ADOFF high in the ADCR0 register. When this happens, the internal A/D converter circuits will not consume power irrespective of what analog voltage is applied to their input lines. If the A/D converter input lines are used as normal I/Os, then care must be taken as if the input voltage is not at a valid logic level, then this may lead to some increase in power consumption.

## 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  ADE                    ; disable ADC interrupt
mov  a,03H
mov  ADCR1,a                ; select fSYS/8 as A/D clock and switch off 1.25V
clr  ADOFF
mov  a,0Fh                  ; setup ACER to configure analog pins AN0~AN3
mov  ACER,a
mov  a,00h
mov  ADCR0,a                ; enable and connect AN0 channel to A/D converter
:
start_conversion:
     clr START              ; high pulse on start bit to initiate conversion
     set START              ; reset A/D
     clr START              ; start A/D
polling_EOC:
     sz EOCB                ; poll the ADCR0 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
```

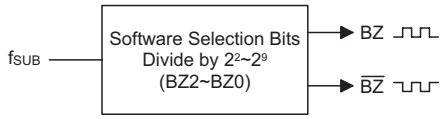
**Example 2: using the interrupt method to detect the end of conversion**

```
clr  ADE                    ; disable ADC interrupt
mov  a,03H
mov  ADCR1,a                ; select fSYS/8 as A/D clock and switch off 1.25V
Clr  ADOFF
mov  a,0Fh                  ; setup ACER to configure analog pins AN0~AN3
mov  ACER,a
mov  a,00h
mov  ADCR0,a                ; enable and connect AN0 channel to A/D converter
Start_conversion:
     clr START              ; high pulse on START bit to initiate conversion
     set START              ; reset A/D
     clr START              ; start A/D
     clr ADF                ; clear ADC interrupt request flag
     set ADE                ; enable ADC interrupt
     set EMI                ; enable global interrupt
:
:
                            ; ADC interrupt service routine
ADC_ISR:
     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_INT_ISR:
     mov a,status_stack
     mov STATUS,a           ; restore STATUS from user defined memory
     mov a,acc_stack        ; restore ACC from user defined memory
     reti
```

## 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 complementary pair, and are pin-shared with I/O pins, PA0 and PA1. Note that the $\overline{BZ}$ pin is the inverse of the BZ pin which together generates a differential output which can supply more power to connected interfaces such as buzzers.

The buzzer is driven by the internal clock source, $f_{SUB}$, which then passes through a divider, the division ratio of which is selected by configuration options to provide a range of buzzer frequencies from $f_{SUB}/2^2$ to $f_{SUB}/2^9$. The clock source that generates $f_{SUB}$, which in turn controls the buzzer frequency, originates from the LXT oscillator. Note that the buzzer frequency is controlled by software selection bits, which select the internal division ratio.




**Buzzer Function**

### PA0/PA1 Pin Function Control

The pin function of BZ or $\overline{BZ}$ is selected by ENBZ or ENBZB software control bits respectively. When the software control bit, ENBZ or ENBZB, is set to 1, the corresponding buzzer output function will be enabled. Otherwise, the buzzer output function will be disabled if the corresponding software control bit is cleared to 0.

If the pin-shared software control bits 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 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 $\overline{BZ}$ buzzer pin outputs. Note that the PA1 data bit in the PA register has no control over the buzzer pin, PA1.

If the pin-shared software control bit has selected that the PA0 pin is to function as a BZ buzzer pin, then the 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 software control bit has configured it as a BZ output.

If the pin-shared software control bit has selected that the PA1 pin is to function as a $\overline{BZ}$ buzzer pin, then the PA1 must be setup as an output by setting bit PAC1 of the PAC port control register to zero. Note that 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 $\overline{BZ}$ buzzer pin PA1. If the PAC1 bit of the PAC port control register is set high, then pin PA1 can still be used as an input even though the software control bit has configured it as a $\overline{BZ}$ output. Note that the PA1 data bit in the PA register still has no control over the buzzer pin, PA1.

Note that no matter what configuration is chosen for the buzzer, if the port control register has setup the pin to function as an input, then this will override the software 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 software selection; the actual function of the pin can be changed dynamically by the application program by programming the appropriate port control register bit.

| PAC0 Control bit | PA0 Data bit | PA0 Output Function |
|---|---|---|
| 0 | 0 | PA0 = "0" |
| 0 | 1 | PA0 = BZ |
| 1 | 0 | PA0 = input line |
| 1 | 1 | PA0 = input line |

**PA0 Pin Function Control (ENBZ=1)**

| PAC1 Control bit | PA0 Data bit | PA1 Output Function |
|---|---|---|
| 0 | 0 | PA1 = "0" |
| 0 | 1 | PA1 = $\overline{BZ}$ |
| 1 | 0 | PA1 = input line |
| 1 | 1 | PA1 = input line |

**PA1 Pin Function Control (ENBZB=1)**

### BZC Register

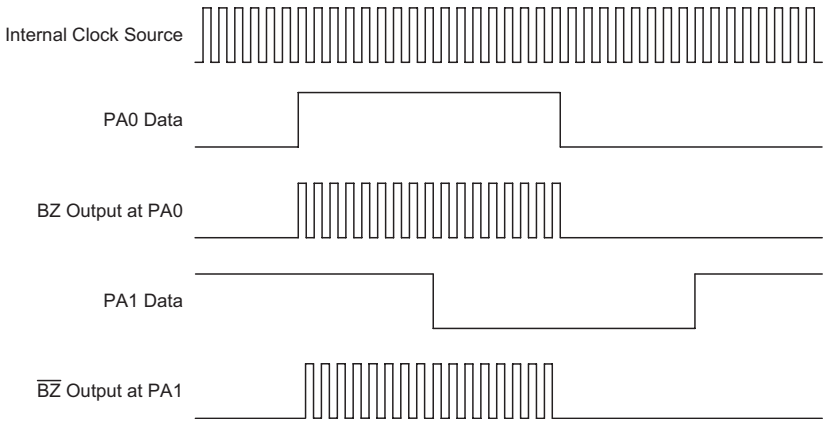
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | ENBZ | ENBZB | BZ2 | BZ1 | BZ0 |
| R/W | — | — | — | R/W | R/W | R/W | R/W | R/W |
| POR | — | — | — | 0 | 0 | 0 | 0 | 0 |

Bit 7~5 unimplemented, read as "0"

Bit 4 **ENBZ:** BZ Pin Function Selection
0: I/O function
1: BZ function

Bit 3 **ENBZB:** $\overline{BZ}$ Pin Function Selection
0: I/O function
1: $\overline{BZ}$ function

Bit 2~0 **BZ2 ~ BZ0:** Buzzer Output Frequency Selection
000: $f_{SUB}/2^2$
001: $f_{SUB}/2^3$
010: $f_{SUB}/2^4$
011: $f_{SUB}/2^5$
100: $f_{SUB}/2^6$
101: $f_{SUB}/2^7$
110: $f_{SUB}/2^8$
111: $f_{SUB}/2^9$




**Buzzer Output Pin Control**

Note: The above drawing shows the situation where both pins PA0 and PA1 are selected by software control bits to be BZ and $\overline{BZ}$ 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

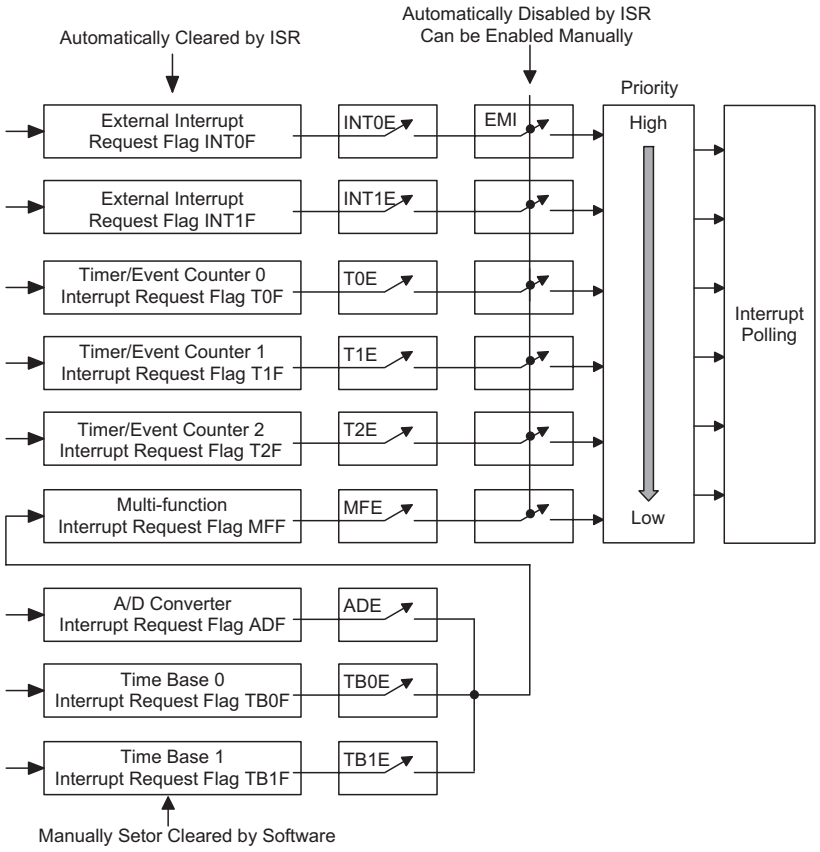
Interrupts are an important part of any microcontroller system. When an external event or an internal function such as a Timer/Event Counter or an A/D converter requires microcontroller attention, their corresponding interrupt will enforce a temporary suspension of the main program allowing the microcontroller 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 and INT1 pins, while the internal interrupts are controlled by the Timer/Event Counter overflows, the Time Base interrupts and the A/D converter interrupt.

## Interrupt Operation

A Timer/Event Counter overflow, Time Base, 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. When this happens and if their appropriate interrupt enable bit is set, 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 Structure**

## 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 |
| Timer/Event Counter 2 Overflow | | 5 | 14H |
| Multi-function Interrupt | A/D Conversion Completion | 6 | 18H |
| | Time Base 0 Interrupt | | |
| | Time Base 1 Interrupt | | |

The A/D converter interrupt and two Time Base interrupts share the same interrupt vector which is 18H. Each of these interrupts have their own 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.

