Xinqun

## High Accuracy HIRC OTP MCU

# BX68R2420

Revision: V1.01 Date: October 19, 2026

# Table of Contents

**Features** ..... **5**
CPU Features ..... 5
Peripheral Features ..... 5
**General Description** ..... **6**
**Block Diagram** ..... **6**
**Pin Assignment** ..... **7**
**Pin Description** ..... **8**
**Absolute Maximum Ratings** ..... **8**
**D.C. Characteristics** ..... **9**
Operating Voltage Characteristics ..... 9
Standby Current Characteristics ..... 9
Operating Current Characteristics ..... 9
**A.C. Characteristics** ..... **10**
High Speed Internal Oscillator – HIRC – Frequency Accuracy ..... 10
Low Speed Internal Oscillator – LIRC – Frequency Accuracy ..... 10
Operating Frequency Characteristic Curves ..... 10
System Start Up Time Characteristics ..... 11
**Input/Output Characteristics** ..... **11**
**Memory Characteristics** ..... **12**
**REM/REMDRV Pin Characteristics** ..... **12**
**LVR Electrical Characteristics** ..... **12**
**Power-on Reset Characteristics** ..... **13**
**System Architecture** ..... **13**
Clocking and Pipelining ..... 13
Program Counter ..... 14
Stack ..... 15
Arithmetic and Logic Unit – ALU ..... 15
**OTP Program Memory** ..... **16**
Structure ..... 16
Special Vectors ..... 16
Look-up Table ..... 16
Table Program Example ..... 17
In Circuit Programming – ICP ..... 18
On-Chip Debug Support – OCDS ..... 18
**Data Memory** ..... **19**
Structure ..... 19
General Purpose Data Memory ..... 19
Special Purpose Data Memory ..... 20

**Special Function Register Description ..... 21**
Indirect Addressing Register – IAR0 ..... 21
Memory Pointer – MP0 ..... 21
Accumulator – ACC ..... 21
Program Counter Low Byte Register – PCL ..... 22
Look-up Table Registers – TBLP, TBLH ..... 22
Status Register – STATUS ..... 22
**Oscillators ..... 23**
Oscillator Overview ..... 23
System Clock Configurations ..... 24
Internal High Speed RC Oscillator – HIRC ..... 24
Internal 32kHz Oscillator – LIRC ..... 24
**Operating Modes and System Clocks ..... 25**
System Clocks ..... 25
System Operation Modes ..... 26
Control Register ..... 27
Operating Mode Switching ..... 28
Standby Current Considerations ..... 31
Wake-up ..... 31
**Watchdog Timer ..... 32**
Watchdog Timer Clock Source ..... 32
Watchdog Timer Control Register ..... 32
Watchdog Timer Operation ..... 33
**Reset and Initialisation ..... 34**
Reset Functions ..... 34
Reset Initial Conditions ..... 37
**Input/Output Ports ..... 39**
Pull-high Resistors ..... 39
I/O Pin Wake-up ..... 39
I/O Port Control Registers ..... 40
I/O Pin Structure ..... 40
Programming Considerations ..... 41
**9-bit Timer with Carrier Output ..... 42**
Timer and Carrier Output Conctrol Register ..... 42
Timer Operation ..... 44
Carrier Output ..... 45
Carrier Output Pins ..... 47
**Interrupts ..... 49**
Interrupt Register ..... 49
Interrupt Operation ..... 49
Time Base Interrupt ..... 50
Interrupt Wake-up Function ..... 51
Programming Considerations ..... 51

**Configuration Options** ..... **52**
**Application Circuits** ..... **53**
**Instruction Set** ..... **54**
Introduction ..... 54
Instruction Timing ..... 54
Moving and Transferring Data ..... 54
Arithmetic Operations ..... 54
Logical and Rotate Operation ..... 55
Branches and Control Transfer ..... 55
Bit Operations ..... 55
Table Read Operations ..... 55
Other Operations ..... 55
**Instruction Set Summary** ..... **56**
Table Conventions ..... 56
**Instruction Definition** ..... **58**
**Package Information** ..... **67**
8-pin SOP (150mil) Outline Dimensions ..... 68
16-pin NSOP (150mil) Outline Dimensions ..... 69
20-pin NSOP (150mil) Outline Dimensions ..... 70
20-pin SSOP (150mil) Outline Dimensions ..... 71

# Features

## CPU Features

- Operating voltage
  - $f_{SYS}$=4MHz: 2.0V~5.5V
- Up to 1μs instruction cycle with 4MHz system clock at $V_{DD}$=5V
- Power down and wake-up functions to reduce power consumption
- Oscillator types
  - Internal High Speed 4MHz RC – HIRC
  - Internal Low Speed 32kHz RC – LIRC
- Multi-mode operation: FAST, SLOW, IDLE and SLEEP
- Fully integrated internal oscillators require no external components
- All instructions executed in one or two instruction cycles
- Table read instructions
- 61 powerful instructions
- 2-level subroutine nesting
- Bit manipulation instruction

## Peripheral Features

- OTP Program Memory: (1K-16)×14
- Data Memory: 32×8
- Watchdog Timer function
- 16 bidirectional I/O lines
- One programmable carrier output – using 9-bit timer
- High Driving Current Output pin
- Single Time-Base function for generation of fixed time interrupt signals
- Low voltage reset function
- Package types: 8-pin SOP, 16-pin NSOP, 20-pin NSOP/SSOP

## General Description

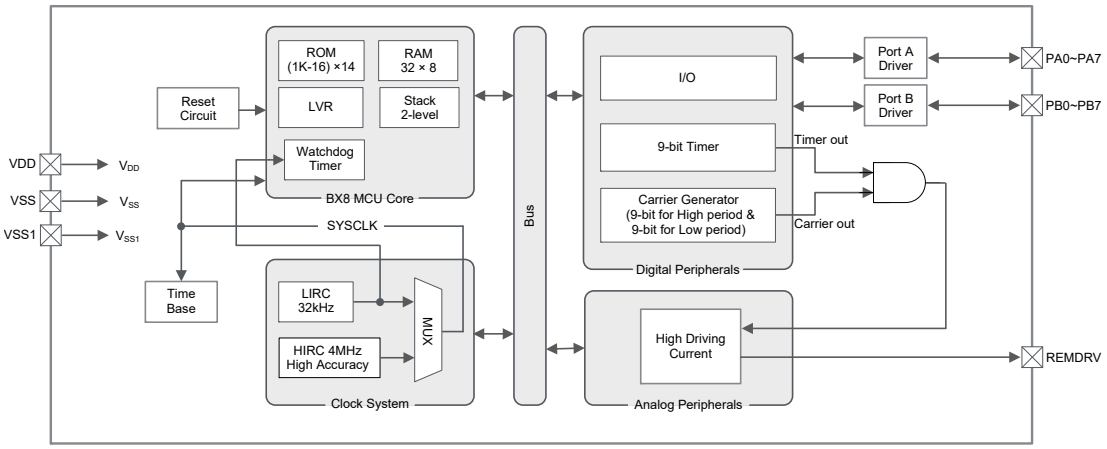
The device is an OTP Memory type 8-bit high performance RISC architecture microcontroller, designed for IR remote controllers and IR transmission related products.

For memory features, the device includes an OTP Program Memory and an area of RAM Data Memory. 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.

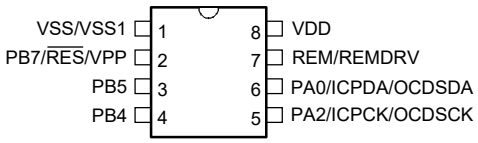
A full choice of internal high and low speed oscillators are provided and the two fully integrated system oscillators require no external components for their implementation. The ability to operate and switch dynamically between a range of operating modes using different clock sources gives users the ability to optimize microcontroller operation and minimize power consumption.

The device includes a 9-bit timer for IR carrier output and a Time-base function which can generate fixed time interrupt. The inclusion of flexible I/O programming features, high driving current capacity with many other features ensure that the device will find excellent use in IR remote controller and timer applications which require a high accuracy clock, a timing function or a high driving current.

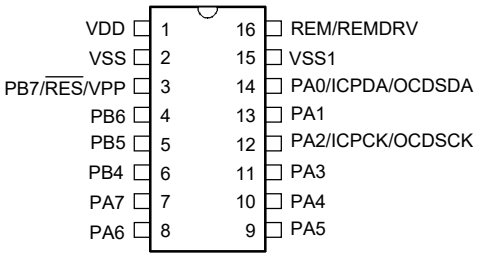
## Block Diagram



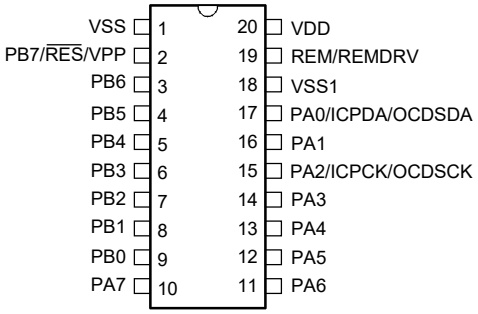
## Pin Assignment




**BX68R2420/BX68V2420**
**8 SOP-A**




**BX68R2420/BX68V2420**
**16 NSOP-A**




**BX68R2420/BX68V2420**
**20 NSOP/SSOP-A**

Note: 1. If the pin-shared pin functions have multiple outputs, the desired pin-shared function is determined by the corresponding software control bits.

2. The OCDSDA and OCDSCK pins are supplied for the OCDS dedicated pins and only available for the BX68V2420 device (Flash type) which is the OCDS EV chip for the BX68R2420 device (OTP type).

3. For less pin-count package types there will be unbonded pins which should be properly configured to avoid unwanted current consumption resulting from floating input conditions. Refer to the “Standby Current Considerations” and “Input/Output Ports” sections.

4. The VPP pin is the High Voltage input for OTP programming and only available for the BX68R2420 device.

## Pin Description

The function of each pin is listed in the following table, however the details behind how each pin is configured is contained in other sections of the datasheet. As the Pin Description table shows the situation for the package with the most pins, not all pins in the tables will be available on smaller package sizes.

| Pin Name | Function | OPT | I/T | O/T | Description |
|---|---|---|---|---|---|
| PA0/ICPDA/ OCDSDA | PA0 | PAPU PAWU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| | ICPDA | — | ST | CMOS | ICP Data/Address pin |
| | OCDSDA | — | ST | CMOS | OCDS Address/Data pin, for EV chip only |
| PA1 | PA1 | PAPU PAWU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| PA2/ICPCK/ OCDSCK | PA2 | PAPU PAWU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| | ICPCK | — | ST | — | ICP clock pin |
| | OCDSCK | — | ST | — | OCDS clock pin, for EV chip only |
| PA3~PA7 | PA3~PA7 | PAPU PAWU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| PB0~PB6 | PB0~PB6 | PBPU PBWU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. |
| PB7/$\overline{\text{RES}}$/VPP | PB7 | PBPU PBWU | ST | CMOS | General purpose I/O. Register enabled pull-up and wake-up. This I/O is pin-shared with VPP and could result in increased current consumption if it is set to output high. |
| | $\overline{\text{RES}}$ | CO | ST | — | External reset input, select by option table |
| | VPP | — | PWR | — | High Voltage input for OTP programming, not available for EV chip |
| REM/REMDRV | REM | — | — | CMOS | CMOS carrier output pin |
| | REMDRV | — | — | NMOS | NMOS carrier output pin |
| VDD | VDD | — | PWR | — | Digital positive power supply |
| VSS | VSS | — | PWR | — | Digital negative power supply, ground |
| VSS1 | VSS1 | — | PWR | — | REMDRV negative power supply |

Legend: I/T: Input type;
OPT: Optional by register option;
CO: Configuration option;
CMOS: CMOS output;
O/T: Output type;
PWR: Power;
ST: Schmitt Trigger input;
NMOS: NMOS output.

