Touch A/D Flash MCU

# BS84B08A-3/BS84C12A-3

Revision: V1.50 Date: December 26, 2018

www.holtek.com

# Table of Contents

**Features** ..... **6**
CPU Features ..... 6
Peripheral Features ..... 6
**General Description** ..... **7**
**Selection Table** ..... **8**
**Block Diagram** ..... **8**
**Pin Assignment** ..... **9**
**Pin Descriptions** ..... **10**
**Absolute Maximum Ratings** ..... **12**
**A.C. Characteristics** ..... **12**
**D.C. Characteristics** ..... **13**
**ADC Electrical Characteristics** ..... **14**
**Power-on Reset Characteristics** ..... **15**
**System Architecture** ..... **16**
Clocking and Pipelining ..... 16
Program Counter ..... 17
Stack ..... 18
Arithmetic and Logic Unit – ALU ..... 18
**Flash Program Memory** ..... **19**
Structure ..... 19
Special Vectors ..... 19
Look-up Table ..... 20
Table Program Example ..... 21
In Circuit Programming ..... 22
On-Chip Debug Support – OCDS ..... 23
**RAM Data Memory** ..... **23**
Structure ..... 23
**Special Function Register Description** ..... **24**
Indirect Addressing Registers – IAR0, IAR1 ..... 24
Memory Pointers – MP0, MP1 ..... 25
Bank Pointer – BP ..... 26
Accumulator – ACC ..... 26
Program Counter Low Register – PCL ..... 27
Look-up Table Registers – TBLP, TBHP, TBLH ..... 27
Status Register – STATUS ..... 27

**EEPROM Data Memory ..... 29**
EEPROM Data Memory Structure ..... 29
EEPROM Registers ..... 29
Reading Data from the EEPROM ..... 31
Writing Data to the EEPROM ..... 31
Write Protection ..... 31
EEPROM Interrupt ..... 31
Programming Considerations ..... 32
**Oscillator ..... 33**
Oscillator Overview ..... 33
System Clock Configurations ..... 33
Internal RC Oscillator – HIRC ..... 34
Internal 32kHz Oscillator – LIRC ..... 34
**Operating Modes and System Clocks ..... 34**
System Clocks ..... 34
System Operation Modes ..... 35
Control Register ..... 36
Operating Mode Switching ..... 38
Standby Current Considerations ..... 40
Wake-up ..... 41
**Watchdog Timer ..... 42**
Watchdog Timer Clock Source ..... 42
Watchdog Timer Control Register ..... 42
Watchdog Timer Operation ..... 43
**Reset and Initialisation ..... 44**
Reset Functions ..... 44
Reset Initial Conditions ..... 47
**Input/Output Ports ..... 52**
I/O Register List ..... 52
Pull-high Resistors ..... 53
Port A Wake-up ..... 54
I/O Port Control Registers ..... 54
I/O Pin Structures ..... 56
Programming Considerations ..... 57
**Timer/Event Counter ..... 57**
Configuring the Timer/Event Counter Input Clock Source ..... 57
Timer Register – TMR ..... 58
Timer Control Register – TMRC ..... 58
Timer Operation ..... 59
Prescaler ..... 59
Programming Considerations ..... 59

**Analog to Digital Converter ..... 60**
A/D Overview ..... 60
A/D Converter Register Description ..... 60
A/D Converter Data Registers – ADRL, ADRH ..... 61
A/D Converter Control Registers – ADCR0, ADCR1, ACERL ..... 61
A/D Operation ..... 64
A/D Input Pins ..... 65
Summary of A/D Conversion Steps ..... 66
Programming Considerations ..... 67
A/D Transfer Function ..... 67
A/D Programming Examples ..... 68
**Touch Key Function ..... 70**
Touch Key Structure ..... 70
Touch Key Register Definition ..... 70
Touch Key Operation ..... 76
Touch Key Interrupt ..... 78
Programming Considerations ..... 79
**Serial Interface Module – SIM ..... 79**
SPI Interface ..... 79
$I^2C$ Interface ..... 85
**Interrupts ..... 94**
Interrupt Registers ..... 94
Interrupt Operation ..... 96
External Interrupt ..... 97
Time Base Interrupt ..... 97
Timer/Event Counter Interrupt ..... 98
EEPROM Interrupt ..... 98
Touch Key Interrupt ..... 99
SIM Interrupt ..... 99
A/D Converter Interrupt ..... 99
Interrupt Wake-up Function ..... 99
Programming Considerations ..... 100
**Application Circuits ..... 100**
**Instruction Set ..... 101**
Introduction ..... 101
Instruction Timing ..... 101
Moving and Transferring Data ..... 101
Arithmetic Operations ..... 101
Logical and Rotate Operation ..... 102
Branches and Control Transfer ..... 102
Bit Operations ..... 102
Table Read Operations ..... 102
Other Operations ..... 102

**Instruction Set Summary ..... 103**
Table Conventions..... 103
**Instruction Definition..... 105**
**Package Information .....114**
16-pin NSOP (150mil) Outline Dimensions.....115
20-pin SOP (300mil) Outline Dimensions .....116
20-pin NSOP (150mil) Outline Dimensions.....117
20-pin SSOP (150mil) Outline Dimensions.....118
24-pin SOP (300mil) Outline Dimensions .....119
24-pin SSOP(150mil) Outline Dimensions..... 120
28-pin SOP (300mil) Outline Dimensions ..... 121
28-pin SSOP (150mil) Outline Dimensions..... 122

# Features

## CPU Features

- Operating Voltage
  - $f_{SYS}$ = 8MHz: 2.7V~5.5V
  - $f_{SYS}$ = 12MHz: 2.7V~5.5V
  - $f_{SYS}$ = 16MHz: 4.5V~5.5V
- Up to 0.25μs instruction cycle with 16MHz system clock at $V_{DD}$=5V
- Fully integrated 8/12 touch key functions -- require no external components
- Power down and wake-up functions to reduce power consumption
- Fully integrated low and high speed internal oscillators
  - Low Speed – 32kHz
  - High speed – 8MHz, 12MHz, 16MHz
- Multi-mode operation: NORMAL, SLOW, IDLE and SLEEP
- All instructions executed in one or two instruction cycles
- Table read instructions
- 63 powerful instructions
- Up to 6-level subroutine nesting
- Bit manipulation instruction

## Peripheral Features

- Flash Program Memory: 3K×16 ~ 4K×16
- RAM Data Memory: 288×8 ~ 384×8
- True EEPROM Memory: 64×8 ~ 128×8
- Watchdog Timer function
- Up to 26 bidirectional I/O lines
- External interrupt line shared with I/O pin
- Single 8-bit Timer/Event Counter
- Single Time-Base function for generation of fixed time interrupt signals
- Multi-channel 12-bit resolution A/D converter
- $I^2C$ and SPI interfaces
- Low voltage reset function
- 8/12 touch key functions
- High current LED driver

## General Description

These devices are a series of Flash Memory A/D type 8-bit high performance RISC architecture microcontrollers with fully integrated touch key functions. With all touch key functions provided internally and with the convenience of Flash Memory multi-programming features, this device range has all the features to offer designers a reliable and easy means of implementing Touch Keyes within their products applications.

Analog feature include a multi-channel 12-bit A/D converter. Protective features such as an internal Watchdog Timer and Low Voltage Reset coupled with excellent noise immunity and ESD protection ensure that reliable operation is maintained in hostile electrical environments.

The touch key functions are fully integrated completely eliminating the need for external components. In addition to the flash program memory, other memory includes an area of RAM Data Memory as well as an area of true EEPROM memory for storage of non-volatile data such as serial numbers, calibration data etc.

All devices include fully integrated low and high speed oscillators which require no external components for their implementation. The ability to operate and switch dynamically between a range of operating modes using different clock sources gives users the ability to optimise microcontroller operation and minimise power consumption. Easy communication with the outside world is provided using the internal $I^2C$ and SPI interfaces, while the inclusion of flexible I/O programming features, Timer/Event Counter and many other features further enhance device functionality and flexibility.

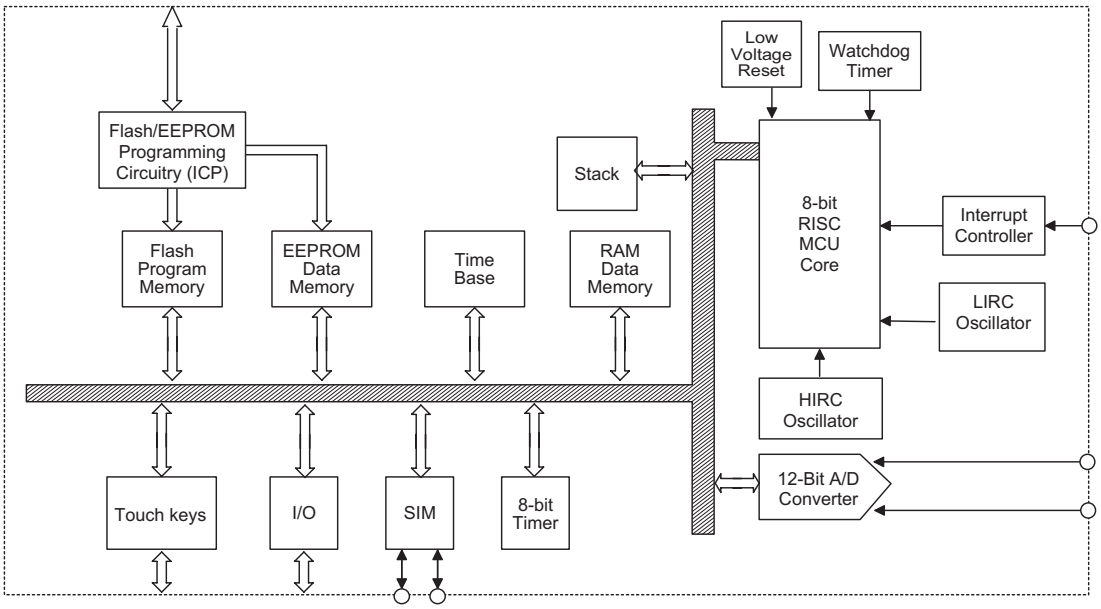
These touch key devices will find excellent use in a huge range of modern Touch Key product applications such as instrumentation, household appliances, electronically controlled tools to name but a few.

## Selection Table

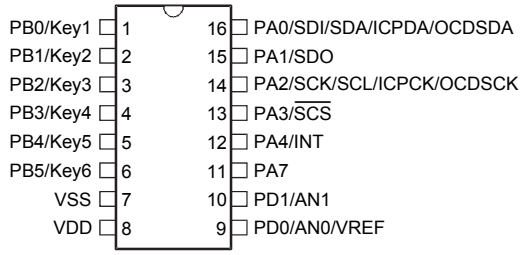
Most features are common to all devices, the main distinguishing feature is the number of I/Os and Touch Keys. The following table summarises the main features of each device.

| Part No. | Internal Clock | VDD | System Clock | Program Memory | Data Memory | Data EEPROM | I/O | High Current LED Output | A/D | 8-bit Timer | Touch Key | SPI/ I²C | LVR | Stack | Package |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| BS84B08A-3 | 8MHz<br>12MHz<br>16MHz | 2.7V~<br>5.5V | 8MHz~<br>16MHz | 3K×16 | 288×8 | 64×8 | 22 | 22 | 12-bit<br>×8 | 1 | 8 | 1 | 2.55V | 6 | 16NSOP<br>20SOP/NSOP/SSOP<br>24SOP |
| BS84C12A-3 | 8MHz<br>12MHz<br>16MHz | 2.7V~<br>5.5V | 8MHz~<br>16MHz | 4K×16 | 384×8 | 128×8 | 26 | 26 | 12-bit<br>×8 | 1 | 12 | 1 | 2.55V | 6 | 20/24/28<br>SOP/SSOP |

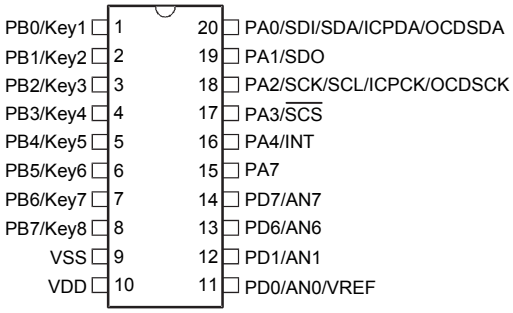
## Block Diagram



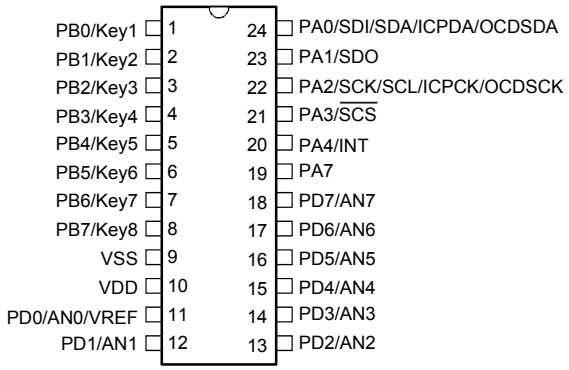
## Pin Assignment



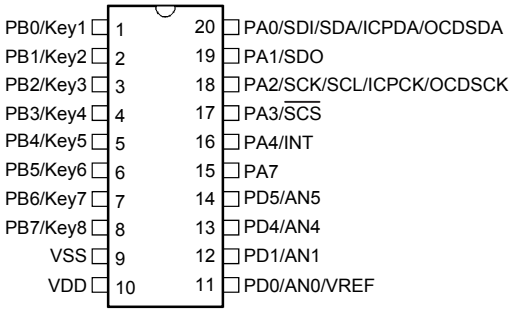

**BS84B08A-3/BS84V08A**
**16 NSOP-A**



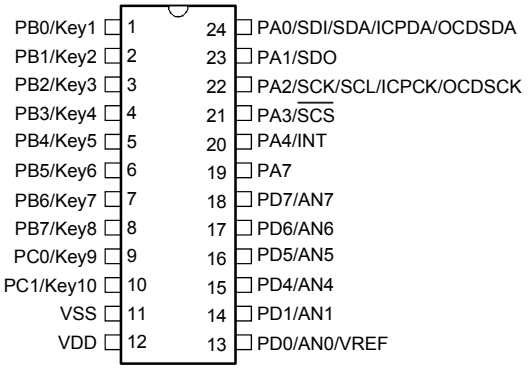

**BS84B08A-3/BS84V08A**
**20 SOP-A/NSOP-A/SSOP-A**



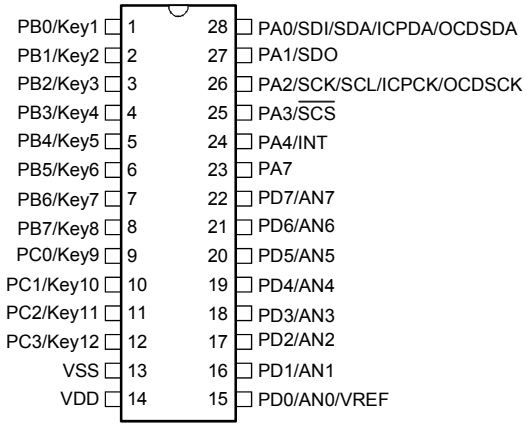

**BS84B08A-3/BS84V08A**
**24 SOP-A**




**BS84C12A-3/BS84V12A**
**20 SOP-A/SSOP-A**




**BS84C12A-3/BS84V12A**
**24 SOP-A/SSOP-A**




**BS84C12A-3/BS84V12A**
**28 SOP-A/SSOP-A**

Note: 1. The OCDSDA and OCDSCK pins are used for OCDS function while the ICPDA and ICPCK pins are used for ICP function.

2. For the less pin count package type there will be unbounded pins which should be properly configured to avoid unwanted power consumption resulting from floating input conditions. Refer to the Standby Current Considerations" and "Input/Output Ports" sections.

## Pin Descriptions

The function of each pin is listed in the following tables, however the details behind how each pin is configured is contained in other sections of the datasheet.

**BS84B08A-3**

| Pin Name | Function | OPT | I/T | O/T | Description |
|---|---|---|---|---|---|
| PA0/SDI/ SDA/ICPDA/ OCDSDA | PA0 | PAWU PAPU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| | SDI | — | ST | — | SPI data input |
| | SDA | — | ST | NMOS | I²C data |
| | ICPDA | — | ST | CMOS | In-circuit programming address/data pin |
| | OCDSDA | — | ST | CMOS | On-chip debug support data/address pin, for EV chip only. |
| PA1/SDO | PA1 | PAWU PAPU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| | SDO | SIMC0 | — | CMOS | SPI data output |
| PA2/SCK/ SCL/ICPCK/ OCDSCK | PA2 | PAWU PAPU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| | SCK | SIMC0 | ST | CMOS | SPI serial clock |
| | SCL | SIMC0 | ST | NMOS | I²C clock |
| | ICPCK | — | ST | — | In-circuit programming clock pin |
| | OCDSCK | — | ST | — | On-chip debug support clock pin, for EV chip only. |
| PA3/$\overline{SCS}$ | PA3 | PAWU PAPU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| | $\overline{SCS}$ | SIMC0 | ST | CMOS | SPI slave select |
| PA4/INT | PA4 | PAWU PAPU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| | INT | INTEG | ST | — | External interrupt |
| PA7 | PA7 | PAWU PAPU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| PB0/KEY1~ PB3/KEY4 | PB0~PB3 | PBPU | ST | CMOS | General purpose I/O. Register enabled pull-up. |
| | KEY1~ KEY4 | TKM0C1 | NSI | — | Touch key inputs |
| PB4/KEY5~ PB7/KEY8 | PB4~PB7 | PBPU | ST | CMOS | General purpose I/O. Register enabled pull-up. |
| | KEY5~ KEY8 | TKM1C1 | NSI | — | Touch key inputs |
| PD0/AN0/ VREF | PD0 | PDPU | ST | CMOS | General purpose I/O. Register enabled pull-up. |
| | AN0 | ACERL | AN | — | ADC input |
| | VREF | ADCR1 | AN | — | ADC reference input |
| PD1/AN1~ PD7/AN7 | PD1~PD7 | PDPU | ST | CMOS | General purpose I/O. Register enabled pull-up. |
| | AN1~AN7 | ACERL | AN | — | ADC input |
| VDD | VDD | — | PWR | — | Power supply |
| VSS | VSS | — | PWR | — | Ground |

Legend: I/T: Input type; O/T: Output type

OP: Optional by register selection

AN: Analog input pin; PWR: Power

ST: Schmitt Trigger input; CMOS: CMOS output

NMOS: NMOS output; NSI: Non-standard input

**BS84C12A-3**

| Pin Name | Function | OPT | I/T | O/T | Description |
|---|---|---|---|---|---|
| PA0/SDI/SDA/ICPDA/OCDSDA | PA0 | PAWU PAPU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| | SDI | — | ST | — | SPI data input |
| | SDA | — | ST | NMOS | I²C data |
| | ICPDA | — | ST | CMOS | In-circuit programming address/data pin |
| | OCDSDA | — | ST | CMOS | On-chip debug support data/address pin, for EV chip only. |
| PA1/SDO | PA1 | PAWU PAPU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| | SDO | SIMC0 | — | CMOS | SPI data output |
| PA2/SCK/SCL/ICPCK/OCDSCK | PA2 | PAWU PAPU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| | SCK | SIMC0 | ST | CMOS | SPI serial clock |
| | SCL | SIMC0 | ST | NMOS | I²C clock |
| | ICPCK | — | ST | — | In-circuit programming clock pin |
| | OCDSCK | — | ST | — | On-chip debug support clock pin, for EV chip only. |
| PA3/$\overline{SCS}$ | PA3 | PAWU PAPU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| | $\overline{SCS}$ | SIMC0 | ST | CMOS | SPI slave select |
| PA4/INT | PA4 | PAWU PAPU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| | INT | INTEG | ST | — | External interrupt |
| PA7 | PA7 | PAWU PAPU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| PB0/KEY1~PB3/KEY4 | PB0~PB3 | PBPU | ST | CMOS | General purpose I/O. Register enabled pull-up. |
| | KEY1~KEY4 | TKM0C1 | NSI | — | Touch key inputs |
| PB4/KEY5~PB7/KEY8 | PB4~PB7 | PBPU | ST | CMOS | General purpose I/O. Register enabled pull-up. |
| | KEY5~KEY8 | TKM1C1 | NSI | — | Touch key inputs |
| PC0/KEY9~PC3/KEY12 | PC0~PC3 | PCPU | ST | CMOS | General purpose I/O. Register enabled pull-up. |
| | KEY9~KEY12 | TKM2C1 | NSI | — | Touch key inputs |
| PD0/AN0/VREF | PD0 | PDPU | ST | CMOS | General purpose I/O. Register enabled pull-up. |
| | AN0 | ACERL | AN | — | ADC input |
| | VREF | ADCR1 | AN | — | ADC reference input |
| PD1/AN1~PD7/AN7 | PD1~PD7 | PDPU | ST | CMOS | General purpose I/O. Register enabled pull-up. |
| | AN1~AN7 | ACERL | AN | — | ADC input |
| VDD | VDD | — | PWR | — | Power supply |
| VSS | VSS | — | PWR | — | Ground |

Legend: I/T: Input type; O/T: Output type

OP: Optional by register selection

AN: Analog input pin; PWR: Power

ST: Schmitt Trigger input; CMOS: CMOS output

NMOS: NMOS output; NSI: Non-standard input

## 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
$I_{OH}$ Total ..... -80mA
$I_{OL}$ 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.

## A.C. Characteristics

Ta= 25˚C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $f_{SYS}$ | System Clock (HIRC) | 3.3V/5V | Ta=25˚C | -2% | 8 | +2% | MHz |
| | | | | -2% | 12 | +2% | MHz |
| | | 5V | Ta=25˚C | -2% | 16 | +2% | MHz |
| $f_{TIMER}$ | Timer Input Pin Frequency | 2.7V~5.5V | — | — | — | 8 | MHz |
| | | 2.7V~5.5V | | — | — | 12 | MHz |
| | | 4.5V~5.5V | | — | — | 16 | MHz |
| $f_{LIRC}$ | System Clock (32kHz) | 5V | Ta=25˚C | -10% | 32 | +10% | kHz |
| $t_{INT}$ | Interrupt Pulse Width | — | — | 1 | 5 | 10 | μs |
| $t_{LVR}$ | Low Voltage Width to Reset | — | — | 120 | 240 | 480 | μs |
| $t_{EERD}$ | EEPROM Read Time | — | — | 1 | 2 | 4 | $t_{SYS}$ |
| $t_{EEWR}$ | EEPROM Write Time | — | — | 1 | 2 | 4 | ms |
| $t_{SST}$ | System Start-up Timer Period (Wake-up from HALT) | — | $f_{SYS}$=HIRC | — | 15~16 | 20 | $t_{SYS}$ |
| | | | $f_{SYS}$=LIRC | — | 1~2 | 3 | |

Note: 1. $t_{SYS} = 1/f_{SYS}$
2. To maintain the accuracy of the internal HIRC oscillator frequency, a 0.1μF decoupling capacitor should be connected between VDD and VSS and located as close to the device as possible.
3. 16MHz can not be used when the supply voltage is below 3.3V.