## 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.

### INTEDGE Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | — | INT1S1 | INT1S0 | INT0S1 | INT0S0 |
| R/W | — | — | — | — | R/W | R/W | R/W | R/W |
| POR | — | — | — | — | 0 | 0 | 0 | 0 |

Bit 7~4 unimplemented, read as 0

Bit 3~2 **INT1S1~INT1S0:** interrupt edge control for INT1 pin
00: disable
01: rising edge
10: falling edge
01: both rising and falling edges

Bit 1~0 **INT0S1~INT0S0:** interrupt edge control for INT0 pin
00: disable
01: rising edge
10: falling edge
01: both rising and falling edges

### INTC0 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | T0F | INT1F | INT0F | T0E | INT1E | INT0E | EMI |
| R/W | — | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | — | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7 unimplemented, read as 0

Bit 6 **T0F:** Timer/Event Counter 0 interrupt request flag
0: no request
1: interrupt request

Bit 5 **INT1F:** INT1 interrupt request flag
0: no request
1: interrupt request

Bit 4 **INT0F:** INT0 pin interrupt request flag
0: no request
1: interrupt request

Bit 3 **T0E:** Timer/Event Counter 0 interrupt control
0: disable
1: enable

Bit 2 **INT1E:** INT1 interrupt control
0: disable
1: enable

Bit 1 **INT0E:** INT0 interrupt control
0: disable
1: enable

Bit 0 **EMI:** Global interrupt control
0: disable
1: enable

### INTC1 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | MFF | T2F | T1F | — | MFE | T2E | T1E |
| R/W | — | R/W | R/W | R/W | — | R/W | R/W | R/W |
| POR | — | 0 | 0 | 0 | — | 0 | 0 | 0 |

Bit 7 unimplemented, read as 0

Bit 6 **MFF:** Multi-function interrupt request flag
0: no request
1: interrupt request

Bit 5 **T2F:** Timer/Event Counter 2 interrupt request flag
0: no request
1: interrupt request

Bit 4 **T1F:** Timer/Event Counter 1 interrupt request flag
0: no request
1: interrupt request

Bit 3 unimplemented, read as 0

Bit 2 **MFE:** Multi-function interrupt control
0: disable
1: enable

Bit 1 **T2E:** Timer/Event Counter 2 interrupt control
0: disable
1: enable

Bit 0 **T1E:** Timer/Event Counter 1 interrupt control
0: disable
1: enable

### MFIC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | TB1F | TB0F | ADF | — | TB1E | TB0E | ADE |
| R/W | — | R/W | R/W | R/W | — | R/W | R/W | R/W |
| POR | — | 0 | 0 | 0 | — | 0 | 0 | 0 |

Bit 7 unimplemented, read as "0"

Bit 6 **TB1F:** Time Base 1 interrupt request flag
0: no request
1: interrupt request

Bit 5 **TB0F:** Time Base 0 interrupt request flag
0: no request
1: interrupt request

Bit 4 **ADF:** A/D conversion completion interrupt request flag
0: no request
1: interrupt request

Bit 3 unimplemented, read as "0"

Bit 2 **TB1E:** Time Base 1 interrupt control
0: disable
1: enable

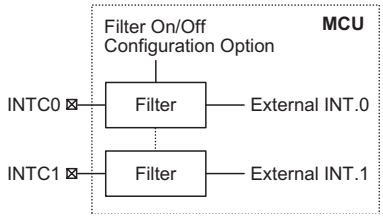
Bit 1 **TB0E:** Timer Base 0 interrupt control
0: disable
1: enable

Bit 0 **ADE:** A/D conversion completion interrupt control
0: disable
1: enable

## External Interrupt

For an external interrupt to occur, the global interrupt enable bit, EMI, and external interrupt enable bits, INT0E and INT1E, 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, INT0F or INT1F, 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 PA2 and PA7 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 PAC.2 and PAC.7 bits in the port control register. When the interrupt is enabled, the stack is not full and the correct transition type appears on the 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, INT0F or INT1F, 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.

## Timer/Event Counter Interrupt

For a Timer/Event Counter 0, Timer/Event Counter 1 or Timer/Event Counter 2 interrupt to occur, the global interrupt enable bit, EMI, and the corresponding timer interrupt enable bit, TnE, must first be set. An actual Timer/Event Counter interrupt will take place when the Timer/Event Counter request flag, TnF is set, a situation that will occur when the 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, 10C or 14H, will take place. When the interrupt is serviced, the timer interrupt request flag, TnF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

### Multi-function Interrupt

An additional interrupt known as the Multi-function interrupt is provided. Unlike the other interrupts, this interrupt has no independent source, but rather is formed from three other existing interrupt sources, namely the A/D Converter interrupt and two Time Base interrupts.

For a Multi-function interrupt to occur, the global interrupt enable bit, EMI, and the Multi-function interrupt enable bit, MFE, 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 or an A/D conversion completion 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 or A/D Converter interrupt will not be automatically reset and must be manually reset by the application program.

### 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, ADE, and Multi-function interrupt enable bit, MFE, must first be 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 location18H, 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.

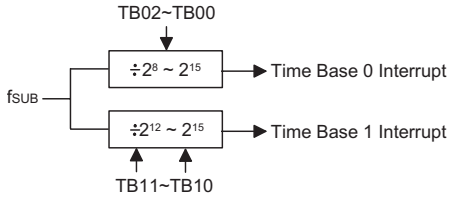
### Time Base Interrupt

Two Time Base Interrupts are contained within the Multi-function Interrupt.

For a Time Base Interrupt to be generated, the global interrupt enable bit, EMI, the specific Time Base Interrupt enable bit, TBnE, and Multi-function interrupt enable bit, MFE, must first be set. An actual Time Base Interrupt will take place when the Time Base Interrupt request flag, TBnF, is set, a situation that will occur when the Time Base overflows. When the interrupt is enabled, the stack is not full and the specific 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 TBnF 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 internal clock source $f_{SUB}$. This $f_S$ input clock first passes through a divider, the division ratio of which is selected by software selection bits to provide longer Time Base interrupt periods. The Time Base 0 interrupt time-out period ranges from $2^{12}/f_{SUB}$~$2^{15}/f_{SUB}$ while the Time Base 1 interrupt time-out period ranges from $2^{8}/f_{SUB}$~$2^{15}/f_{SUB}$. The clock source, $f_{SUB}$, which controls the Time Base interrupt period, is derived from the LXT oscillator.

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




**Time Base Interrupt**

### TBC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | TB11 | TB10 | LXTLP | PFDC | TB02 | TB01 | TB00 |
| R/W | — | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | — | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7 unimplemented, read as "0"

Bit 6~5 **TB11~TB10:** Time Base 1 Time-out Period selection
00: 4096/$f_{SUB}$
01: 8192/$f_{SUB}$
10: 16384/$f_{SUB}$
11: 32768/$f_{SUB}$

Bit 4 **LXTLP:** LXT Low Power Control
0: Disable
1: Enable

Bit 3 **PFDC:** PFD function Control
0: I/O
1: PFD

Bit 2~0 **TB02~TB00:** Time Base 0 Time-out Period selection
000: 256/$f_{SUB}$
001: 512/$f_{SUB}$
010: 1024/$f_{SUB}$
011: 2048/$f_{SUB}$
100: 4096/$f_{SUB}$
101: 8192/$f_{SUB}$
110: 16384/$f_{SUB}$
111: 32768/$f_{SUB}$

## 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.

## Reset and Initialisation

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

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

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

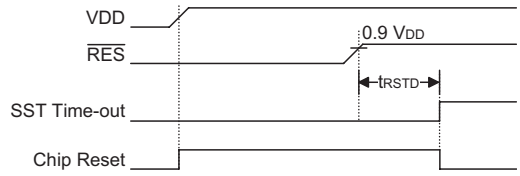
### Reset Functions

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

#### Power-on Reset

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

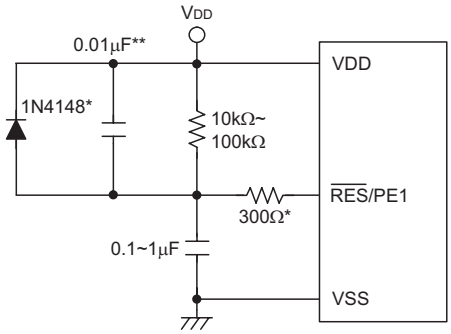
Although the microcontroller has an internal RC reset function, if the VDD power supply rise time is not fast enough or does not stabilise quickly at power-on, the internal reset function may be incapable of providing proper reset operation. For this reason it is recommended that an external RC network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




**Power-On Reset Timing Chart**

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

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



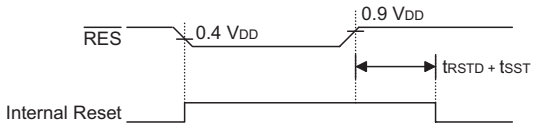

Note: "*" It is recommended that this component is added for added ESD protection

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

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

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

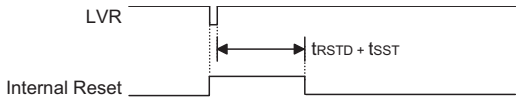
This type of reset occurs when the microcontroller is already running and the $\overline{RES}$ pin is forcefully pulled low by external hardware such as an external switch. In this case as in the case of other reset, the Program Counter will reset to zero and program execution initiated from this point.




**$\overline{RES}$ Reset Timing Chart**

### Low Voltage Reset − LVR

The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device. The LVR function is always enabled with a specific LVR voltage, $V_{LVR}$. If the supply voltage of the device drops to within a range of 0.9V~$V_{LVR}$ such as might occur when changing the battery, the LVR will automatically reset the device internally and the LVRF bit in the SMOD1 register will also be set to 1. For a valid LVR signal, a low supply voltage, i.e., a voltage in the range between 0.9V~$V_{LVR}$ must exist for a time greater than that specified by $t_{LVR}$ in the A.C. characteristics. If the low supply voltage state does not exceed this value, the LVR will ignore the low supply voltage and will not perform a reset function. The actual $V_{LVR}$ value is fixed by the LVS bits in the LVRC register. If the LVS7~LVS0 bits have any other value, which may perhaps occur due to adverse environmental conditions such as noise, the LVR will reset the device after 2~3 $f_{LIRC}$ clock cycles. When this happens, the LRF bit in the SMOD1 register will be set to 1. After power on the register will have the value of 01010101B. Note that the LVR function will be automatically disabled when the device enters the power down mode.