## Absolute Maximum Ratings

Supply Voltage ..... $V_{SS}$-0.3V to 6.0V

Input Voltage ..... $V_{SS}$-0.3V to $V_{DD}$+0.3V

Storage Temperature ..... -60°C to 150°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 the device. Functional operation of the device at other conditions beyond those listed in the specification is not implied and prolonged exposure to extreme conditions may affect device reliability.

## D.C. Characteristics

For data in the following tables, note that factors such as oscillator type, operating voltage, operating frequency, pin load conditions, temperature and program instruction type, etc., can all exert an influence on the measured values.

### Operating Voltage Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|
| $V_{DD}$ | Operating Voltage – HIRC | $f_{SYS}=f_{HIRC}$=4MHz | 2.0 | — | 5.5 | V |
| | Operating Voltage – LIRC | $f_{SYS}=f_{LIRC}$=32kHz | 2.0 | — | 5.5 | V |

### Standby Current Characteristics

Ta=25°C, unless otherwise specified.

| Symbol | Standby Mode | Test Conditions | | Min. | Typ. | Max. | Max. @85°C | Unit |
|---|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | | |
| $I_{STB}$ | SLEEP Mode | 2V | WDT off | — | 0.45 | 0.80 | 7.00 | μA |
| | | 3V | | — | 0.45 | 0.90 | 8.00 | |
| | | 5V | | — | 0.5 | 2.0 | 10.0 | |
| | | 2V | WDT on | — | 1.5 | 3.0 | 5.5 | μA |
| | | 3V | | — | 1.8 | 3.6 | 6.5 | |
| | | 5V | | — | 3 | 5 | 10 | |
| | IDLE0 Mode – LIRC | 2V | $f_{SUB}$ on | — | 2.4 | 4.0 | 8.0 | μA |
| | | 3V | | — | 3 | 5 | 9 | |
| | | 5V | | — | 5 | 10 | 11 | |
| | IDLE1 Mode – HIRC | 2V | $f_{SUB}$ on, $f_{SYS}$=4MHz | — | 450 | 500 | 600 | μA |
| | | 3V | | — | 500 | 600 | 700 | |
| | | 5V | | — | 600 | 800 | 960 | |

Note: When using the characteristic table data, the following notes should be taken into consideration:

1. Any digital inputs are set in a non-floating condition and the I/O pin-shared with VPP is not setup in an output high condition.
2. All measurements are taken under conditions of no load and with all peripherals in an off state.
3. There are no DC current paths.
4. All Standby Current values are taken after a HALT instruction execution thus stopping all instruction execution.

### Operating Current Characteristics

Ta=25°C

| Symbol | Operating Mode | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $I_{DD}$ | SLOW Mode – LIRC | 2V | $f_{SYS}$=32kHz | — | 12 | 24 | μA |
| | | 3V | | — | 15 | 30 | |
| | | 5V | | — | 30 | 50 | |
| | FAST Mode – HIRC | 2V | $f_{SYS}$=4MHz | — | 0.6 | 1.0 | mA |
| | | 3V | | — | 0.8 | 1.2 | |
| | | 5V | | — | 1.6 | 2.4 | |

Note: When using the characteristic table data, the following notes should be taken into consideration:

1. Any digital inputs are set in a non-floating condition and the I/O pin-shared with VPP is not setup in an output high condition.
2. All measurements are taken under conditions of no load and with all peripherals in an off state.
3. There are no DC current paths.
4. All Operating Current values are measured using a continuous NOP instruction program loop.

## A.C. Characteristics

For data in the following tables, note that factors such as oscillator type, operating voltage, operating frequency and temperature, etc., can all exert an influence on the measured values.

### High Speed Internal Oscillator – HIRC – Frequency Accuracy

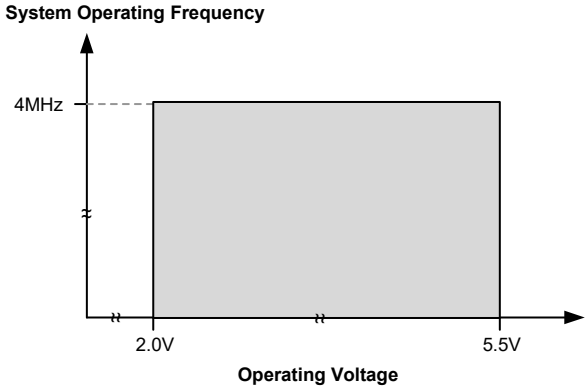
| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $f_{HIRC}$ | HIRC Frequency (System Frequency) | 2.2V~3.6V | -10°C~50°C | -0.8% | 4 | +0.8% | MHz |
| | | | -40°C~85°C | -1.5% | 4 | +1.5% | |
| | | 2.0V~3.6V | -10°C~50°C | -0.8% | 4 | +0.8% | |
| | | | -40°C~85°C | -1.5% | 4 | +1.5% | |
| | | 2.2V~5.5V | -10°C~50°C | -0.8% | 4 | +0.8% | |
| | | | -40°C~85°C | -1.5% | 4 | +1.5% | |
| | | 2.0V~5.5V | -40°C~85°C | -1.5% | 4 | +1.5% | |

### Low Speed Internal Oscillator – LIRC – Frequency Accuracy

Ta=-40°C~85°C, unless otherwise specified.

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Temp. | | | | |
| $f_{LIRC}$ | LIRC Frequency | 2.0V~5.5V | 25°C | -20% | 32 | +20% | kHz |
| | | | -40°C~85°C | -50% | 32 | +60% | |
| $t_{START}$ | LIRC Start Up Time | — | — | — | — | 500 | μs |

### Operating Frequency Characteristic Curves



### System Start Up Time Characteristics

Ta=-40°C~85°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $t_{SST}$ | System Start-up Time Wake-up from Condition Where $f_{SYS}$ is Off | — | $f_{SYS}=f_H$~$f_H/64$, $f_H=f_{HIRC}$ | — | 128 | — | $t_{SYS}$ |
| | | — | $f_{SYS}=f_{SUB}=f_{LIRC}$ | — | 2 | — | $t_{SUB}$ |
| | System Start-up Time Wake-up from Condition Where $f_{SYS}$ is On | — | $f_{SYS}=f_H$~$f_H/64$, $f_H=f_{HIRC}$ | — | 2 | — | $t_{SYS}$ |
| | | — | $f_{SYS}=f_{SUB}=f_{LIRC}$ | — | 2 | — | $t_{SUB}$ |
| | System Speed Switch Time FAST to SLOW Mode or SLOW to FAST Mode | — | $f_{HIRC}$ switches from off to on | — | 128 | — | $t_{HIRC}$ |
| $t_{RSTD}$ | System Reset Delay Time Reset Source from Power-on Reset or LVR Hardware Reset | — | $RR_{POR}$=5V/ms | 5 | 16 | 80 | ms |
| | System Reset Delay Time Reset Source from WDT Overflow or $\overline{RES}$ Pin Reset | — | — | | | | |
| $t_{SRESET}$ | Minimum Software Reset Width to Reset | — | — | 45 | 90 | 120 | μs |

Note: 1. For the System Start-up time values, whether $f_{SYS}$ is on or off depends upon the mode type and the chosen $f_{SYS}$ system oscillator. Details are provided in the System Operating Modes section.
2. The time units, shown by the symbols, $t_{HIRC}$, etc., are the inverse of the corresponding frequency values as provided in the frequency tables. For example $t_{HIRC}=1/f_{HIRC}$, $t_{LIRC}=1/f_{LIRC}$, etc.
3. If the LIRC is used as the system clock and if it is off when in the SLEEP Mode, then an additional LIRC start up time, $t_{START}$, as provided in the LIRC frequency table, must be added to the $t_{SST}$ time in the table above.
4. The System Speed Switch Time is effectively the time taken for the newly activated oscillator to start up.

## Input/Output Characteristics

Ta=-40°C~85°C, unless otherwise specify.

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $V_{IL}$ | Input Low Voltage for I/O Ports | 5V | — | 0 | — | 1.5 | V |
| | | — | — | 0 | — | $0.2V_{DD}$ | |
| | Input Low Voltage for $\overline{RES}$ Pin | — | $V_{DD}$≥2.7 | 0 | — | 0.4 $V_{DD}$ | V |
| | | — | 2.0≤$V_{DD}$<2.7 | 0 | — | $0.3V_{DD}$ | V |
| $V_{IH}$ | Input High Voltage for I/O Ports | 5V | — | 3.5 | — | 5 | V |
| | | — | — | $0.8V_{DD}$ | — | $V_{DD}$ | |
| | Input High Voltage for $\overline{RES}$ Pin | — | — | $0.9V_{DD}$ | — | $V_{DD}$ | V |
| $I_{OL}$ | Sink Current for I/O Ports | 3V | $V_{OL}=0.1V_{DD}$ | 5 | 10 | — | mA |
| | | 5V | | 10 | 20 | — | |
| $I_{OH}$ | Source Current for I/O Ports | 3V | $V_{OH}=0.9V_{DD}$ | -2.5 | -5.0 | — | mA |
| | | 5V | | -5.0 | -10 | — | |
| $R_{PH}$ | Pull-high Resistance for I/O Ports (Note) | 3V | — | 20 | 60 | 100 | kΩ |
| | | 5V | — | 10 | 30 | 50 | |
| $I_{LEAK}$ | Input Leakage Current for I/O Ports | 5V | $V_{IN}=V_{DD}$ or $V_{IN}=V_{SS}$ | — | — | ±1 | μA |
| $t_{RES}$ | External Reset Minimum Low Pulse Width | — | — | 0.3 | — | — | μs |

Note: The $R_{PH}$ internal pull high resistance value is calculated by connecting to ground and enabling the input pin with a pull-high resistor and then measuring the pin current at the specified supply voltage level. Dividing the voltage by this measured current provides the $R_{PH}$ value.

## Memory Characteristics

Ta=-40°C~85°C, unless otherwise specify.

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| **OTP Program Memory** | | | | | | | |
| $t_{RETD}$ | ROM Data Retention Time | — | Ta=25°C | — | 40 | — | Year |
| **Flash Program Memory – for BX68V2420 only** | | | | | | | |
| $t_{ACTV}$ | ROM Activation Time – Wake-up from Power Down Mode | — | — | 32 | — | 64 | μs |
| **RAM Data Memory** | | | | | | | |
| $V_{DR}$ | RAM Data Retention Voltage | — | — | 1.0 | — | — | V |

Note: The ROM activation time $t_{ACTV}$ should be added when calculating the total system start-up time of a wake-up from the IDLE/SLEEP mode.