## D.C. Characteristics

Ta= 25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $V_{DD}$ | Operating Voltage (HIRC) | — | $f_{SYS}$ = 8MHz | 2.7 | — | 5.5 | V |
| | | | $f_{SYS}$ = 12MHz | 2.7 | — | 5.5 | V |
| | | | $f_{SYS}$ = 16MHz | 4.5 | — | 5.5 | V |
| $I_{DD1}$ | Operating Current (HIRC, $f_{SYS}=f_H, f_S=f_{SUB}=f_{LIRC}$) | 3V | No load, $f_H$ = 8MHz ADC off, WDT enable | — | 1.2 | 1.8 | mA |
| | | 5V | | — | 2.2 | 3.3 | mA |
| | | 3V | No load, $f_H$ =12MHz ADC off, WDT enable | — | 1.6 | 2.4 | mA |
| | | 5V | | — | 3.3 | 5.0 | mA |
| | | 5V | No load, $f_H$ = 16MHz ADC off, WDT enable | — | 4.0 | 6.0 | mA |
| $I_{DD2}$ | Operating Current (HIRC, $f_{SYS}=f_L, f_S=f_{SUB}=f_{LIRC}$) | 3V | No load, $f_H$ =12MHz, $f_L$= $f_H$/2, ADC off, WDT enable | — | 1.2 | 2.0 | mA |
| | | 5V | | — | 2.2 | 3.3 | mA |
| | | 3V | No load, $f_H$=12MHz, $f_L$= $f_H$/4, ADC off, WDT enable | — | 1.0 | 1.5 | mA |
| | | 5V | | — | 1.8 | 2.7 | mA |
| | | 3V | No load, $f_H$ =12MHz, $f_L$= $f_H$/8, ADC off, WDT enable | — | 0.9 | 1.4 | mA |
| | | 5V | | — | 1.6 | 2.4 | mA |
| | | 3V | No load, $f_H$ =12MHz, $f_L$= $f_H$/16, ADC off, WDT enable | — | 0.8 | 1.2 | mA |
| | | 5V | | — | 1.5 | 2.3 | mA |
| | | 3V | No load, $f_H$=12MHz, $f_L$= $f_H$/32, ADC off, WDT enable | — | 0.8 | 1.2 | mA |
| | | 5V | | — | 1.5 | 2.3 | mA |
| | | 3V | No load, $f_H$ =12MHz, $f_L$= $f_H$/64, ADC off, WDT enable | — | 0.8 | 1.2 | mA |
| | | 5V | | — | 1.5 | 2.3 | mA |
| $I_{DD3}$ | Operating Current (LIRC, $f_{SYS}=f_L=f_{LIRC}$, $f_S=f_{SUB}=f_{LIRC}$) | 3V | No load, ADC off, WDT enable, LVR enable | — | 50 | 100 | μA |
| | | 5V | | — | 70 | 150 | μA |
| $I_{STB1}$ | IDLE Mode Standby Current (HIRC, $f_{SYS}=f_H, f_S=f_{SUB}=f_{LIRC}$) | 3V | No load, system HALT, ADC off, WDT enable, $f_{SYS}$ = 12MHz | — | 0.9 | 1.4 | mA |
| | | 5V | | — | 1.4 | 2.1 | mA |
| $I_{STB2}$ | IDLE Mode Standby Current (HIRC, $f_{SYS}$=off, $f_S=f_{SUB}=f_{LIRC}$) | 3V | No load, system HALT, ADC off, WDT enable, $f_{SYS}$ = 12MHz, LVR enable | — | 40 | 80 | μA |
| | | 5V | | — | 50 | 100 | μA |
| $I_{STB3}$ | IDLE Mode Standby Current (HIRC, $f_{SYS}= f_L, f_S=f_{SUB}=f_{LIRC}$) | 3V | No load, system HALT, ADC off, WDT enable, $f_{SYS}$ = 12MHz/64 | — | 0.7 | 1.1 | mA |
| | | 5V | | — | 1.4 | 2.1 | mA |
| $I_{STB4}$ | IDLE Mode Standby Current (HIRC, $f_{SYS}$=off, $f_S=f_{SUB}=f_{LIRC}$) | 3V | No load, system HALT, ADC off, WDT enable, $f_{SYS}$ = 12MHz/64, LVR enable | — | 40 | 80 | μA |
| | | 5V | | — | 50 | 100 | μA |

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $I_{STB5}$ | IDLE Mode Standby Current (LIRC, $f_{SYS}=f_L=f_{LIRC}$, $f_S=f_{SUB}=f_{LIRC}$) | 3V | No load, system HALT, ADC off, WDT enable,$f_{SYS}$ = 32kHz | — | 1.9 | 4.0 | μA |
| | | 5V | | — | 3.3 | 7.0 | μA |
| $I_{STB6}$ | IDLE Mode Standby Current (LIRC, $f_{SYS}$=off, $f_S=f_{SUB}=f_{LIRC}$) | 3V | No load, system HALT, ADC off, WDT enable, $f_{SYS}$=32kHz, LVR enable | — | 40 | 80 | μA |
| | | 5V | | — | 50 | 100 | μA |
| $I_{STB7}$ | SLEEP Mode Standby Current (LIRC, $f_{SYS}$=off, $f_S=f_{SUB}=f_{LIRC}$) | 3V | No load, system HALT, ADC off, WDT enable, $f_{SYS}$ = 32kHz | — | 1.3 | 3.0 | μA |
| | | 5V | | — | 2.4 | 5.0 | μA |
| $V_{IL}$ | Input Low Voltage for I/O Ports or Input Pins | 5V | — | 0 | — | 1.5 | V |
| | | — | | 0 | — | $0.2V_{DD}$ | V |
| $V_{IH}$ | Input High Voltage for I/O Ports or Input Pins | 5V | — | 3.5 | — | 5.0 | V |
| | | — | | $0.8V_{DD}$ | — | $V_{DD}$ | V |
| $V_{LVR}$ | Low Voltage Reset Voltage | — | LVR enable, 2.55V | -5% | 2.55 | +5% | V |
| $I_{LVR}$ | Low Voltage Reset Current | — | LVR enable | — | 62 | 90 | μA |
| $I_{OL}$ | Sink Current for I/O Port | 3V | $V_{OL}=0.1V_{DD}$ | 8 | 16 | — | mA |
| | | 5V | $V_{OL}=0.1V_{DD}$ | 16 | 32 | — | mA |
| $I_{OH}$ | Source Current for I/O Port | 3V | $V_{OH}=0.9V_{DD}$ | -3.75 | -7.5 | — | mA |
| | | 5V | $V_{OH}=0.9V_{DD}$ | -7.5 | -15 | — | mA |
| $R_{PH}$ | Pull-high Resistance for I/O Ports | 3V | — | 20 | 60 | 100 | kΩ |
| | | 5V | — | 10 | 30 | 50 | kΩ |

## ADC Electrical Characteristics

Ta= 25˚C

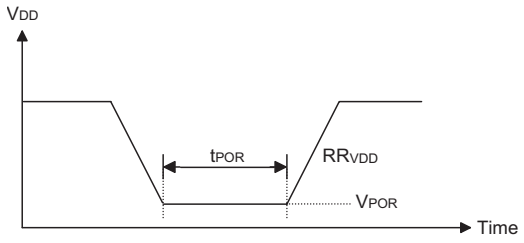
| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $AV_{DD}$ | A/D Converter Operating Voltage | — | — | 2.7 | — | 5.5 | V |
| $V_{ADI}$ | A/D Converter Input Voltage | — | — | 0 | — | $V_{REF}$ | V |
| $V_{REF}$ | A/D Converter Reference Voltage | — | — | 2 | — | $AV_{DD}$ | V |
| $V_{BG}$ | Reference Voltage | — | — | -3% | 1.19 | +3% | V |
| DNL1 | Differential Non-linearity | 3V | $V_{REF}=AV_{DD}=V_{DD}$<br>$t_{ADCK}$ =0.5μs<br>Ta=25°C | -2.0 | — | +2.0 | LSB |
| | | 5V | | | | | |
| DNL2 | Differential Non-linearity | 3V | $V_{REF}=AV_{DD}=V_{DD}$<br>$t_{ADCK}$ =0.5μs<br>Ta=-20°C ~ 85°C | -4.0 | — | +4.0 | LSB |
| | | 5V | | | | | |
| DNL3 | Differential Non-linearity | 3V | $V_{REF}=AV_{DD}=V_{DD}$<br>$t_{ADCK}$ =0.5μs<br>Ta=-40°C ~ -20°C | -8.5 | — | +8.5 | LSB |
| | | 5V | | | | | |
| INL1 | Integral Non-linearity | 3V | $V_{REF}=AV_{DD}=V_{DD}$<br>$t_{ADCK}$ =0.5μs<br>Ta=25°C | -4.0 | — | +4.0 | LSB |
| | | 5V | | | | | |

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| INL2 | Integral Non-linearity | 3V<br>5V | $V_{REF}$=$AV_{DD}$=$V_{DD}$<br>$t_{ADCK}$ =0.5μs<br>Ta=-20°C ~ 85°C | -4.2 | — | +4.2 | LSB |
| INL3 | Integral Non-linearity | 3V<br>5V | $V_{REF}$=$AV_{DD}$=$V_{DD}$<br>$t_{ADCK}$ =0.5μs<br>Ta=-40°C ~ -20°C | -8.5 | — | +8.5 | LSB |
| $I_{ADC}$ | Additional Power Consumption if A/D Converter is used | 3V | No load ($t_{ADCK}$ =0.5μs ) | — | 0.9 | 1.35 | mA |
| | | 5V | No load ($t_{ADCK}$ =0.5μs ) | — | 1.2 | 1.8 | mA |
| $t_{ADCK}$ | A/D Converter Clock Period | — | — | 0.5 | — | 100 | μs |
| $t_{ADC}$ | A/D Conversion Time<br>(Include Sample and Hold Time) | — | 12-bit ADC | — | 16 | — | $t_{ADCK}$ |
| $t_{ADS}$ | A/D Converter Sampling Time | — | — | — | 4 | — | $t_{ADCK}$ |
| $t_{ON2ST}$ | A/D Converter On-to-Start Time | — | — | 2 | — | — | μs |
| $t_{BGS}$ | $V_{BG}$ Turn on Stable Time | — | — | 200 | — | — | μs |

## Power-on Reset Characteristics

Ta= 25˚C

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

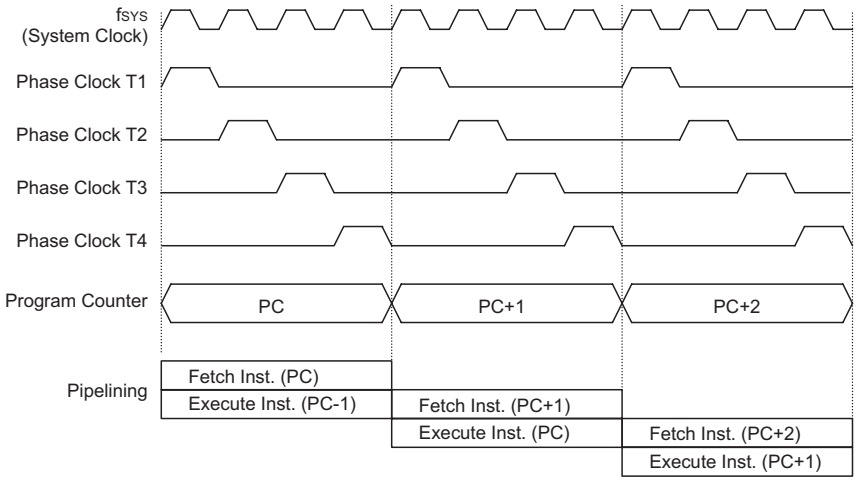


## System Architecture

A key factor in the high-performance features of the Holtek range of microcontrollers is attributed to their internal system architecture. The range of devices take advantage of the usual features found within RISC microcontrollers providing increased speed of operation and Periodic 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 these devices suitable for low-cost, high-volume production for controller applications.

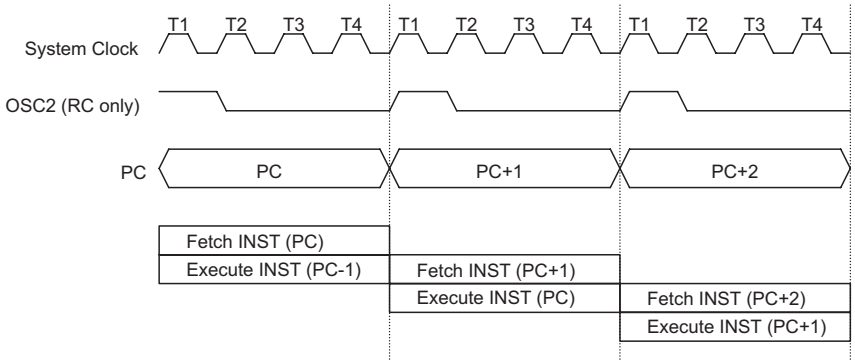
### Clocking and Pipelining

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




**System Clock and Pipelining**

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.




**Instruction Fetching**

## Program Counter

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

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.

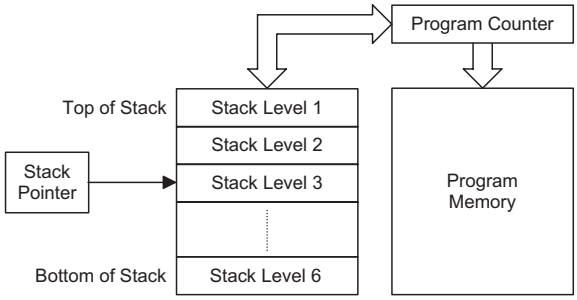
| Device | Program Counter | |
|---|---|---|
| | Program Counter High Byte | PCL Register |
| BS84B08A-3 | PC10~PC8 | PCL7~PCL0 |
| BS84C12A-3 | PC10~PC8 | |

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

## Stack

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

If the stack is full and an enabled interrupt takes place, the interrupt request flag will be recorded but the acknowledge signal will be inhibited. When the Stack Pointer is decremented, by RET or RETI, the interrupt will be serviced. This feature prevents stack overflow allowing the programmer to use the structure more easily. However, when the stack is full, a CALL subroutine instruction can still be executed which will result in a stack overflow. Precautions should be taken to avoid such cases which might cause unpredictable program branching. If the stack is overflow, the first Program Counter save in the stack will be lost.




## 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

## Flash Program Memory

The Program Memory is the location where the user code or program is stored. For this device series the Program Memory is Flash type, which means it can be programmed and re-programmed a large number of times, allowing the user the convenience of code modification on the same device. By using the appropriate programming tools, these Flash devices offer users the flexibility to conveniently debug and develop their applications while also offering a means of field programming and updating.

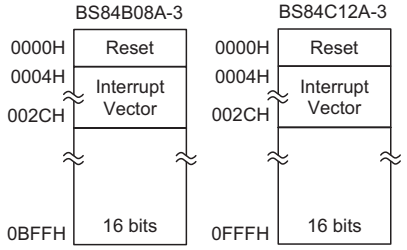
### Structure

The Program Memory has a capacity of 3K×16 or 4K×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.

| Device | Capacity | Banks |
|---|---|---|
| BS84B08A-3 | 3K×16 | 0,1 |
| BS84C12A-3 | 4K×16 | 0,1,2 |

### Special Vectors

Within the Program Memory, certain locations are reserved for the reset and interrupts. The location 000H 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.



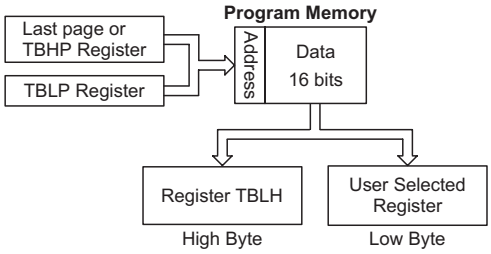

**Program Memory Structure**

## Look-up Table

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

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

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




| Instruction | Table Location Bits | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | b10 | b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| TABRDC [m] | @10 | @9 | @8 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |
| TABRDL [m] | 1 | 1 | 1 | @7 | @6 | @5 | @4 | @3 | @2 | @1 | @0 |

**Table Location**

Note: b10~b0: Table location bits

@7~@0: Table pointer (TBLP) bits

@10~@8: Table pointer (TBHP) bits

## Table Program Example

The following example shows how the table pointer and table data is defined and retrieved from the microcontroller. This example uses raw table data located in the Program Memory which is stored there using the ORG statement. The value at this ORG statement is "F00H" which refers to the start address of the last page within the 4K words Program Memory of the device. The table pointer is setup here to have an initial value of "06H". This will ensure that the first data read from the data table will be at the Program Memory address "F06H" or 6 locations after the start of the last page. Note that the value for the table pointer is referenced to the first address of the present page if the "TABRDC [m]" instruction is being used. The high byte of the table data which in this case is equal to zero will be transferred to the TBLH register automatically when the "TABRDC [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

```
tempreg1 db ?      ; temporary register #1
tempreg2 db ?      ; temporary register #2
:
:
mov a,06h          ; initialise low table pointer - note that this address is referenced
mov tblp,a
mov a,0Fh          ; initialise high table pointer
mov tbhp,a
:
:
tabrdc tempreg1    ; transfers value in table referenced by table pointer data at program
                   ; memory address "F06H" transferred to tempreg1 and TBLH
dec tblp           ; reduce value of table pointer by one
tabrdc tempreg2    ; transfers value in table referenced by table pointer data at program
                   ; memory address "F05H" transferred to tempreg2 and TBLH in this
                   ; example the data "1AH" is transferred to tempreg1 and data "0FH" to
                   ; register tempreg2
:
:
org F00h           ; sets initial address of program memory
dc 00Ah, 00Bh, 00Ch, 00Dh, 00Eh, 00Fh, 01Ah, 01Bh
:
:
```

## In Circuit Programming

The provision of Flash type Program Memory provides the user with a means of convenient and easy upgrades and modifications to their programs on the same device. As an additional convenience, Holtek has provided a means of programming the microcontroller in-circuit using a 4-pin interface. This provides manufacturers with the possibility of manufacturing their circuit boards complete with a programmed or un-programmed microcontroller, and then programming or upgrading the program at a later stage. This enables product manufacturers to easily keep their manufactured products supplied with the latest program releases without removal and re-insertion of the device.

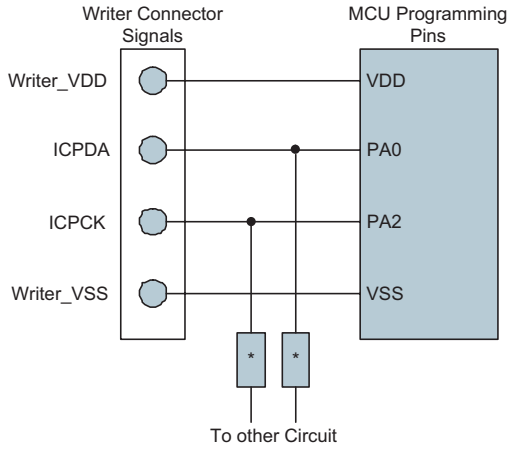
The Holtek Flash MCU to Writer Programming Pin correspondence table is as follows:

| Holtek Write Pins | MCU Programming Pins | Function |
|---|---|---|
| ICPDA | PA0 | Serial Address and data -- read/write |
| ICPCK | PA2 | Programming Serial Clock |
| VDD | VDD | Power Supply(5.0V) |
| VSS | VSS | Ground |

During the programming process, the user must there take care to ensure that no other outputs are connected to these two pins.

The Program Memory can be programmed serially in-circuit using this 4-wire interface. Data is downloaded and uploaded serially on a single pin with an additional line for the clock. Two additional lines are required for the power supply. The technical details regarding the in-circuit programming of the device are beyond the scope of this document and will be supplied in supplementary literature.

During the programming process the PA0 and PA2 I/O pins for data and clock programming purposes. The user must there take care to ensure that no other outputs are connected to these two pins.




Note: * may be resistor or capacitor. The resistance of * must be greater than 1kΩ or the capacitance of * must be less than 1nF.

### On-Chip Debug Support – OCDS

There is an EV chip named BS84V08A/BS84V12A which is used to emulate the BS84B08A-3/BS84C12A-3 device. The BS84V08A/BS84V12A device also provides the "On-Chip Debug" function to debug the BS84B08A-3/BS84C12A-3 device during development process. The device, BS84B08A-3/BS84C12A-3, is almost functional compatible except the "On-Chip Debug" function and package types. Users can use the BS84V08A/BS84V12A device to emulate the BS84B08A-3/BS84C12A-3 device behaviors by connecting the OCDSDA and OCDSCK pins to the Holtek HT-IDE development tools. The OCDSDA pin is the OCDS Data/Address input/output pin while the OCDSCK pin is the OCDS clock input pin. When users use the BS84V08A/BS84V12A EV chip for debugging, the corresponding pin functions shared with the OCDSDA and OCDSCK pins in the BS84B08A-3/BS84C12A-3 device will have no effect in the BS84V08A/BS84V12A EV chip. However, the two OCDS pins which are pin-shared with the ICP programming pins are still used as the Flash Memory programming pins for ICP. For more detailed OCDS information, refer to the corresponding document named "Holtek e-Link for 8-bit MCU OCDS User's Guide".

| Holtek e-Link Pins | MCU Pins | Pin Description |
|---|---|---|
| OCDSDA | OCDSDA | On-chip Debug Support Data/Address input/output |
| OCDSCK | OCDSCK | On-chip Debug Support Clock input |
| VDD | VDD | Power Supply |
| GND | VSS | Ground |

## RAM Data Memory

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

### Structure

Divided into two sections, the first of these is an area of RAM, known as the Special Function Data Memory. Here are located registers which 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 known as the General Purpose Data Memory, which is reserved for general purpose use. All locations within this area are read and write accessible under program control.

The overall Data Memory is subdivided into several banks for the devices. The Special Purpose Data Memory registers are accessible in all banks, with the exception of the EEC register at address 40H, which is only accessible in Bank 1. Switching between the different Data Memory banks is achieved by setting the Bank Pointer to the correct value. The start address of the Data Memory for all devices is the address 00H.

| Device | Capacity | Bank 0 | Bank 1 | Bank 2 |
|---|---|---|---|---|
| BS84B08A-3 | 288×8 | 60H~FFH | 80H~FFH | — |
| BS84C12A-3 | 384×8 | 60H~FFH | 80H~FFH | 80H~DFH |

**General Purpose Data Memory**

## Special Function Register Description

Most of the Special Function Register details will be described in the relevant functional section, however several registers require a separate description in this section.

### 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.

**BS84B08A-3**

| | Bank 0, 1 |
|---|---|
| 00H | IAR0 |
| 01H | MP0 |
| 02H | IAR1 |
| 03H | MP1 |
| 04H | BP |
| 05H | ACC |
| 06H | PCL |
| 07H | TBLP |
| 08H | TBLH |
| 09H | TBHP |
| 0AH | STATUS |
| 0BH | SMOD |
| 0CH | CTRL |
| 0DH | INTEG |
| 0EH | INTC0 |
| 0FH | INTC1 |
| 10H | Unused |
| 11H | Unused |
| 12H | Unused |
| 13H | LVRC |
| 14H | PA |
| 15H | PAC |
| 16H | PAPU |
| 17H | PAWU |
| 18H | Unused |
| 19H | Unused |
| 1AH | WDTC |
| 1BH | TBC |
| 1CH | TMR |
| 1DH | TMRC |
| 1EH | EEA |
| 1FH | EED |
| 20H | PB |
| 21H | PBC |
| 22H | PBPU |
| 23H | I2CTOC |
| 24H | SIMC0 |
| 25H | SIMC1 |
| 26H | SIMD |
| 27H | SIMC2/SIMA |
| 28H | Unused |
| 29H | Unused |
| 2AH | Unused |
| 2BH | Unused |
| 2CH | Unused |
| 2DH | Unused |
| 2EH | ADRL |
| 2FH | ADRH |

| | Bank 0 | Bank 1 |
|---|---|---|
| 30H | ADCR0 | |
| 31H | ADCR1 | |
| 32H | ACERL | |
| 33H | Unused | |
| 34H | Unused | |
| 35H | PD | |
| 36H | PDC | |
| 37H | PDPU | |
| 38H | Unused | |
| 39H | Unused | |
| 3AH | Unused | |
| 3BH | Unused | |
| 3CH | Unused | |
| 3DH | Unused | |
| 3EH | Unused | |
| 3FH | Unused | |
| 40H | Unused | EEC |
| 41H | Unused | |
| 42H | Unused | |
| 43H | TKTMR | |
| 44H | TKC0 | |
| 45H | TK16DL | |
| 46H | TK16DH | |
| 47H | TKC1 | |
| 48H | TKM016DL | |
| 49H | TKM016DH | |
| 4AH | TKM0ROL | |
| 4BH | TKM0ROH | |
| 4CH | TKM0C0 | |
| 4DH | TKM0C1 | |
| 4EH | TKM116DL | |
| 4FH | TKM116DH | |
| 50H | TKM1ROL | |
| 51H | TKM1ROH | |
| 52H | TKM1C0 | |
| 53H | TKM1C1 | |
| 54H | Unused | |
| 55H | Unused | |
| 56H | Unused | |
| 57H | Unused | |
| 58H | Unused | |
| 59H | Unused | |
| 5AH | Unused | |
| 5BH | Unused | |
| 5CH | Unused | |
| 5DH | Unused | |
| 5EH | Unused | |
| 5FH | Unused | |

**BS84C12A-3**

| | Bank 0, 1, 2 |
|---|---|
| 00H | IAR0 |
| 01H | MP0 |
| 02H | IAR1 |
| 03H | MP1 |
| 04H | BP |
| 05H | ACC |
| 06H | PCL |
| 07H | TBLP |
| 08H | TBLH |
| 09H | TBHP |
| 0AH | STATUS |
| 0BH | SMOD |
| 0CH | CTRL |
| 0DH | INTEG |
| 0EH | INTC0 |
| 0FH | INTC1 |
| 10H | Unused |
| 11H | Unused |
| 12H | Unused |
| 13H | LVRC |
| 14H | PA |
| 15H | PAC |
| 16H | PAPU |
| 17H | PAWU |
| 18H | Unused |
| 19H | Unused |
| 1AH | WDTC |
| 1BH | TBC |
| 1CH | TMR |
| 1DH | TMRC |
| 1EH | EEA |
| 1FH | EED |
| 20H | PB |
| 21H | PBC |
| 22H | PBPU |
| 23H | I2CTOC |
| 24H | SIMC0 |
| 25H | SIMC1 |
| 26H | SIMD |
| 27H | SIMC2/SIMA |
| 28H | Unused |
| 29H | Unused |
| 2AH | Unused |
| 2BH | Unused |
| 2CH | Unused |
| 2DH | Unused |
| 2EH | ADRL |
| 2FH | ADRH |

| | Bank 0, 2 | Bank 1 |
|---|---|---|
| 30H | ADCR0 | |
| 31H | ADCR1 | |
| 32H | ACERL | |
| 33H | Unused | |
| 34H | Unused | |
| 35H | PD | |
| 36H | PDC | |
| 37H | PDPU | |
| 38H | PC | |
| 39H | PCC | |
| 3AH | PCPU | |
| 3BH | Unused | |
| 3CH | Unused | |
| 3DH | Unused | |
| 3EH | Unused | |
| 3FH | Unused | |
| 40H | Unused | EEC |
| 41H | Unused | |
| 42H | Unused | |
| 43H | TKTMR | |
| 44H | TKC0 | |
| 45H | TK16DL | |
| 46H | TK16DH | |
| 47H | TKC1 | |
| 48H | TKM016DL | |
| 49H | TKM016DH | |
| 4AH | TKM0ROL | |
| 4BH | TKM0ROH | |
| 4CH | TKM0C0 | |
| 4DH | TKM0C1 | |
| 4EH | TKM116DL | |
| 4FH | TKM116DH | |
| 50H | TKM1ROL | |
| 51H | TKM1ROH | |
| 52H | TKM1C0 | |
| 53H | TKM1C1 | |
| 54H | TKM216DL | |
| 55H | TKM216DH | |
| 56H | TKM2ROL | |
| 57H | TKM2ROH | |
| 58H | TKM2C0 | |
| 59H | TKM2C1 | |
| 5AH | Unused | |
| 5BH | Unused | |
| 5CH | Unused | |
| 5DH | Unused | |
| 5EH | Unused | |
| 5FH | Unused | |

**Special Purpose Data Memory**

## 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 according to BP register. Direct Addressing can only be used with Bank 0, all other Banks must be addressed indirectly using MP1 and IAR1.