**Low Voltage Reset Timing Chart**

• **LVRC Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | LVS7 | LVS6 | LVS5 | LVS4 | LVS3 | LVS2 | LVS1 | LVS0 |
| R/W | R/W | R/W | R/W | R/W | R/w | R/w | R/W | R/W |
| POR | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Bit 7~0 **LVS7 ~ LVS0:** LVR voltage select
01010101: 2.1V
00110011: 2.1V
10011001: 2.1V
10101010: 2.1V
Any other values: Generates MCU reset – register is reset to POR value

When an actual low voltage condition occurs, as specified by the defined LVR voltage values above, an MCU reset will be generated. The reset operation will be activated after 2~3 $f_{LIRC}$ clock cycles. In this situation the register contents will remain the same after such a reset occurs.

Any register value, other than the four defined register values above, will also result in the generation of an MCU reset. The reset operation will be activated after 2~3 $f_{LIRC}$ clock cycles. However in this situation the register contents will be reset to the POR value.

• **SMOD1 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | FSYSON | — | — | — | — | LVRF | LRF | WRF |
| R/W | R/W | — | — | — | — | R/W | R/W | R/W |
| POR | 0 | — | — | — | — | x | 0 | 0 |

'x' unknown

Bit 7 **FSYSON:** $f_{SYS}$ Control in IDLE Mode
Described elsewhere.

Bit 6~3 Unimplemented, read as "0"

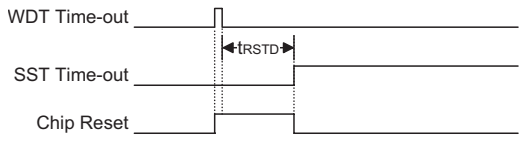
Bit 2 **LVRF:** LVR function reset flag
0: not occurred
1: occurred
This bit is set to 1 when a specific Low Voltage Reset situation condition occurs. This bit can only be cleared to 0 by the application program.

Bit 1 **LRF:** LVR Control register software reset flag
0: not occurred
1: occurred
This bit is set to 1 if the LVRC register contains any non defined LVR voltage register values. This in effect acts like a software-reset function. This bit can only be cleared to 0 by the application program.

bit 0 **WRF:** WDT Control register software reset flag
Described elsewhere.

### Watchdog Time-out Reset during Normal Operation

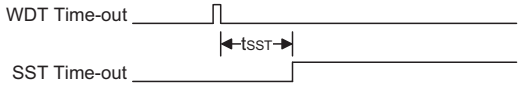
The Watchdog time-out Reset during normal operation is the same as a hardware $\overline{RES}$ pin reset except that the Watchdog time-out flag TO will be set to ″1″.




**WDT Time-out Reset during 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 | $\overline{RES}$ reset during power-on |
| u | u | $\overline{RES}$ or LVR reset during normal operation |
| 1 | u | WDT time-out reset during normal operation |
| 1 | 1 | WDT time-out reset during Power Down |

Note: "u" stands for unchanged

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

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

The different kinds of resets all affect the internal registers of the microcontroller in different ways. To ensure reliable continuation of normal program execution after a reset occurs, it is important to know what condition the microcontroller is in after a particular reset occurs. The following table describes how each type of reset affects each of the microcontroller internal registers. Note that where more than one package type exists the table will reflect the situation for the larger package type.

| Register | Reset (Power-On) | $\overline{RES}$ Reset (Normal Oper.) | WDT Time-out (Normal Oper.) | WDT Time-out (HALT)* |
|---|---|---|---|---|
| MP0 | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| MP1 | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| BP | --0- 0000 | --0- 0000 | --0- 0000 | --u- uuuu |
| ACC | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| PCL | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| TBLP | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TBLH | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TBHP | --xx xxxx | --uu uuuu | --uu uuuu | --uu uuuu |
| STATUS | --00 xxxx | --uu uuuu | --1u uuuu | --11 uuuu |
| INTC0 | -000 0000 | -000 0000 | -000 0000 | -uuu uuuu |
| LVDC | --00 -000 | --00 -000 | --00 -000 | --uu -uuu |

| Register | Reset (Power-On) | $\overline{\text{RES}}$ Reset (Normal Oper.) | WDT Time-out (Normal Oper.) | WDT Time-out (HALT)* |
|---|---|---|---|---|
| TMR0 | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| TMR0C | 00-0 1000 | 00-0 1000 | 00-0 1000 | uu-u uuuu |
| BZC | ---0 0000 | ---0 0000 | ---0 0000 | ---u uuuu |
| TBC | -000 0000 | -000 0000 | -000 0000 | -uuu uuuu |
| PA | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PAC | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PB | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PBC | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PC | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PCC | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PD | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PDC | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PE | ---- --11 | ---- --11 | ---- --11 | ---- --uu |
| PEC | ---- --11 | ---- --11 | ---- --11 | ---- --uu |
| PEPU | ---- ---0 | ---- ---0 | ---- ---0 | ---- --u |
| INTC1 | -000 -000 | -000 -000 | -000 -000 | -uuu -uuu |
| LVRC | 0101 0101 | 0101 0101 | 0101 0101 | uuuu uuuu |
| ADCR0 | 0110 -000 | 0110 -000 | 0110 -000 | uuuu -uuu |
| ADCR1 | -000 -000 | -000 -000 | -000 -000 | -uuu -uuu |
| ADRL (ADRFS=0) | xxxx ---- | xxxx ---- | xxxx ---- | uuuu ---- |
| ADRL (ADRFS=1) | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| ADRH (ADRFS=0) | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| ADRH (ADRFS=1) | ---- xxxx | ---- xxxx | ---- xxxx | ---- xxxx |
| ACER | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| SMOD | 0000 0011 | 0000 0011 | 0000 0011 | uuuu uuuu |
| SMOD1 | 0--- -x00 | 0--- -x00 | 0--- -x00 | u--- -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 |
| PCPU | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| PDPU | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| INTEDGE | ---- 0000 | ---- 0000 | ---- 0000 | ---- uuuu |
| LCDCTRL | 0--- ---0 | 0--- ---0 | 0--- ---0 | u--- ---u |
| WDTC | 0101 0011 | 0101 0011 | 0101 0011 | uuuu uuuu |
| MFIC | -000 -000 | -000 -000 | -000 -000 | -uuu -uuu |
| PCFS | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 |
| PDFS | 1111 1111 | 1111 1111 | 1111 1111 | 1111 1111 |
| TMR2 | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| TMR2C | 00-0 1000 | 00-0 1000 | 00-0 1000 | uu-u uuuu |
| TMR1 | xxxx xxxx | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| TMR1C | 00-0 1000 | 00-0 1000 | 00-0 1000 | uu-u uuuu |

Note: "-" not implement
"u" means "unchanged"
"x" means "unknown"

## Oscillator

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

### System Clock Configurations

There are three methods of generating the system clock, two high speed oscillators, one low speed oscillator and an externally supplied clock. The two high speed oscillators are the external crystal/ceramic oscillator, HXT, and the internal 4MHz RC oscillator, HIRC. The low speed oscillator is the external 32.768 kHz crystal oscillator, LXT. Selecting whether the low or high oscillator is used as the system oscillator is implemented using the HLCLK bit and CKS2 ~ CKS0 bits in the SMOD register and as the system clock can be dynamically selected.

The actual source clock for the high speed oscillators is chosen via a configuration option. The frequency of the slow or high speed oscillator is also determined using the HLCLK and CKS2~CKS0 bits in the SMOD register. Note that two oscillator selections must be made namely one high speed and one low speed system oscillators. It is not possible to choose a no-oscillator selection for either the high or low speed oscillator.

### External Crystal/Ceramic Oscillator – HXT

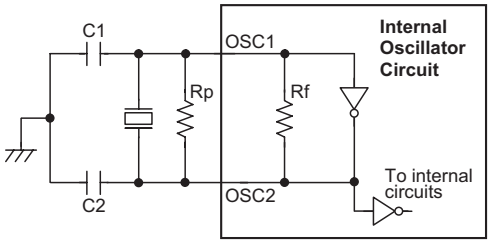
After selecting the external crystal configuration option, the simple connection of a crystal across OSC1 and OSC2, is normally all that is required to create the necessary phase shift and feedback for oscillation, without requiring external capacitors. However, for some crystal types and frequencies, 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. An additional configuration option must be setup to configure the device according to whether the oscillator frequency is high, defined as equal to or above 1MHz, or low, which is defined as below 1 MHz.

More information regarding oscillator applications is located on the Holtek website.

| Crystal Oscillator C1 and C2 Values | | |
|---|---|---|
| **Crystal Frequency** | **C1** | **C2** |
| 4MHz | — | — |
| 1MHz | — | — |
| 455kHz (see Note 2) | 100pF | 100pF |
| Note: 1. C1 and C2 values are for guidance only.<br>2. XTAL mode configuration option: 455kHz.<br>3. $R_{P1}$=5MΩ~10MΩ is recommended. | | |

**Crystal Recommended Capacitor Values**

For oscillator stability and to minimise the effects of noise and crosstalk, it is important to ensure that the crystal and any associated resistors and capacitors along with inter connecting lines are all located as close to the MCU as possible.




Note: 1. Rp is normally not required. C1 and C2 are required.
2. Although not shown OSC1/OSC2 pins have a parasitic capacitance of around 7pF.

**Crystal/Resonator Oscillator – HXT**

## Internal High Speed RC Oscillator – HIRC

The internal RC oscillator is a fully integrated system oscillator requiring no external components. The internal RC oscillator has three fixed frequencies of 4MHz. Device trimming during the manufacturing process and the inclusion of internal frequency compensation circuits are used to ensure that the influence of the power supply voltage, temperature and process variations on the oscillation frequency are minimised. As a result, at a power supply of 3V and at a temperature of 25˚C degrees, the fixed oscillation frequency of 4MHz will have a specific tolerance described in the A.C. Characteristics. Note that if this internal system clock option is selected, as it requires no external pins for its operation, I/O pins PB0 and PB1 are free for use as normal I/O pins.