## REM/REMDRV Pin Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $I_{OL}$ | Sink Current for REMDRV Pin | 3V | $V_{OL}$=0.5V | — | 500 | — | mA |
| | Sink Current for REM Pin | 3V | $V_{OL}$=0.1$V_{DD}$ | 5 | 10 | — | mA |
| | | 5V | | 10 | 20 | — | |
| $I_{OH}$ | Source Current for REM Pin | 3V | $V_{OL}$=0.9$V_{DD}$ | -1.25 | -2.50 | — | mA |
| | | 5V | | -2.5 | -5.0 | — | |
| $t_{READYB}$ | REMDRV Output Function Stable Time (by polling the READYB bit=0) | — | Wake-up from IDLE or SLEEP mode or REMDRV bit from 1 to 0 | — | 250 | — | μs |

## LVR Electrical Characteristics

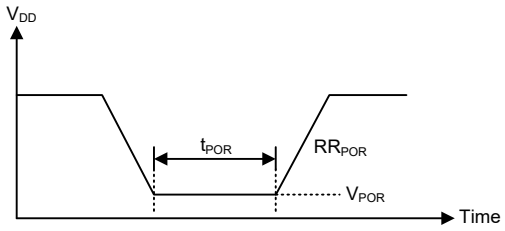
Ta=-40°C~85°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---|---|---|---|---|---|---|---|
| | | $V_{DD}$ | Conditions | | | | |
| $V_{LVR}$ | Low Voltage Reset Voltage | — | LVR enable, voltage select 1.90V | -5% | 1.9 | +5% | V |
| | | | LVR enable, voltage select 2.10V | -5% | 2.1 | +5% | |
| | | | LVR enable, voltage select 3.15V | -5% | 3.15 | +5% | |
| | | | LVR enable, voltage select 4.20V | -5% | 4.2 | +5% | |
| $I_{LVR}$ | Operating Current | 3V | LVR enable, $V_{LVR}$=1.9V | — | — | 20 | μA |
| | | 5V | LVR enable, $V_{LVR}$=1.9V | — | 25 | 35 | |
| $t_{LVR}$ | Minimum Low Voltage Width to Reset | — | — | 100 | 240 | 1250 | μs |

## Power-on Reset Characteristics

Ta=-40°C~85°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 |
| $RR_{POR}$ | $V_{DD}$ Rising 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 |



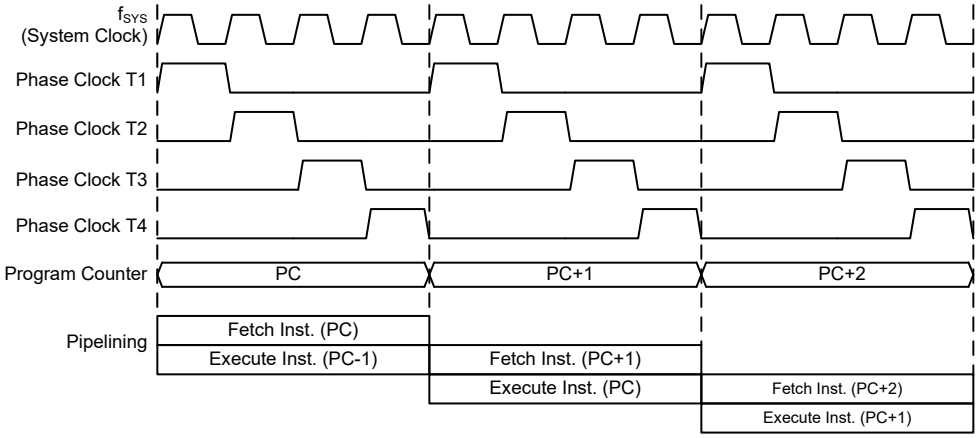

## System Architecture

A key factor in the high-performance features of the microcontrollers is attributed to their internal system architecture. The device takes advantage of the usual features found within RISC microcontrollers providing increased speed of operation and enhanced performance. The pipelining scheme is implemented in such a way that instruction fetching and instruction execution are overlapped, hence instructions are effectively executed in one cycle, with the exception of branch or call instructions which need one more cycle. 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 control system with maximum reliability and flexibility. This makes the device suitable for affordable, high-volume production for controller applications.

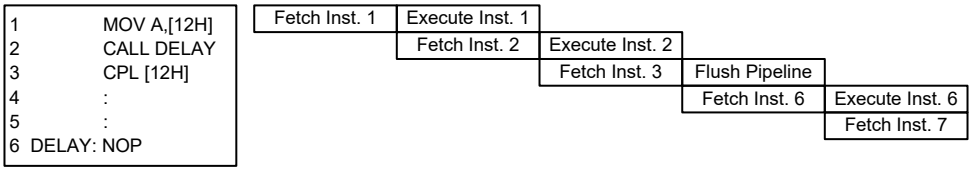
### Clocking and Pipelining

The main system clock, derived from either a HIRC or LIRC 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.

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




**System Clocking and Pipelining**




**Instruction Fetching**

## Program Counter

During program execution, the Program Counter is used to keep track of the address of the next instruction to be executed. It is automatically incremented by one each time an instruction is executed except for instructions, such as “JMP” or “CALL” that demand a jump to a non-consecutive Program Memory address. 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.

| Program Counter | |
|---|---|
| **Program Counter High Byte** | **PCL Register** |
| PC9~PC8 | PCL7~PCL0 |

**Program Counter**

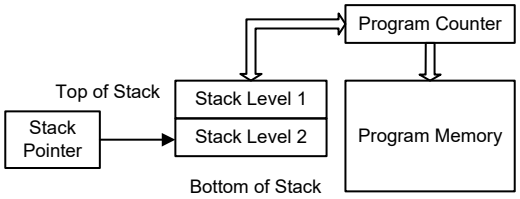
The lower byte of the Program Counter, known as the Program Counter Low register or PCL, is available for program control and is a readable and writeable register. By transferring data directly into this register, a short program jump can be executed directly; however, as only this low byte is available 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, organized into two levels, 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:

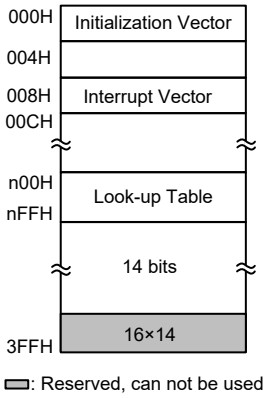
- Arithmetic operations: ADD, ADDM, ADC, ADCM, SUB, SUBM, SBC, SBCM, DAA
- Logic operations: AND, OR, XOR, ANDM, ORM, XORM, CPL, CPLA
- Rotation: RRA, RR, RRCA, RRC, RLA, RL, RLCA, RLC
- Increment and Decrement: INCA, INC, DECA, DEC
- Branch decision: JMP, SZ, SZA, SNZ, SIZ, SDZ, SIZA, SDZA, CALL, RET,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

## OTP Program Memory

The Program Memory is the location where the user code or program is stored. The device is supplied with One-Time Programmable, OTP memory where users can program their application code into the device.

### Structure

The Program Memory has a capacity of (1K-16)×14 bits. Note that the subtractive 16×14 bits space is reserved and cannot be used. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries. Table data, which can be set in any location within the Program Memory, is addressed by a separate table pointer register.




**Program Memory Structure**

### Special Vectors

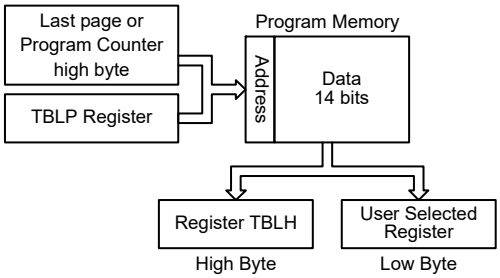
Within the Program Memory, certain locations are reserved for the reset and the interrupt. The location 000H is reserved for use by the device reset for program initialisation. After a device reset is initiated, the program will jump to this location and begin execution.

### 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 configured by placing the address of the look up data to be retrieved in the table pointer register, TBLP. This register defines the lower 8-bit address of the look-up table.

After setting up the table pointer, the table data can be retrieved from the Program Memory using the “TABRD [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.

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




## Table Program Example

The following example shows how the table pointer and table data is defined and retrieved from the microcontrollers. 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 "300H" which refers to the start address of the last page within the 1K words Program Memory of the device. The table pointer low byte register is set here to have an initial value of "06H". This will ensure that the first data read from the data table will be at the Program Memory address "306H" or 6 locations after the start of the last page. Note that the value for the table pointer is referenced to the specific address of the TBLP register if the "TABRD [m]" instruction is being used. The high byte of the table data which in this case is equal to zero will be transferred to the TBLH register automatically when the "TABRD [m] instruction is executed.

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

### Table Read Program Example

```
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         ; to the last page or present page
:
:
tabrd tempreg1     ; transfers value in table referenced by table pointer, data at program
                   ; memory address "306H" transferred to tempreg1 and TBLH
dec tblp           ; reduce value of table pointer by one
tabrd tempreg2     ; transfers value in table referenced by table pointer, data at program
                   ; memory address "305H" transferred to tempreg2 and TBLH
                   ; in this example the data "1AH" is transferred to tempreg1 and data "0FH"
                   ; to register tempreg2
                   ; the value "00H" will be transferred to the high byte register TBLH
:
:
org 300h           ; sets initial address of program memory
dc 00Ah, 00Bh, 00Ch, 00Dh, 00Eh, 00Fh, 01Ah, 01Bh
```

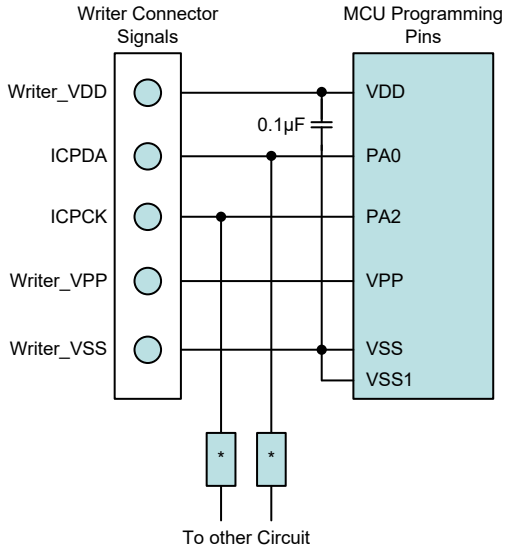
## In Circuit Programming – ICP

The provision of OTP type Program Memory, users can program their application One-Time into the device. As an additional convenience. The device has provided a means of programming in-circuit using a 5-pin interface. This provides manufacturers with the possibility of manufacturing their circuit boards complete with an un-programmed microcontroller, and then programming the program at a later stage.

| Writer Pins | MCU Programming Pins | Pin Description |
|---|---|---|
| ICPDA | PA0 | Programming Serial Data/Address |
| ICPCK | PA2 | Programming Clock |
| VPP | VPP | Programming OTP ROM power supply (8.5V) |
| VDD | VDD | Power Supply. A 0.1μF capacitor between VDD and VSS is required for programming. |
| VSS | VSS&VSS1 | Ground |

The Program Memory can be programmed serially in-circuit using this 5-wire interface. Data is downloaded and uploaded serially on a single pin with an additional line for the clock. Three additional lines are required for the power supply. The technical details regarding the in-circuit programming of the device is beyond the scope of this document and will be supplied in supplementary literature.