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

### Indirect Addressing Program Example

```
data .section ´data´
adres1   db ?
adres2   db ?
adres3   db ?
adres4   db ?
block    db ?
code .section at 0 ´code´
org00h
start:
     mov a,04h            ; setup size of block
     mov block,a
     mov a,offset adres1  ; Accumulator loaded with first RAM address
     mov mp0,a            ; setup memory pointer with first RAM address
loop:
     clr IAR0             ; clear the data at address defined by mp0
     inc mp0              ; increment memory pointer
     sdz block            ; check if last memory location has been cleared
     jmp loop
continue:
```

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

## Bank Pointer – BP

Depending upon which device is used, the Program and Data Memory are divided into several banks, Bank0, Bank1 and Bank2. Selecting the required Data Memory area is achieved using the Bank Pointer. Bit 0 and bit 1 of the Bank Pointer are used to select Data Memory Banks 0~2.

The Data Memory is initialised to Bank 0 after a reset, except for a 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 Bank1 must be implemented using Indirect Addressing.

As both the Program Memory and Data Memory share the same Bank Pointer Register, care must be taken during programming.

### BP Register – BS84B08A-3

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | — | — | — | — | DMBP0 |
| R/W | — | — | — | — | — | — | — | R/W |
| POR | — | — | — | — | — | — | — | 0 |

Bit 7 ~ 2 Unimplemented, read as "0"

Bit 1~0 **DMBP0**: Select Data Memory Banks

0: Bank 0

1: Bank 1

### BP Register – BS84C12A-3

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | — | — | — | DMBP1 | DMBP0 |
| R/W | — | — | — | — | — | — | R/W | R/W |
| POR | — | — | — | — | — | — | 0 | 0 |

Bit 7 ~ 2 Unimplemented, read as "0"

Bit 1~0 **DMBP1 ~ DMBP0**: Select Data Memory Banks

00: Bank 0

01: Bank 1

10: Bank 2

11: Reserved

## Accumulator – ACC

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

### Program Counter Low Register – PCL

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

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

These three special function registers are used to control operation of the look-up table which is stored in the Program Memory. TBLP and TBHP are the table pointers and indicate 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.

**STATUS Register**

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

"×" unknown

Bit 7 ~ 6 Unimplemented, read as "0"

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

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

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

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

Bit 1 **AC**: Auxiliary flag
0: no auxiliary carry
1: an operation results in a carry out of the low nibbles in addition, or no borrow from the high nibble into the low nibble in subtraction

Bit 0 **C**: Carry flag
0: no carry-out
1: an operation results in a carry during an addition operation or if a borrow does not take place during a subtraction operation

**C** is also affected by a rotate through carry instruction.

## EEPROM Data Memory

One of the special features in the device is its internal EEPROM Data Memory. EEPROM, which stands for Electrically Erasable Programmable Read Only Memory, is by its nature a non-volatile form of memory, with data retention even when its power supply is removed. By incorporating this kind of data memory, a whole new host of application possibilities are made available to the designer. The availability of EEPROM storage allows information such as product identification numbers, calibration values, specific user data, system setup data or other product information to be stored directly within the product microcontroller. The process of reading and writing data to the EEPROM memory has been reduced to a very trivial affair.

### EEPROM Data Memory Structure

The EEPROM Data Memory capacity is up to 128×8 bits. Unlike the Program Memory and RAM Data Memory, the EEPROM Data Memory is not directly mapped and is therefore not directly accessible in the same way as the other types of memory. Read and Write operations to the EEPROM are carried out in single byte operations using an address and data register in Bank 0 and a single control register in Bank 1.

| Device | Capacity | Address |
|---|---|---|
| BS84B08A-3 | 64×8 | 00H~3FH |
| BS84C12A-3 | 128×8 | 00H~3FH |

### EEPROM Registers

Three registers control the overall operation of the internal EEPROM Data Memory. These are the address register, EEA, the data register, EED and a single control register, EEC. As both the EEA and EED registers are located in Bank 0, they can be directly accessed in the same way as any other Special Function Register. The EEC register however, being located in Bank1, cannot be directly addressed directly and can only be read from or written to indirectly using the MP1 Memory Pointer and Indirect Addressing Register, IAR1. Because the EEC control register is located at address 40H in Bank 1, the MP1 Memory Pointer must first be set to the value 40H and the Bank Pointer register, BP, set to the value, 01H, before any operations on the EEC register are executed.

#### EEPROM Control Registers List

| Name | Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| EEA | — | — | D5 | D4 | D3 | D2 | D1 | D0 |
| EED | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| EEC | — | — | — | — | WREN | WR | RDEN | RD |

#### EEA Register

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

"×" unknown

Bit 7 ~ 6 Unimplemented, read as "0"

Bit 5 ~ 0 Data EEPROM address

Data EEPROM address bit 5 ~ bit 0

### EED 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 Data EEPROM data

Data EEPROM data bit 7 ~ bit 0

### EEC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | — | WREN | WR | RDEN | RD |
| R/W | — | — | — | — | R/W | R/W | R/W | R/W |
| POR | — | — | — | — | 0 | 0 | 0 | 0 |

Bit 7 ~ 4 Unimplemented, read as "0"

Bit 3 **WREN**: Data EEPROM Write Enable

0: Disable

1: Enable

This is the Data EEPROM Write Enable Bit which must be set high before Data EEPROM write operations are carried out. Clearing this bit to zero will inhibit Data EEPROM write operations.

Bit 2 **WR**: EEPROM Write Control

0: Write cycle has finished

1: Activate a write cycle

This is the Data EEPROM Write Control Bit and when set high by the application program will activate a write cycle. This bit will be automatically reset to zero by the hardware after the write cycle has finished. Setting this bit high will have no effect if the WREN has not first been set high.

Bit 1 **RDEN**: Data EEPROM Read Enable

0: Disable

1: Enable

This is the Data EEPROM Read Enable Bit which must be set high before Data EEPROM read operations are carried out. Clearing this bit to zero will inhibit Data EEPROM read operations.

Bit 0 **RD**: EEPROM Read Control

0: Read cycle has finished

1: Activate a read cycle

This is the Data EEPROM Read Control Bit and when set high by the application program will activate a read cycle. This bit will be automatically reset to zero by the hardware after the read cycle has finished. Setting this bit high will have no effect if the RDEN has not first been set high.

Note: The WREN, WR, RDEN and RD can not be set to "1" at the same time in one instruction.
The WR and RD can not be set to "1" at the same time.

### Reading Data from the EEPROM

To read data from the EEPROM, the read enable bit, RDEN, in the EEC register must first be set high to enable the read function. The EEPROM address of the data to be read must then be placed in the EEA register. If the RD bit in the EEC register is now set high, a read cycle will be initiated. Setting the RD bit high will not initiate a read operation if the RDEN bit has not been set. When the read cycle terminates, the RD bit will be automatically cleared to zero, after which the data can be read from the EED register. The data will remain in the EED register until another read or write operation is executed. The application program can poll the RD bit to determine when the data is valid for reading.

### Writing Data to the EEPROM

To write data to the EEPROM, the write enable bit, WREN, in the EEC register must first be set high to enable the write function. The EEPROM address of the data to be written must then be placed in the EEA register and the data placed in the EED register. If the WR bit in the EEC register is now set high, an internal write cycle will then be initiated. Setting the WR bit high will not initiate a write cycle if the WREN bit has not been set. As the EEPROM write cycle is controlled using an internal timer whose operation is asynchronous to microcontroller system clock, a certain time will elapse before the data will have been written into the EEPROM. Detecting when the write cycle has finished can be implemented either by polling the WR bit in the EEC register or by using the EEPROM interrupt. When the write cycle terminates, the WR bit will be automatically cleared to zero by the microcontroller, informing the user that the data has been written to the EEPROM. The application program can therefore poll the WR bit to determine when the write cycle has ended.

### Write Protection

Protection against inadvertent write operation is provided in several ways. After the device is powered-on the Write Enable bit in the control register will be cleared preventing any write operations. Also at power-on the Bank Pointer, BP, will be reset to zero, which means that Data Memory Bank 0 will be selected. As the EEPROM control register is located in Bank 1, this adds a further measure of protection against spurious write operations. During normal program operation, ensuring that the Write Enable bit in the control register is cleared will safeguard against incorrect write operations.

### EEPROM Interrupt

The EEPROM write interrupt is generated when an EEPROM write cycle has ended. The EEPROM interrupt must first be enabled by setting the DEE bit in the relevant interrupt register. When an EEPROM write cycle ends, the DEF request flag will be set. If the global, EEPROM is enabled and the stack is not full, a jump to the associated Interrupt vector will take place. When the interrupt is serviced, the EEPROM interrupt flag will automatically reset. More details can be obtained in the Interrupt section.

## Programming Considerations

Care must be taken that data is not inadvertently written to the EEPROM. Protection can be Periodic by ensuring that the Write Enable bit is normally cleared to zero when not writing. Also the Bank Pointer could be normally cleared to zero as this would inhibit access to Bank 1 where the EEPROM control register exist. Although certainly not necessary, consideration might be given in the application program to the checking of the validity of new write data by a simple read back process. When writing data the WR bit must be set high immediately after the WREN bit has been set high, to ensure the write cycle executes correctly. The global interrupt bit EMI should also be cleared before a write cycle is executed and then re-enabled after the write cycle starts.

### Programming Examples

- **Reading data from the EEPROM – polling method**

```
MOV A, EEPROM_ADRES          ; user defined address
MOV EEA, A
MOV A, 040H                  ; setup memory pointer MP1
MOV MP1, A                   ; MP1 points to EEC register
MOV A, 01H                   ; setup Bank Pointer
MOV BP, A
SET IAR1.1                   ; set RDEN bit, enable read operations
SET IAR1.0                   ; start Read Cycle - set RD bit
BACK:
SZ IAR1.0                    ; check for read cycle end
JMP BACK
CLR IAR1                     ; disable EEPROM write
CLR BP
MOV A, EED                   ; move read data to register
MOV READ_DATA, A
```

- **Writing Data to the EEPROM – polling method**

```
CLR   EMI
MOV A, EEPROM_ADRES          ; user defined address
MOV EEA, A
MOV A, EEPROM_DATA           ; user defined data
MOV EED, A
MOV A, 040H                  ; setup memory pointer MP1
MOV MP1, A                   ; MP1 points to EEC register
MOV A, 01H                   ; setup Bank Pointer
MOV BP, A
SET IAR1.3                   ; set WREN bit, enable write operations
SET IAR1.2                   ; start Write Cycle - set WR bit
SET EMI
BACK:
SZ IAR1.2                    ; check for write cycle end
JMP BACK
CLR IAR1                     ; disable EEPROM write
CLR BP
```

# Oscillator

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

## Oscillator Overview

In addition to being the source of the main system clock the oscillators also provide clock sources for the Watchdog Timer and Time Base Interrupts. Fully integrated internal oscillators, requiring no external components, are provided to form a wide range of both fast and slow system oscillators. The higher frequency oscillators provide higher performance but carry with it the disadvantage of higher power requirements, while the opposite is of course true for the lower frequency oscillators. With the capability of dynamically switching between fast and slow system clock, the device has the flexibility to optimize the performance/power ratio, a feature especially important in power sensitive portable applications.

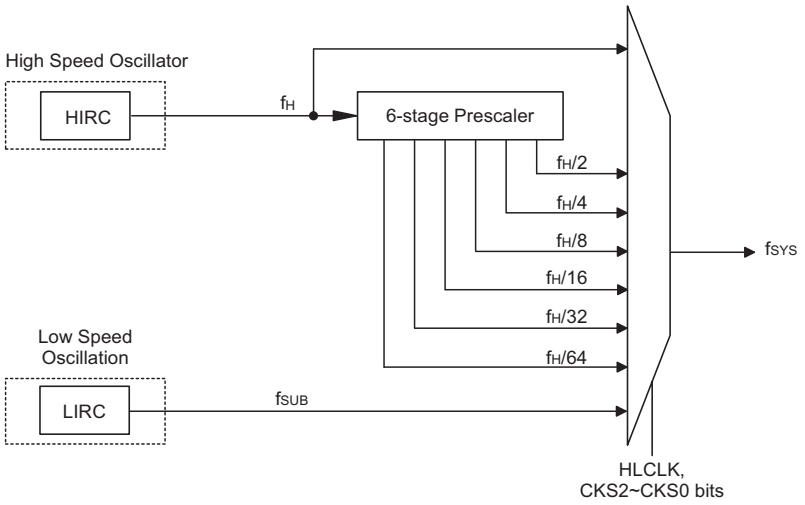
| Type | Name | Freq. |
|---|---|---|
| Internal High Speed RC | HIRC | 8/12/16MHz |
| Internal Low Speed RC | LIRC | 32kHz |

**Oscillator Types**

## System Clock Configurations

There are two methods of generating the system clock, a high speed oscillator and a low speed oscillator. The high speed oscillator is the internal 8MHz, 12MHz, 16MHz RC oscillator. The low speed oscillator is the internal 32kHz (LIRC) oscillator. Selecting whether the low or high speed 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 used for the high speed and the low speed oscillators is chosen via registers. The frequency of the slow speed or high speed system clock is also determined using the HLCLK bit 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.




**System Clock Configurations**

### Internal RC Oscillator – HIRC

The internal RC oscillator is a fully integrated system oscillator requiring no external components. The internal RC oscillator has a power on default frequency of 8 MHz but can be selected to be either 8MHz, 12MHz or 16MHz using the HIRCS1 and HIRCS0 bits in the CTRL register. 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.

### Internal 32kHz Oscillator – LIRC

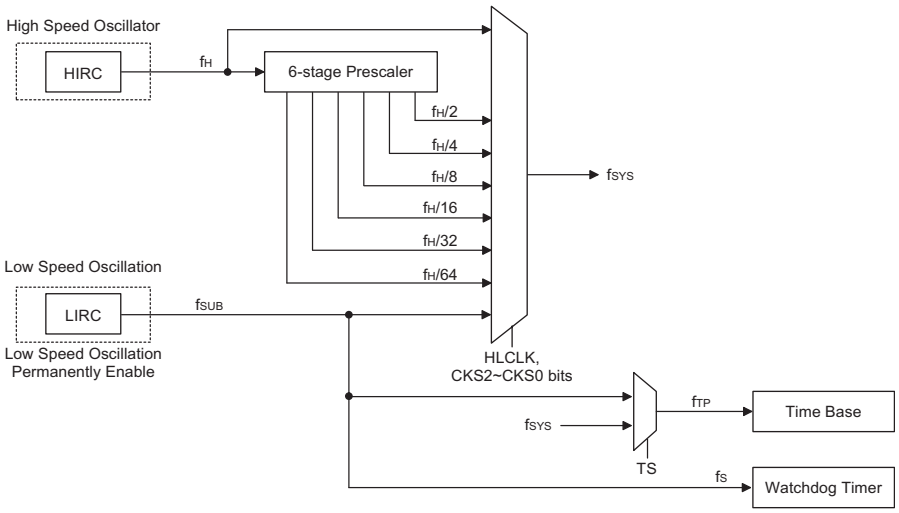
The Internal 32kHz System Oscillator is the low frequency oscillator. It is a fully integrated RC oscillator with a typical frequency of 32kHz at 5V, requiring no external components for its implementation. Device trimming during the manufacturing process and the inclusion of internal frequency compensation circuits are used to ensure that the influence of the power supply voltage, temperature and process variations on the oscillation frequency are minimised. After power on this LIRC oscillator will be permanently enabled; there is no provision to disable the oscillator using.

## 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 this device 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 main system clock, can come from either a high frequency, $f_H$, or low frequency, $f_{SUB}$, source, and is selected using the HLCLK bit and CKS2~CKS0 bits in the SMOD register. Both the high and low speed system clocks are sourced from internal RC oscillators.




**System Clock Configurations**

Note: When the system clock source $f_{SYS}$ is switched to $f_{SUB}$ from $f_H$, the high speed oscillation will stop to conserve the power. Thus there is no $f_H$~$f_H/64$ for peripheral circuit to use.

## System Operation Modes

There are five 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 three modes, the SLEEP, IDLE0 and IDLE1 Mode are used when the microcontroller CPU is switched off to conserve power.

| Operating Mode | Description | | | |
|---|---|---|---|---|
| | CPU | $f_{SYS}$ | $f_{SUB}$ | $f_S$ |
| NORMAL mode | On | $f_H$~$f_H$/64 | On | On |
| SLOW mode | On | $f_{SUB}$ | On | On |
| ILDE0 mode | Off | Off | On | On |
| IDLE1 mode | Off | On | On | On |
| SLEEP mode | Off | Off | On | 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 the high speed oscillator. This mode operates allowing the microcontroller to operate normally with a clock source will come from the high speed oscillator, HIRC. 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 $f_{SUB}$. Running the microcontroller in this mode allows it to run with much lower operating currents. In the SLOW Mode, the $f_H$ is off.

### SLEEP 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 SLEEP mode the CPU will be stopped. However the $f_{SUB}$ clocks will continue to run the Watchdog Timer will continue to operate.

### 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 CTRL register is low. In the IDLE0 Mode the system oscillator will be stop and will therefore be inhibited from driving the CPU but some peripheral functions will remain operational such as the Watchdog Timer, Timer/Event Counter.

### IDLE1 Mode

The IDLE1 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 CTRL 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. In the IDLE1 Mode, the system oscillator will continue to run, and this system oscillator may be the high speed or low speed system oscillator. In the IDLE1 Mode the Watchdog Timer clock, $f_S$, will be on. The source comes from $f_{SUB}$ then $f_S$ will be on.

## Control Register

The SMOD register is used to control the internal clocks within the device.

### SMOD Register

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

Bit 7 ~ 5 **CKS2 ~ CKS0**: The system clock selection when HLCLK is "0"
000: $f_{SUB}$ ($f_{LIRC}$)
001: $f_{SUB}$ ($f_{LIRC}$)
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 can be LIRC, a divided version of the high speed system oscillator can also be chosen as the system clock source.

Bit 4 Unimplemented, read as 0.

Bit 3 **LTO**: LIRC System OSC SST ready flag
0: Not ready
1: Ready
This is the low speed system oscillator SST 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 change to a high level after 1~2 cycles.

Bit 2 **HTO**: HIRC System OSC SST ready flag
0: Not ready
1: Ready
This is the high speed system oscillator SST ready flag which indicates when the high speed system oscillator is stable after a wake-up has occurred. 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 power on reset or a wake-up has occurred, the flag will change to a high level after 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: $f_H/2$ ~ $f_H/64$ or $f_{SUB}$
1: $f_H$
This bit is used to select if the $f_H$ clock or the $f_H/2$ ~ $f_H/64$ or $f_{SUB}$ clock is used as the system clock. When the bit is high the $f_H$ clock will be selected and if low the $f_H/2$ ~ $f_H/64$ or $f_{SUB}$ clock will be selected. When system clock switches from the $f_H$ clock to the $f_{SUB}$ clock and the $f_H$ clock will be automatically switched off to conserve power.

### CTRL Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | FSYSON | — | HIRCS1 | HIRCS0 | — | LVRF | LRF | WRF |
| R/W | R/W | — | R/W | R/W | — | R/W | R/W | R/W |
| POR | 0 | — | 0 | 0 | — | × | 0 | 0 |

"×"Unknown

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

Bit 6 Unimplemented, read as 0.

Bit 5~4 **HIRCS1~HIRCS0**: High frequency clock select
00: 8MHz
01: 16 MHz
10: 12 MHz
11: 8 MHz

Bit 3 Unimplemented, read as 0.

Bit 2 **LVRF**: LVR function reset flag
0: Not occur
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**: LVRC Control register software reset flag
0: Not occur
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.

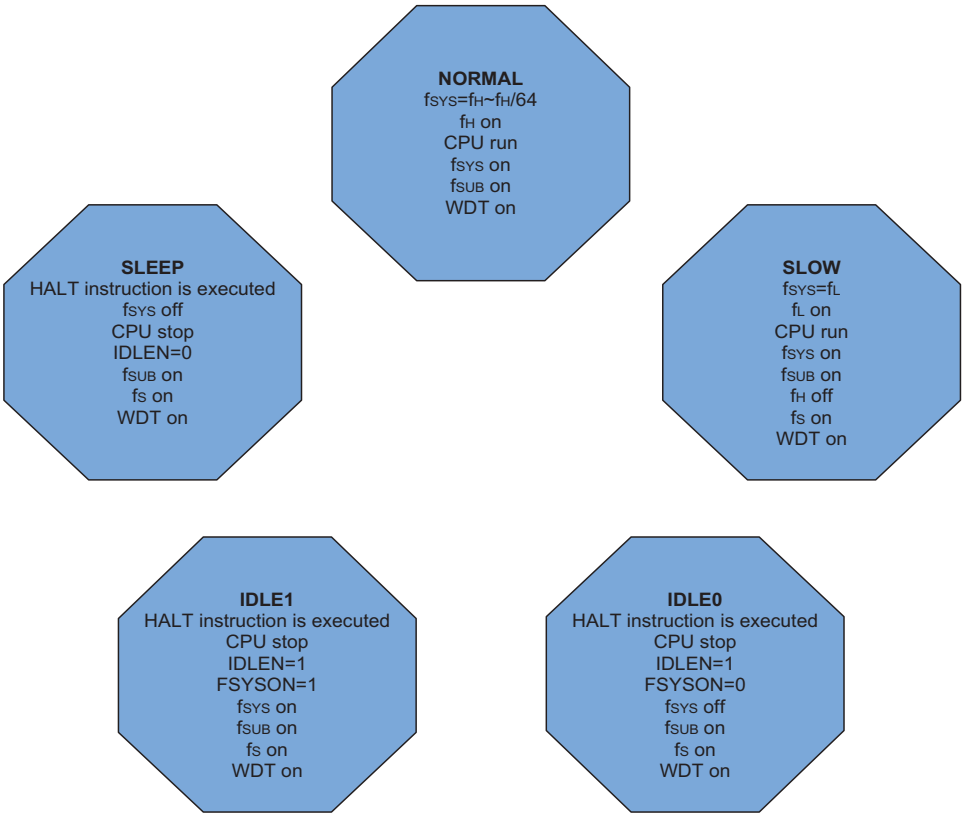
Bit 0 **WRF**: WDT Control register software reset flag
0: Not occur
1: Occurred
This bit is set to 1 by the WDT Control register software reset and cleared by the application program. Note that this bit can only be cleared to 0 by the application program.

## 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 CTRL 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. When this happens it must be noted that the $f_H/16$ and $f_H/64$ internal clock sources will also stop running. 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 setting the HLCLK bit to "0" and setting 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 LIRC oscillator and therefore requires this oscillator 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 LIRC 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.

### Entering the SLEEP Mode

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

- The system clock and Time Base 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.
- 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

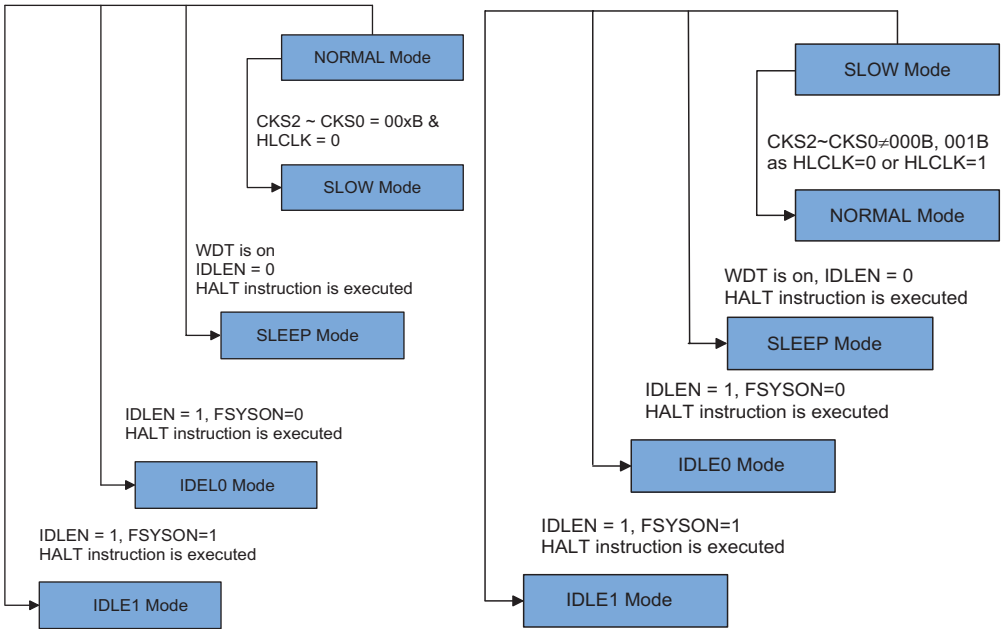
There is only one way for the device to enter the IDLE0 Mode and that is to execute the "HALT" instruction in the application program with the IDLEN bit in SMOD register equal to "1" and the FSYSON bit in CTRL register equal to "0". When this instruction is executed under the conditions described above,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

- The system clock will be stopped and the application program will stop at the "HALT" instruction, but the Time Base and the low frequency $f_{SUB}$ clock will be on.
- The Data Memory contents and registers will maintain their present condition.
- The WDT will be cleared and resume counting.
- 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 CTRL register equal to "1". When this instruction is executed under the conditions described above, the following will occur:

- The system clock and the low frequency $f_{SUB}$ 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.
- 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 the device 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 the device. All high-impedance input pins must be connected to either a fixed high or low level as any floating input pins could create internal oscillations and result in increased current consumption. This also applies to devices which have different package types, as there may be 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. 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 falling edge on Port A
- A system interrupt
- A WDT overflow

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 the device will not be immediately serviced, but will rather be serviced later when the related interrupt is finally enabled or when a stack level becomes free. The other situation is where the related interrupt is enabled and the stack is not full, in which case the regular interrupt response takes place. If an interrupt request flag is set high before entering the SLEEP or IDLE Mode, the wake-up function of the related interrupt will be disabled.

| System Oscillator | Wake-up Time (SLEEP Mode) | Wake-up Time (IDLE0 Mode) | Wake-up Time (IDLE1 Mode) |
|---|---|---|---|
| HIRC | 15~16 HIRC cycles | | 1~2 HIRC cycles |
| LIRC | 1~2 LIRC cycles | | 1~2 LIRC cycles |

**Wake-Up Time**

# 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 internal fSUB clock which is in turn supplied by the LIRC oscillator. The Watchdog Timer source clock is then subdivided by a ratio of 28 to 218 to give longer timeouts, the actual value being chosen using the WS2~WS0 bits in the WDTC register. The LIRC internal oscillator has an approximate period of 32kHz at a supply voltage of 5V. However, it should be noted that this specified internal clock period can vary with VDD, temperature and process variations.