## External 32.768kHz Crystal Oscillator – LXT

The External 32.768kHz Crystal System Oscillator is the low frequency oscillator. This clock source has a fixed frequency of 32.768 kHz and requires a 32.768 kHz crystal to be connected between pins XT1 and XT2. The external resistor and capacitor components connected to the 32.768 kHz crystal are necessary to provide oscillation. For applications where precise frequencies are essential, these components may be required to provide frequency compensation due to different crystal manufacturing tolerances. During power-up there is a time delay associated with the LXT oscillator waiting for it to start-up.

When the microcontroller enters the SLEEP or IDLE Mode, the system clock is switched off to stop microcontroller activity and to conserve power. However, in many microcontroller applications it may be necessary to keep the internal timers operational even when the microcontroller is in the SLEEP or IDLE Mode. To do this, another clock, independent of the system clock, must be provided.

However, for some crystals, to ensure oscillation and accurate frequency generation, it is necessary to add two small value external capacitors, C1 and C2. The exact values of C1 and C2 should be selected in consultation with the crystal or resonator manufacturer's specification. The external parallel feedback resistor, $R_{P2}$, is required.

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

**32768 Hz Crystal Recommended Capacitor Values**

### LXT Oscillator Low Power Function

The LXT oscillator can function in one of two modes, the Quick Start Mode and the Low Power Mode. The mode selection is executed using the LXTLP bit in the TBC register.

| LXTLP Bit | LXT Mode |
|---|---|
| 0 | Quick Start |
| 1 | Low-power |

After power on, the LXTLP bit will be automatically cleared to zero ensuring that the LXT oscillator is in the Quick Start operating mode. In the Quick Start Mode the LXT oscillator will power up and stabilise quickly. However, after the LXT oscillator has fully powered up it can be placed into the Low-power mode by setting the LXTLP bit high. The oscillator will continue to run but with reduced current consumption, as the higher current consumption is only required during the LXT oscillator start-up. In power sensitive applications, such as battery applications, where power consumption must be kept to a minimum, it is therefore recommended that the application program sets the LXTLP bit high about 2 seconds after power-on.

It should be noted that, no matter what condition the LXTLP bit is set to, the LXT oscillator will always function normally and the only difference is that it will take more time to start up if in the Low-power mode.

### External Oscillator – EC

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 oscillator 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.

# Operating Modes and System Clocks

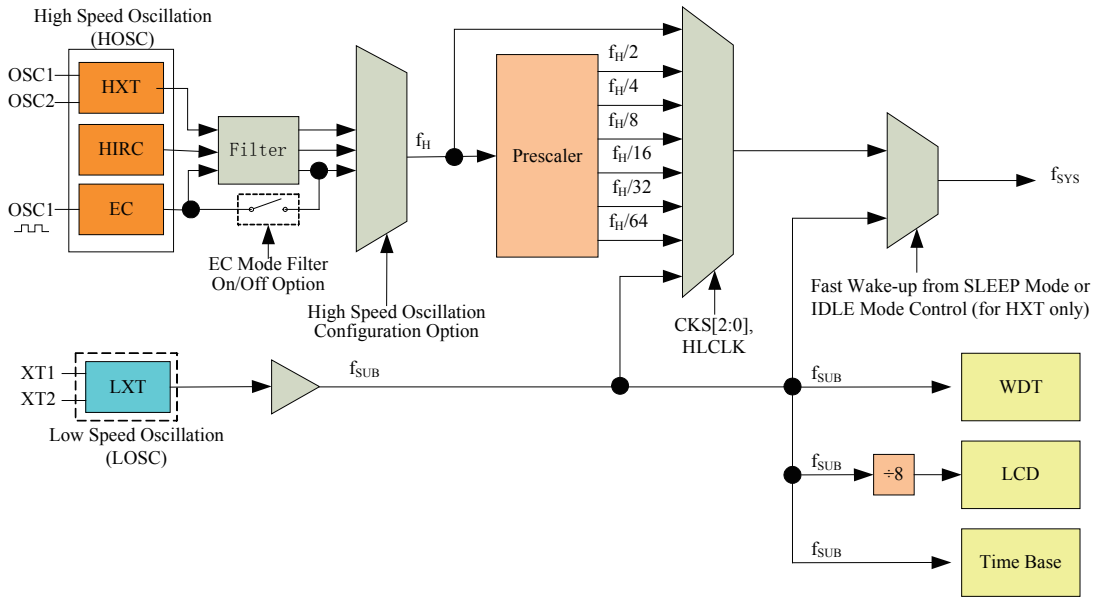
Present day applications require that their microcontrollers have high performance but often still demand that they consume as little power as possible, conflicting requirements that are especially true in battery powered portable applications. The fast clocks required for high performance will by their nature increase current consumption and of course vice-versa, lower speed clocks reduce current consumption. As Holtek has provided these devices with both high and low speed clock sources and the means to switch between them dynamically, the user can optimise the operation of their microcontroller to achieve the best performance/power ratio.

## System Clocks

The device has many different clock sources for both the CPU and peripheral function operation. By providing the user with a wide range of clock options using configuration options and register programming, a clock system can be configured to obtain maximum application performance.

The main system clock, can come from either a high frequency, $f_H$, or low frequency, $f_L$, source, and is selected using the HLCLK bit and CKS2~CKS0 bits in the SMOD register. The high speed system clock can be sourced from either the HXT or HIRC oscillator, selected via a configuration option. The low speed system clock source can be sourced from internal clock $f_{SUB}$. The $f_{SUB}$ clock source is derived from the LXT oscillator. The other choice, which is a divided version of the high speed system oscillator has a range of $f_H/2$~$f_H/64$.

The $f_{SUB}$ clock is also used as the Time Base and Watchdog timer clock source. The $f_{SUB}$ clock is used to provide a substitute clock for the microcontroller just after a wake-up has occurred to enable faster wake-up times.




**System Clock Configurations**

## System Operating Modes

There are six different modes of operation for the microcontroller, each one with its own special characteristics and which can be chosen according to the specific performance and power requirements of the application. There are two modes allowing normal operation of the microcontroller, the NORMAL Mode and SLOW Mode. The remaining four modes, the SLEEP0, SLEEP1, IDLE0 and IDLE1 Mode are used when the microcontroller CPU is switched off to conserve power.

| Operating Mode | Description | | |
|---|---|---|---|
| | CPU | $f_{SYS}$ | $f_{SUB}$ |
| NORMAL Mode | On | $f_H \sim f_H/64$ | On |
| SLOW Mode | On | $f_{SUB}$ | On |
| IDLE0 Mode | Off | Off | On |
| IDLE1 Mode | Off | On | On |
| SLEEP0 Mode | Off | Off | Off |
| SLEEP1 Mode | Off | Off | On |

### NORMAL Mode

As the name suggests this is one of the main operating modes where the microcontroller has all of its functions operational and where the system clock is provided by one of the high speed oscillators. This mode operates allowing the microcontroller to operate normally with a clock source will come from one of the high speed oscillators, either the HXT or HIRC oscillators. The high speed oscillator will however first be divided by a ratio ranging from 1 to 64, the actual ratio being selected by the CKS2~CKS0 and HLCLK bits in the SMOD register. Although a high speed oscillator is used, running the microcontroller at a divided clock ratio reduces the operating current.

### SLOW Mode

This is also a mode where the microcontroller operates normally although now with a slower speed clock source. The clock source used will be from the low speed oscillator, LXT. Running the microcontroller in this mode allows it to run with much lower operating currents. In the SLOW Mode, the $f_H$ is off.

### SLEEP0 Mode

The SLEEP Mode is entered when an HALT instruction is executed and when the IDLEN bit in the SMOD register is low. In the SLEEP0 mode the CPU will be stopped, and the $f_{SUB}$ clock will be stopped too, and the Watchdog Timer function is disabled. Since the WDT function is always enabled, this device will not enter the SLEEP0 mode.

### SLEEP1 Mode

The SLEEP Mode is entered when an HALT instruction is executed and when the IDLEN bit in the SMOD register is low. In the SLEEP1 mode the CPU will be stopped. However, the $f_{SUB}$ clock will continue to operate if the Watchdog Timer function is enabled as its clock source comes from the $f_{SUB}$.

### IDLE0 Mode

The IDLE0 Mode is entered when a HALT instruction is executed and when the IDLEN bit in the SMOD register is high and the FSYSON bit in the SMOD1 register is low. In the IDLE0 Mode the system oscillator will be inhibited from driving the CPU but some peripheral functions will remain operational such as the Watchdog Timer and Time Base. In the IDLE0 Mode, the system oscillator will be stopped. In the IDLE0 Mode the $f_{SUB}$ clock source will be still on.

### IDLE1 Mode

The IDLE1 Mode is entered when an HALT instruction is executed and when the IDLEN bit in the SMOD register is high and the FSYSON bit in the WDTC register is high. In the IDLE1 Mode the system oscillator will be inhibited from driving the CPU but may continue to provide a clock source to keep some peripheral functions operational such as the Timers. In the IDLE1 Mode, the system oscillator will continue to run, and this system oscillator may be high speed or low speed system oscillator. In the IDLE1 Mode the $f_{SUB}$ clock source will be on.

## Control Register

A register pair, SMOD and SMOD1, is used for overall control of the internal clocks within the device.

### SMOD Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | CKS2 | CKS1 | CKS0 | FSTEN | LTO | HTO | IDLEN | HLCLK |
| R/W | R/W | R/W | R/W | R/W | R | R | R/W | R/W |
| POR | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

Bit 7~5 **CKS2~CKS0:** The system clock selection when HLCLK is "0"
000: $f_{SUB}$ ($f_{LXT}$)
001: $f_{SUB}$ ($f_{LXT}$)
010: $f_H/64$
011: $f_H/32$
100: $f_H/16$
101: $f_H/8$
110: $f_H/4$
111: $f_H/2$

These three bits are used to select which clock is used as the system clock source. In addition to the system clock source, which is derived from the LXT oscillator, a divided version of the high speed system oscillator can also be chosen as the system clock source.

Bit 4 **FSTEN:** Fast Wake-up Control (only for HXT)
0: Disable
1: Enable

This is the Fast Wake-up Control bit which determines if the $f_{SUB}$ clock source is initially used after the device wakes up. When the bit is high, the $f_{SUB}$ clock source can be used as a temporary system clock to provide a faster wake up time as the $f_{SUB}$ clock is available.