During the programming process, the user must take care of the ICPDA and ICPCK pins for data and clock programming purposes to ensure that no other outputs are connected to these two pins.




Note: 1. A 0.1μF capacitor is required to be connected between VDD and VSS for ICP programming and located as close to these pins as possible.

2. * 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 BX68V2420 which is used to emulate the real MCU device named BX68R2420. The EV chip device also provides an “On-Chip Debug” function to debug the real MCU device during the development process. The EV chip and the real MCU device are almost functionally compatible except for “On-Chip Debug” function. Users can use the EV chip device to emulate the real chip device behavior by connecting the OCDSDA and OCDSCK pins to the

ICE 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 EV chip for debugging, other functions which are shared with the OCDSDA and OCDSCK pins in the device will have no effect in the EV chip.

| ICE Pins | EV Chip 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 |
| VSS | VSS&VSS1 | Ground |

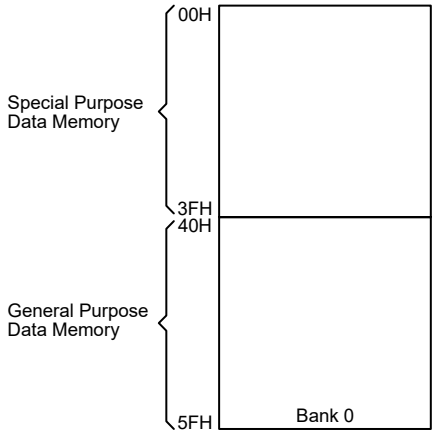
## Data Memory

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

### Structure

Categorised into two types, the first of these is an area of RAM, known as the Special Function Data Memory. These registers have fixed locations and are necessary for correct operation of the device. Many of these registers can be read from and written to directly under program control, however, some remain protected from user manipulation. The second area of Data Memory is 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 start address of the Data Memory for the device is 00H. The address range of the Special Purpose Data Memory for the device is from 00H to 3FH while the address range of the General Purpose Data Memory is from 40H to 5FH.



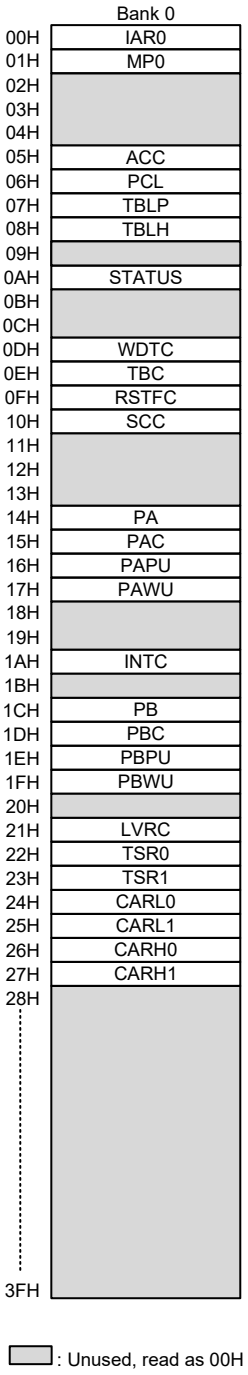

**Data Memory Structure**

### General Purpose Data Memory

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

## Special Purpose Data Memory

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




**Special Purpose Data Memory**

## Special Function 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 Register – IAR0

The Indirect Addressing Register, IAR0, although having its location in normal RAM register space, do not actually physically exist as normal register. The method of indirect addressing for RAM data manipulation uses this Indirect Addressing Register and Memory Pointer, in contrast to direct memory addressing, where the actual memory address is specified. Actions on the IAR0 register will result in no actual read or write operation to this register but rather to the memory location specified by the corresponding Memory Pointer, MP0. Acting as a pair, IAR0 and MP0 can together access data from Bank 0. As the Indirect Addressing Register is not physically implemented, reading the Indirect Addressing Register will return a result of “00H” and writing to the register will result in no operation.

### Memory Pointer – MP0

A Memory Pointer, known as MP0 is provided. The Memory Pointer is 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 Register 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.