The WDT is always enabled.

## Watchdog Timer Control Register

A single register, WDTC, controls the required timeout period as well as the enable operation. The WDTC register is initiated to 01010011B at any reset but keeps unchanged at the WDT time-out occurrence in a power down state.

### 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 software control

10101B or 01010B: Enabled

Other values: Reset MCU (Reset will be active after 2~3 LIRC clock for debounce time.)

If the MCU reset caused by the WE [4:0] in WDTC software reset, the WRF flag of CTRL register will be set).

Bit 2~ 0 **WS2 ~ WS0**: WDT Time-out period selection

000: $2^8/f_{SUB}$

001: $2^{10}/f_{SUB}$

010: $2^{12}/f_{SUB}$

011: $2^{14}/f_{SUB}$(default)

100: $2^{15}/f_{SUB}$

101: $2^{16}/f_{SUB}$

110: $2^{17}/f_{SUB}$

111: $2^{18}/f_{SUB}$

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

**CTRL Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | FSYSON | — | HIRCS1 | HIRCS0 | — | LVRF | LRF | WRF |
| R/W | R/W | — | R/W | R/W | — | R/W | R/W | R/W |
| POR | 0 | — | 0 | 0 | — | x | 0 | 0 |

"×"unknown

Bit 7 **FSYSON**: $f_{SYS}$ Control IDLE Mode
Describe elsewhere

Bit 6 Unimplemented, read as "0"

Bit 5~ 4 **HIRCS1~HIRCS0**: High frequency clock select
Describe elsewhere

Bit 3 Unimplemented, read as "0"

Bit 2 **LVRF**: LVR function reset flag
Describe elsewhere

Bit 1 **LRF**: LVR Control register software reset flag
Describe elsewhere

Bit 0 **WRF**: WDT Control register software reset flag
0: Not occur
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.

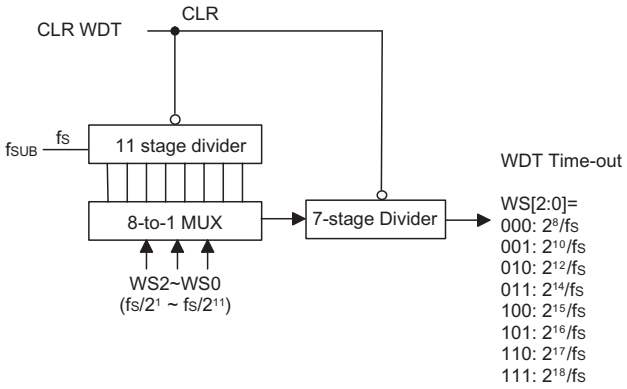
## Watchdog Timer Operation

In these devices the Watchdog Timer supplied by the $f_{SUB}$ oscillator and is therefore always on. 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, the clear WDT instruction will not be executed in the correct manner, in which case the Watchdog Timer will overflow and reset the device. There are five bits, WE4~WE0, in the WDTC register to enable the WDT function. When the WE4~WE0 bits value is equal to 01010B or 10101B, the WDT function is enabled. However, if the WE4~WE0 bits are changed to any other values except 01010B and 10101B, which is caused by the environmental noise, it will reset the microcontroller after 2~3 LIRC clock cycles.

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. Four methods can be adopted to clear the contents of the Watchdog Timer. The first is a WDT reset, which means a certain value is written into the WE4~WE0 bit filed except 01010B and 10101B, the second is using the Watchdog Timer software clear instructions 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.

The maximum time-out period is when the $2^{18}$ division ratio is selected. As an example, with a 32 kHz LIRC oscillator as its source clock, this will give a maximum watchdog period of around 8 seconds for the $2^{18}$ division ratio, and a minimum timeout of 7.8ms for the $2^8$ division ration.




**Watchdog Timer**

## 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.

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 is implemented in situations where the power supply voltage falls below a certain threshold.

### Reset Functions

There are four ways in which a microcontroller reset can occur, through events occurring internally:

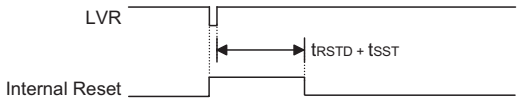
#### Power-on Reset

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.


**Power-On Reset Timing Chart**

### Low Voltage Reset – LVR

The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device, which is selected via a configuration option. If the supply voltage of the device drops to within a range of 0.9V~$V_{LVR}$ such as might occur when changing the battery, the LVR will automatically reset the device internally and the LVRF bit in the CTRL register will also be set. For a valid LVR signal, a low voltage, i.e., a voltage in the range between 0.9V~ $V_{LVR}$ must exist for greater than the value $t_{LVR}$ specified in the A.C. characteristics. If the low voltage state does not exceed this value, the LVR will ignore the low supply voltage and will not perform a reset function. The actual $V_{LVR}$ is set by the LVRC register. If the LVS7~LVS0 bits are changed to some certain values by the environmental noise, the LVR will reset the device after 2~3 LIRC clock cycles. When this happens, the LRF bit in the CTRL 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 |
| POR | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Bit 7~0 **LVS7 ~ LVS0**: LVR Voltage Select control

01010101: 2.55V(default)

00110011: 2.55V

10011001: 2.55V

10101010: 2.55V

Other values: MCU reset (reset will be active after 2~3 LIRC clock for debounce time)

Note: S/W can write 00H~FFH to control LVR voltage, even to S/W reset MCU. If the MCU reset caused LVRC software reset, the LRF flag of CTRL register will be set.

• **CTRL Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | FSYSON | — | HIRCS1 | HIRCS0 | — | LVRF | LRF | WRF |
| R/W | R/W | — | R/W | R/W | — | R/W | R/W | R/W |
| POR | 0 | — | 0 | 0 | — | x | 0 | 0 |

"×" unknown

Bit 7 **FSYSON:** $f_{SYS}$ Control IDLE Mode
Describe elsewhere

Bit 6 Unimplemented, read as "0"

Bit 5~ 4 **HIRCS1~HIRCS0**: High frequency clock select
Describe elsewhere

Bit 3 Unimplemented, read as "0"

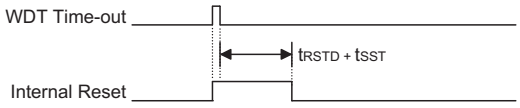
Bit 2 **LVRF:** LVR function reset flag
0: Not occur
1: Occurred
This bit is set to 1 when a specific Low Voltage Reset situation condition occurs. This bit can only be cleared to 0 by the application program.

Bit 1 **LRF:** LVR Control register software reset flag
0: Not occur
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
Describe elsewhere

### Watchdog Time-out Reset during Normal Operation

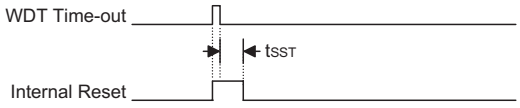
The Watchdog time-out Reset during normal operation is the same as a LVR reset except that the Watchdog time-out flag TO will be set to "1".




**WDT Time-out Reset during Normal Operation Timing Chart**

### Watchdog Time-out Reset during SLEEP or IDLE Mode

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




Note: The $t_{SST}$ is 15~16 clock cycles if the system clock source is provided by the HIRC.

The $t_{SST}$ is 1~2 clock for the LIRC.

**WDT Time-out Reset during SLEEP or IDLE Timing Chart**

## Reset Initial Conditions

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

| TO | PDF | RESET Conditions |
|---|---|---|
| 0 | 0 | Power-on reset |
| u | u | LVR reset during NORMAL or SLOW Mode operation |
| 1 | u | WDT time-out reset during NORMAL or SLOW Mode operation |
| 1 | 1 | WDT time-out reset during IDLE or SLEEP Mode operation |

Note: "u" stands for unchanged

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

| Item | Condition After RESET |
|---|---|
| Program Counter | Reset to zero |
| Interrupts | All interrupts will be disabled |
| WDT | Clear after reset, WDT begins counting |
| Timer/Event Counter | Timer/Event Counter will be turned off |
| Input/Output Ports | I/O ports will be setup as inputs and AN0~AN7 as A/D input pins |
| 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.

**BS84B08A-3 Register**

| Register | LVR&power on | WDT Overflow (Normal Mode) | WDT Overflow (HALT Mode) |
|---|---|---|---|
| IAR0 | ---- ---- | ---- ---- | ---- ---- |
| MP0 | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| IAR1 | ---- ---- | ---- ---- | ---- ---- |
| MP1 | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| BP | ---- ---0 | ---- ---0 | ---- ---u |
| ACC | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| PCL | 0000 0000 | 0000 0000 | 0000 0000 |
| TBLP | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| TBLH | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| TBHP | ---- xxxx | ---- uuuu | ---- uuuu |
| STATUS | --00 xxxx | --1u uuuu | --11 uuuu |
| SMOD | 0000 0011 | 0000 0011 | uuuu uuuu |
| CTRL | 0-00 -x00 | 0-00 -x00 | u-uu -uuu |
| INTEG | ---- --00 | ---- --00 | ---- --uu |
| INTC0 | -000 0000 | -000 0000 | -uuu uuuu |
| INTC1 | -000 -000 | -000 -000 | -uuu -uuu |
| LVRC | 0000 0000 | 0000 0000 | uuuu uuuu |
| PA | 1--1 1111 | 1--1 1111 | u--u uuuu |
| PAC | 1--1 1111 | 1--1 1111 | u--u uuuu |
| PAPU | 0--0 0000 | 0--0 0000 | u--u uuuu |
| PAWU | 0--0 0000 | 0--0 0000 | u--u uuuu |
| WDTC | 0101 0011 | 0101 0011 | uuuu uuuu |
| TBC | --00 ---- | --00 ---- | --uu ---- |
| TMR | 0000 0000 | 0000 0000 | uuuu uuuu |
| TMRC | --00 -000 | --00 -000 | --uu -uuu |
| EEA | --11 1111 | --11 1111 | --uu uuuu |
| EED | 0000 0000 | 0000 0000 | uuuu uuuu |
| PB | 1111 1111 | 1111 1111 | uuuu uuuu |
| PBC | 1111 1111 | 1111 1111 | uuuu uuuu |
| PBPU | 0000 0000 | 0000 0000 | uuuu uuuu |
| I2CTOC | 0000 0000 | 0000 0000 | uuuu uuuu |
| SIMC0 | 0000 -00- | 0000 -00- | uuuu –uu- |
| SIMC1 | 1000 0001 | 1000 0001 | uuuu -uuu |
| SIMD | 0000 0000 | 0000 0000 | uuuu uuuu |
| SIMC2 | --11 1111 | --11 1111 | --uu uuuu |
| SIMA | 0000 0000 | 0000 0000 | uuuu uuuu |
| ADRL(ADRFS=0) | xxxx ---- | xxxx ---- | uuuu ---- |
| ADRL(ADRFS=1) | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| ADRH(ADRFS=0) | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| ADRH(ADRFS=1) | ---- xxxx | ---- xxxx | ---- uuuu |
| ADCR0 | 0110 0000 | 0110 0000 | uuuu uuuu |
| ADCR1 | 00-0 -000 | 00-0 -000 | uu-u -uuu |
| ACERL | 1111 1111 | 1111 1111 | uuuu uuuu |
| PD | 1111 1111 | 1111 1111 | uuuu uuuu |
| PDC | 1111 1111 | 1111 1111 | uuuu uuuu |
| PDPU | 0000 0000 | 0000 0000 | uuuu uuuu |

| Register | LVR&power on | WDT Overflow (Normal Mode) | WDT Overflow (HALT Mode) |
|---|---|---|---|
| TKTMR | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKC0 | -000 0000 | -000 0000 | -uuu uuuu |
| TK16DL | 0000 0000 | 0000 0000 | uuuu uuuu |
| TK16DH | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKC1 | ---- --11 | ---- --11 | ---- --uu |
| TKM016DL | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM016DH | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM0ROL | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM0ROH | ---- --00 | ---- --00 | ---- --uu |
| TKM0C0 | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM0C1 | 0-00 0000 | 0-00 0000 | u-uu uuuu |
| TKM116DL | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM116DH | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM1ROL | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM1ROH | ---- --00 | ---- --00 | ---- --uu |
| TKM1C0 | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM1C1 | 0000 0000 | 0000 0000 | uuuu uuuu |
| EEC | ---- 0000 | ---- 0000 | ---- uuuu |

Note: "-" not implement

"u" stands for "unchanged"

"x" stands for "unknown"

**BS84C12A-3 Register**

| Register | LVR&power on | WDT Overflow (Normal Mode) | WDT Overflow (HALT Mode) |
|---|---|---|---|
| IAR0 | ---- ---- | ---- ---- | ---- ---- |
| MP0 | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| IAR1 | ---- ---- | ---- ---- | ---- ---- |
| MP1 | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| BP | ---- ---0 | ---- ---0 | ---- ---u |
| ACC | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| PCL | 0000 0000 | 0000 0000 | 0000 0000 |
| TBLP | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| TBLH | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| TBHP | ---x xxxx | ---u uuuu | ---u uuuu |
| STATUS | --00 xxxx | --1u uuuu | --11 uuuu |
| SMOD | 0000 0011 | 0000 0011 | uuuu uuuu |
| CTRL | 0-00 -x00 | 0-00 -x00 | u-uu -uuu |
| INTEG | ---- --00 | ---- --00 | ---- --uu |
| INTC0 | -000 0000 | -000 0000 | -uuu uuuu |
| INTC1 | -000 -000 | -000 -000 | -uuu -uuu |
| LVRC | 0000 0000 | 0000 0000 | uuuu uuuu |
| PA | 1--1 1111 | 1--1 1111 | u--u uuuu |
| PAC | 1--1 1111 | 1--1 1111 | u--u uuuu |
| PAPU | 0--0 0000 | 0--0 0000 | u--u uuuu |
| PAWU | 0--0 0000 | 0--0 0000 | u--u uuuu |
| WDTC | 0101 0011 | 0101 0011 | uuuu uuuu |
| TBC | --00 ---- | --00 ---- | --uu ---- |
| TMR | 0000 0000 | 0000 0000 | uuuu uuuu |
| TMRC | --00 -000 | --00 -000 | --uu -uuu |
| EEA | --11 1111 | --11 1111 | --uu uuuu |
| EED | 0000 0000 | 0000 0000 | uuuu uuuu |
| PB | 1111 1111 | 1111 1111 | uuuu uuuu |
| PBC | 1111 1111 | 1111 1111 | uuuu uuuu |
| PBPU | 0000 0000 | 0000 0000 | uuuu uuuu |
| I2CTOC | 0000 0000 | 0000 0000 | uuuu uuuu |
| SIMC0 | 0000 -00- | 0000 -00- | uuuu –uu- |
| SIMC1 | 1000 0001 | 1000 0001 | uuuu -uuu |
| SIMD | 0000 0000 | 0000 0000 | uuuu uuuu |
| SIMC2 | --11 1111 | --11 1111 | --uu uuuu |
| SIMA | 0000 0000 | 0000 0000 | uuuu uuuu |
| ADRL(ADRFS=0) | xxxx ---- | xxxx ---- | uuuu ---- |
| ADRL(ADRFS=1) | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| ADRH(ADRFS=0) | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| ADRH(ADRFS=1) | ---- xxxx | ---- xxxx | ---- uuuu |
| ADCR0 | 0110 0000 | 0110 0000 | uuuu uuuu |
| ADCR1 | 00-0 -000 | 00-0 -000 | uu-u -uuu |
| ACERL | 1111 1111 | 1111 1111 | uuuu uuuu |
| PD | 1111 1111 | 1111 1111 | uuuu uuuu |
| PDC | 1111 1111 | 1111 1111 | uuuu uuuu |
| PDPU | 0000 0000 | 0000 0000 | uuuu uuuu |

| Register | LVR&power on | WDT Overflow (Normal Mode) | WDT Overflow (HALT Mode) |
|---|---|---|---|
| PC | 1111 1111 | 1111 1111 | uuuu uuuu |
| PCC | 1111 1111 | 1111 1111 | uuuu uuuu |
| PCPU | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKTMR | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKC0 | -000 0000 | -000 0000 | -uuu uuuu |
| TK16DL | 0000 0000 | 0000 0000 | uuuu uuuu |
| TK16DH | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKC1 | ---- --11 | ---- --11 | ---- --uu |
| TKM016DL | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM016DH | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM0ROL | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM0ROH | ---- --00 | ---- --00 | ---- --uu |
| TKM0C0 | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM0C1 | 0-00 0000 | 0-00 0000 | u-uu uuuu |
| TKM116DL | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM116DH | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM1ROL | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM1ROH | ---- --00 | ---- --00 | ---- --uu |
| TKM1C0 | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM1C1 | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM216DL | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM216DH | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM2ROL | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM2ROH | ---- --00 | ---- --00 | ---- --uu |
| TKM2C0 | 0000 0000 | 0000 0000 | uuuu uuuu |
| TKM2C1 | 0000 0000 | 0000 0000 | uuuu uuuu |
| EEC | ---- 0000 | ---- 0000 | ---- uuuu |

Note: "-" not implement

"u" stands for "unchanged"

"x" stands for "unknown"

## Input/Output Ports

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

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

### I/O Register List

**BS84B08A-3**

| Register Name | Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PAWU | D7 | — | — | D4 | D3 | D2 | D1 | D0 |
| PAPU | D7 | — | — | D4 | D3 | D2 | D1 | D0 |
| PA | D7 | — | — | D4 | D3 | D2 | D1 | D0 |
| PAC | D7 | — | — | D4 | D3 | D2 | D1 | D0 |
| PBPU | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| PB | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| PBC | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| PDPU | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| PD | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| PDC | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

**BS84C12A-3**

| Register Name | Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PAWU | D7 | — | — | D4 | D3 | D2 | D1 | D0 |
| PAPU | D7 | — | — | D4 | D3 | D2 | D1 | D0 |
| PA | D7 | — | — | D4 | D3 | D2 | D1 | D0 |
| PAC | D7 | — | — | D4 | D3 | D2 | D1 | D0 |
| PBPU | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| PB | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| PBC | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| PCPU | — | — | — | — | D3 | D2 | D1 | D0 |
| PC | — | — | — | — | D3 | D2 | D1 | D0 |
| PCC | — | — | — | — | D3 | D2 | D1 | D0 |
| PDPU | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| PD | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| PDC | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

## 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~PDPU, and are implemented using weak PMOS transistors.

### PAPU Register

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

Bit 7 I/O Port A bit 7 Pull-High Control
0: Disable
1: Enable

Bit 6 ~ 5 Unimplemented, read as "0"

Bit 4 ~ 0 I/O Port A bit 4~ bit 0 Pull-High Control
0: Disable
1: Enable

### 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 |

Bit 7 ~ 0 I/O Port B bit 7~ bit 0 Pull-High Control
0: Disable
1: Enable

### PCPU Register – BS84C12A-3

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

Bit 7 ~ 4 Unimplemented, read as "0"

Bit 3 ~ 0 I/O Port C bit 3~ bit 0 Pull-High Control
0: Disable
1: Enable

### 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 D bit 7~ bit 0 Pull-High Control
0: Disable
1: Enable

## Port A Wake-up

The HALT instruction forces the microcontroller into the SLEEP or IDLE Mode 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. 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 | — | — | D4 | D3 | D2 | D1 | D0 |
| R/W | R/W | — | — | R/W | R/W | R/W | R/W | R/W |
| POR | 0 | — | — | 0 | 0 | 0 | 0 | 0 |

Bit 7 I/O Port A bit 7 Pull-High Control
0: Disable
1: Enable

Bit 6 ~ 5 Unimplemented, read as "0"

Bit 4 ~ 0 I/O Port A bit 4 ~ 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~PDC, to control the input/output configuration. With this control register, each CMOS output or input 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 | — | — | D4 | D3 | D2 | D1 | D0 |
| R/W | R/W | — | — | R/W | R/W | R/W | R/W | R/W |
| POR | 1 | — | — | 1 | 1 | 1 | 1 | 1 |

Bit 7 I/O Port A bit 7 Input/Output Control
0: Disable
1: Enable

Bit 6 ~ 5 Unimplemented, read as "0"

Bit 4 ~ 0 I/O Port A bit 4 ~ bit 0 Input/Output Control
0: Disable
1: Enable

### PBC 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 B bit 7~ bit 0 Input/Output Control
0: Output
1: Input

### PCC Register – BS84C12A-3

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

Bit 7 ~ 4 Unimplemented, read as "0"

Bit 3 ~ 0 I/O Port C bit 3~ bit 0 Input/Output Control
0: Disable
1: Enable

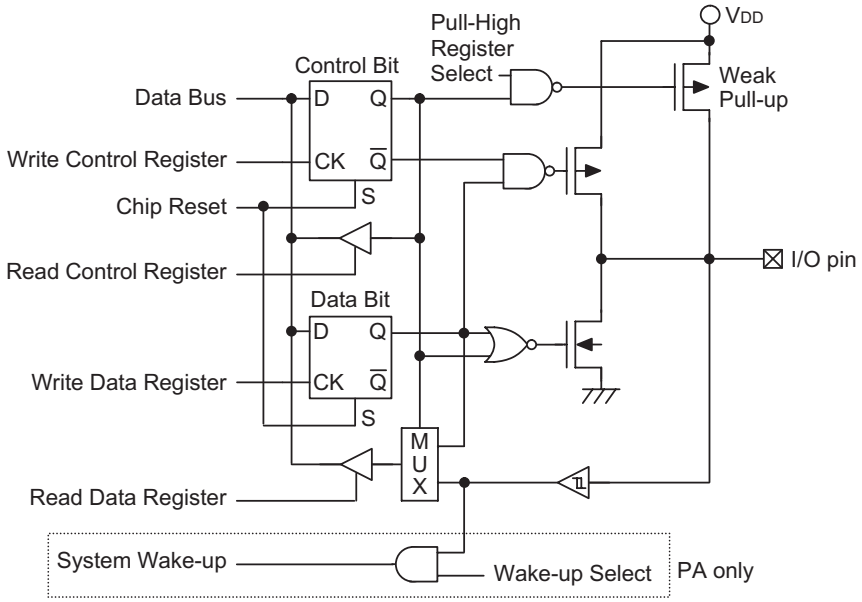
### PDC Register

| 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 |

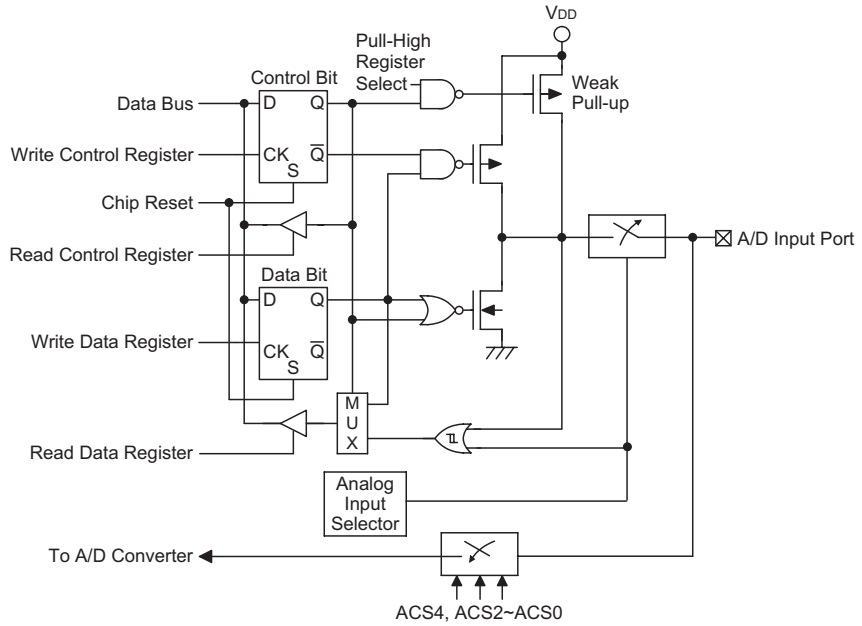
Bit 7 ~ 0 I/O Port D bit 7~ bit 0 Input/Output Control
0: Disable
1: Enable

## I/O Pin Structures

The accompanying diagrams illustrate the internal structures of some generic 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

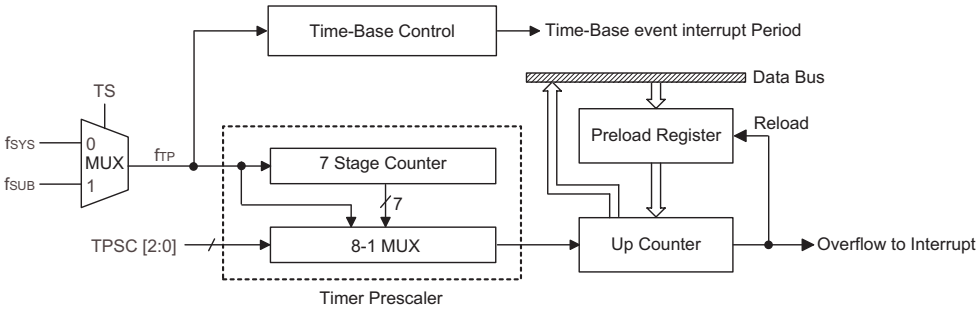
Within the user program, one of the first things to consider is port initialisation. After a reset, all of the I/O data and port control registers will be set high. This means that all I/O pins will default to an input state, the level of which depends on the other connected circuitry and whether pull-high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

## Timer/Event Counter

The provision of timers form an important part of any microcontroller, giving the designer a means of carrying out time related functions. The devices contain one 8-bit. The 8-bit timer is a general timer. The provision of an internal prescaler to the clock circuitry on gives added range to the timers.