Bit 3 **LTO:** Low speed system oscillator ready flag
0: Not ready
1: Ready

This is the low speed system oscillator ready flag which indicates when the low speed system oscillator is stable after power on reset or a wake-up has occurred. The flag will be low when in the SLEEP0 Mode but after a wake-up has occurred, the flag will change to a high level after 1024 clock cycles as the LXT oscillator is used.

Bit 2 **HTO:** High speed system oscillator ready flag
0: Not ready
1: Ready

This is the high speed system oscillator ready flag which indicates when the high speed system oscillator is stable. This flag is cleared to "0" by hardware when the device is powered on and then changes to a high level after the high speed system oscillator is stable. Therefore this flag will always be read as "1" by the application program after device power-on. The flag will be low when in the SLEEP or IDLE0 Mode but after a wake-up has occurred, the flag will change to a high level after 1024 clock cycles if the HXT oscillator is used or 15~16 clock cycles if the HIRC oscillator is used.

bit 1 **IDLEN:** IDLE Mode control
0: Disable
1: Enable
This is the IDLE Mode Control bit and determines what happens when the HALT instruction is executed. If this bit is high, when a HALT instruction is executed the device will enter the IDLE Mode. In the IDLE1 Mode the CPU will stop running but the system clock will continue to keep the peripheral functions operational, if FSYSON bit is high. If FSYSON bit is low, the CPU and the system clock will all stop in IDLE0 mode. If the bit is low the device will enter the SLEEP Mode when a HALT instruction is executed.

bit 0 **HLCLK:** system clock selection
0: $f_H/2 \sim f_H/64$ or $\mathbf{f_{SUB}}$
1: $f_H$
This bit is used to select if the $f_H$ clock or the $f_H/2 \sim f_H/64$ or $\mathbf{f_{SUB}}$ clock is used as the system clock. When the bit is high the $f_H$ clock will be selected and if low the $f_H/2 \sim f_H/64$ or $\mathbf{f_{SUB}}$ clock will be selected. When system clock switches from the $f_H$ clock to the $\mathbf{f_{SUB}}$ clock and the $f_H$ clock will be automatically switched off to conserve power.

### SMOD1 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | FSYSON | — | — | — | — | LVRF | LRF | WRF |
| R/W | R/W | — | — | — | — | R/W | R/W | R/W |
| POR | 0 | — | — | — | — | x | 0 | 0 |

'x' unknown

Bit 7 **FSYSON:** $f_{SYS}$ Control in IDLE Mode
0: Disable
1: Enable

Bit 6~3 Unimplemented, read as "0"

Bit 2 **LVRF:** LVR function reset flag
0: not occurred
1: occurred
This bit is set to 1 when a specific Low Voltage Reset situation occurs. This bit can only be cleared to 0 by the application program.

Bit 1 **LRF:** LVR Control register software reset flag
0: not occurred
1: occurred
This bit is set to 1 if the LVRC register contains any non defined LVR voltage register values. This in effect acts like a software-reset function. This bit can only be cleared to 0 by the application program.

bit 0 **WRF:** WDT Control register software reset flag
0: not occurred
1: occurred
This bit is set to 1 by the WDT Control register software reset and cleared by the application program. Note that this bit can only be cleared to 0 by the application program.

## Fast Wake-up

To minimise power consumption the device can enter the SLEEP or IDLE0 Mode, where the system clock source to the device will be stopped. However when the device is woken up again, it can take a considerable time for the original system oscillator to restart, stabilise and allow normal operation to resume. To ensure the device is up and running as fast as possible a Fast Wake-up function is provided, which allows $f_{SUB}$, namely the LXT oscillator, to act as a temporary clock to first drive the system until the original system oscillator has stabilised. As the clock source for the Fast Wake-up function is $f_{SUB}$, the Fast Wake-up function is only available in the SLEEP1 and IDLE0 modes. When the device is woken up from the SLEEP0 mode, the Fast Wake-up function has no effect because the $f_{SUB}$ clock is stopped. The Fast Wake-up enable/disable function is controlled using the FSTEN bit in the SMOD register.

If the HXT oscillator is selected as the NORMAL Mode system clock, and if the Fast Wake-up function is enabled, then it will take one to two $t_{SUB}$ clock cycles of the LXT oscillator for the system to wake-up. The system will then initially run under the $f_{SUB}$ clock source until 1024 HXT clock cycles have elapsed, at which point the HTO flag will switch high and the system will switch over to operating from the HXT oscillator.

If the HIRC oscillator is used as the system oscillator then it will take 15~16 clock cycles of the HIRC to wake up the system from the SLEEP or IDLE0 Mode. The Fast Wake-up bit, FSTEN will have no effect in this case.

| System Oscillator | FSTEN Bit | Wake-up Time (SLEEP0 Mode) | Wake-up Time (SLEEP1 Mode) | Wake-up Time (IDLE0 Mode) | Wake-up Time (IDLE1 Mode) |
|---|---|---|---|---|---|
| HXT | 0 | 1024 HXT cycles | 1024 HXT cycles | | 1~2 HXT cycles |
| | 1 | 1024 HXT cycles | 1~2 $f_{SUB}$ cycles (System runs with $f_{SUB}$ first for 1024 HXT cycles and then switches over to run with the HXT clock ) | | 1~2 HXT cycles |
| HIRC | x | 15~16 HIRC cycles | 15~16 HIRC cycles | | 1~2 HIRC cycles |
| LXT | x | 1024 LXT cycles | 1024 LXT cycles | | 1~2 LXT cycles |

**Wake-up Times**

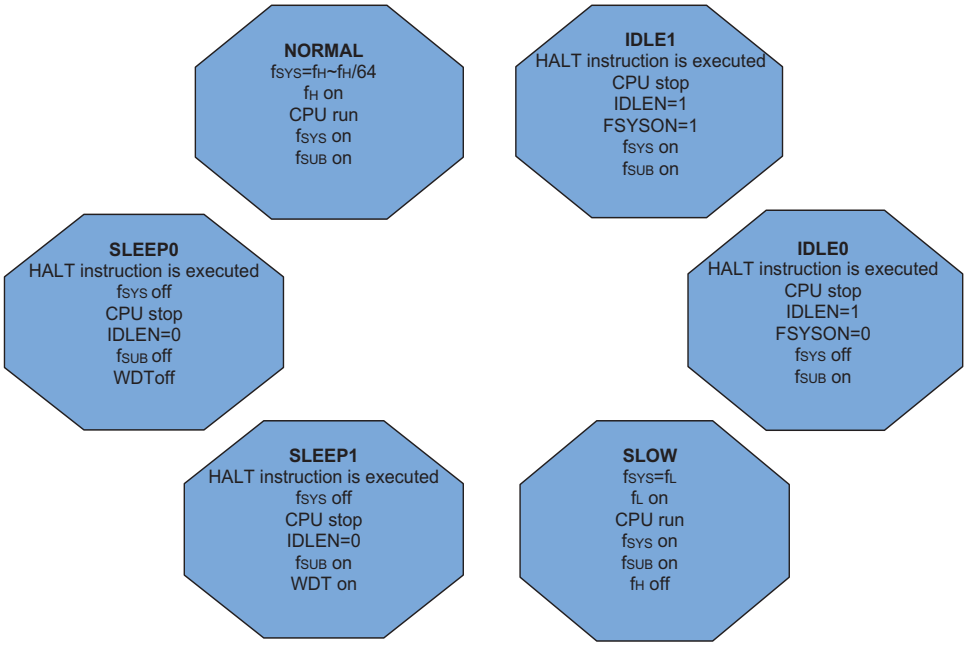
Note that if the Watchdog Timer is disabled, which means that the $f_{SUB}$ clock derived from the LXT is off, then there will be no Fast Wake-up function available when the device wakes up from the SLEEP0 Mode.

## Operating Mode Switching

The device can switch between operating modes dynamically allowing the user to select the best performance/power ratio for the present task in hand. In this way microcontroller operations that do not require high performance can be executed using slower clocks thus requiring less operating current and prolonging battery life in portable applications.

In simple terms, Mode Switching between the NORMAL Mode and SLOW Mode is executed using the HLCLK bit and CKS2~CKS0 bits in the SMOD register while Mode Switching from the NORMAL/SLOW Modes to the SLEEP/IDLE Modes is executed via the HALT instruction. When a HALT instruction is executed, whether the device enters the IDLE Mode or the SLEEP Mode is determined by the condition of the IDLEN bit in the SMOD register and FSYSON in the SMOD1 register.

When the HLCLK bit switches to a low level, which implies that clock source is switched from the high speed clock source, $f_H$, to the clock source, $f_H/2$~$f_H/64$ or $f_{SUB}$. If the clock is from the $f_{SUB}$, the high speed clock source will stop running to conserve power. The accompanying flowchart shows what happens when the device moves between the various operating modes.