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

#### Indirect Addressing Program Example

```
data .section 'data'
adres1   db ?
adres2   db ?
adres3   db ?
adres4   db ?
block    db ?
code .section at 0 'code'
org 00h
start:
     mov a, 04h              ; set size of block
     mov block, a
     mov a, offset adres1    ; Accumulator loaded with first RAM address
     mov MP0, a              ; set memory pointer with first RAM address
loop:
     clr IAR0                ; clear the data at address defined by MP0
     inc MP0                 ; increase 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 examples shown above, no reference is made to specific Data Memory addresses.

### Accumulator – ACC

The Accumulator is central to the operation of any microcontroller and is closely related with operations carried out by the ALU. The Accumulator is the place where all intermediate results

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

### Program Counter Low Byte Register – PCL

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

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

The special function register TBLP is used to control operation of the look-up table which is stored in the Program Memory. TBLP is the table pointer and indicates the location where the table data is located. Its value must be set before any table read commands are executed. Its value can be changed, for example using the “INC” or “DEC” instructions, allowing for easy table data pointing and reading. TBLH is the location where the high order byte of the table data is stored after a table read data instruction has been executed. Note that the lower order table data byte is transferred to a user defined location.

### Status Register – STATUS

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

With the exception of the TO and PDF flags, bits in the status register can be altered by instructions like most 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 | x | x | x | x |

"x": unknown

Bit 7~6 Unimplemented, read as "0"

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

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

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

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

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

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

The "C" flag is also affected by a rotate through carry instruction.

## Oscillators

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 optimisation can be achieved in terms of speed and power saving. Oscillator operations are selected through the relevant control registers.

### Oscillator Overview

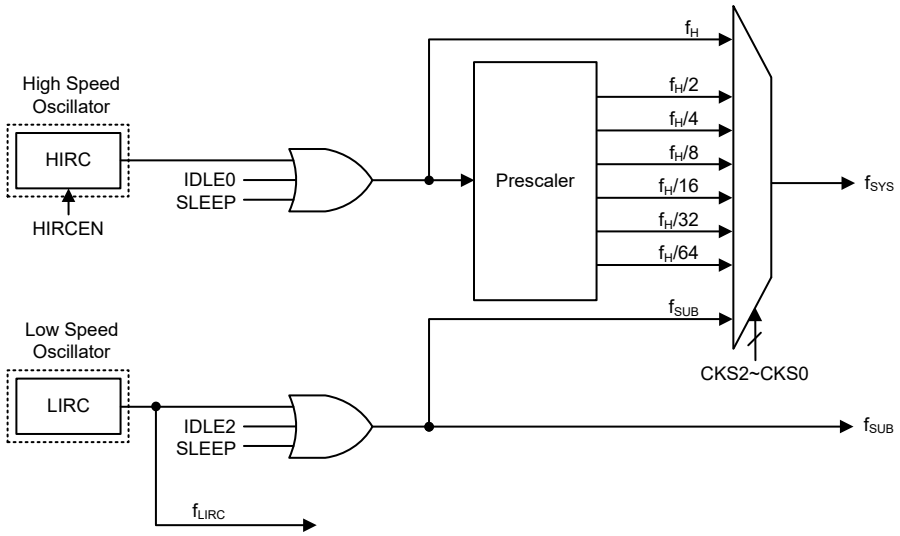
In addition to being the source of the main system clock the oscillators also provide clock sources for the Watchdog Timer and Time Base Interrupt. The fully integrated internal oscillators, requiring no external components, are provided to form a wide range of both fast and slow system oscillators. The higher frequency oscillator provides higher performance but carry with it the disadvantage of higher power requirements, while the opposite is of course true for the lower frequency oscillator. 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 | Frequency |
|---|---|---|
| Internal High Speed RC | HIRC | 4MHz |
| Internal Low Speed RC | LIRC | 32kHz |

**Oscillator Types**

## System Clock Configurations

There are two oscillator sources, one high speed oscillator and one low speed oscillator. The high speed system clock is sourced from the internal 4MHz RC oscillator, HIRC. The low speed oscillator is the internal 32kHz RC oscillator, LIRC. Selecting whether the low or high speed oscillator is used as the system oscillator is implemented using the CKS2~CKS0 bits in the SCC register and the system clock can be dynamically selected.




**System Clock Configuration**

## Internal High Speed RC Oscillator – HIRC

The internal RC oscillator is a fully integrated system oscillator requiring no external components. The internal RC oscillator has one fixed frequency of 4MHz. Device trimming during the manufacturing process and the inclusion of internal frequency compensation circuits are used to ensure that the influence of the power supply voltage, temperature and process variations on the oscillation frequency are minimised.

## Internal 32kHz Oscillator – LIRC

The Internal 32kHz System Oscillator is a fully integrated low frequency RC oscillator with a typical frequency of 32kHz, 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.

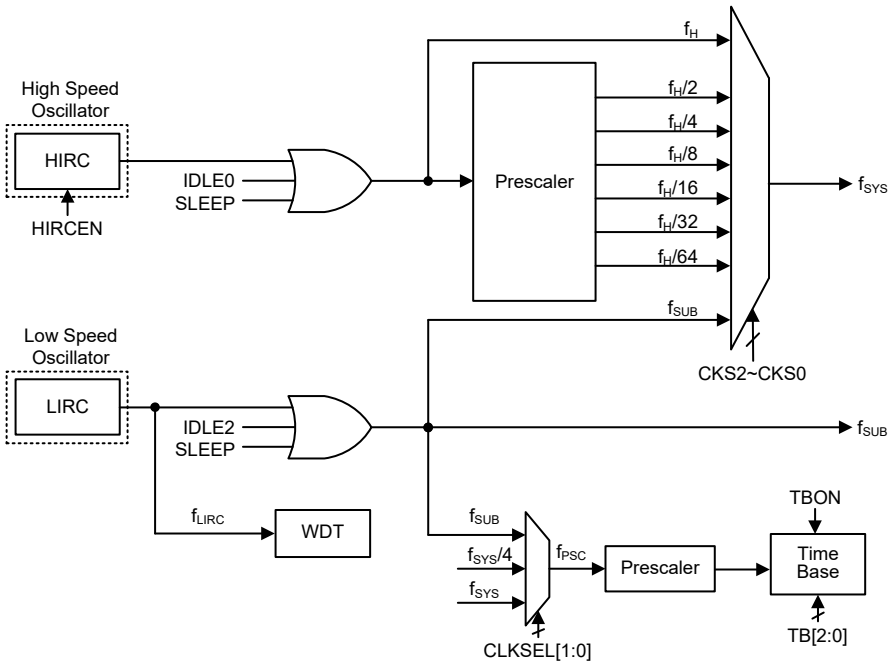
## Operating Modes and System Clocks

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 the device has provided both high and low speed clock sources and the means to switch between them dynamically, the user can optimise the operation of their microcontroller to achieve the best performance/power ratio.

### System Clocks

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

The main system clock, can come from either a high frequency, $f_H$, or low frequency, $f_{SUB}$, source, and is selected using the CKS2~CKS0 bits in the SCC register. The high speed system clock is sourced from the HIRC oscillator. The low speed system clock source is sourced from the LIRC oscillator. The other choice, which is a divided version of the high speed system oscillator has a range of $f_H/2$~$f_H/64$.




**Device Clock Configurations**

Note: When the system clock source $f_{SYS}$ is switched to $f_{SUB}$ from $f_H$, the high speed oscillator will stop to conserve the power or continue to oscillate to provide the clock source, $f_H$~$f_H/64$, for peripheral circuit to use, which is determined by configuring the corresponding high speed oscillator enable control bit.

## System Operation Modes

There are six different modes of operation for the microcontrollers, 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 FAST Mode and SLOW Mode. The remaining four modes, the SLEEP, IDLE0, IDLE1 and IDLE2 Mode are used when the microcontroller CPU is switched off to conserve power.

| Operation Mode | CPU | Register Setting | | | $f_{SYS}$ | $f_H$ | $f_{SUB}$ | $f_{LIRC}$ |
|---|---|---|---|---|---|---|---|---|
| | | FHIDEN | FSIDEN | CKS2~CKS0 | | | | |
| FAST | On | x | x | 000~110 | $f_H$~$f_H$/64 | On | On | On |
| SLOW | On | x | x | 111 | $f_{SUB}$ | On/Off (1) | On | On |
| IDLE0 | Off | 0 | 1 | 000~110 | Off | Off | On | On |
| | | | | 111 | On | | | |
| IDLE1 | Off | 1 | 1 | xxx | On | On | On | On |
| IDLE2 | Off | 1 | 0 | 000~110 | On | On | Off | On |
| | | | | 111 | Off | | | |
| SLEEP | Off | 0 | 0 | xxx | Off | Off | Off | On/Off (2) |

"x": Don't care

Note: 1. The $f_H$ clock will be switched on or off by configuring the corresponding oscillator enable bit in the SLOW mode.

2. The $f_{LIRC}$ clock can be switched on or off which is controlled by the WDT function being enabled or disabled in the SLEEP mode.

### FAST Mode

This is one of the main operating modes where the microcontrollers have all of their functions operational and where the system clock is provided by the high speed oscillator. This mode operates allowing the microcontrollers to operate normally with a clock source which 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 bits in the SCC register. Although a high speed oscillator is used, running the microcontrollers 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 $f_{SUB}$, which is derived from the LIRC oscillator.

### SLEEP Mode

The SLEEP Mode is entered when an HALT instruction is executed and when the FHIDEN and FSIDEN bit are low. In the SLEEP mode the CPU will be stopped. The $f_{SUB}$ clock provided to the peripheral function will also be stopped, too. However the $f_{LIRC}$ clock can continues to operate if the WDT function is enabled.

### IDLE0 Mode

The IDLE0 Mode is entered when an HALT instruction is executed and when the FHIDEN bit in the SCC register is low and the FSIDEN bit in the SCC register is high. In the IDLE0 Mode the CPU will be switched off but the low speed oscillator will be turned on to drive some peripheral functions.

### IDLE1 Mode

The IDLE1 Mode is entered when an HALT instruction is executed and when the FHIDEN bit in the SCC register is high and the FSIDEN bit in the SCC register is high. In the IDLE1 Mode the CPU will be switched off but both the high and low speed oscillators will be turned on to provide a clock source to keep some peripheral functions operational.

### IDLE2 Mode

The IDLE2 Mode is entered when an HALT instruction is executed and when the FHIDEN bit in the SCC register is high and the FSIDEN bit in the SCC register is low. In the IDLE2 Mode the CPU will be switched off but the high speed oscillator will be turned on to provide a clock source to keep some peripheral functions operational.

## Control Register

The SCC register is used to control the system clock and the corresponding oscillator configurations.

- **SCC Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | CKS2 | CKS1 | CKS0 | — | HIRCF | HIRCEN | FHIDEN | FSIDEN |
| R/W | R/W | R/W | R/W | — | R | R/W | R/W | R/W |
| POR | 1 | 1 | 1 | — | 0 | 0 | 0 | 0 |

Bit 7~5 **CKS2~CKS0**: System clock selection
000: $f_H$
001: $f_H/2$
010: $f_H/4$
011: $f_H/8$
100: $f_H/16$
101: $f_H/32$
110: $f_H/64$
111: $f_{SUB}$
These three bits are used to select which clock is used as the system clock source. In addition to the system clock source directly derived from $f_H$ or $f_{SUB}$, 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 **HIRCF**: HIRC oscillator stable flag
0: HIRC unstable
1: HIRC stable
This bit is used to indicate whether the HIRC oscillator is stable or not. When the HIRCEN bit is set to 1 to enable the HIRC oscillator, the HIRCF bit will first be cleared to 0 and then set to 1 after the HIRC oscillator is stable.

Bit 2 **HIRCEN**: HIRC oscillator enable control
0: Disable
1: Enable

Bit 1 **FHIDEN**: High Frequency oscillator control when CPU is switched off
0: Disable
1: Enable
This bit is used to control whether the high speed oscillator is activated or stopped when the CPU is switched off by executing an "HALT" instruction.

Bit 0 **FSIDEN**: Low Frequency oscillator control when CPU is switched off
0: Disable
1: Enable
This bit is used to control whether the low speed oscillator is activated or stopped when the CPU is switched off by executing an "HALT" instruction.

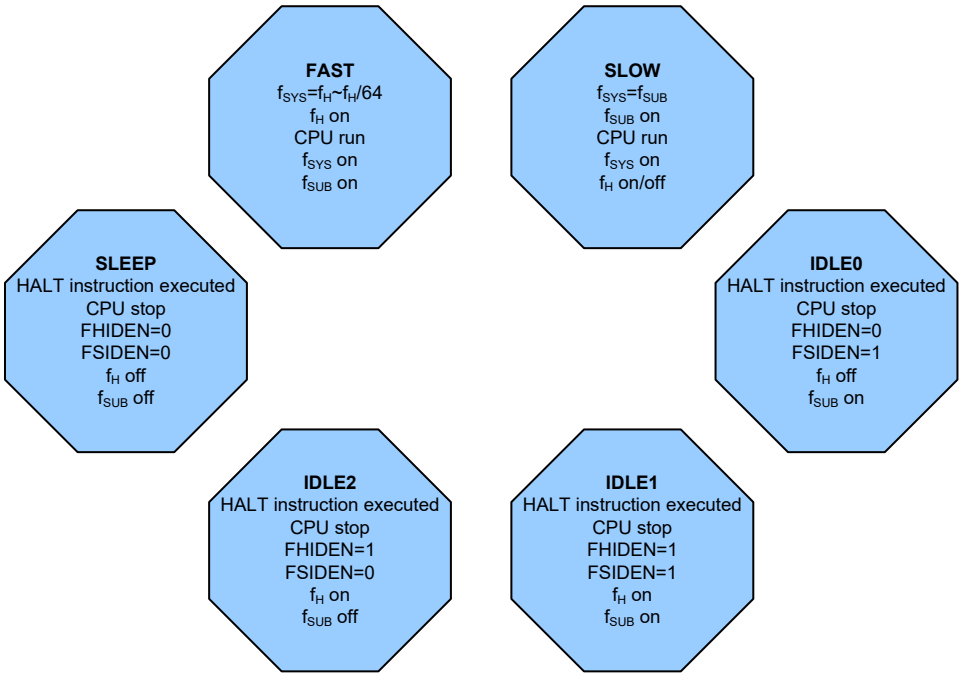
Note: A certain delay is required before the relevant clock is successfully switched to the target clock source after any clock switching setup using the CKS2~CKS0 bits. A proper delay time must be arranged before executing the following operations which require immediate reaction with the target clock source.

Clock switching delay time=$4\times t_{SYS}+[0\sim(1.5\times t_{Curr.}+0.5\times t_{Tar.})]$, where $t_{Curr.}$ indicates the current clock period, $t_{Tar.}$ indicates the target clock period and the $t_{SYS}$ indicates the current system clock period.

## 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.

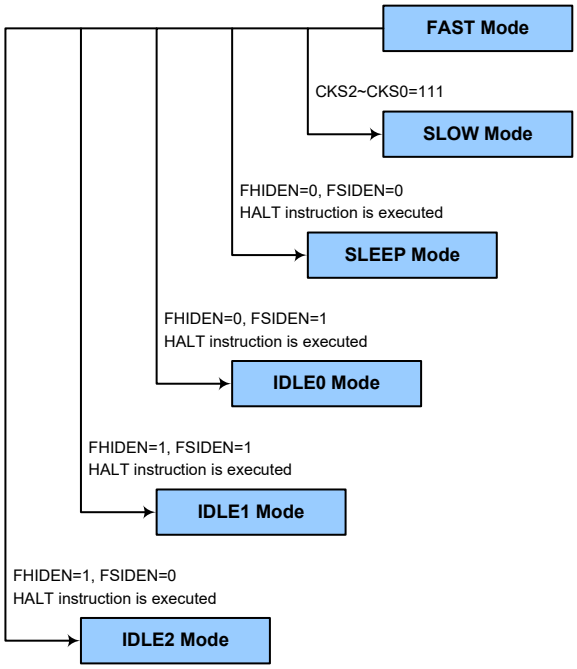
In simple terms, mode switching between the FAST Mode and SLOW Mode is executed using the CKS2~CKS0 bits in the SCC register while mode switching from the FAST/SLOW Modes to the SLEEP/IDLE Modes is executed via the HALT instruction. When an HALT instruction is executed, whether the device enters the IDLE Mode or the SLEEP Mode is determined by the condition of the FHIDEN and FSIDEN bits in the SCC register.




### FAST Mode to SLOW Mode Switching

When running in the FAST Mode, which uses the high speed system oscillator, and therefore consumes more power, the system clock can switch to run in the SLOW Mode by set the CKS2~CKS0 bits to “111” in the SCC 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.

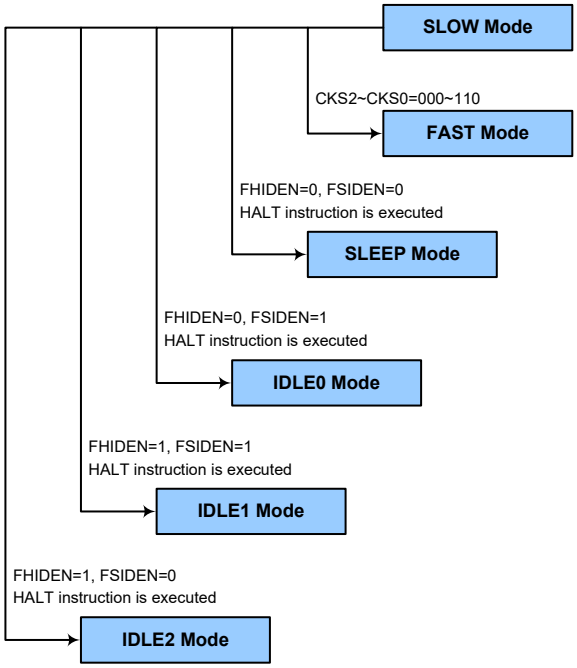
The SLOW Mode system clock is sourced from the LIRC oscillator and therefore requires this oscillator to be stable before full mode switching occurs.




### SLOW Mode to FAST Mode Switching

In the SLOW mode the system clock is derived from $f_{SUB}$. When system clock is switched back to the FAST mode from $f_{SUB}$, the CKS2~CKS0 bits should be set to "000"~"110" and then the system clock will respectively be switched to $f_H$~$f_H$/64.

However, if $f_H$ is not used in the SLOW mode and thus switched off, it will take some time to re-oscillate and stabilise when switching to the FAST mode from the SLOW Mode. This is monitored using the HIRCF bit in the SCC register. The time duration required for the high speed system oscillator stabilisation is specified in the System Start Up Time Characteristics.