There are two types of registers related to the Timer/Event Counter. The first is the register that contains the actual value of the timer and into which an initial value can be preloaded. Reading from this register retrieves the contents of the Timer/Event Counter. The second type of associated register is the Timer Control Register which defines the timer options.




**Timer/Event Counter**

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

The Timer/Event Counter clock source can originate from either the system clock $f_{SYS}$ or the $f_{SUB}$ oscillator, the choice of which is determined by the TS bit in the TMRC register. This internal clock source is first divided by a prescaler, the division ratio of which is conditioned by the Timer Control Register bits TPSC0~TPSC2.

## Timer Register – TMR

The timer register is a special function register located in the Special Purpose Data Memory and is the place where the actual timer value is stored, it is known as TMR. The value in the timer register increases by one each time an internal clock pulse is received The timer will count from the initial value loaded by the preload register to the full count of FFH at which point the timer overflows and an internal interrupt signal is generated. The timer value will then be reset with the initial preload register value and continue counting.

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

## Timer Control Register – TMRC

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

The timer-on bit, which is bit 4 of the Timer Control Register and known as TON, provides the basic on/off control of the timer. Setting the bit high allows the counter to run, clearing the bit stops the counter. Bits 0~2 of the Timer Control Register determine the division ratio of the input clock prescaler. The TS bit selects the internal clock source.

### TMRC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | TS | TON | — | TPSC2 | TPSC1 | TPSC0 |
| R/W | — | — | R/W | R/W | — | R/W | R/W | R/W |
| POR | — | — | 0 | 0 | — | 0 | 0 | 0 |

Bit 7 ~ 6 Unimplemented, read as "0"

Bit 5 **TS**: Timer/Event Counter Clock Source
0: $f_{SYS}$
1: $f_{SUB}$

Bit 4 **TON**: Timer/Event Counter Counting Enable
0: disable
1: enable

Bit 3 Unimplemented, read as "0"

Bits 2 ~ 0 **TPSC2~TPSC0**: Timer prescaler rate selection
Timer internal clock=
000: $f_{TP}$
001: $f_{TP}/2$
010: $f_{TP}/4$
011: $f_{TP}/8$
100: $f_{TP}/16$
101: $f_{TP}/32$
110: $f_{TP}/64$
111: $f_{TP}/128$

### Timer Operation

The Timer/Event Counter is utilized to measure fixed time intervals, providing an internal interrupt signal each time the Timer/Event Counter overflows. The timer input clock source is either $f_{SYS}$ or $f_{SUB}$, however, this timer clock source is further divided by a prescaler, the value of which is determined by the bits TPSC2~TPSC0 in the Timer Control Register. The timer-on bit, TON must be set high to enable the timer to run. Each time an internal clock transition occurs, the timer increments by one; when the timer is full and overflows, an interrupt signal is generated and the timer will reload the value already loaded into the preload register and continue counting. A timer overflow condition and corresponding internal interrupt is one of the wake-up sources, however, the internal interrupts can be disabled by ensuring that the timer enable bit in the interrupt register is reset to zero.

### Prescaler

Bits TPSC0~TPSC2 of the TMRC register can be used to define a division ratio for the internal clock source of the Timer/Event Counter enabling longer time out periods to be setup.

### Programming Considerations

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 timer is properly initialised before using it 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. It is also important to ensure that an initial value is first loaded into the timer registers before the timer is switched on; this is because after power-on the initial values of the timer registers are unknown.

After the timer has been initialised the timer can be turned on and off by controlling the enable bit in the timer control register. When the Timer/Event Counter overflows, its corresponding interrupt request flag in the interrupt control register will be set. If the Timer/Event Counter interrupt is enabled this will in turn generate an interrupt signal. However irrespective of whether the interrupts are enabled or not, a Timer/Event Counter overflow will also generate a wake-up signal if the device is in a 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 Idle/Sleep Mode.

## Analog to Digital Converter

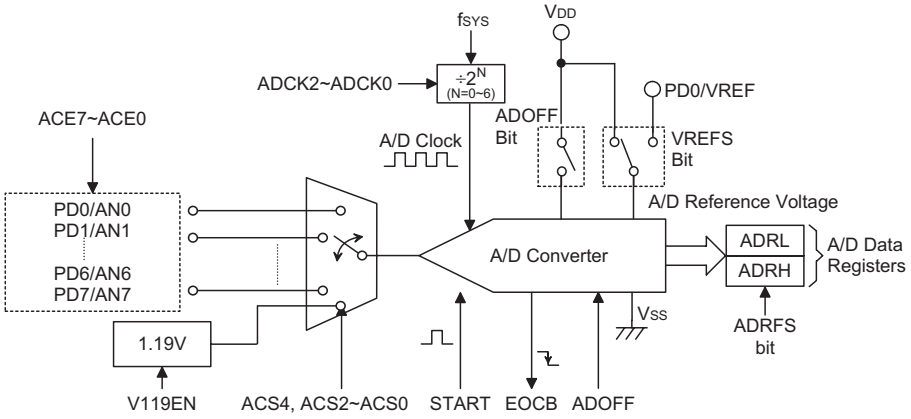
The need to interface to real world analog signals is a common requirement for many electronic systems. However, to properly process these signals by a microcontroller, they must first be converted into digital signals by A/D converters. By integrating the A/D conversion electronic circuitry into the 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 a multi-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 a 12-bit digital value.

| Part No. | Input Channels | A/D Channel Select Bits | Input Pins |
|---|---|---|---|
| BS84B08A-3 | 8 | ACS4,ACS2~ACS0 | AN0~AN7 |
| BS84C12A-3 | 8 | ACS4,ACS2~ACS0 | AN0~AN7 |

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




**A/D Converter Structure**

### A/D Converter Register Description

Overall operation of the A/D converter is controlled using five registers. A read only register pair exists to store the ADC data 12-bit value. The remaining three registers are control registers which setup the operating and control function of the A/D converter.

| Name | Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ADRL(ADRFS=0) | D3 | D2 | D1 | D0 | — | — | — | — |
| ADRL(ADRFS=1) | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| ADRH(ADRFS=0) | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 |
| ADRH(ADRFS=1) | — | — | — | — | D11 | D10 | D9 | D8 |
| ADCR0 | START | EOCB | ADOFF | ADRFS | — | ACS2 | ACS1 | ACS0 |
| ADCR1 | ACS4 | V119EN | — | VREFS | — | ADCK2 | ADCK1 | ADCK0 |
| ACERL | ACE7 | ACE6 | ACE5 | ACE4 | ACE3 | ACE2 | ACE1 | ACE0 |

**A/D Converter Register List**

## A/D Converter Data Registers – ADRL, ADRH

As the devices contain 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 digitized conversion value. As only 12 bits of the 16-bit register space is utilized, 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 Registers – ADCR0, ADCR1, ACERL

To control the function and operation of the A/D converter, three control registers known as ADCR0, ADCR1, ACERL are provided. These 8-bit registers define functions such as the selection of which analog channel is connected to the internal A/D converter, the digitized data format, 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 signals or internal 1.19V is actually connected to the internal A/D converter.

The ACERL control register contains the ACE7~ACE0 bits which determine which pins on Port D 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 | ADRFS | — | 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 **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 register section.

Bit 3 Unimplemented, read as "0"

Bit 2 ~ 0 **ACS2 ~ 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 1.19V will be routed to the A/D Converter.

### ADCR1 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | ACS4 | V119EN | — | 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 **ACS4**: Select Internal 1.19V as ADC input Control

0: Disable

1: Enable

This bit enables 1.19V to be connected to the A/D converter. The V119EN bit must first have been set to enable the bandgap circuit 1.19V voltage to be used by the A/D converter. When the ACS4 bit is set high, the bandgap 1.19V voltage will be routed to the A/D converter and the other A/D input channels disconnected.

Bit 6 **V119EN**: Internal 1.19V 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 1.19V voltage can be used by the A/D converter. If 1.19V is not used by the A/D converter and the LVR function is disabled then the bandgap reference circuit will be automatically switched off to conserve power. When 1.19V 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 Unimplemented, read as "0"

Bit 4 **VREFS**: Select 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 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 ($f_{SUB}$ for test)

These three bits are used to select the clock source for the A/D converter.

**ACERL Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | ACE7 | ACE6 | ACE5 | ACE4 | ACE3 | ACE2 | ACE1 | ACE0 |
| 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 **ACE7**: Define PD7 is A/D input or not
0: Not A/D input
1: A/D input, AN7

Bit 6 **ACE6**: Define PD6 is A/D input or not
0: Not A/D input
1: A/D input, AN6

Bit 5 **ACE5**: Define PD5 is A/D input or not
0: Not A/D input
1: A/D input, AN5

Bit 4 **ACE4**: Define PD4is A/D input or not
0: Not A/D input
1: A/D input, AN4

Bit 3 **ACE3**: Define PD3 is A/D input or not
0: Not A/D input
1: A/D input, AN3

Bit 2 **ACE2**: Define PD2 is A/D input or not
0: Not A/D input
1: A/D input, AN2

Bit 1 **ACE1**: Define PD1 is A/D input or not
0: Not A/D input
1: A/D input, AN1

Bit 0 **ACE0**: Define PD0 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, $t_{ADCK}$, is from 0.5μs, care must be taken for system clock frequencies. For example, if the system clock operates at a frequency of 4MHz, the ADCK2~ADCK0 bits should not be set to 000B or 110B. Doing so will give A/D clock periods that are less than the minimum A/D clock period or greater than the maximum 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 |
| 8MHz | 125ns* | 250ns* | 500ns | 1μs | 2μs | 4μs | 8μs | Undefined |
| 12MHz | 83ns* | 167ns* | 333ns* | 667ns | 1.33μs | 2.67μs | 5.33μs | Undefined |
| 16MHz | 62.5ns* | 125ns* | 250ns* | 500ns | 1μs | 2μs | 4μs | Undefined |
| 20MHz | 50ns* | 100ns* | 200ns* | 400ns* | 800ns | 1.6μs | 3.2μs | Undefined |

**A/D Clock Period Examples**

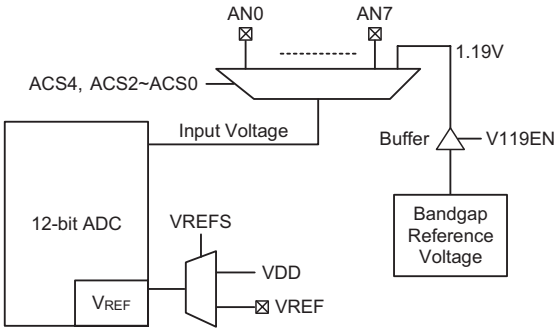
Controlling the power on/off function of the A/D converter circuitry is implemented using the ADOFF bit in the ADCR0 register. This bit must be zero to power on the A/D converter. When the ADOFF bit is cleared to zero to power on the A/D converter internal circuitry a certain delay, as indicated in the 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 ACERL registers, 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 Port A as well as other functions. The ACE7~ACE0 bits in the ACERL registers, 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 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 VREF.




**A/D Input Structure**

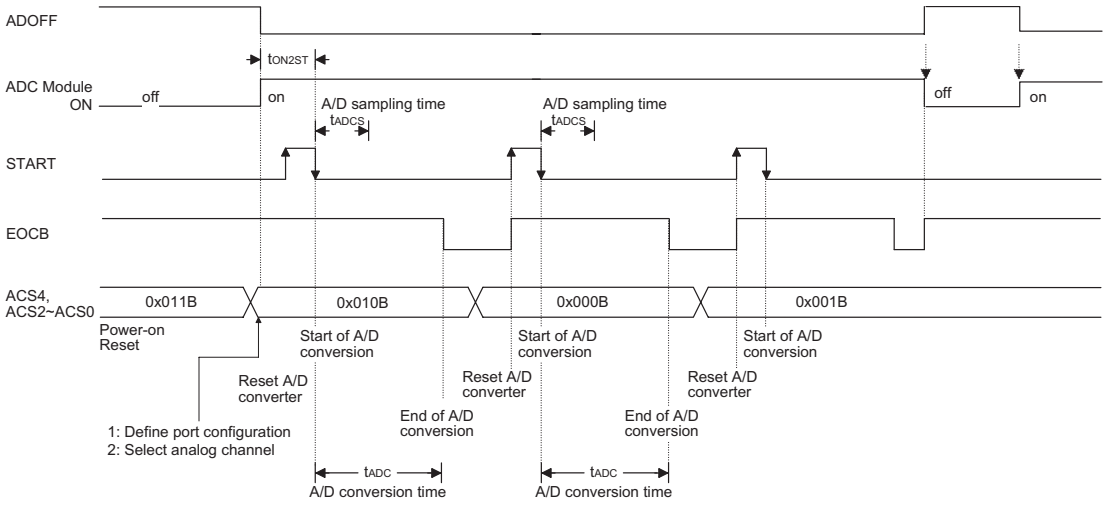
## Summary of A/D Conversion Steps

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, ACS2~ACS0 bits which are also contained in the ADCR1 and 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 ACERL 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 ADCR0 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 taken for the A/D conversion is 16 $t_{ADCK}$ where $t_{ADCK}$ is equal to the A/D clock period.




**A/D Conversion Timing**

## 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 Transfer Function

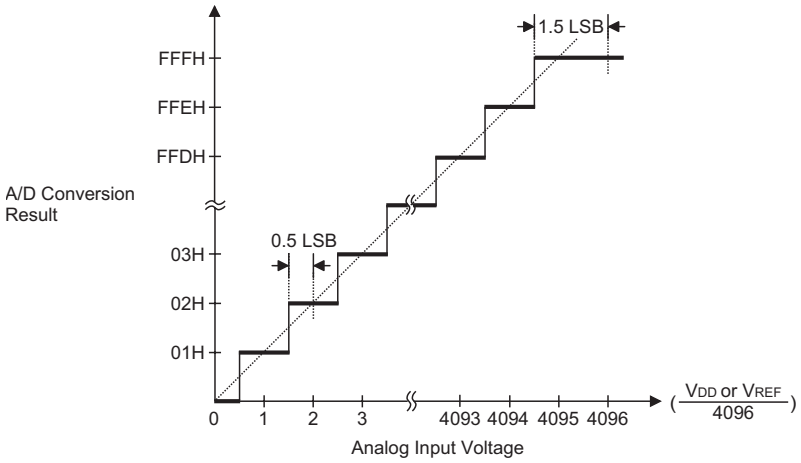
As the device contains a 12-bit A/D converter, its full-scale converted digitized value is equal to FFFH. Since the full-scale analog input value is equal to the $V_{DD}$ or $V_{REF}$ voltage, this gives a single bit analog input value of $V_{DD}$ or $V_{REF}$ divided by 4096.

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

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

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

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 digitized value will change at a point 1.5 LSB below the $V_{DD}$ or $V_{REF}$ level.