### NORMAL Mode to SLOW Mode Switching

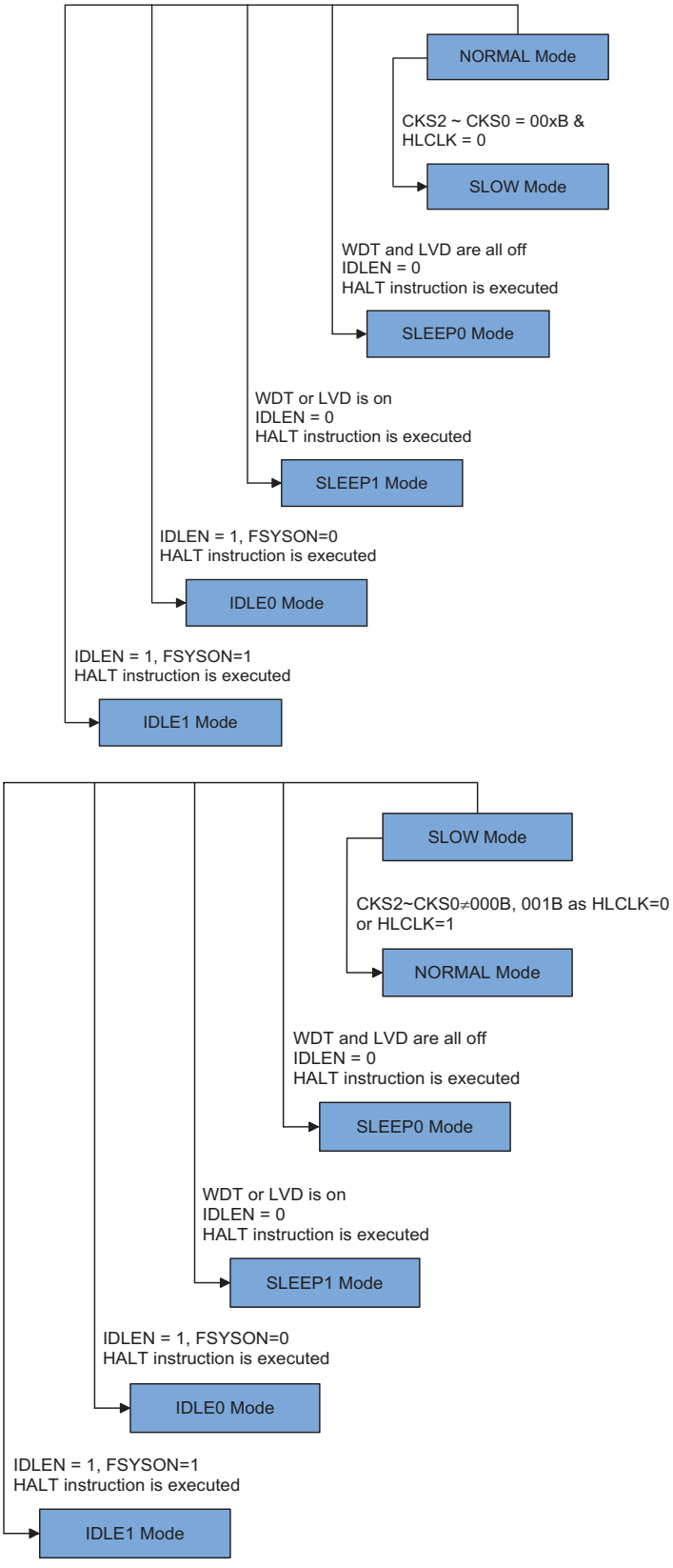
When running in the NORMAL Mode, which uses the high speed system oscillator, and therefore consumes more power, the system clock can switch to run in the SLOW Mode by set the HLCLK bit to "0" and set the CKS2~CKS0 bits to "000" or "001" in the SMOD register. This will then use the low speed system oscillator which will consume less power. Users may decide to do this for certain operations which do not require high performance and can subsequently reduce power consumption.

The SLOW Mode is sourced from the LXT oscillator and therefore requires these oscillators to be stable before full mode switching occurs. This is monitored using the LTO bit in the SMOD register.

### SLOW Mode to NORMAL Mode Switching

In SLOW Mode the system uses the LXT low speed system oscillator. To switch back to the NORMAL Mode, where the high speed system oscillator is used, the HLCLK bit should be set to "1" or HLCLK bit is "0", but CKS2~CKS0 is set to "010", "011", "100", "101", "110" or "111". As a certain amount of time will be required for the high frequency clock to stabilise, the status of the HTO bit is checked. The amount of time required for high speed system oscillator stabilization depends upon which high speed system oscillator type is used.

Note: This device not support Sleep 0 mode (WDT always enabled).


NORMAL Mode
CKS2 ~ CKS0 = 00xB &
HLCLK = 0
SLOW Mode
WDT and LVD are all off
IDLEN = 0
HALT instruction is executed
SLEEP0 Mode
WDT or LVD is on
IDLEN = 0
HALT instruction is executed
SLEEP1 Mode
IDLEN = 1, FSYSON=0
HALT instruction is executed
IDLE0 Mode
IDLEN = 1, FSYSON=1
HALT instruction is executed
IDLE1 Mode
SLOW Mode
CKS2~CKS0≠000B, 001B as HLCLK=0
or HLCLK=1
NORMAL Mode
WDT and LVD are all off
IDLEN = 0
HALT instruction is executed
SLEEP0 Mode
WDT or LVD is on
IDLEN = 0
HALT instruction is executed
SLEEP1 Mode
IDLEN = 1, FSYSON=0
HALT instruction is executed
IDLE0 Mode
IDLEN = 1, FSYSON=1
HALT instruction is executed
IDLE1 Mode

### Entering the SLEEP1 Mode

There is only one way for the device to enter the SLEEP1 Mode and that is to execute the "HALT" instruction in the application program with the IDLEN bit in SMOD register equal to "0" and the WDT is on. When this instruction is executed under the conditions described above, the following will occur:

- The system clock will be stopped and the application program will stop at the "HALT" instruction, but the WDT function will remain with the clock source coming from the $f_{SUB}$ clock.
- The Data Memory contents and registers will maintain their present condition.
- The WDT will be cleared and resume counting as the WDT is enabled.
- The I/O ports will maintain their present conditions.
- In the status register, the Power Down flag, PDF, will be set and the Watchdog time-out flag, TO, will be cleared.

### Entering the IDLE0 Mode

There is only one way for the device to enter the IDLE0 Mode and that is to execute the "HALT" instruction in the application program with the IDLEN bit in SMOD register equal to "1" and the FSYSON bit in SMOD1 register equal to "0". When this instruction is executed under the conditions described above, the following will occur:

- The system clock will be stopped and the application program will stop at the "HALT" instruction, but the $f_{SUB}$ clock will be on.
- The Data Memory contents and registers will maintain their present condition.
- The WDT will be cleared and resume counting as the WDT clock source is derived from the $f_{SUB}$ clock.
- The I/O ports will maintain their present conditions.
- In the status register, the Power Down flag, PDF, will be set and the Watchdog time-out flag, TO, will be cleared.

### Entering the IDLE1 Mode

There is only one way for the device to enter the IDLE1 Mode and that is to execute the "HALT" instruction in the application program with the IDLEN bit in SMOD register equal to "1" and the FSYSON bit in SMOD1 register equal to "1". When this instruction is executed under the conditions described above, the following will occur:

- The system clock and $f_{SUB}$ clock will be on 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 as the WDT clock source is derived from the $f_{SUB}$ clock.
- The I/O ports will maintain their present conditions.
- In the status register, the Power Down flag, PDF, will be set and the Watchdog time-out flag, TO, will be cleared.

### Standby Current Considerations

As the main reason for entering the SLEEP or IDLE Mode is to keep the current consumption of these devices to as low a value as possible, perhaps only in the order of several micro-amps except in the IDLE1 Mode, there are other considerations which must also be taken into account by the circuit designer if the power consumption is to be minimised. Special attention must be made to the I/O pins on these devices. All high-impedance input pins must be connected to either a fixed high or low level as any floating input pins could create internal oscillations and result in increased current consumption. This also applies to devices which have different package types, as there may be unbonded pins. These must either be setup as outputs or if setup as inputs must have pull-high resistors connected.

Care must also be taken with the loads, which are connected to I/O pins, which are setup as outputs. These should be placed in a condition in which minimum current is drawn or connected only to external circuits that do not draw current, such as other CMOS inputs. Also note that additional standby current will also be required if the LXT oscillator has been enabled.

In the IDLE1 Mode the system oscillator is on, if the system oscillator is from the high speed system oscillator, the additional standby current will also be perhaps in the order of several hundred micro-amps.

### Wake-up

After the system enters the SLEEP or IDLE 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 using the PAWU register to permit a negative transition on the pin to wake-up the system. When a Port A pin wake-up occurs, the program will resume execution at the instruction following the "HALT" instruction. If the system is woken up by an interrupt, then two possible situations may occur. The first is where the related interrupt is disabled or the interrupt is enabled but the stack is full, in which case the program will resume execution at the instruction following the "HALT" instruction. In this situation, the interrupt which woke-up these devices will not be immediately serviced, but will rather be serviced later when the related interrupt is finally enabled or when a stack level becomes free. The other situation is where the related interrupt is enabled and the stack is not full, in which case the regular interrupt response takes place. If an interrupt request flag is set high before entering the SLEEP or IDLE Mode, the wake-up function of the related interrupt will be disabled.

## Programming Considerations

The HXT and LXT oscillators both use the same SST counter. For example, if the system is woken up from the SLEEP0 Mode and both the HXT and LXT oscillators need to start-up from an off state. The LXT oscillator uses the SST counter after HXT oscillator has finished its SST period.

- If the device is woken up from the SLEEP0 Mode to the NORMAL Mode, the high speed system oscillator needs an SST period. The device will execute first instruction after HTO is "1". At this time, the LXT oscillator may not be stable as the $f_{SUB}$ is from the LXT oscillator. The same situation occurs in the power-on state. The LXT oscillator is not ready yet when the first instruction is executed.
- If the device is woken up from the SLEEP1 Mode to NORMAL Mode, and the system clock source is from HXT oscillator and FSTEN is "1", the system clock can be switched to the LXT oscillator after wake up.
- There are peripheral functions, such as Timer/Event counters, for which the $f_{SYS}$ is used. If the system clock source is switched from $f_H$ to $f_{SUB}$, the clock source to the peripheral functions mentioned above will change accordingly.
- The on/off condition of $f_{SUB}$ depends upon whether the WDT is enabled or disabled as the WDT clock source is selected from $f_{SUB}$.

## Low Voltage Detector − LVD

The 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.

### LVD Operation

The overall function of the LVD is enabled using the LVDEN bit in the LVDC register. The LVDEN bit is the enable/disable control bit when set low the overall function of the LVD will be disabled. The LVDO bit is the LVD detector output bit. 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 LVDC 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 LVDEN bit in the LVDC 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 LVDEN 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. Note that the LVD function will be automatically disabled when the device enters the power down mode.

There are a range of voltage values, selected using the software selection bits, which can be chosen to activate the LVD.

#### LVDC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | LVDO | LVDEN | — | VLVD2 | VLVD1 | VLVD0 |
| R/W | — | — | R | R/W | — | R/W | R/W | R/W |
| POR | — | — | 0 | 0 | — | 0 | 0 | 0 |

Bit 7~6 unimplemented, read as "0"

Bit 5 **LVDO:** LVD Output Flag
0: No Low Voltage Detected
1: Low Voltage Detected

Bit 4 **LVDEN:** Low Voltage Detector Enable/Disable
0: Disable
1: Enable

Bit 3 unimplemented, read as "0"

Bit 2~0 **VLVD2 ~ VLVD0:** Select LVD Voltage
000: 2.0V
001: 2.2V
010: 2.4V
011: 2.7V
100: 3.0V
101: 3.3V
110: 3.6V
111: 4.0V

# 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.

## Watchdog Timer Clock Source

The Watchdog Timer clock source is provided by the $f_{SUB}$ clock. The $f_{SUB}$ clock is sourced from the LXT oscillator selected. The LXT oscillator is supplied by an external 32.768 kHz crystal. The Watchdog Timer source clock is then subdivided by a ratio of $2^8$ to $2^{18}$ to give longer timeouts, the actual value being chosen using the WS2~WS0 bits in the WDTC register.