### 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 both the FHIDEN and FSIDEN bits in the SCC register equal to "0". In this mode all the clocks and functions will be switched off except the WDT function. 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.
- The Data Memory contents and registers will maintain their present condition.
- 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.
- The WDT will be cleared and resume counting if the WDT function is enabled. If the WDT function is disabled, the WDT will be cleared and then stopped.

### 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 FHIDEN bit in the SCC register equal to "0" and the FSIDEN bit in the SCC register equal to "1". When this instruction is executed under the conditions described above, the following will occur:

- The $f_H$ 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 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.
- The WDT will be cleared and resume counting if the WDT function is enabled. If the WDT function is disabled, the WDT will be cleared and then stopped.

### 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 both the FHIDEN and FSIDEN bits in the SCC register equal to "1". When this instruction is executed under the conditions described above, the following will occur:

- The $f_H$ and $f_{SUB}$ clocks will be on but the application program will stop at the "HALT" instruction.
- The Data Memory contents and registers will maintain their present condition.
- 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.
- The WDT will be cleared and resume counting if the WDT function is enabled. If the WDT function is disabled, the WDT will be cleared and then stopped.

### Entering the IDLE2 Mode

There is only one way for the device to enter the IDLE2 Mode and that is to execute the "HALT" instruction in the application program with the FHIDEN bit in the SCC register equal to "1" and the FSIDEN bit in the SCC register equal to "0". When this instruction is executed under the conditions described above, the following will occur:

- The $f_H$ clock will be on but the $f_{SUB}$ clock will be off and the application program will stop at the "HALT" instruction.
- The Data Memory contents and registers will maintain their present condition.
- 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.
- The WDT will be cleared and resume counting if the WDT function is enabled. If the WDT function is disabled, the WDT will be cleared and then stopped.

### 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 and IDLE2 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 the device which has different package types, as there may be unbonded pins. These must either be set as outputs or if set as inputs must have pull-high resistors connected. In addition, the I/O pin-shared with VPP must not be set to output high, as this could result in increased current consumption.

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

In the IDLE1 and IDLE2 Mode the high speed oscillator is on, if the peripheral function clock source is derived from the high speed oscillator, the additional standby current will also be perhaps in the order of several hundred micro-amps.

### Wake-up

To minimise power consumption the device can enter the SLEEP or any IDLE Mode, where the CPU will be switched off. However, when the device is woken up again, it will take a considerable time for the original system oscillator to restart, stabilise and allow normal operation to resume.

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 I/O Pins
- An external $\overline{RES}$ pin reset
- A system interrupt
- A WDT overflow

When the device executes the "HALT" instruction, it will enter the IDLE or SLEEP mode and the PDF flag will be set to 1. The PDF flag will be cleared to 0 if the device experiences a system power-up or executes the clear Watchdog Timer instruction.

If the system is woken up by an external $\overline{RES}$ pin reset, the device will experience a full system reset, however, If the system is woken up by a WDT overflow, a Watchdog Timer reset will be initiated and the TO flag will be set to 1. 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, other flags remain in their original status.

Each pin can be set using the PAWU or PBWU register to permit a negative transition on the pin to wake-up the system. When 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.

## 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 clock, $f_{LIRC}$ which is sourced from the LIRC oscillator. The LIRC internal oscillator has an approximate frequency of 32kHz and this specified internal clock period can vary with $V_{DD}$, temperature and process variations. The Watchdog Timer source clock is then subdivided by a ratio of $[(2^8-2^0)\sim2^8]\sim[(2^{15}-2^7)\sim2^{15}]$ to give longer timeouts, the actual value being chosen using the WS2~WS0 bits in the WDTC register.

### Watchdog Timer Control Register

A single register, WDTC, controls the required time-out period as well as the enable/disable operation.

- **WDTC Register**

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

Bit 7~4 Unimplemented, read as "0"

Bit 3 **WDTEN**: WDT function enable control
0: Disable
1: Enable

Bit 2~0 **WS2~WS0**: WDT time-out period selection
000: $[(2^8-2^0)\sim2^8]/f_{LIRC}$
001: $[(2^9-2^1)\sim2^9]/f_{LIRC}$
010: $[(2^{10}-2^2)\sim2^{10}]/f_{LIRC}$
011: $[(2^{11}-2^3)\sim2^{11}]/f_{LIRC}$
100: $[(2^{12}-2^4)\sim2^{12}]/f_{LIRC}$
101: $[(2^{13}-2^5)\sim2^{13}]/f_{LIRC}$
110: $[(2^{14}-2^6)\sim2^{14}]/f_{LIRC}$
111: $[(2^{15}-2^7)\sim2^{15}]/f_{LIRC}$
These three bits determine the division ratio of the Watchdog Timer source clock, which in turn determines the timeout period.

• **RSTFC Register**

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

Bit 7~3 Unimplemented, read as "0"

Bit 2 **LVRF**: LVR function reset flag
Refer to the Low Voltage Reset section.

Bit 1 **LRF**: LVR control register software reset flag
Refer to the Low Voltage Reset section.

Bit 0 Unimplemented, read as "0"

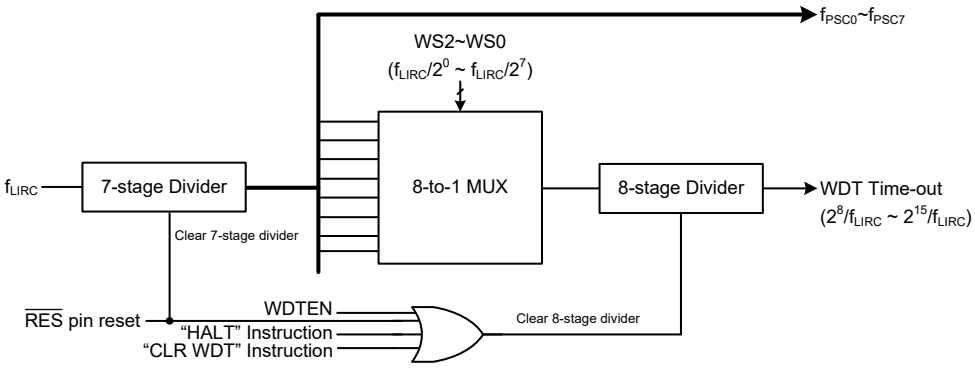
## Watchdog Timer Operation

The Watchdog Timer operates by providing a device reset when its timer overflows. This means that in the application program and during normal operation the user has to strategically clear the Watchdog Timer before it overflows to prevent the Watchdog Timer from executing a reset. This is done using the clear watchdog instruction. If the program malfunctions for whatever reason, jumps to an unknown location, or enters an endless loop, the clear instruction will not be executed in the correct manner, in which case the Watchdog Timer will overflow and reset the device. With regard to the Watchdog Timer enable/disable function, there is one bit, WDTEN, in the WDTC register to offer the enable/disable control.

Under normal program operation, a Watchdog Timer time-out will initialise a device reset and set the status bit TO. However, if the system is in the SLEEP or IDLE Mode, when a Watchdog Timer time-out occurs, the TO bit in the status register will be set and only the Program Counter and Stack Pointer will be reset. Four methods can be adopted to clear the contents of the Watchdog Timer. The first is using the Watchdog Timer software clear instruction, the second is via a HALT instruction. The third is an external hardware reset, which means a low level on the external reset pin. The fourth is that the WDTEN bit is cleared to zero to disable the Watchdog Timer.

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 $2^{15}$ division ratio is selected. As an example, with a 32kHz LIRC oscillator as its source clock, this will give a maximum watchdog period of around 1 second for the $2^{15}$ division ratio, and a minimum timeout of 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 microcontrollers. In this case, internal circuitry will ensure that the microcontrollers, after a short delay, will be in a well-defined state and ready to execute the first program instruction. After this power-on reset, certain important internal registers will be set to defined states before the program commences. One of these registers is the Program Counter, which will be reset to zero forcing the microcontroller to begin program execution from the lowest Program Memory address.

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

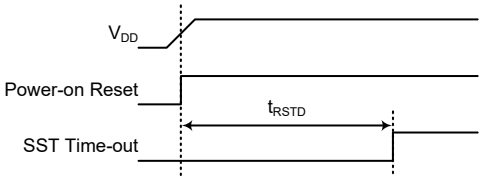
Another reset exists in the form of a Low Voltage Reset, LVR, where a full reset, similar to the $\overline{RES}$ reset is implemented in situations where the power supply voltage falls below a certain threshold. Another type of reset is when the Watchdog Timer overflows and resets the microcontroller. All types of reset operations result in different register conditions being setup.

## Reset Functions

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

### Power-on Reset

The most fundamental and unavoidable reset is the one that occurs after power is first applied to the microcontrollers. 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.



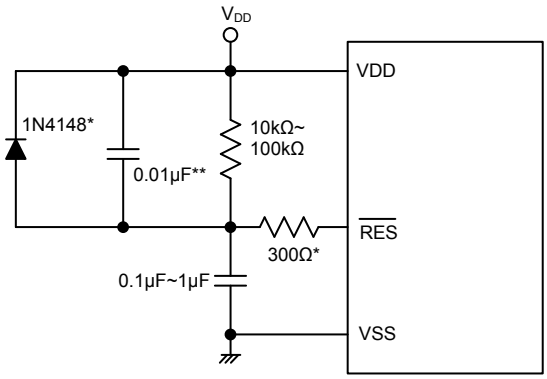

**Power-on Reset Timing Chart**

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

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

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

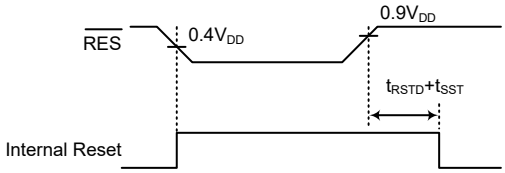
For applications that operate within an environment where more noise is present the Enhanced Reset Circuit shown is recommended.




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

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

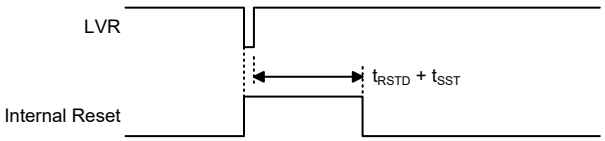
Pulling the $\overline{RES}$ pin low using external hardware will also execute a device reset. In this case, as in the case of other resets, the Program Counter will reset to zero and program execution initiated from this point.




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

### Low Voltage Reset – LVR

The microcontroller contains a low voltage reset circuit in order to monitor the supply voltage of the device and provides an MCU reset should the value fall below a certain predefined level. 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 in battery powered applications, the LVR will automatically reset the device internally and the LVRF bit in the RSTFC register will also be set high. For a valid LVR signal, a low supply voltage, i.e., a voltage in the range between 0.9V~$V_{LVR}$ must exist for a time greater than that specified by $t_{LVR}$ in the LVR Electrical characteristics. If the low supply voltage state does not exceed this value, the LVR will ignore the low supply voltage and will not perform a reset function. If the LVS7~LVS0 bits are set to 01100110B, the LVR function is enabled with a fixed LVR voltage of 1.9V. The actual $V_{LVR}$ value can be selected by the LVS7~LVS0 bits in the LVRC register. If the LVS7~LVS0 bits are changed to some different values by environmental noise, the LVR will reset the device after a delay time, $t_{SRESET}$. When this happens, the LRF bit in the RSTFC register will be set high. After power on the register will have the value of 01100110B. Note that the LVR function will be automatically disabled when the device enters the IDLE or SLEEP mode.




**Low Voltage Reset Timing Chart**

- **LVRC Register**

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

Bit 7~0 **LVS7~LVS0**: LVR Voltage Select control
01100110: 1.9V
01010101: 2.1V
00110011: 3.15V
10011001: 4.2V
11110000: LVR disable
Any other value: Generates MCU reset – register is reset to POR value

When an actual low voltage condition as specified above occurs, an MCU reset will be generated. The reset operation will be activated after the low voltage condition keeps more than a $t_{LVR}$ time. In this situation the register contents will remain the same after such a reset occurs.

Any register value, other than the five defined LVR values above, will also result in the generation of an MCU reset. The reset operation will be activated after a delay time, $t_{SRESET}$. However in this situation the register contents will be reset to the POR value.

- **RSTFC Register**

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

"x": unknown

Bit 7~3 Unimplemented, read as "0"

Bit 2 **LVRF**: LVR function reset flag
0: Not occured
1: Occurred

This bit is set high when an actual Low Voltage Reset situation condition occurs. This bit can only be cleared to zero by the application program.

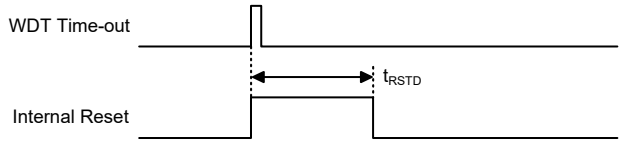
Bit 1 **LRF**: LVR control register software reset flag
0: Not occured
1: Occurred

This bit is set high by the LVR control register containing any undefined LVR voltage register values. This in effect acts like a software reset function. Note that this bit can only be cleared to zero by the application program.