**Ideal A/D Transfer Function**

## A/D Programming Examples

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 ADCR0 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 1.19V
clr     ADOFF
mov    a,0Fh              ; setup ACERL to configure pins AN0~AN3
mov    ACERL,a
mov    a,01h
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: 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 1.19V
Clr     ADOFF
mov    a,0Fh              ; setup ACERL to configure pins AN0~AN3
mov    ACERL,a
mov    a,01h
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
```

# Touch Key Function

Each device provides multiple touch key functions. The touch key function is fully integrated and requires no external components, allowing touch key functions to be implemented by the simple manipulation of internal registers.

## Touch Key Structure

The touch keys are pin shared with the PB and PC logic I/O pins, with the desired function chosen via register bits. Keys are organized into groups of four, with each group known as a module and having a module number, M0 to M2. Each module is a fully independent set of four Touch Keys and each Touch Key has its own oscillator. Each module contains its own control logic circuits and register set. Examination of the register names will reveal the module number it is referring to.

| Device | Keys - n | Touch Key Module | Touch Key | Shared I/O Pin |
|---|---|---|---|---|
| BS84B08A-3 | 8 | M0 | K1~K4 | PB0~PB3 |
| | | M1 | K5~K8 | PB4~PB7 |
| BS84C12A-3 | 12 | M0 | K1~K4 | PB0~PB3 |
| | | M1 | K5~K8 | PB4~PB7 |
| | | M2 | K9~K12 | PC0~PC3 |

## Touch Key Register Definition

Each touch key module, which contains four touch key functions, has its own suite registers. The following table shows the register set for each touch key module. The Mn within the register name refers to the Touch Key module number, BS84B08A-3 has a range of M0 to M1, BS84C12A-3 has a range of M0 to M2.

| Name | Usage |
|---|---|
| TKTMR | Touch Key 8-bit timer/counter register |
| TKC0 | Counter on-off and clear control/reference clock control/Start bit |
| TK16DL | Touch key module 16-bit counter low byte contents |
| TK16DH | Touch key module 16-bit counter high byte contents |
| TKC1 | Touch key OSC frequency select |
| TKMn16DL | Module n 16-bit counter low byte contents |
| TKMn16DH | Module n 16-bit counter high byte contents |
| TKMnROL | Reference OSC internal capacitor select |
| TKMnROH | Reference OSC internal capacitor select |
| TKMnC0 | Control Register 0<br>Multiplexer Key Select |
| TKMnC1 | Control Register 1<br>Key oscillator control/Reference oscillator control/ Touch key or I/O select |

**Register Listing**

| Register Name | Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TKTMR | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| TKC0 | — | TKRCOV | TKST | TKCFOV | TK16OV | TSCS | TK16S1 | TK16S0 |
| TK16DL | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| TK16DH | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| TKC1 | — | — | — | — | — | — | TKFS1 | TKFS0 |
| TK-Mn16DL | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| TK-Mn16DH | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| TKMn-ROL | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| TKMn-ROH | — | — | — | — | — | — | D9 | D8 |
| TKMnC0 | MnMXS1 | MnMXS0 | MnDFEN | MnFILEN | MnSOFC | MnSOF2 | MnSOF1 | MnSOF0 |
| TKMnC1 | MnTSS | — | MnROEN | MnKOEN | MnK4IO | MnK3IO | MnK2IO | MnK1IO |

**Touch Key Module(n=0~2)**

### TKTMR 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 Touch Key 8-bit timer/counter register
Time slot counter overflow set-up time is (256-TKTMR[7:0]) x 32

### TKC0 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | TKRCOV | TKST | TKCFOV | TK16OV | TSCS | TK16S1 | TK16S0 |
| 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 **TKRCOV**: Time slot counter overflow flag
0: No overflow
1: Overflow
If module 0 or all module (select by TSCS bit) time slot counter is overflow, the Touch Key Interrupt request flag will be set (TKMF) and all module key OSC and ref OSC auto stop. All module 16-bit C/F counter, 16-bit counter, 5-bit time slot counter and 8-bit time slot timer counter will be automatically switched off.

Bit 5 **TKST**: Start Touch Key detection control bit
0: Stopped
0→1: Started
In all modules the16-bit C/F counter, 16-bit counter, 5-bit time slot counter will be automatically cleared when this bit is cleared to "0" (8-bit programmable time slot counter will not be cleared, which overflow time is setup by user). When this bit changes from low to high, the 16-bit C/F counter, 16-bit counter, 5-bit time slot counter and 8-bit time slot timer counter will be automatically on and enable key OSC and ref OSC output clock input these counter.

Bit 4 **TKCFOV**: Touch key module 16-bit C/F counter overflow flag
0: Not overflow
1: Overflow
This bit must be cleared by software.

Bit 3 **TK16OV**: Touch key module 16-bit counter overflow flag
0: Not overflow
1: Overflow
This bit must be cleared by software.

Bit 2 **TSCS**: Touch Key time slot counter select
0: Each Module use own time slot counter.
1: All Touch Key Module use Module 0 time slot counter.

Bit 1~0 **TK16S1~ TK16S0**: The touch key module 16-bit counter clock source select
00: $f_{SYS}$
01: $f_{SYS}/2$
10: $f_{SYS}/4$
11: $f_{SYS}/8$

### TKC1 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | — | — | — | TKFS1 | TKFS0 |
| R/W | — | — | — | — | — | — | R/W | R/W |
| POR | — | — | — | — | — | — | 0 | 0 |

Bit 7 ~2 Unimplemented, read as "0"

Bit 1~0 **TKFS1~TKFS0**: Touch key OSC frequency select
00: 500kHz
01: 1000 kHz
10: 1500 kHz
11: 2000 kHz

### TK16DL Register

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

Bit 7~0 Touch key module 16-bit counter low byte contents

### TK16DH Register

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

Bit 7~0 Touch key module 16-bit counter high byte contents

### TKMn16DL Register

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

Bit 7~0 Module n 16-bit counter low byte contents

### TKMn16DH Register

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

Bit 7~0 Module n 16-bit counter high byte contents

### TKMnROL 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 Reference OSC internal capacitor select
OSC internal capacitor select : (TKMnRO[9:0] x 50pF) / 1024

### TKMnROH Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | — | — | — | D9 | D8 |
| R/W | — | — | — | — | — | — | R/W | R/W |
| POR | — | — | — | — | — | — | 0 | 0 |

Bit 7 ~2 Unimplemented, read as “0”
Bit 1~0 Reference OSC internal capacitor select
OSC internal capacitor select : (TKMnRO[9:0] x 50pF) / 1024

**TKMnC0 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | MnMXS1 | MnMXS0 | MnDFEN | MnFILEN | MnSOFC | MnSOF2 | MnSOF1 | MnSOF0 |
| 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 **MnMXS1~MnMXS0**: Multiplexer key select

Bit 5 **MnDFEN**: Multi-frequency control
0: Disable
1: Enable

Bit 4 **MnFILEN**: Filter function control
0: Disable
1: Enable

Bit3 **MnSOFC**: C to F OSC frequency hopping function control
0: The frequency hopping function is controlled by MnSOF2~ MnSOF0 bits
1: The frequency hopping function is controlled by hardware regardless of what is the state of MnSOF2~ MnSOF0 bits

Bit 2~0 **MnSOF2~ MnSOF0**: Selecting key OSC and ref OSC frequency as C to F OSC is controlled by software
000: 1380kHz
001: 1500kHz
010: 1670kHz
011: 1830kHz
100: 2000kHz
101: 2230kHz
110: 2460kHz
111: 2740kHz

The frequency which is mentioned here will be changed when the external or internal capacitor is with different value. if the touch key operates at 2MHz frequency, users can adjust the frequency in scale when select other frequency.

| Bit | | Module Number | | |
|---|---|---|---|---|
| MnMXS1 | MnMXS0 | M0 | M1 | M2 |
| 0 | 0 | Key 1 | Key 5 | Key 9 |
| 0 | 1 | Key 2 | Key 6 | Key 10 |
| 1 | 0 | Key 3 | Key 7 | Key 11 |
| 1 | 1 | Key 4 | Key 8 | Key 12 |

### TKMnC1 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | MnTSS | — | MnROEN | MnKOEN | MnK4IO | MnK3IO | MnK2IO | MnK1IO |
| 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 **MnTSS**: Time slot counter clock select
0:Reference OSC
1:$f_{SYS}/4$

Bit 6 Unimplemented, read as "0"

Bit 5 **MnROEN**: Reference OSC control
0: Disable
1: Enable

Bit 4 **MnKOEN**: Key OSC control
0: Disable
1: Enable

Bit 3~0 **MnK4IO~ MnK1IO**: I/O pin or touch key function select

| MnK4OEN | M0 | M1 | M2 |
|---|---|---|---|
| | PB3/Key4 | PB7/Key8 | PC3/Key12 |
| 0 | I/O | | |
| 1 | Touch key input | | |

| MnK3OEN | M0 | M1 | M2 |
|---|---|---|---|
| | PB2/Key3 | PB6/Key7 | PC2/Key11 |
| 0 | I/O | | |
| 1 | Touch key input | | |

| MnK2OEN | M0 | M1 | M2 |
|---|---|---|---|
| | PB1/Key2 | PB5/Key6 | PC1/Key10 |
| 0 | I/O | | |
| 1 | Touch key input | | |

| MnK1OEN | M0 | M1 | M2 |
|---|---|---|---|
| | PB0/Key1 | PB4/Key5 | PC0/Key9 |
| 0 | I/O | | |
| 1 | Touch key input | | |

## Touch Key Operation

When a finger touches or is in proximity to a touch pad, the capacitance of the pad will increase. By using this capacitance variation to change slightly the frequency of the internal sense oscillator, touch actions can be sensed by measuring these frequency changes. Using an internal programmable divider the reference clock is used to generate a fixed time period. By counting a number of generated clock cycles from the sense oscillator during this fixed time period touch key actions can be determined.

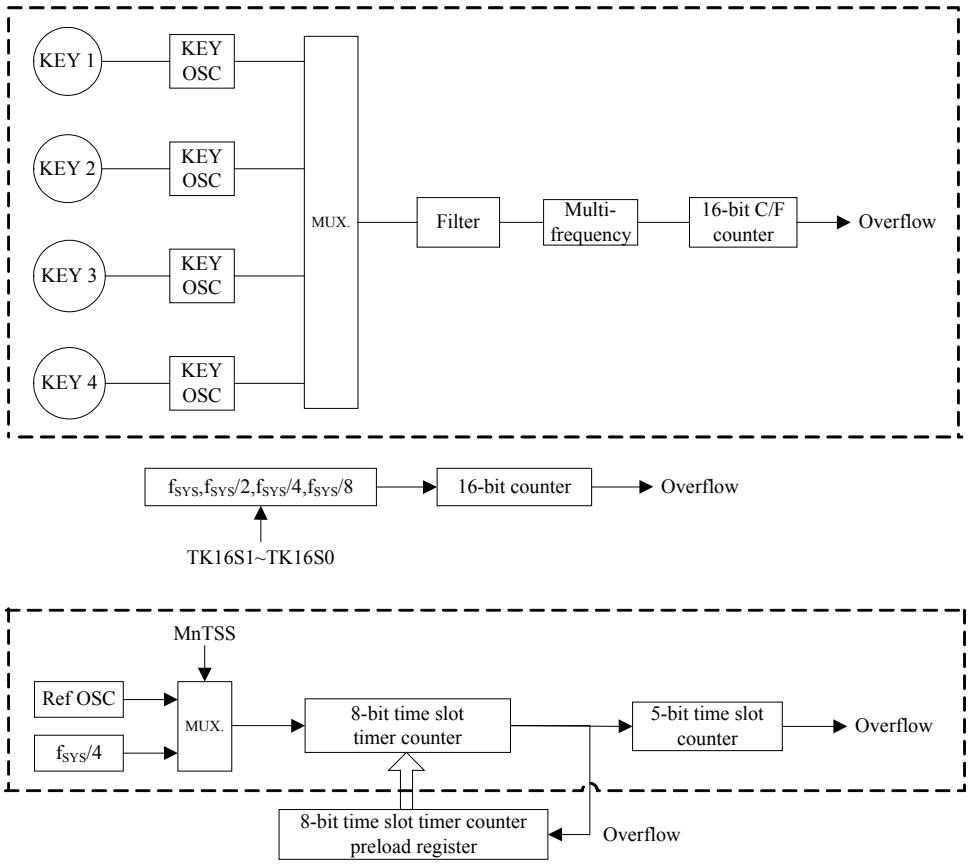
During this reference clock fixed interval, the number of clock cycles generated by the sense oscillator is measured, and it is this value that is used to determine if a touch action has been made or not. These devices contain four touch key inputs which are shared with logical I/O pins, with the desired function selected using register bits.

Using the TSCS bit in the TKC0 register can select the module 0 time slot counter as the time slot counter for all modules. All modules use the same started signal. The16-bit C/F counter, 16-bit counter, 5-bit time slot counter in all modules will be automatically cleared when this bit is cleared to "0", but the 8-bit programmable time slot counter will not be cleared. The overflow time is setup by user. When this bit changes from low to high, the 16-bit C/F counter, 16-bit counter, 5-bit time slot counter and 8-bit time slot timer counter will be automatically switched on.

The key oscillator and reference oscillator in all modules will be automatically stopped and the 16-bit C/F counter, 16-bit counter, 5-bit time slot counter and 8-bit time slot timer counter will be automatically switched off when the 5-bit time slot counter overflows. The clock source for the time slot counter and 8+5 bit counter, is sourced from the reference oscillator or $f_{SYS}/4$. The reference oscillator and key oscillator will be enabled by setting the MnROEN bit and MnKOEN bits in the TKMnC1 register .

When the time slot counter in all the touch key modules or in the touch key module 0 overflows, an actual touch key interrupt will take place. The touch keys mentioned here are the keys which are enabled.

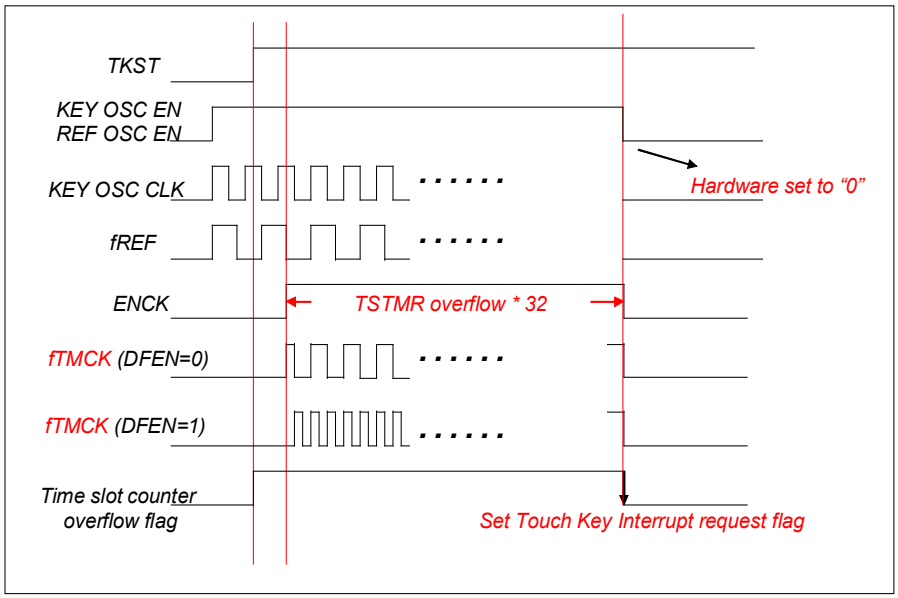
Each touch key module, which consists of four touch keys, Key 1~Key 4 is contained in module 0, Key 5~Key 8 is contained in module 1 and Key 9~Key 12 is contained in module 2. Each touch key module has an identical structure.

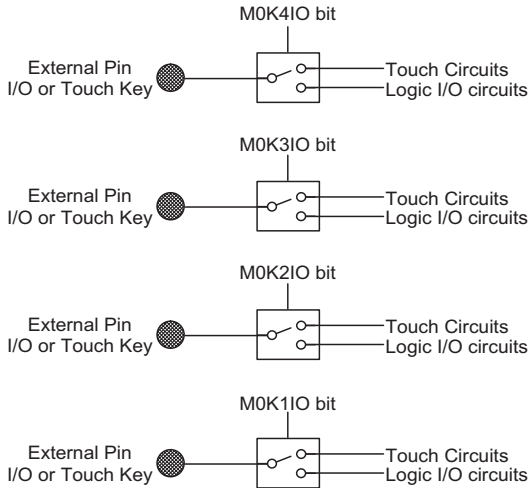



Note: Each touch key module contains the content in the dash line.

**Touch Key Module Block Diagram**

The touch key sense oscillator and reference oscillator timing diagram is shown in the following figure:




**Touch Key or I/O Function Select**

## Touch Key Interrupt

The touch key only has single interrupt, when the time slot counter in all the touch key modules or in the touch key module 0 overflows, an actual touch key interrupt will take place. The touch keys mentioned here are the keys which are enabled. The16-bit C/F counter, 16-bit counter, 5-bit time slot counter and 8-bit time slot counter in all modules will be automatically cleared.

The TKCFOV flag, which is the 16-bit C/F counter overflow flag will go high when any of the Touch Key Module 16-bit C/F counter overflows. As this flag will not be automatically cleared, it has to be cleared by the application program.

The module 0 only contains one 16-bit counter. The TK16OV flag, which is the 16-bit counter overflow flag will go high when the 16-bit counter overflows. As this flag will not be automatically cleared, it has to be cleared by the application program. More details regarding the touch key interrupt is located in the interrupt section of the datasheet.

### Programming Considerations

After the relevant registers are setup, the touch key detection process is initiated the changing the TKST bit from low to high. This will enable and synchronise all relevant oscillators. The TKRCOV flag, which is the time slot counter flag will go high and remain high until the counter overflows. When this happens an interrupt signal will be generated.

When the external touch key size and layout are defined, their related capacitances will then determine the sensor oscillator frequency.

## Serial Interface Module – SIM

These devices contain a Serial Interface Module, which include both the four line SPI interface and the two line I2C interface types, to allow an easy method of communication with external peripheral hardware. Having relatively simple communication protocols, these serial interface types allow the microcontroller to interface to external SPI or I2C based hardware such as sensors, Flash memory or EEPROM memory, etc. The SIM interface pins are pin-shared with other I/O pins and must be selected using the SIMEN bit in the SIMC0 register. As both interface types share the same pins and registers, the choice of whether the SPI or I2C type is used is made using the SIM operating mode control bits, named SIM2~SIM0, in the SIMC0 register.

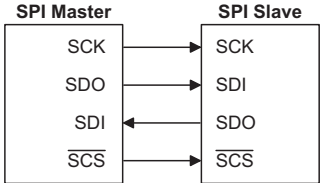
### SPI Interface

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

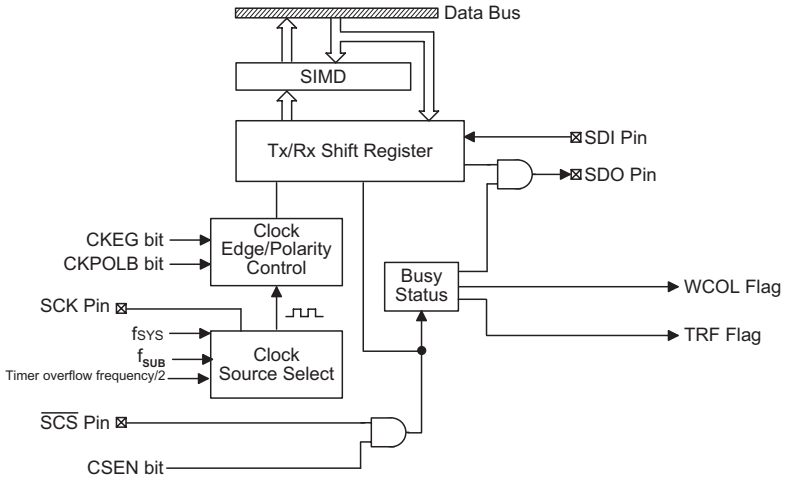
The communication is full duplex and operates as a slave/master type, where the device can be either master or slave. Although the SPI interface specification can control multiple slave devices from a single master, but this device provided only one SCS pin. If the master needs to control multiple slave devices from a single master, the master can use I/O pin to select the slave devices.

#### SPI Interface Operation

The SPI interface is a full duplex synchronous serial data link. It is a four line interface with pin names SDI, SDO, SCK and $\overline{SCS}$. Pins SDI and SDO are the Serial Data Input and Serial Data Output lines, SCK is the Serial Clock line and $\overline{SCS}$ is the Slave Select line. As the SPI interface pins are pin-shared with normal I/O pins and with the $I^2C$ function pins, the SPI interface must first be enabled by setting the correct bits in the SIMC0 and SIMC2 registers. Communication between devices connected to the SPI interface is carried out in a slave/master mode with all data transfer initiations being implemented by the master. The Master also controls the clock signal. As the device only contains a single $\overline{SCS}$ pin only one slave device can be utilized. The $\overline{SCS}$ pin is controlled by software, set CSEN bit to "1" to enable SCS pin function, set CSEN bit to "0" the SCS pin will be as I/O function.




**SPI Master/Slave Connection**




**SPI Block Diagram**

The SPI function in this device offers the following features:

- Full duplex synchronous data transfer
- Both Master and Slave modes
- LSB first or MSB first data transmission modes
- Transmission complete flag
- Rising or falling active clock edge

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

## SPI Registers

There are three internal registers which control the overall operation of the SPI interface. These are the SIMD data register and two registers SIMC0 and SIMC2. Note that the SIMC1 register is only used by the $I^2C$ interface.

| Register Name | Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SIMC0 | SIM2 | SIM1 | SIM0 | — | — | — | SIMEN | — |
| SIMD | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| SIMC2 | — | — | CKPOLB | CKEG | MLS | CSEN | WCOL | TRF |

**SIM Registers List**

The SIMD register is used to store the data being transmitted and received. The same register is used by both the SPI and $I^2C$ functions. Before the device writes data to the SPI bus, the actual data to be transmitted must be placed in the SIMD register. After the data is received from the SPI bus, the device can read it from the SIMD register. Any transmission or reception of data from the SPI bus must be made via the SIMD register.

• **SIMD Register**

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

"x" unknown

There are also two control registers for the SPI interface, SIMC0 and SIMC2. Note that the SIMC2 register also has the name SIMA which is used by the I²C function. The SIMC1 register is not used by the SPI function, only by the I²C function. Register SIMC0 is used to control the enable/disable function and to set the data transmission clock frequency. Although not connected with the SPI function, the SIMC0 register is also used to control the Peripheral Clock Prescaler. Register SIMC2 is used for other control functions such as LSB/MSB selection, write collision flag etc.

• **SIMC0 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | SIM2 | SIM1 | SIM0 | — | — | — | SIMEN | — |
| R/W | R/W | R/W | R/W | — | — | — | R/W | — |
| POR | 1 | 1 | 1 | — | — | — | 0 | — |

Bit 7 ~ 5 **SIM2~SIM0**: SIM Operating Mode Control

000: SPI master mode; SPI clock is $f_{SYS}/4$

001: SPI master mode; SPI clock is $f_{SYS}/16$

010: SPI master mode; SPI clock is $f_{SYS}/64$

011: SPI master mode; SPI clock is $f_{SUB}$

100: SPI master mode; SPI clock is TIMER overflow frequency/2

101: SPI slave mode

110: I²C mode

111: Reserved

These bits setup the overall operating mode of the SIM function. As well as selecting if the I²C or SPI function, they are used to control the SPI Master/Slave selection and the SPI Master clock frequency. The SPI clock is a function of the system clock but can also be chosen to be sourced from the Timer/Event counter. If the SPI Slave Mode is selected then the clock will be supplied by an external Master device.

Bit 4~2 Unimplemented, read as "0"

Bit 1 **SIMEN**: SIM Control

0: Disable

1: Enable

The bit is the overall on/off control for the SIM interface. When the SIMEN bit is cleared to zero to disable the SIM interface, the SDI, SDO, SCK and $\overline{SCS}$, or SDA and SCL lines will be as I/O function and the SIM operating current will be reduced to a minimum value. If the SIM is configured to operate as an SPI interface via the SIM2~SIM0 bits, the contents of the SPI control registers will remain at the previous settings when the SIMEN bit changes from low to high and should therefore be first initialised by the application program. If the SIM is configured to operate as an I²C interface via the SIM2~SIM0 bits and the SIMEN bit changes from low to high, the contents of the I²C control bits such as HTX and TXAK will remain at the previous settings and should therefore be first initialised by the application program while the relevant I²C flags such as HCF, HAAS, HBB, SRW and RXAK will be set to their default states.

Bit 0 Unimplemented, read as "0"

• **SIMC2 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | CKPOLB | CKEG | MLS | CSEN | WCOL | TRF |
| R/W | — | — | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | — | — | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7 ~ 6 Unimplemented, read as "0"

Bit 5 **CKPOLB**: Determines the base condition of the clock line

0: the SCK line will be high when the clock is inactive

1: the SCK line will be low when the clock is inactive

The CKPOLB bit determines the base condition of the clock line, if the bit is high, then the SCK line will be low when the clock is inactive. When the CKPOLB bit is low, then the SCK line will be high when the clock is inactive.

Bit 4 **CKEG**: Determines SPI SCK active clock edge type

CKPOLB=0

0: SCK is high base level and data capture at SCK rising edge

1: SCK is high base level and data capture at SCK falling edge

CKPOLB=1

0: SCK is low base level and data capture at SCK falling edge

1: SCK is low base level and data capture at SCK rising edge

The CKEG and CKPOLB bits are used to setup the way that the clock signal outputs and inputs data on the SPI bus. These two bits must be configured before data transfer is executed otherwise an erroneous clock edge may be generated. The CKPOLB bit determines the base condition of the clock line, if the bit is high, then the SCK line will be low when the clock is inactive. When the CKPOLB bit is low, then the SCK line will be high when the clock is inactive.

Bit 3 **MLS**: SPI Data shift order

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

Bit 2 **CSEN**: SPI $\overline{\text{SCS}}$ pin Control

0: Disable

1: Enable

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

Bit 1 **WCOL**: SPI Write Collision flag

0: No collision

1: Collision

The WCOL flag is used to detect if a data collision has occurred. If this bit is high it means that data has been attempted to be written to the SIMD register during a data transfer operation. This writing operation will be ignored if data is being transferred. The bit can be cleared by the application program.

Bit 0 **TRF**: SPI Transmit/Receive Complete flag

0: Data is being transferred

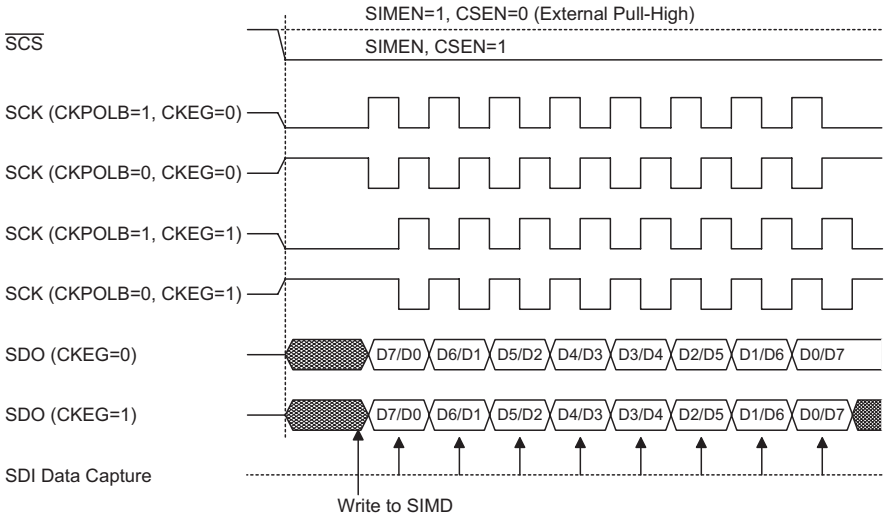
1: SPI data transmission is completed

The TRF bit is the Transmit/Receive Complete flag and is set "1" automatically when an SPI data transmission is completed, but must set to "0" by the application program. It can be used to generate an interrupt.

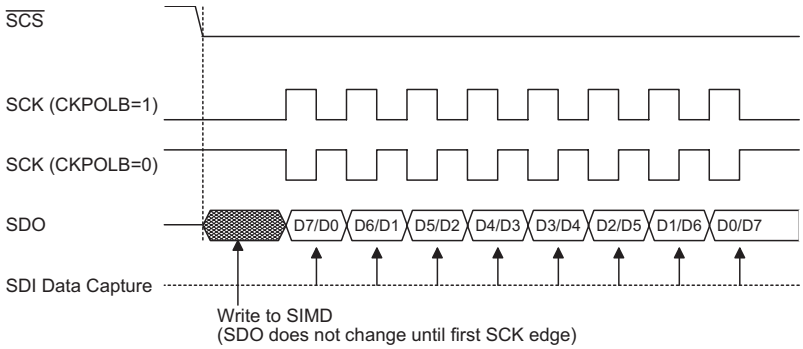
### SPI Communication

After the SPI interface is enabled by setting the SIMEN bit high, then in the Master Mode, when data is written to the SIMD register, transmission/reception will begin simultaneously. When the data transfer is complete, the TRF flag will be set automatically, but must be cleared using the application program. In the Slave Mode, when the clock signal from the master has been received, any data in the SIMD register will be transmitted and any data on the SDI pin will be shifted into the SIMD register. The master should output an $\overline{SCS}$ signal to enable the slave device before a clock signal is provided. The slave data to be transferred should be well prepared at the appropriate moment relative to the $\overline{SCS}$ signal depending upon the configurations of the CKPOLB bit and CKEG bit. The accompanying timing diagram shows the relationship between the slave data and $\overline{SCS}$ signal for various configurations of the CKPOLB and CKEG bits.