## Watchdog Timer Control Regsiter

A single register, WDTC, controls the required timeout period as well as the enable operation. This register controls the overall operation of the Watchdog Timer.

### WDTC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | WE4 | WE3 | WE2 | WE1 | WE0 | WS2 | WS1 | WS0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |

Bit 7~3 **WE4 ~ WE0:** WDT function enable control
01010: Enabled
10101: Enabled (WDT function is always enable)
Other Values: Reset MCU

If these bits are changed due to adverse environmental conditions, the microcontroller will be reset. The reset operation will be activated after 2~3 LIRC clock cycles and the WRF bit in the SMOD1 register will be set to 1.

Bit 2~0 **WS2 ~ WS0:** Select WDT Time-out Period
000: $2^7 / f_{SUB} \sim 2^8 / f_{SUB}$
001: $2^9 / f_{SUB} \sim 2^{10} / f_{SUB}$
010: $2^{11} / f_{SUB} \sim 2^{12} / f_{SUB}$
011: $2^{13} / f_{SUB} \sim 2^{14} / f_{SUB}$
100: $2^{14} / f_{SUB} \sim 2^{15} / f_{SUB}$
101: $2^{15} / f_{SUB} \sim 2^{16} / f_{SUB}$
110: $2^{16} / f_{SUB} \sim 2^{17} / f_{SUB}$
111: $2^{17} / f_{SUB} \sim 2^{18} / f_{SUB}$

These three bits determine the division ratio of the Watchdog Timer source clock, which in turn determines the timeout period.

**SMOD1 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | FSYSON | — | — | — | — | LVRF | LRF | WRF |
| R/W | R/W | — | — | — | — | R/W | R/W | R/W |
| POR | 0 | — | — | — | — | x | 0 | 0 |

Bit 7 **FSYSON:** $f_{SYS}$ Control in IDLE Mode
Described elsewhere.

Bit 6~3 Unimplemented, read as "0"

Bit 2 **LVRF:** LVR function reset flag
Described elsewhere.

Bit 1 **LVF:** LVR Control register software reset flag
Described elsewhere.

Bit 0 **WRF:** WDT Control register software reset flag
0: not occurred
1: occurred
This bit is set to 1 by the WDT Control register software reset and cleared by the application program. Note that this bit can only be cleared to 0 by the application program.

## Watchdog Timer Operation

The Watchdog Timer operates by providing a device reset when its timer overflows. This means that in the application program and during normal operation the user has to strategically clear the Watchdog Timer before it overflows to prevent the Watchdog Timer from executing a reset. This is done using the clear watchdog instructions. If the program malfunctions for whatever reason, jumps to an unknown location, or enters an endless loop, these clear instructions will not be executed in the correct manner, in which case the Watchdog Timer will overflow and reset the device. With regard to the Watchdog Timer enable function, there are five bits, WE4~WE0, in the WDTC register to offer the enable control and reset control of the Watchdog Timer. The WDT function is always enabled when the WE4~WE0 bits are set to a value of 10101B or 01010B. If the WE4~WE0 bits are set to any other values, other than 01010B and 10101B, it will reset the device after 2~3 LIRC clock cycles. The LIRC internal oscillator has an approximate frequency of 32kHz and this specified internal clock period can vary with $V_{DD}$, temperature and process variations. After power on the WE4~WE0 bits will have a value of 01010B.

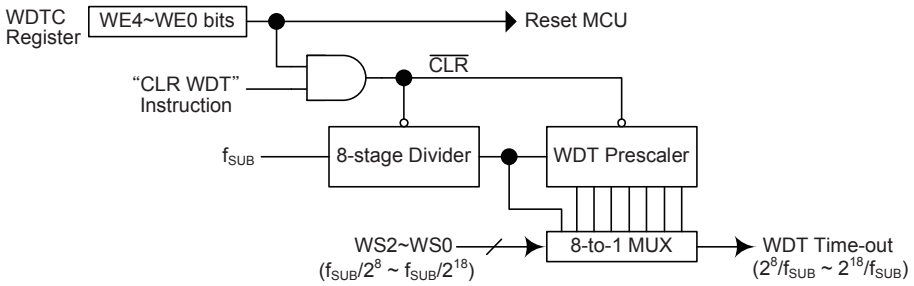
| WDT Function Control | WE4 ~ WE0 Bits | WDT Function |
|---|---|---|
| Always Enabled | 01010B | Enable |
| | 10101B | Enable |
| | Any other value | Reset MCU |

**Watchdog Timer Enable/Disable Control**

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 SLEEP or IDLE 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 a WDT reset, which means a certain value except 01010B and 10101B written into the WE4~WE0 field, the second is using the Watchdog Timer software clear instruction and the third is via a HALT instruction.

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

The maximum time out period is when the $2^{18}$ division ratio is selected. As an example, with a 32.768kHz LXT oscillator as its source clock, this will give a maximum watchdog period of around 8 second for the $2^{18}$ division ratio, and a minimum time-out of 7.8ms for the $2^8$ division ration.

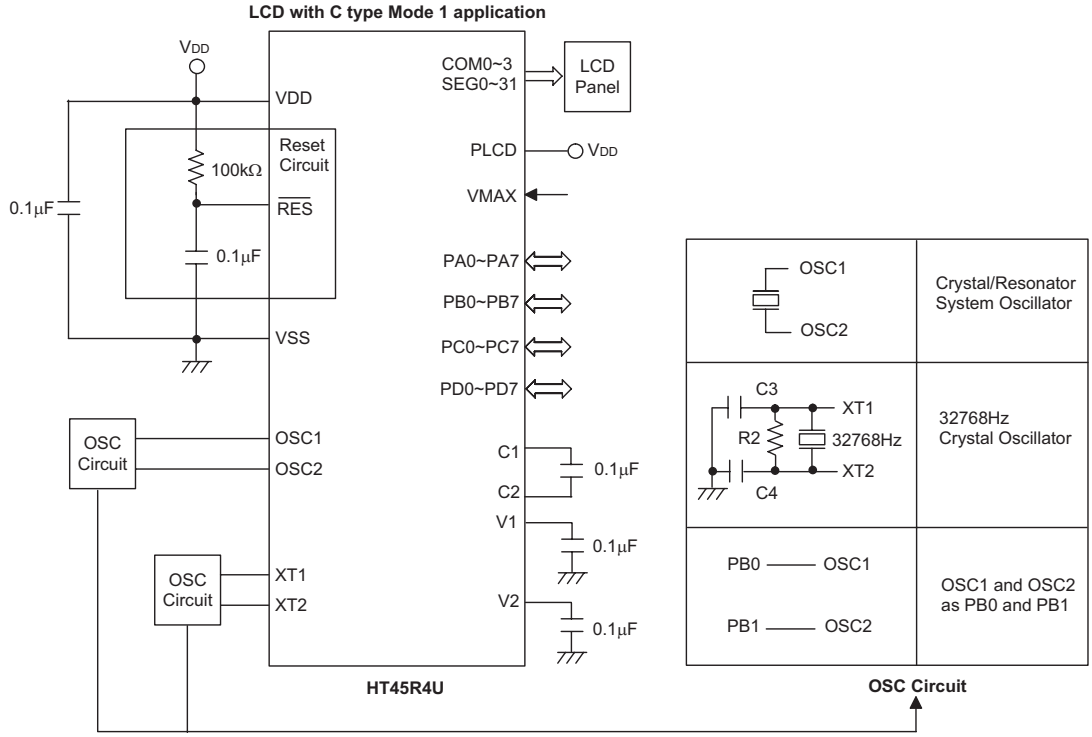


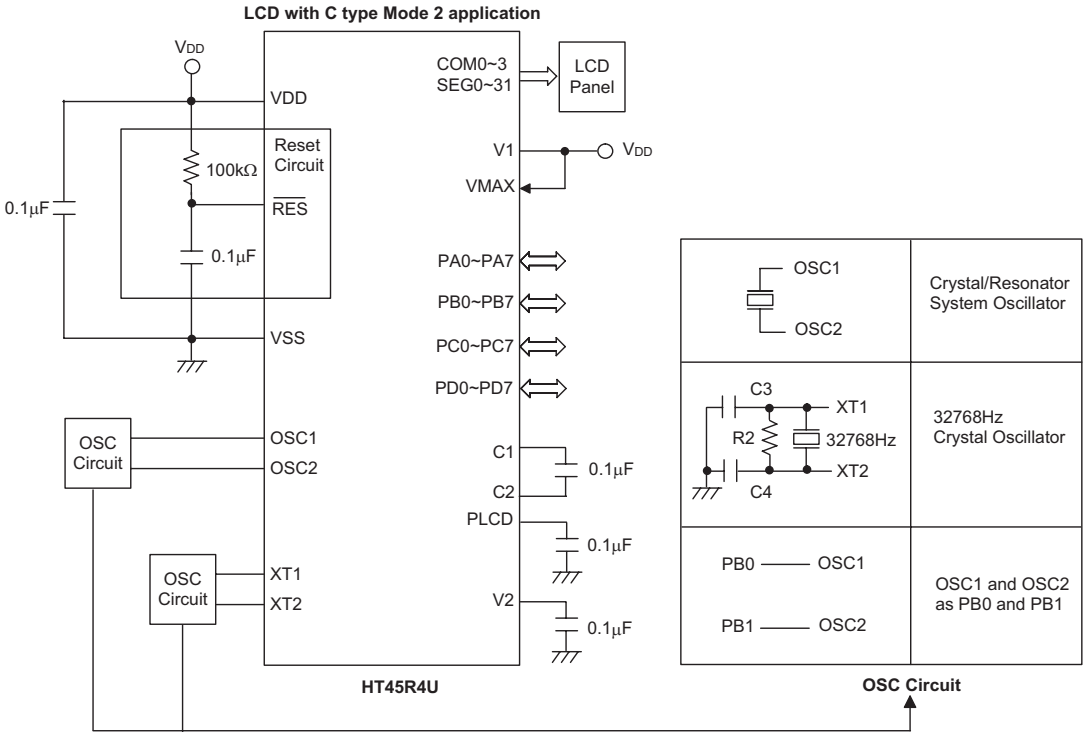

**Watchdog Timer**

## Configuration Options

| No. | Options |
|---|---|
| 1 | High Speed System Oscillator Selection – $f_H$<br>HXT, HIRC, External clock with filter on, External clock with filter off |
| 2 | HXT mode Selection<br>455kHz, 1MHz~12MHz |
| 3 | I/O or Reset pin Selection<br>I/O pin, External Reset pin |
| 4 | Interrupt and Timer/Event Counter Input Pins Filter Control<br>Filter on, Filter off |

## Application Circuits



# Instruction Set

## Introduction

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

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

## Instruction Timing

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

## Moving and Transferring Data

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

## Arithmetic Operations

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

### Logical and Rotate Operation

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

### Branches and Control Transfer

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

### Bit Operations

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

### Table Read Operations

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

### Other Operations

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

## Instruction Set Summary

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

### Table Conventions

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

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

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

Note: 1. For skip instructions, if the result of the comparison involves a skip then two cycles are required, if no skip takes place only one cycle is required.