Bit 0 Unimplemented, read as "0"

### Watchdog Time-out Reset during Normal Operation

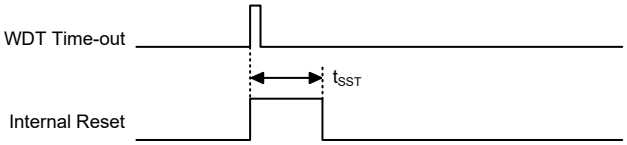
When the Watchdog time-out Reset during normal operations in the FAST or SLOW mode occurs, the Watchdog time-out flag TO will be set to "1".




**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 and PDF flags will be set to "1". Refer to the System Start Up Time Characteristics for $t_{SST}$ details.




**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 | $\overline{\text{RES}}$ or LVR reset during FAST or SLOW Mode operation |
| 1 | u | WDT time-out reset during FAST or SLOW Mode operation |
| 1 | 1 | WDT time-out reset during IDLE or SLEEP Mode operation |

"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, Time Base | Cleared after reset, WDT begins counting |
| Input/Output Ports | I/O ports will be set as inputs |
| Stack Pointer | Stack Pointer will point to the top of the stack |

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

| Register | Reset (Power On) | WDT Time-out (Normal Operation) | $\overline{\text{RES}}$ Reset (Normal Operation) | WDT Time-out (IDLE/SLEEP) |
|---|---|---|---|---|
| IAR0 | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| MP0 | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| ACC | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| PCL | 0000 0000 | 0000 0000 | 0000 0000 | 0000 0000 |
| TBLP | xxxx xxxx | uuuu uuuu | uuuu uuuu | uuuu uuuu |
| TBLH | --xx xxxx | --uu uuuu | --uu uuuu | --uu uuuu |
| STATUS | --00 xxxx | --1u uuuu | --uu uuuu | --11 uuuu |
| WDTC | ---- 1111 | ---- 1111 | ---- 1111 | ---- uuuu |
| TBC | 0000 -000 | 0000 -000 | 0000 -000 | uuuu -uuu |
| RSTFC | ---- -x0- | ---- -uu- | ---- -uu- | ---- -uu- |
| SCC | 111- 0000 | 111- 0000 | 111- 0000 | uuu- uuuu |
| PA | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PAC | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PAPU | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| PAWU | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| INTC | --0- -0-0 | --0- -0-0 | --0- -0-0 | --u- -u-u |
| PB | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PBC | 1111 1111 | 1111 1111 | 1111 1111 | uuuu uuuu |
| PBPU | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| PBWU | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| LVRC | 0110 0110 | 0110 0110 | 0110 0110 | uuuu uuuu |
| TSR0 | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| TSR1 | 100- --00 | 100- --00 | 100- --00 | uuu- --uu |
| CARL0 | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| CARL1 | ---- --00 | ---- --00 | ---- --00 | ---- --uu |
| CARH0 | 0000 0000 | 0000 0000 | 0000 0000 | uuuu uuuu |
| CARH1 | ---- --10 | ---- --10 | ---- --10 | ---- --uu |

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

## Input/Output Ports

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

| Register Name | Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PA | PA7 | PA6 | PA5 | PA4 | PA3 | PA2 | PA1 | PA0 |
| PAC | PAC7 | PAC6 | PAC5 | PAC4 | PAC3 | PAC2 | PAC1 | PAC0 |
| PAPU | PAPU7 | PAPU6 | PAPU5 | PAPU4 | PAPU3 | PAPU2 | PAPU1 | PAPU0 |
| PAWU | PAWU7 | PAWU6 | PAWU5 | PAWU4 | PAWU3 | PAWU2 | PAWU1 | PAWU0 |
| PB | PB7 | PB6 | PB5 | PB4 | PB3 | PB2 | PB1 | PB0 |
| PBC | PBC7 | PBC6 | PBC5 | PBC4 | PBC3 | PBC2 | PBC1 | PBC0 |
| PBPU | PBPU7 | PBPU6 | PBPU5 | PBPU4 | PBPU3 | PBPU2 | PBPU1 | PBPU0 |
| PBWU | PBWU7 | PBWU6 | PBWU5 | PBWU4 | PBWU3 | PBWU2 | PBWU1 | PBWU0 |

**I/O Logic Function Register List**

### 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 a digital input have the capability of being connected to an internal pull-high resistor. These pull-high resistors are selected using the relevant pull-high control registers and are implemented using weak PMOS transistors.

Note that the pull-high resistor can be controlled by the relevant pull-high control register only when the pin-shared functional pin is selected as a digital input or NMOS output. Otherwise, the pull-high resistors cannot be enabled.

- **PxPU Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | PxPU7 | PxPU6 | PxPU5 | PxPU4 | PxPU3 | PxPU2 | PxPU1 | PxPU0 |
| 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 |

**PxPUn**: I/O Port x.n Pin pull-high function control
0: Disable
1: Enable
The PxPUn bit is used to control the pin pull-high function. Here the "x" can be A or B. However, the actual available bits for each I/O Port may be different.

### I/O Pin 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 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 and Port B can be selected individually to have this wake-up feature using the PxWU register.

Note that the wake-up function can be controlled by the wake-up control registers only when the pin is selected as a general purpose input and the MCU enters the IDLE or SLEEP mode.

- **PxWU Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | PxWU7 | PxWU6 | PxWU5 | PxWU4 | PxWU3 | PxWU2 | PxWU1 | PxWU0 |
| 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 |

**PxWUn**: I/O Port x.n Pin wake-up function control
0: Disable
1: Enable
The PxWUn bit is used to control the Px.n pin wake-up function. Here the “x” can be A or B. However, the actual available bits for each I/O Port may be different.

## I/O Port Control Registers

Each I/O port has its own control register known as PAC~PBC, 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 set as a CMOS output. If the pin is currently set as an output, instructions can still be used to read the output register. However, it should be noted that the program will in fact only read the status of the output data latch and not the actual logic status of the output pin.

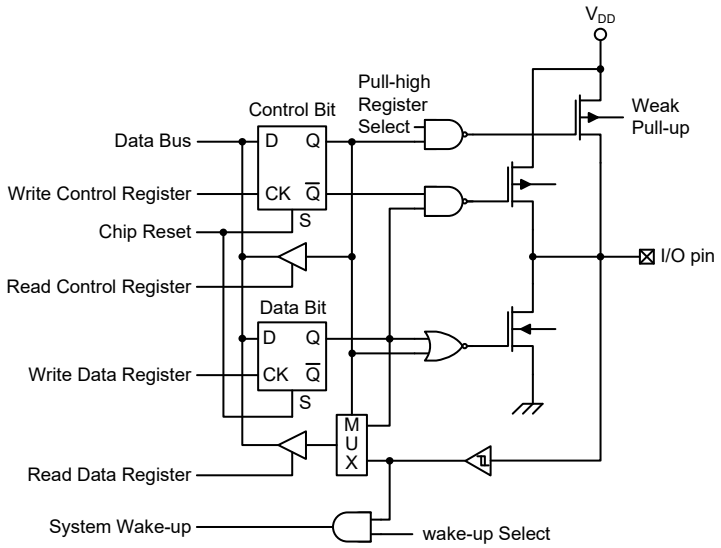
- **PxC Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | PxC7 | PxC6 | PxC5 | PxC4 | PxC3 | PxC2 | PxC1 | PxC0 |
| 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 |

**PxCn**: I/O Px.n Pin type selection
0: Output
1: Input
The PxCn bit is used to control the pin type selection. Here the “x” can be A or B. However, the actual available bits for each I/O Port may be different.

## I/O Pin Structure

The accompanying diagram illustrates the internal structure of the I/O logic function. As the exact logical construction of the I/O pin will differ from this drawing, it is supplied as a guide only to assist with the functional understanding of the I/O logic function.




**Logic Function Input/Output Structure**

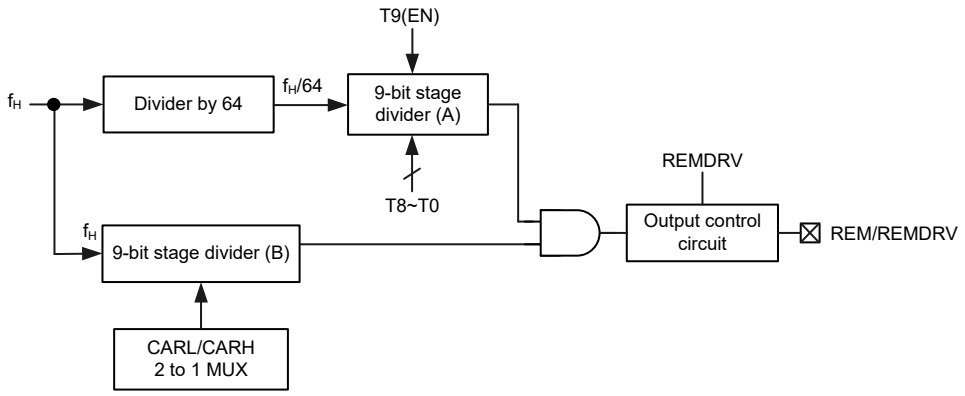
## Programming Considerations

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

Each Pin 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 pins. Single or multiple pins can be set to have this function.

## 9-bit Timer with Carrier Output

The timer is an internal unit for creating a remote control transmission pattern. It consists of a 9-bit count-down counter for timing and two pair of registers which are the CARL1&CARL0, CARH1&CARH0, for the carrier signal low level and high level period control. A dual function pin named REM/REMDRV is provided for the carrier output.




**Timer with Carrier Output Block Diagram**

### Timer and Carrier Output Conctrol Register

The Timer operation and the Carrier output generator functions are controlled by a serial registers. The T8~T0 bits are used to set the 9-bit down counter value, T9 bit is used to enable the timer operation. TOEF is the Timer operation end flag. REMDRV bit is the REM/REMDRV pin function selection bit while the READYB bit is used to indicate whether the REMDRV output function is ready or not. The CARL1&CARL0 register pair is for the carrier output low level period control while the CARH1&CARH0 register pair is for the carrier output high level period control. The bit 1 of the CARH1 register named CH9 is used to start the carrier output.

| Register Name | Bit | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TSR0 | T7 | T6 | T5 | T4 | T3 | T2 | T1 | T0 |
| TSR1 | TOEF | REMDRV | READYB | — | — | — | T9 | T8 |
| CARL0 | CL7 | CL6 | CL5 | CL4 | CL3 | CL2 | CL1 | CL0 |
| CARL1 | — | — | — | — | — | — | CL9 | CL8 |
| CARH0 | CH7 | CH6 | CH5 | CH4 | CH3 | CH2 | CH1 | CH0 |
| CARH1 | — | — | — | — | — | — | CH9 | CH8 |

**Timer and Carrier Output Control Register List**

- **TSR0 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | T7 | T6 | T5 | T4 | T3 | T2 | T1 | T0 |
| 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 9-bit Timer down counter bit 7~0

Writing to TSR0 will only put the written data to the TSR0 register (T7~T0) and writing to the TSR1 (T8) bit will transfer the specified data and contents of TSR0 to the Down Counter. The TOEF bit will be cleared after the data transferred from TSR1 and TSR0 to the Down Counter is completed.

• **TSR1 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | TOEF | REMDRV | READYB | — | — | — | T9 | T8 |
| R/W | R | R/W | R | — | — | — | R/W | R/W |
| POR | 1 | 0 | 0 | — | — | — | 0 | 0 |

Bit 7 **TOEF**: Timer operation end flag
0: Timer operation is in progress
1: Timer operation is ended

Bit 6 **REMDRV**: REM or REMDRV output function selection
0: REMDRV
1: REM

Bit 5 **READYB**: REMDRV output function ready flag, check this bit before output carrier
0: REMDRV is ready for carrier output
1: REMDRV is not ready for carrier output

This bit is used to indicate that whether the REMDRV output driver gets ready to deliver the carrier signal or not. When the REMDRV function is first enabled, including a wake-up from HALT instruction or an output function switch from REM to REMDRV mode, a certain period delay is necessary for the output driver to become stable before the REMDRV carrier signal is sent on the REMDRV pin. Users should make sure that the REMDRV output driver is ready by polling the READYB bit before the Timer is enabled. Note that in REM output function the READYB bit is not available and is always read as 0.

Bit 4~2 Unimplemented, read as “0”

Bit 1 **T9**: Timer enable control
0: Disable
1: Enable

When the T9 bit is set high, the timer will start counting. The timer will stop when its counter is equal to “0” and then TOEF is set equal to “1”. If the T9 bit is cleared to zero during the timer counting, the timer will also be stopped. Once the T9 bit is set from 1→0→1, the down counter will reload data from T8~T0 bits, and then the down counter begins counting down with the new load data.

Bit 0 **T8**: Timer down counter bit 8

Writing to TSR0 will only put the written data to the TSR0 register (T7~T0) and writing to the TSR1 (T8) bit will transfer the specified data and contents of TSR0 to the Down Counter. The TOEF bit will be cleared after the data transferred from TSR1 and TSR0 to the Down Counter is completed.

• **CARL0 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | CL7 | CL6 | CL5 | CL4 | CL3 | CL2 | CL1 | CL0 |
| 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 **CL7~CL0**: Carrier low period control bits 7~0

• **CARL1 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | — | — | — | CL9 | CL8 |
| R/W | — | — | — | — | — | — | R | R/W |
| POR | — | — | — | — | — | — | 0 | 0 |

Bit 7~2 Unimplemented, read as “0”

Bit 1 **CL9**: Fixed to “0”

Bit 0 **CL8**: Carrier low period control bit 8

• **CARH0 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | CH7 | CH6 | CH5 | CH4 | CH3 | CH2 | CH1 | CH0 |
| 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 **CH7~CH0**: Carrier high period control bits 7~0

• **CARH1 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | — | — | — | — | — | — | CH9 | CH8 |
| R/W | — | — | — | — | — | — | R/W | R/W |
| POR | — | — | — | — | — | — | 1 | 0 |

Bit 7~2 Unimplemented, read as "0"

Bit 1 **CH9**: Carrier Output Control
0: With Carrier
1: Without Carrier

Bit 0 **CH8**: Carrier high period control bit 8

## Timer Operation

The timer starts counting down when a value other than "0" is set for the down counter with the timer enable bit T9 set high.

Note that if the content of the Down counter is 000H, set the T9 bit high to start the timer counting, the timer will only count 1 step. The timer output time is $64/f_H$ which is calculated by the formulae:

$$(0+1)\times 64/f_H=64/f_H$$

The down counter is decreased by one in one cycle of 64/fH. If the value of the count-down counter becomes "0", the zero detector generates the timer operation end signal to stop the timer operation. At this time, the TOEF bit will be set to "1". The output of the timer operation end signal is continued while the down counter is "0" and the timer is stopped.

The following relational expression applies between the timer's output time and the down counter's set value of T8~T0 bits value.

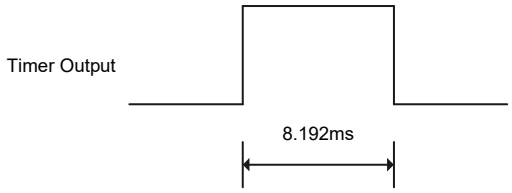
$$\text{Timer output time}=(T[8:0]+1)\times 64/f_H$$

An example is shown below for $f_H$=4MHz