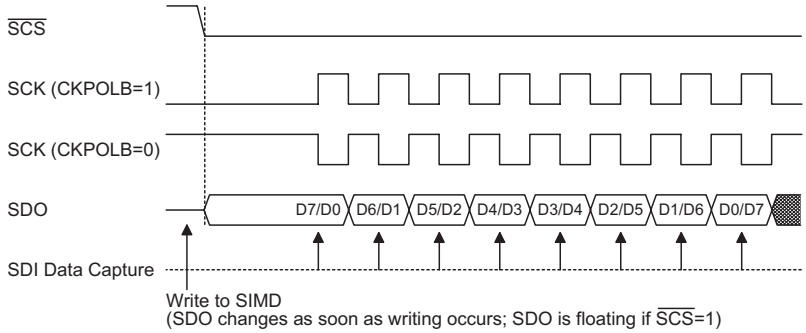
The SPI will continue to function even in the IDLE Mode.




**SPI Master Mode Timing**

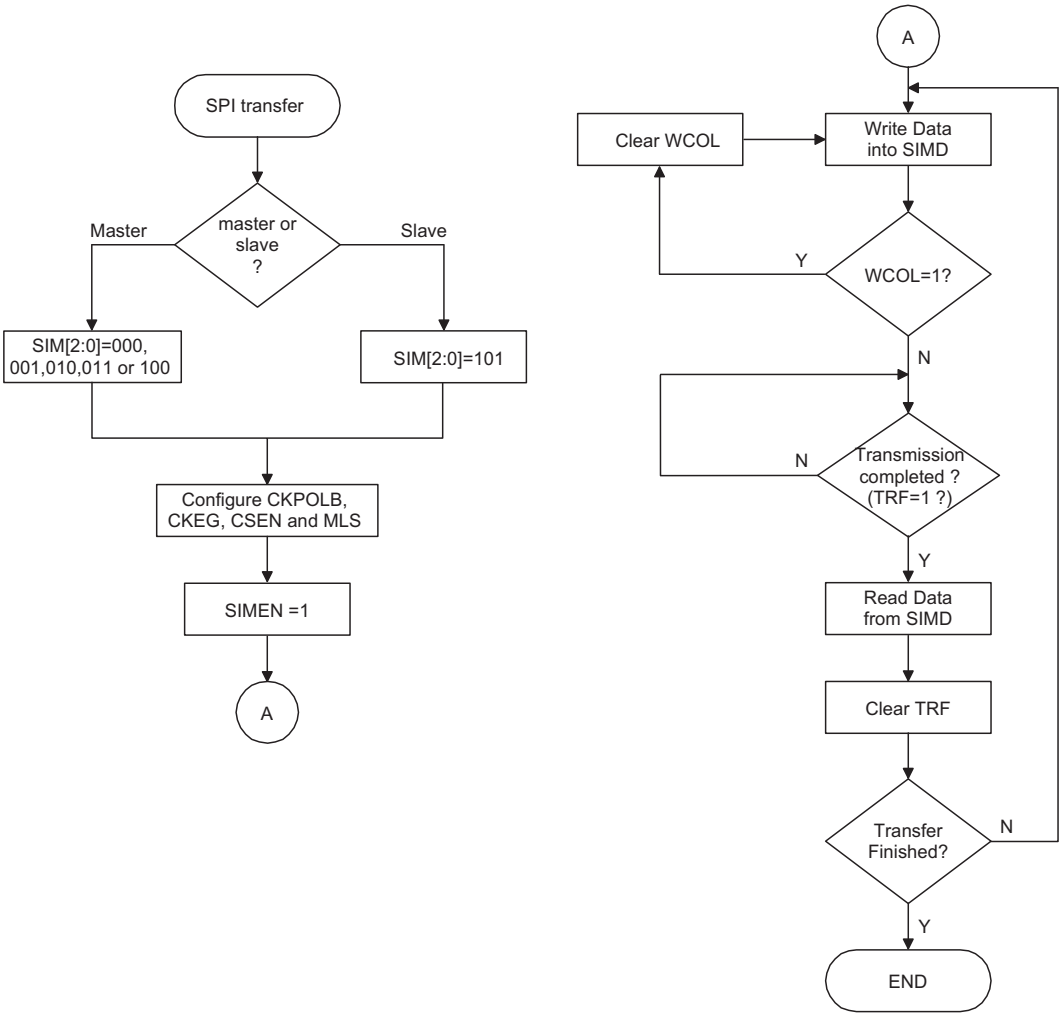



**SPI Slave Mode Timing – CKEG=0**




Note: For SPI slave mode, if SIMEN=1 and CSEN=0, SPI is always enabled and ignores the SCS level.

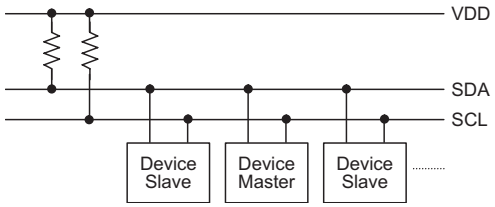
**SPI Slave Mode Timing – CKEG=1**




**SPI Transfer Control Flowchart**

## I²C Interface

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




**I²C Master/Slave Bus Connection**

### I²C Interface Operation

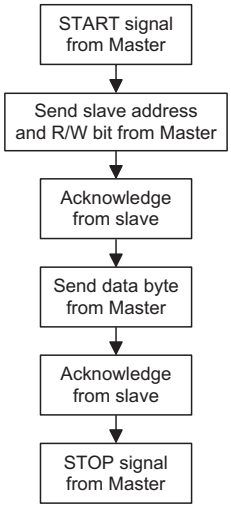
The I²C serial interface is a two line interface, a serial data line, SDA, and serial clock line, SCL. As many devices may be connected together on the same bus, their outputs are both open drain types. For this reason it is necessary that external pull-high resistors are connected to these outputs.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

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

The debounce time of the I²C interface uses the system clock to in effect add a debounce time to the external clock to reduce the possibility of glitches on the clock line causing erroneous operation. The debounce time, is 2 system clocks. To achieve the required I²C data transfer speed, there exists a relationship between the system clock, $f_{SYS}$, and the I²C debounce time. For either the I²C Standard or Fast mode operation, users must take care of the selected system clock frequency and the configured debounce time to match the criterion shown in the following table.

| I²C Debounce Time Selection | I²C Standard Mode (100kHz) | I²C Fast Mode (400kHz) |
|---|---|---|
| 2 system clock debounce | $f_{SYS}$ > 4MHz | $f_{SYS}$ > 10MHz |

**I²C Minimum $f_{SYS}$ Frequency**



### I²C Registers

There are four control registers associated with the I²C bus, SIMC0, SIMC1, SIMA and I2CTOC and one data register, SIMD. The SIMD register, which is shown in the above SPI section, is used to store the data being transmitted and received on the I²C bus. Before the microcontroller writes data to the I²C bus, the actual data to be transmitted must be placed in the SIMD register. After the data is received from the I²C bus, the microcontroller can read it from the SIMD register. Any transmission or reception of data from the I²C bus must be made via the SIMD register. The SIM pins are pin shared with other I/O pins and must be selected using the SIMEN bit in the SIMC0 register.

Note that the SIMA register also has the name SIMC2 which is used by the SPI function. Bit SIMEN and bits SIM2~SIM0 in register SIMC0 are used by the I²C interface.

| Register Name | Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SIMC0 | SIM2 | SIM1 | SIM0 | — | — | — | SIMEN | — |
| SIMC1 | HCF | HAAS | HBB | HTX | TXAK | SRW | RNIC | RXAK |
| SIMD | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| SIMA | A6 | A5 | A4 | A3 | A2 | A1 | A0 | — |
| I2CTOC | I2CTOEN | I2CTOF | I2CTOS5 | I2CTOS4 | I2CTOS3 | I2CTOS2 | I2CTOS1 | I2CTOS0 |

**I²C Registers List**

- **SIMC0 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | SIM2 | SIM1 | SIM0 | — | — | — | SIMEN | — |
| R/W | R/W | R/W | R/W | — | — | — | R/W | — |
| POR | 1 | 1 | 1 | — | — | — | 0 | — |

Bit 7 ~ 5 **SIM2~SIM0**: SIM Operating Mode Control

000: SPI master mode; SPI clock is $f_{SYS}/4$

001: SPI master mode; SPI clock is $f_{SYS}/16$

010: SPI master mode; SPI clock is $f_{SYS}/64$

011: SPI master mode; SPI clock is $f_{SUB}$

100: SPI master mode; SPI clock is TIMER Overflow frequency/2

101: SPI slave mode

110: I²C mode

111: Reserved

These bits setup the overall operating mode of the SIM function. As well as selecting if the I²C or SPI function, they are used to control the SPI Master/Slave selection and the SPI Master clock frequency. The SPI clock is a function of the system clock but can also be chosen to be sourced from the Timer/Event counter. If the SPI Slave Mode is selected then the clock will be supplied by an external Master device.

Bit 4~2 unimplemented, read as “0”

Bit 1 **SIMEN**: SIM Control
0: Disable
1: Enable
The bit is the overall on/off control for the SIM interface. When the SIMEN bit is cleared to zero to disable the SIM interface, the SDI, SDO, SCK and $\overline{SCS}$, or SDA and SCL lines will be as I/O function and the SIM operating current will be reduced to a minimum value. If the SIM is configured to operate as an SPI interface via the SIM2~SIM0 bits, the contents of the SPI control registers will remain at the previous settings when the SIMEN bit changes from low to high and should therefore be first initialised by the application program. If the SIM is configured to operate as an $I^2C$ interface via the SIM2~SIM0 bits and the SIMEN bit changes from low to high, the contents of the $I^2C$ control bits such as HTX and TXAK will remain at the previous settings and should therefore be first initialised by the application program while the relevant $I^2C$ flags such as HCF, HAAS, HBB, SRW and RXAK will be set to their default states.

Bit 0 Unimplemented, read as "0"

- **SIMC1 Register**

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

Bit 7 **HCF**: $I^2C$ Bus data transfer completion flag
0: Data is being transferred
1: Completion of an 8-bit data transfer
The HCF flag is the data transfer flag. This flag will be zero when data is being transferred. Upon completion of an 8-bit data transfer the flag will go high and an interrupt will be generated.

Bit 6 **HAAS**: $I^2C$ Bus address match flag
0: Not address match
1: Address match
The HASS flag is the address match flag. This flag is used to determine if the slave device address is the same as the master transmit address. If the addresses match then this bit will be high, if there is no match then the flag will be low.

Bit 5 **HBB**: $I^2C$ Bus busy flag
0: $I^2C$ Bus is not busy
1: $I^2C$ Bus is busy
The HBB flag is the $I^2C$ busy flag. This flag will be "1" when the $I^2C$ bus is busy which will occur when a START signal is detected. The flag will be set to "0" when the bus is free which will occur when a STOP signal is detected.

Bit 4 **HTX**: Select $I^2C$ slave device is transmitter or receiver
0: Slave device is the receiver
1: Slave device is the transmitter

Bit 3 **TXAK**: $I^2C$ Bus transmit acknowledge flag
0: Slave send acknowledge flag
1: Slave do not send acknowledge flag
The TXAK bit is the transmit acknowledge flag. After the slave device receipt of 8-bits of data, this bit will be transmitted to the bus on the 9th clock from the slave device. The slave device must always set TXAK bit to "0" before further data is received.

Bit 2 **SRW**: I²C Slave Read/Write flag

0: Slave device should be in receive mode

1: Slave device should be in transmit mode

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

Bit 1 **RNIC**: I²C running using Internal Clock Control

0: I²C running using internal clock

1: I²C running not using Internal Clock

The I²C module can run without using internal clock, and generate an interrupt if the SIM interrupt is enabled, which can be used in SLEEP Mode, IDLE Mode, NORMAL(SLOW) Mode. If this bit is set to "1" and MCU is in "HALT", slave-receiver can work well but slave-transmitter doesn't work since it needs system clock .

Bit 0 **RXAK**: I²C Bus Receive acknowledge flag

0: Slave receive acknowledge flag

1: Slave do not receive acknowledge flag

The RXAK flag is the receiver acknowledge flag. When the RXAK flag is "0", it means that a acknowledge signal has been received at the 9th clock, after 8 bits of data have been transmitted. When the slave device in the transmit mode, the slave device checks the RXAK flag to determine if the master receiver wishes to receive the next byte. The slave transmitter will therefore continue sending out data until the RXAK flag is "1". When this occurs, the slave transmitter will release the SDA line to allow the master to send a STOP signal to release the I²C Bus.

**• I2CTOC Register**

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

Bit 7 **I2CTOEN**: I2C Time-out Control

0: disable

1: enable

Bit 6 **I2CTOF**: Time-out flag

0: no time-out

1: time-out occurred

Bit 5~0 **I2CTOS5~I2CTOS0**: Time-out Definition

I²C time-out clock source is $f_{SUB}/32$.

I²C time-out time is given by: ([I2CTOS5 : I2CTOS0]+1) x $(32/f_{SUB})$

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

• **SIMD Register**

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

"x" unknown

• **SIMA Register**

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

"x" unknown

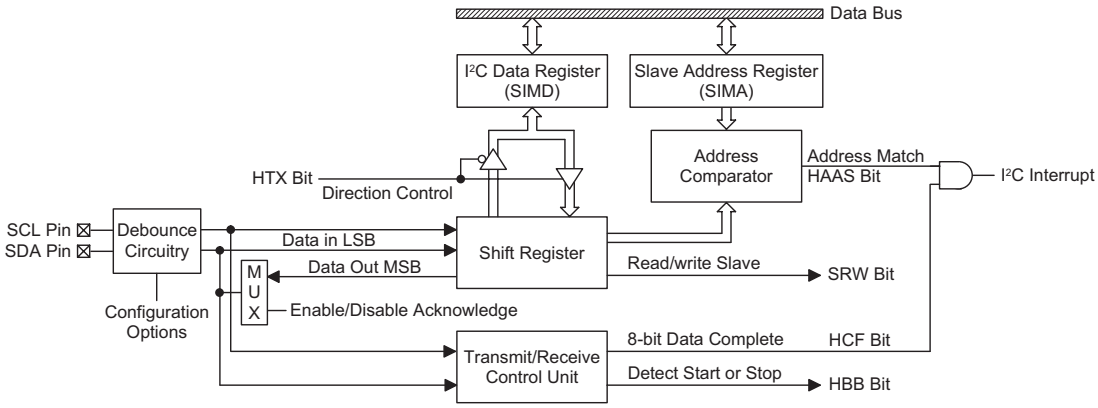
Bit 7~1 **A6~A0**: I²C slave address

A6~ A0 is the I²C slave address bit 6 ~ bit 0.

The SIMA register is also used by the SPI interface but has the name SIMC2. The SIMA register is the location where the 7-bit slave address of the slave device is stored. Bits 7~ 1 of the SIMA register define the device slave address. Bit 0 is not defined.

When a master device, which is connected to the I²C bus, sends out an address, which matches the slave address in the SIMA register, the slave device will be selected. Note that the SIMA register is the same register address as SIMC2 which is used by the SPI interface.

Bit 0 Unimplemented, read as "0"

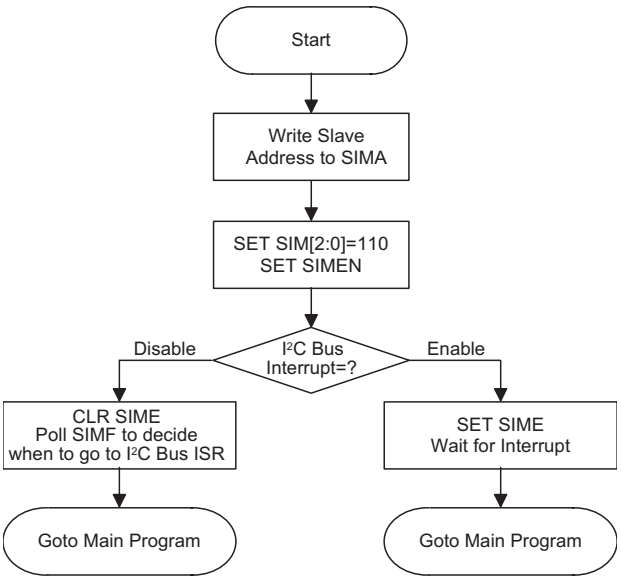



**I²C Block Diagram**

### I²C Bus Communication

Communication on the I²C bus requires four separate steps, a START signal, a slave device address transmission, a data transmission and finally a STOP signal. When a START signal is placed on the I²C bus, all devices on the bus will receive this signal and be notified of the imminent arrival of data on the bus. The first seven bits of the data will be the slave address with the first bit being the MSB. If the address of the slave device matches that of the transmitted address, the HAAS bit in the SIMC1 register will be set and an I²C interrupt will be generated. After entering the interrupt service routine, the slave device must first check the condition of the HAAS bit to determine whether the interrupt source originates from an address match or from the completion of an 8-bit data transfer. During a data transfer, note that after the 7-bit slave address has been transmitted, the following bit, which is the 8th bit, is the read/write bit whose value will be placed in the SRW bit. This bit will be checked by the slave device to determine whether to go into transmit or receive mode. Before any transfer of data to or from the I²C bus, the microcontroller must initialise the bus, the following are steps to achieve this:

- Step 1
  Set the SIM2~SIM0 and SIMEN bits in the SIMC0 register to "1" to enable the I²C bus.
- Step 2
  Write the slave address of the device to the I²C bus address register SIMA.
- Step 3
  Set the SIME interrupt enable bit of the interrupt control register to enable the SIM interrupt .




**I²C Bus Initialisation Flow Chart**

### I²C Bus Start Signal

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

### Slave Address

The transmission of a START signal by the master will be detected by all devices on the I²C bus. To determine which slave device the master wishes to communicate with, the address of the slave device will be sent out immediately following the START signal. All slave devices, after receiving this 7-bit address data, will compare it with their own 7-bit slave address. If the address sent out by the master matches the internal address of the microcontroller slave device, then an internal I²C bus interrupt signal will be generated. The next bit following the address, which is the 8th bit, defines the read/write status and will be saved to the SRW bit of the SIMC1 register. The slave device will then transmit an acknowledge bit, which is a low level, as the 9th bit. The slave device will also set the status flag HAAS when the addresses match.

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

### I²C Bus Read/Write Signal

The SRW bit in the SIMC1 register defines whether the slave device wishes to read data from the I²C bus or write data to the I²C bus. The slave device should examine this bit to determine if it is to be a transmitter or a receiver. If the SRW flag is "1" then this indicates that the master device wishes to read data from the I²C bus, therefore the slave device must be setup to send data to the I²C bus as a transmitter. If the SRW flag is "0" then this indicates that the master wishes to send data to the I²C bus, therefore the slave device must be setup to read data from the I²C bus as a receiver.

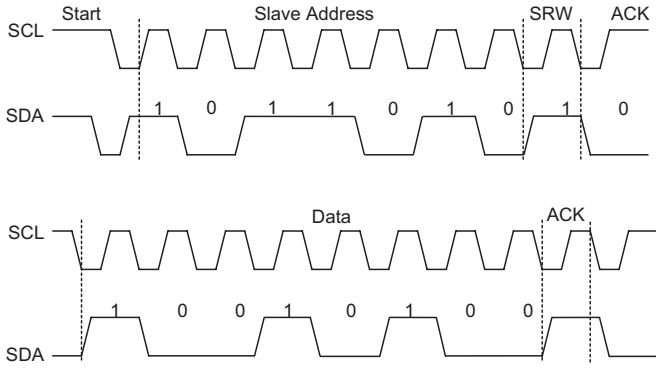
### I²C Bus Slave Address Acknowledge Signal

After the master has transmitted a calling address, any slave device on the I²C bus, whose own internal address matches the calling address, must generate an acknowledge signal. The acknowledge signal will inform the master that a slave device has accepted its calling address. If no acknowledge signal is received by the master then a STOP signal must be transmitted by the master to end the communication. When the HAAS flag is high, the addresses have matched and the slave device must check the SRW flag to determine if it is to be a transmitter or a receiver. If the SRW flag is high, the slave device should be setup to be a transmitter so the HTX bit in the SIMC1 register should be set to "1". If the SRW flag is low, then the microcontroller slave device should be setup as a receiver and the HTX bit in the SIMC1 register should be set to "0".

### I²C Bus Data and Acknowledge Signal

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

When the slave receiver receives the data byte, it must generate an acknowledge bit, known as TXAK, on the 9th clock. The slave device, which is setup as a transmitter will check the RXAK bit in the SIMC1 register to determine if it is to send another data byte, if not then it will release the SDA line and await the receipt of a STOP signal from the master.

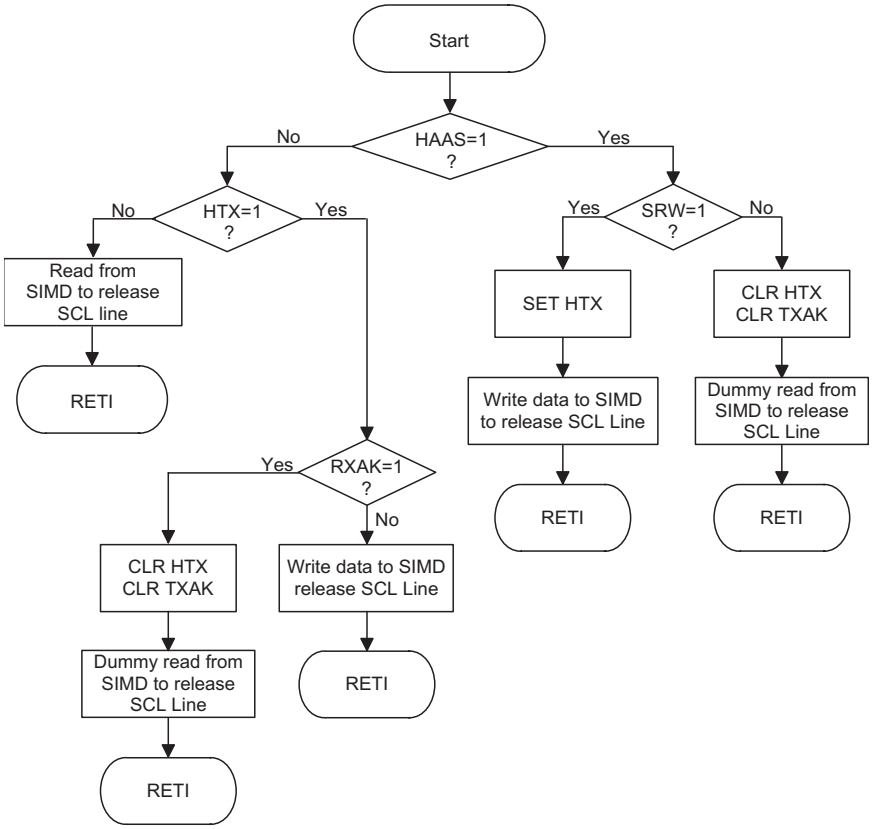



S=Start (1 bit)
SA=Slave Address (7 bits)
SR=SRW bit (1 bit)
M=Slave device send acknowledge bit (1 bit)
D=Data (8 bits)
A=ACK (RXAK bit for transmitter, TXAK bit for receiver 1 bit)
P=Stop (1 bit)

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

**I²C Communication Timing Diagram**

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




**I²C Bus ISR Flow Chart**

### I²C Time-out Control

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

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

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

| Register | After I²C Time-out |
|---|---|
| SIMD, SIMA, SIMC0 | No change |
| SIMC1 | Reset to POR condition |

**I²C Registers After Time-out**

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

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

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

# Interrupts

Interrupts are an important part of any microcontroller system. When an external event or an internal function such as a Touch Action or Timer/Event Counter overflow 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 devices contain several external interrupt and internal interrupts functions. The external interrupt is generated by the action of the external INT pin, while the internal interrupts are generated by various internal functions such as the Touch Keys, Timer/Event Counter, Time Base, SIM etc.

## Interrupt Registers

Overall interrupt control, which basically means the setting of request flags when certain microcontroller conditions occur and the setting of interrupt enable bits by the application program, is controlled by a series of registers, located in the Special Purpose Data Memory, as shown in the accompanying table. The number of registers depends upon the device chosen but fall into three categories. The first is the INTC0~INTC1 registers which setup the primary interrupts, the second is the INTEG registers to setup the external interrupt trigger edge type.

Each register contains a number of enable bits to enable or disable individual registers as well as interrupt flags to indicate the presence of an interrupt request. The naming convention of these follows a specific pattern. First is listed an abbreviated interrupt type, then the (optional) number of that interrupt followed by either an "E" for enable/disable bit or "F" for request flag.

| Function | Enable Bit | Request Flag | Notes |
|---|---|---|---|
| Global | EMI | — | — |
| INT Pin | INTE | INTF | — |
| Touch Key Module | TKME | TKMF | — |
| Timer/Event Counter | TE | TF | — |
| SIM | SIME | SIMF | — |
| Time Base | TBE | TBF | — |
| EEPROM | DEE | DEF | — |
| A/D Converter interrupt | ADE | ADF | — |

**Interrupt Register Bit Naming Conventions**

### Interrupt Register Contents

| Name | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |
|---|---|---|---|---|---|---|---|---|
| INTEG | — | — | — | — | — | — | INTS1 | INTS0 |
| INTC0 | — | TF | TKMF | INTF | TE | TKME | INTE | EMI |
| INTC1 | ADF | DEF | TBF | SIMF | ADE | DEE | TBE | SIME |

### INTEG Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | — | — | — | INTS1 | INTS0 |
| R/W | — | — | — | — | — | — | R/W | R/W |
| POR | — | — | — | — | — | — | 0 | 0 |

Bit 7 ~ 2 Unimplemented, read as "0"

Bit 1 ~ 0 **INTS1, INTS0**: Defines INT interrupt active edge

00: Disabled interrupt

01: Rising Edge interrupt

10: Falling Edge interrupt

11: Dual Edge interrupt

### INTC0 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | TF | TKMF | INTF | TE | TKME | INTE | EMI |
| R/W | — | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | — | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7 Unimplemented, read as "0"
Bit 6 **TF**: Timer/Event Counter interrupt request flag
0: No request
1: Interrupt request
Bit 5 **TKMF**: Touch key module interrupt request flag
0: No request
1: Interrupt request
Bit 4 **INTF**: INT pin interrupt request flag
0: No request
1: Interrupt request
Bit 3 **TE**: Timer/Event Counter interrupt control
0: Disable
1: Enable
Bit 2 **TKME**: Touch key module interrupt control
0: Disable
1: Enable
Bit 1 **INTE**: INT pin 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 | ADF | DEF | TBF | SIMF | ADE | DEE | TBE | SIME |
| 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 **ADF**: A/D Converter interrupt request flag
0: No request
1: Interrupt request
Bit 6 **DEF**: Data EEPROM interrupt request flag
0: No request
1: Interrupt request
Bit 5 **TBF**: Time Base interrupt request flag
0: No request
1: Interrupt request
Bit 4 **SIMF**:SIM interrupt request flag
0: No request
1: Interrupt request
Bit 3 **ADE**: A/D Converter Interrupt control
0: Disable
1: Enable
Bit 2 **DEE**: Data EEPROM control
0: Disable
1: Enable
Bit 1 **TBE**: Time Base interrupt control
0: Disable
1: Enable
Bit 0 **SIME**: SIM interrupt control
0: Disable
1: Enable

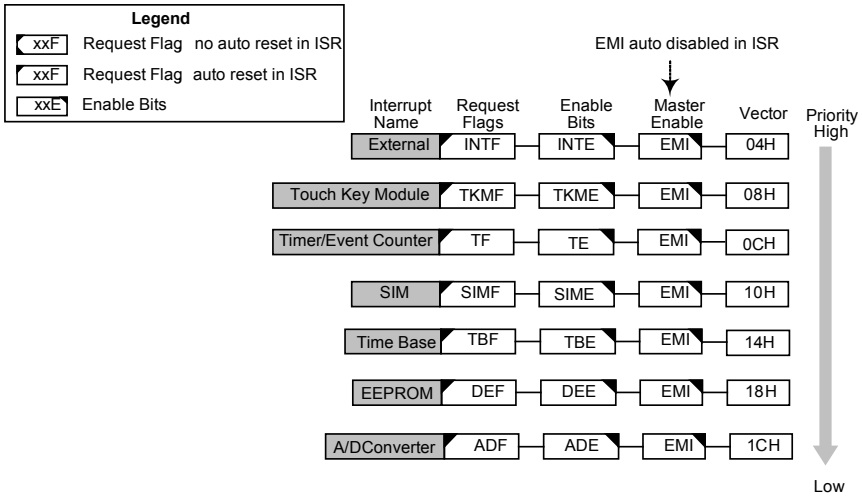
## Interrupt Operation

When the conditions for an interrupt event occur, such as a Touch Key Counter overflow, Timer/ Event Counter overflow, etc, the relevant interrupt request flag will be set. Whether the request flag actually generates a program jump to the relevant interrupt vector is determined by the condition of the interrupt enable bit. If the enable bit is set high then the program will jump to its relevant vector; if the enable bit is zero then although the interrupt request flag is set an actual interrupt will not be generated and the program will not jump to the relevant interrupt vector. The global interrupt enable bit, if cleared to zero, will disable all interrupts.

When an interrupt is generated, 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" 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", 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 diagrams with their order of priority. Some interrupt sources have their own individual vector while others share the same multi-function interrupt vector. Once an interrupt subroutine is serviced, all the other interrupts will be blocked, as the global interrupt enable bit, 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. In case of simultaneous requests, the accompanying diagram shows the priority that is applied. All of the interrupt request flags when set will wake-up the device if it is in SLEEP or IDLE Mode, however to prevent a wake-up from occurring the corresponding flag should be set before the device is in SLEEP or IDLE Mode.




**Interrupt Structure**

### External Interrupt

The external interrupt is controlled by signal transitions on the pin INT. An external interrupt request will take place when the external interrupt request flag, INTF, is set, which will occur when a transition, whose type is chosen by the edge select bits, appears on the external interrupt pin. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and respective external interrupt enable bit, INTE, must first be set. Additionally the correct interrupt edge type must be selected using the INTEG register to enable the external interrupt function and to choose the trigger edge type. As the external interrupt pin is pin-shared with I/O pin, its can only be configured as external interrupt pin if its external interrupt enable bit in the corresponding interrupt register has been set. The pin must also be setup as an input by setting the corresponding bit 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, will take place. When the interrupt is serviced, the external interrupt request flag, INTF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts. Note that any pull-high resistor selections on the external interrupt pin will remain valid even if the pin is used as an external interrupt input.

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

### Time Base Interrupt

The function of the Time Base Interrupt is to provide regular time signal in the form of an internal interrupt. It is controlled by the overflow signal from its timer function. When this happens its interrupt request flags TBF will be set. To allow the program to branch to its interrupt vector address, the global interrupt enable bit, EMI and Time Base enable bit, TBE, must first be set. When the interrupt is enabled, the stack is not full and the Time Base overflows, a subroutine call to its vector location will take place. When the interrupt is serviced, the interrupt request flag, TBF, will be automatically reset and the EMI bit will be cleared to disable other interrupts.