2. Any instruction which changes the contents of the PCL will also require 2 cycles for execution.

3. For the "CLR WDT1" and "CLR WDT2" instructions the TO and PDF flags may be affected by the execution status. The TO and PDF flags are cleared after both "CLR WDT1" and "CLR WDT2" instructions are consecutively executed. Otherwise the TO and PDF flags remain unchanged.

## Instruction Definition

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

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

Operation ACC ← ACC + [m] + C

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

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

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

Operation [m] ← ACC + [m] + C

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

**ADD A,[m]** Add Data Memory to ACC

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

Operation ACC ← ACC + [m]

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

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

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

Operation ACC ← ACC + x

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

**ADDM A,[m]** Add ACC to Data Memory

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

Operation [m] ← ACC + [m]

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

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

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

Operation ACC ← ACC "AND" [m]

Affected flag(s) Z

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

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

Operation ACC ← ACC "AND" x

Affected flag(s) Z

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

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

Operation [m] ← ACC "AND" [m]

Affected flag(s) Z

**CALL addr** Subroutine call

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

Operation Stack ← Program Counter + 1
Program Counter ← addr

Affected flag(s) None

**CLR [m]** Clear Data Memory

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

Operation [m] ← 00H

Affected flag(s) None

**CLR [m].i** Clear bit of Data Memory

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

Operation [m].i ← 0

Affected flag(s) None

**CLR WDT** Clear Watchdog Timer

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

Operation WDT cleared
TO ← 0
PDF ← 0

Affected flag(s) TO, PDF

**CLR WDT1** Pre-clear Watchdog Timer

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

Operation WDT cleared
TO ← 0
PDF ← 0

Affected flag(s) TO, PDF

**CLR WDT2** Pre-clear Watchdog Timer

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

Operation WDT cleared
TO ← 0
PDF ← 0

Affected flag(s) TO, PDF

**CPL [m]** Complement Data Memory

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

Operation [m] ← $\overline{[m]}$

Affected flag(s) Z

**CPLA [m]** Complement Data Memory with result in ACC

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

Operation ACC ← $\overline{[m]}$

Affected flag(s) Z

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

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

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

Affected flag(s) C

**DEC [m]** Decrement Data Memory

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

Operation [m] ← [m] − 1

Affected flag(s) Z

DECA [m] Decrement Data Memory with result in ACC

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

Operation ACC ← [m] − 1

Affected flag(s) Z

**HALT** Enter power down mode

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

Operation TO ← 0
PDF ← 1

Affected flag(s) TO, PDF

**INC [m]** Increment Data Memory

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

Operation [m] ← [m] + 1

Affected flag(s) Z

**INCA [m]** Increment Data Memory with result in ACC

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

Operation ACC ← [m] + 1

Affected flag(s) Z

**JMP addr** Jump unconditionally

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

Operation Program Counter ← addr

Affected flag(s) None

**MOV A,[m]** Move Data Memory to ACC

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

Operation ACC ← [m]

Affected flag(s) None

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

Description The immediate data specified is loaded into the Accumulator.

Operation ACC ← x

Affected flag(s) None

**MOV [m],A** Move ACC to Data Memory

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

Operation [m] ← ACC

Affected flag(s) None

**NOP** No operation

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

Operation No operation

Affected flag(s) None

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

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

Operation ACC ← ACC ″OR″ [m]

Affected flag(s) Z

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

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

Operation ACC ← ACC ″OR″ x

Affected flag(s) Z

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

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

Operation [m] ← ACC ″OR″ [m]

Affected flag(s) Z

**RET** Return from subroutine

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

Operation Program Counter ← Stack

Affected flag(s) None

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

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

Operation Program Counter ← Stack
ACC ← x

Affected flag(s) None

**RETI** Return from interrupt

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

Operation Program Counter ← Stack
EMI ← 1

Affected flag(s) None

**RL [m]** Rotate Data Memory left

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

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

Affected flag(s) None

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

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

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

Affected flag(s) None

**RLC [m]** Rotate Data Memory left through Carry

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

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

Affected flag(s) C

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

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

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

Affected flag(s) C

**RR [m]** Rotate Data Memory right

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

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

Affected flag(s) None

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

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

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

Affected flag(s) None

**RRC [m]** Rotate Data Memory right through Carry

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

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

Affected flag(s) C

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

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

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

Affected flag(s) C

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

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

Operation ACC ← ACC − [m] − C

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

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

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

Operation [m] ← ACC − [m] − C

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

**SDZ [m]** Skip if decrement Data Memory is 0

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

Operation [m] ← [m] − 1
Skip if [m]=0

Affected flag(s) None

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

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

Operation ACC ← [m] − 1
Skip if ACC=0

Affected flag(s) None

**SET [m]** Set Data Memory

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

Operation [m] ← FFH

Affected flag(s) None

**SET [m].i** Set bit of Data Memory

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

Operation [m].i ← 1

Affected flag(s) None

**SIZ [m]** Skip if increment Data Memory is 0

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

Operation [m] ← [m] + 1
Skip if [m]=0

Affected flag(s) None

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

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

Operation ACC ← [m] + 1
Skip if ACC=0

Affected flag(s) None

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

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

Operation Skip if [m].i ≠ 0

Affected flag(s) None

**SUB A,[m]** Subtract Data Memory from ACC

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

Operation ACC ← ACC − [m]

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

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

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

Operation [m] ← ACC − [m]

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

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

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

Operation ACC ← ACC − x

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

**SWAP [m]** Swap nibbles of Data Memory

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

Operation [m].3~[m].0 ↔ [m].7~[m].4

Affected flag(s) None

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

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

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

Affected flag(s) None

**SZ [m]** Skip if Data Memory is 0

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

Operation Skip if [m]=0

Affected flag(s) None

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

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

Operation ACC ← [m]
Skip if [m]=0

Affected flag(s) None

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

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

Operation Skip if [m].i=0

Affected flag(s) None

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

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

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

Affected flag(s) None

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

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

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

Affected flag(s) None

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

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

Operation ACC ← ACC ″XOR″ [m]

Affected flag(s) Z

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

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

Operation [m] ← ACC ″XOR″ [m]

Affected flag(s) Z

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

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

Operation ACC ← ACC ″XOR″ x

Affected flag(s) Z

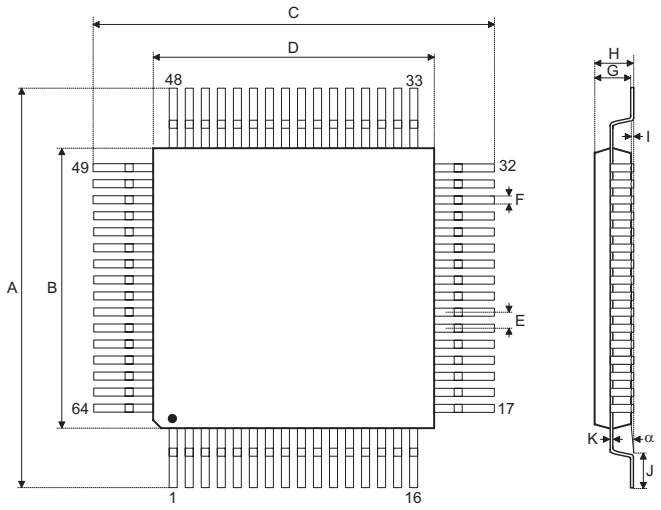
## Package Information

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

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

- Package Information (include Outline Dimensions, Product Tape and Reel Specifications)
- The Operation Instruction of Packing Materials
- Carton information

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




| Symbol | Dimensions in inch | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 0.354 BSC | — |
| B | — | 0.276 BSC | — |
| C | — | 0.354 BSC | — |
| D | — | 0.276 BSC | — |
| E | — | 0.016 BSC | — |
| F | 0.005 | 0.007 | 0.009 |
| G | 0.053 | 0.055 | 0.057 |
| H | — | — | 0.063 |
| I | 0.002 | — | 0.006 |
| J | 0.018 | 0.024 | 0.030 |
| K | 0.004 | — | 0.008 |
| α | 0° | — | 7° |

| Symbol | Dimensions in mm | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 9.0 BSC | — |
| B | — | 7.0 BSC | — |
| C | — | 9.0 BSC | — |
| D | — | 7.0 BSC | — |
| E | — | 0.4 BSC | — |
| F | 0.13 | 0.18 | 0.23 |
| G | 1.35 | 1.40 | 1.45 |
| H | — | — | 1.60 |
| I | 0.05 | — | 0.15 |
| J | 0.45 | 0.60 | 0.75 |
| K | 0.09 | — | 0.20 |
| α | 0° | — | 7° |