The purpose of the Time Base Interrupt is to provide an interrupt signal at fixed time periods. Its clock source originate from the internal clock source $f_{SYS}$ or $f_{SUB}$. This $f_{TP}$ input clock passes through a divider, the division ratio of which is selected by programming the appropriate bits in the TBC register to obtain longer interrupt periods whose value ranges. The clock source that generates $f_{TP}$, which in turn controls the Time Base interrupt period, can originate from several different sources, as shown in the System Operating Mode section.

**TBC Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | TB1 | TB0 | — | — | — | — |
| R/W | — | — | R/W | R/W | — | — | — | — |
| POR | — | — | 0 | 0 | — | — | — | — |

Bit 7~6 Unimplemented, read as "0"

Bit 5~4 **TB1 ~ TB0**: Select Time Base Time-out Period

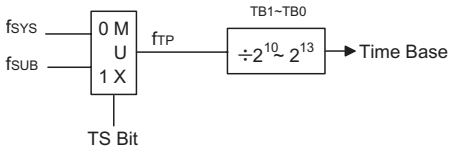
00: $1024/f_{TP}$

01: $2048/f_{TP}$

10: $4096/f_{TP}$

11: $8192/f_{TP}$

Bit 3~ 0 Unimplemented, read as "0"




**Time Base Structure**

## Timer/Event Counter Interrupt

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

## EEPROM Interrupt

An EEPROM Interrupt request will take place when the EEPROM Interrupt request flag, DEF, is set, which occurs when an EEPROM Write cycle ends. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and EEPROM Interrupt enable bit, DEE, must first be set. When the interrupt is enabled, the stack is not full and an EEPROM Write cycle ends, a subroutine call to the respective EEPROM Interrupt vector, will take place. When the EEPROM Interrupt is serviced, the DEF flag will be automatically cleared and the EMI bit will be automatically cleared to disable other interrupts.

### Touch Key Interrupt

For a Touch Key interrupt to occur, the global interrupt enable bit, EMI, and the corresponding Touch Key interrupt enable TKME must be first set. An actual Touch Key interrupt will take place when the Touch Key request flag. TKMF, is set, a situation that will occur when the time slot counter overflows. When the interrupt is enabled, the stack is not full and the Touch Key time slot counter overflow occurs, a subroutine call to the relevant timer interrupt vector, will take place. When the interrupt is serviced, the Touch Key interrupt request flag, TKMF, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

The TKCFOV flag, which is the 16-bit C/F counter overflow flag will go high when any of the Touch Key Module 16-bit C/F counter overflows. As this flag will not be automatically cleared, it has to be cleared by the application program.

Module 0 only contains one 16-bit counter. The TK16OV flag, which is the 16-bit counter overflow flag will go high when the 16-bit counter overflows. As this flag will not be automatically cleared, it has to be cleared by the application program.

### SIM Interrupt

A SIM Interrupt request will take place when the SIM Interrupt request flag, SIMF, is set, which occurs when a byte of data has been received or transmitted by the SIM interface. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and the Serial Interface Interrupt enable bit, SIME, must first be set. When the interrupt is enabled, the stack is not full and a byte of data has been transmitted or received by the SIM interface, a subroutine call to the respective interrupt vector, will take place. When the Serial Interface Interrupt is serviced, the SIM interrupt request flag, SIMF, will be automatically cleared and the EMI bit will be automatically cleared to disable other interrupts.

### A/D Converter Interrupt

The A/D Converter Interrupt is controlled by the termination of an A/D conversion process. An A/D Converter Interrupt request will take place when the A/D Converter Interrupt request flag, ADF, is set, which occurs when the A/D conversion process finishes. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and A/D Interrupt enable bit, ADE, must first be set. When the interrupt is enabled, the stack is not full and the A/D conversion process has ended, a subroutine call to the A/D Converter Interrupt vector, will take place. When the interrupt is serviced, the A/D Converter Interrupt flag, ADF, will be automatically cleared. The EMI bit will also be automatically cleared to disable other interrupts.

### Interrupt Wake-up Function

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

### Programming Considerations

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

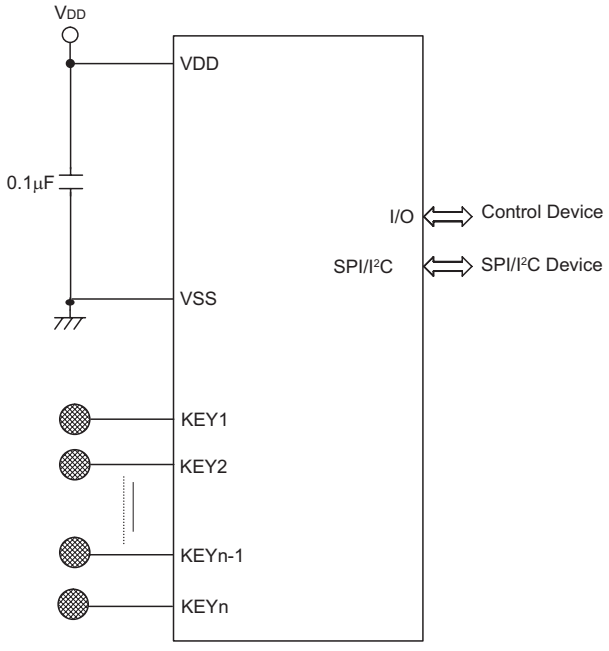
It is recommended that programs do not use the "CALL" instruction within the interrupt service subroutine. Interrupts often occur in an unpredictable manner or need to be serviced immediately. If only one stack is left and the interrupt is not well controlled, the original control sequence will be damaged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

Every interrupt has the capability of waking up the microcontroller when it is in SLEEP or IDLE Mode, the wake up being generated when the interrupt request flag changes from low to high. If it is required to prevent a certain interrupt from waking up the microcontroller then its respective request flag should be first set high before enter SLEEP or IDLE Mode.

As only the Program Counter is pushed onto the stack, then when the interrupt is serviced, if the contents of the accumulator, status register or other registers are altered by the interrupt service program, their contents should be saved to the memory at the beginning of the interrupt service routine.

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

## Application Circuits



# Instruction Set

## Introduction

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

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

## Instruction Timing

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

## Moving and Transferring Data

The transfer of data within the microcontroller program is one of the most frequently used operations. Making use of 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 Operation** | | | |
| TABRD [m] | Read table (specific page or current page) to TBLH and Data Memory | 2Note | None |
| TABRDL [m] | Read table (last page) to TBLH and Data Memory | 2Note | None |
| **Miscellaneous** | | | |
| NOP | No operation | 1 | None |
| CLR [m] | Clear Data Memory | 1Note | None |
| SET [m] | Set Data Memory | 1Note | None |
| CLR WDT | Clear Watchdog Timer | 1 | TO, PDF |
| CLR WDT1 | Pre-clear Watchdog Timer | 1 | TO, PDF |
| CLR WDT2 | Pre-clear Watchdog Timer | 1 | TO, PDF |
| SWAP [m] | Swap nibbles of Data Memory | 1Note | None |
| SWAPA [m] | Swap nibbles of Data Memory with result in ACC | 1 | None |
| HALT | Enter power down mode | 1 | TO, PDF |

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

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

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

## Instruction Definition

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

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

Operation ACC ← ACC + [m] + C

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

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

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

Operation [m] ← ACC + [m] + C

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

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

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

Operation ACC ← ACC + [m]

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

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

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

Operation ACC ← ACC + x

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

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

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

Operation [m] ← ACC + [m]

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

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

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

Operation ACC ← ACC "AND" [m]

Affected flag(s) Z

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

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

Operation ACC ← ACC "AND" x

Affected flag(s) Z

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

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

Operation [m] ← ACC "AND" [m]

Affected flag(s) Z

**CALL addr** Subroutine call

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

Operation: Stack ← Program Counter + 1
Program Counter ← addr

Affected flag(s): None

**CLR [m]** Clear Data Memory

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

Operation: [m] ← 00H

Affected flag(s): None

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

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

Operation: [m].i ← 0

Affected flag(s): None

**CLR WDT** Clear Watchdog Timer

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

Operation: WDT cleared
TO ← 0
PDF ← 0

Affected flag(s): TO, PDF

**CLR WDT1** Pre-clear Watchdog Timer

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

Operation: WDT cleared
TO ← 0
PDF ← 0

Affected flag(s): TO, PDF

**CLR WDT2** Pre-clear Watchdog Timer

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

Operation: WDT cleared
TO ← 0
PDF ← 0

Affected flag(s): TO, PDF

**CPL [m]** Complement Data Memory

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

Operation: [m] ← $\overline{[m]}$

Affected flag(s): Z

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

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

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

Affected flag(s) Z

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

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

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

Affected flag(s) C

**DEC [m]** Decrement Data Memory

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

Operation [m] ← [m] − 1

Affected flag(s) Z

DECA [m] Decrement Data Memory with result in ACC

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

Operation ACC ← [m] − 1

Affected flag(s) Z

**HALT** Enter power down mode

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

Operation TO ← 0
PDF ← 1

Affected flag(s) TO, PDF

**INC [m]** Increment Data Memory

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

Operation [m] ← [m] + 1

Affected flag(s) Z

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

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

Operation ACC ← [m] + 1

Affected flag(s) Z

**JMP addr** Jump unconditionally

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

Operation Program Counter ← addr

Affected flag(s) None

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

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

Operation ACC ← [m]

Affected flag(s) None

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

Description The immediate data specified is loaded into the Accumulator.

Operation ACC ← x

Affected flag(s) None

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

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

Operation [m] ← ACC

Affected flag(s) None

**NOP** No operation

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

Operation No operation

Affected flag(s) None

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

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

Operation ACC ← ACC ″OR″ [m]

Affected flag(s) Z

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

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

Operation ACC ← ACC ″OR″ x

Affected flag(s) Z

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

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

Operation [m] ← ACC ″OR″ [m]

Affected flag(s) Z

**RET** Return from subroutine

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

Operation Program Counter ← Stack

Affected flag(s) None

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

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

Operation Program Counter ← Stack
ACC ← x

Affected flag(s) None

**RETI** Return from interrupt

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

Operation Program Counter ← Stack
EMI ← 1

Affected flag(s) None

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

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

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

Affected flag(s) None

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

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

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

Affected flag(s) None

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

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

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

Affected flag(s) C

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

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

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

Affected flag(s) C

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

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

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

Affected flag(s) None

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

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

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

Affected flag(s) None

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

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

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

Affected flag(s) C

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

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

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

Affected flag(s) C

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

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

Operation ACC ← ACC − [m] − C

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

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

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

Operation [m] ← ACC − [m] − C

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

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

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

Operation [m] ← [m] − 1
Skip if [m]=0

Affected flag(s) None

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

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

Operation ACC ← [m] − 1
Skip if ACC=0

Affected flag(s) None

**SET [m]** Set Data Memory

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

Operation [m] ← FFH

Affected flag(s) None

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

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

Operation [m].i ← 1

Affected flag(s) None

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

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

Operation [m] ← [m] + 1
Skip if [m]=0

Affected flag(s) None

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

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

Operation ACC ← [m] + 1
Skip if ACC=0

Affected flag(s) None

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

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

Operation Skip if [m].i ≠ 0

Affected flag(s) None

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

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

Operation ACC ← ACC − [m]

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

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

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

Operation [m] ← ACC − [m]

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

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

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

Operation ACC ← ACC − x

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

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

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

Operation [m].3~[m].0 ↔ [m].7~[m].4

Affected flag(s) None

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

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

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

Affected flag(s) None

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

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

Operation Skip if [m]=0

Affected flag(s) None

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

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

Operation ACC ← [m]
Skip if [m]=0

Affected flag(s) None

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

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

Operation Skip if [m].i=0

Affected flag(s) None

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

Description The low byte of the program code addressed by the table pointer (TBHP and TBLP or only TBLP if no 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

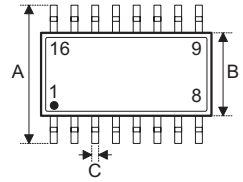
## Package Information

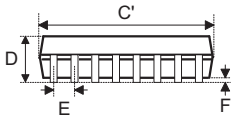
Note that the package information provided here is for consultation purposes only. As this information may be updated at regular intervals users are reminded to consult the Holtek website for the latest version of the Package/Carton Information.

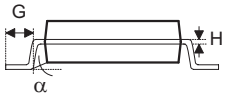
Additional supplementary information with regard to packaging is listed below. Click on the relevant section to be transferred to the relevant website page.

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

## 16-pin NSOP (150mil) Outline Dimensions



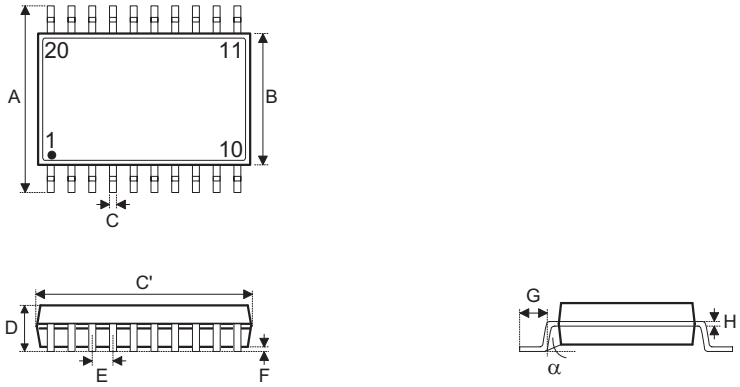




| Symbol | Dimensions in inch | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 0.236 BSC | — |
| B | — | 0.154 BSC | — |
| C | 0.012 | — | 0.020 |
| C' | — | 0.390 BSC | — |
| D | — | — | 0.069 |
| E | — | 0.050 BSC | — |
| F | 0.004 | — | 0.010 |
| G | 0.016 | — | 0.050 |
| H | 0.004 | — |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

| Symbol | Dimensions in mm | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 6.000 BSC | — |
| B | — | 3.900 BSC | — |
| C | 0.31 | — | 0.51 |
| C' | — | 9.900 BSC | — |
| D | — | — | 1.75 |
| E | — | 1.270 BSC | — |
| F | 0.10 | — | 0.25 |
| G | 0.40 | — | 1.27 |
| H | 0.10 | — | 0.25 |
| α | 0° | — | 8° |

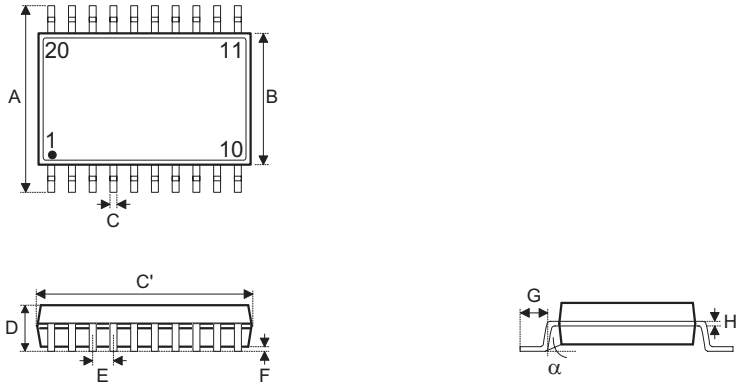
## 20-pin SOP (300mil) Outline Dimensions




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

| Symbol | Dimensions in mm | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 10.30 BSC | — |
| B | — | 7.50 BSC | — |
| C | 0.31 | — | 0.51 |
| C’ | — | 12.80 BSC | — |
| D | — | — | 2.65 |
| E | — | 1.27 BSC | — |
| F | 0.10 | — | 0.30 |
| G | 0.40 | — | 1.27 |
| H | 0.20 | — | 0.33 |
| α | 0° | — | 8° |

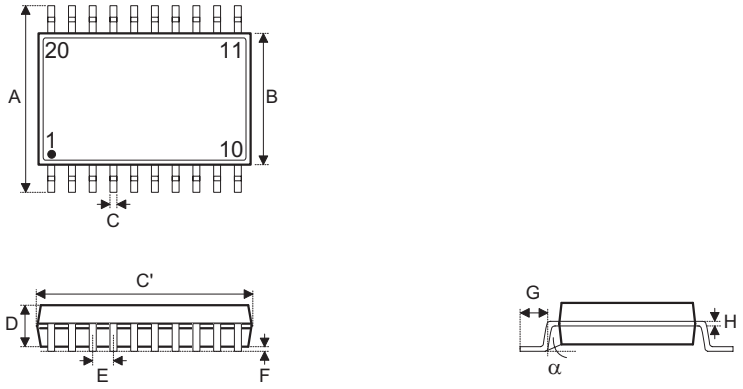
## 20-pin NSOP (150mil) Outline Dimensions




| Symbol | Dimensions in inch | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | 0.228 | 0.236 | 0.244 |
| B | 0.146 | 0.154 | 0.161 |
| C | 0.009 | — | 0.012 |
| C’ | 0.382 | 0.390 | 0.398 |
| D | — | — | 0.069 |
| E | — | 0.032 BSC | — |
| F | 0.002 | — | 0.009 |
| G | 0.020 | — | 0.031 |
| H | 0.008 | — | 0.010 |
| α | 0° | — | 8° |

| Symbol | Dimensions in mm | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | 5.80 | 6.00 | 6.20 |
| B | 3.70 | 3.90 | 4.10 |
| C | 0.23 | — | 0.30 |
| C’ | 9.70 | 9.90 | 10.10 |
| D | — | — | 1.75 |
| E | — | 0.80 BSC | — |
| F | 0.05 | — | 0.23 |
| G | 0.50 | — | 0.80 |
| H | 0.21 | — | 0.25 |
| α | 0° | — | 8° |

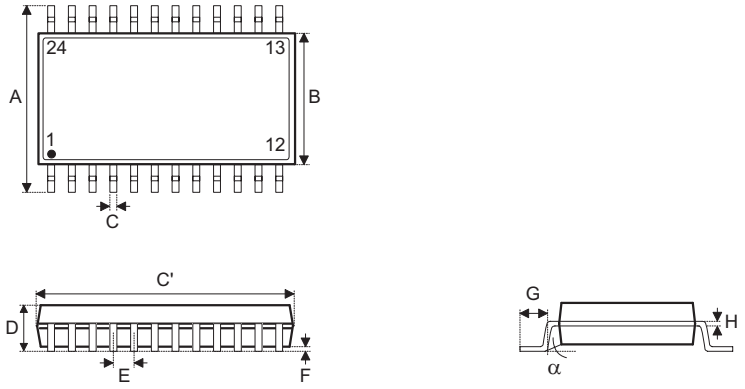
## 20-pin SSOP (150mil) Outline Dimensions




| Symbol | Dimensions in inch | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 0.236 BSC | — |
| B | — | 0.155 BSC | — |
| C | 0.008 | — | 0.012 |
| C’ | — | 0.341 BSC | — |
| D | — | — | 0.069 |
| E | — | 0.025 BSC | — |
| F | 0.004 | — | 0.0098 |
| G | 0.016 | — | 0.05 |
| H | 0.004 | — | 0.01 |
| α | 0° | — | 8° |

| Symbol | Dimensions in mm | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 6.000 BSC | — |
| B | — | 3.900 BSC | — |
| C | 0.20 | — | 0.30 |
| C’ | — | 8.660 BSC | — |
| D | — | — | 1.75 |
| E | — | 0.635 BSC | — |
| F | 0.10 | — | 0.25 |
| G | 0.41 | — | 1.27 |
| H | 0.10 | — | 0.25 |
| α | 0° | — | 8° |

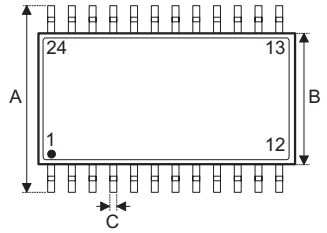
## 24-pin SOP (300mil) Outline Dimensions

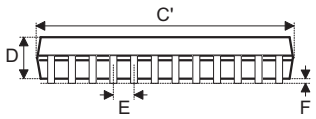


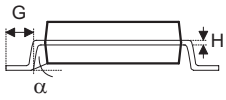

| Symbol | Dimensions in inch | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 0.406 BSC | — |
| B | — | 0.295 BSC | — |
| C | 0.012 | — | 0.020 |
| C’ | — | 0.606 BSC | — |
| D | — | — | 0.104 |
| E | — | 0.050 BSC | — |
| F | 0.004 | — | 0.012 |
| G | 0.016 | — | 0.050 |
| H | 0.008 | — |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

| Symbol | Dimensions in mm | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 10.30 BSC | — |
| B | — | 7.50 BSC | — |
| C | 0.31 | — | 0.51 |
| C’ | — | 15.40 BSC | — |
| D | — | — | 2.65 |
| E | — | 1.27 BSC | — |
| F | 0.10 | — | 0.30 |
| G | 0.40 | — | 1.27 |
| H | 0.20 | — | 0.33 |
| α | 0° | — | 8° |

## 24-pin SSOP(150mil) Outline Dimensions



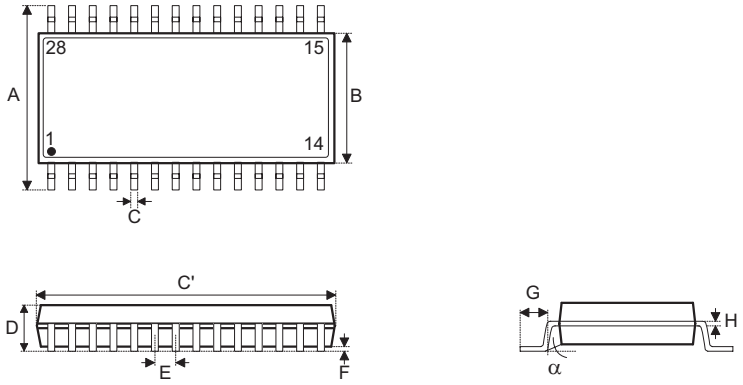




| Symbol | Dimensions in inch | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 0.236 BSC | — |
| B | — | 0.154 BSC | — |
| C | 0.008 | — | 0.012 |
| C’ | — | 0.341 BSC | — |
| D | — | — | 0.069 |
| E | — | 0.025 BSC | — |
| F | 0.004 | — | 0.010 |
| G | 0.016 | — | 0.050 |
| H | 0.004 | — | 0.010 |
| α | 0° | — | 8° |

| Symbol | Dimensions in mm | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 6.000 BSC | — |
| B | — | 3.900 BSC | — |
| C | 0.20 | — | 0.30 |
| C’ | — | 8.660 BSC | — |
| D | — | — | 1.75 |
| E | — | 0.635 BSC | — |
| F | 0.10 | — | 0.25 |
| G | 0.41 | — | 1.27 |
| H | 0.10 | — | 0.25 |
| α | 0° | — | 8° |

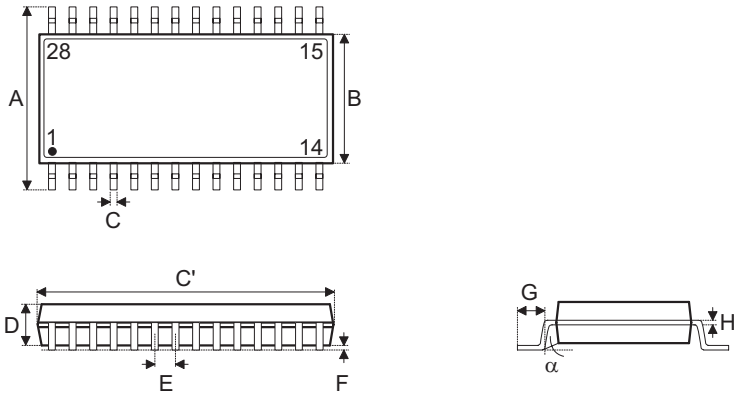
## 28-pin SOP (300mil) Outline Dimensions




| Symbol | Dimensions in inch | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 0.406 BSC | — |
| B | — | 0.295 BSC | — |
| C | 0.012 | — | 0.020 |
| C’ | — | 0.705 BSC | — |
| D | — | — | 0.104 |
| E | — | 0.050 BSC | — |
| F | 0.004 | — | 0.012 |
| G | 0.016 | — | 0.050 |
| H | 0.008 | — | 0.013 |
| α | 0° | — | 8° |

| Symbol | Dimensions in mm | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 10.30 BSC | — |
| B | — | 7.5 BSC | — |
| C | 0.31 | — | 0.51 |
| C’ | — | 17.9 BSC | — |
| D | — | — | 2.65 |
| E | — | 1.27 BSC | — |
| F | 0.10 | — | 0.30 |
| G | 0.40 | — | 1.27 |
| H | 0.20 | — | 0.33 |
| α | 0° | — | 8° |

## 28-pin SSOP (150mil) Outline Dimensions




| Symbol | Dimensions in inch | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 0.236 BSC | — |
| B | — | 0.154 BSC | — |
| C | 0.008 | — | 0.012 |
| C’ | — | 0.390 BSC | — |
| D | — | — | 0.069 |
| E | — | 0.025 BSC | — |
| F | 0.004 | — | 0.010 |
| G | 0.016 | — | 0.050 |
| H | 0.004 | — | 0.010 |
| α | 0° | — | 8° |

| Symbol | Dimensions in mm | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | — | 6.0 BSC | — |
| B | — | 3.9 BSC | — |
| C | 0.20 | — | 0.30 |
| C’ | — | 9.9 BSC | — |
| D | — | — | 1.75 |
| E | — | 0.635 BSC | — |
| F | 0.10 | — | 0.25 |
| G | 0.41 | — | 1.27 |
| H | 0.10 | — | 0.25 |
| α | 0° | — | 8° |