```
MOV A, 0FFH
MOV TSR0, A
MOV A, 01H
MOV TSR1, A
SET TSR1.1
```

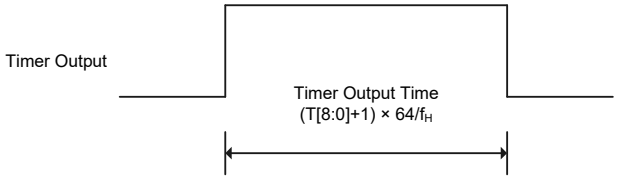
In the case above, the timer output time is as follows.

$$(T[8:0]+1)\times 64/f_H=(511+1)\times 16\mu s=8.192ms$$

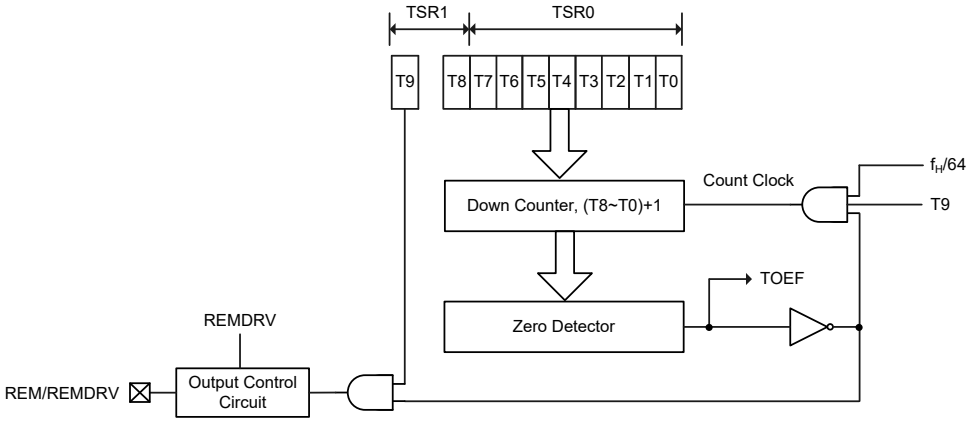



**Timer Output – $f_H$=4MHz, T[8:0]=511**

Setting the T9 bit high channels the timer to the REM pin. The REM pin will be a combination of the timer and carrier signals.



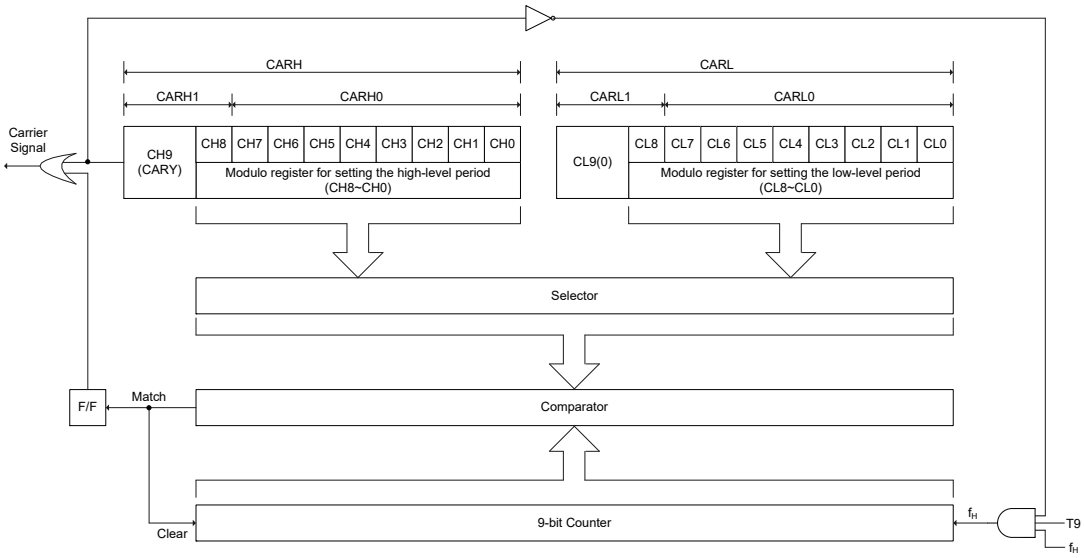

**Timer Output when Carrier is not Output**




**Timer Configuration**

## Carrier Output

The carrier generator consists of a 9-bit counter and two modulo registers which are the CARH1&CARH0 and the CARL1&CARL0 for setting the high-level and low-level periods respectively.




**Remote Controller Carrier Generator Configuration**

Note: 1. The CL9 bit in the CARL1 register is fixed to “0”.
2. The T9 bit is in the TSR1 register which is used to enable the timer output.

### Carrier Periods

The carrier duty ratio and carrier frequency can be determined by setting the high-level and low-level widths using the respective modulo registers. Each of these widths can be set in a range of 500ns to 64μs at $f_H$=4MHz.

The following program gives an example to show how to set the high period and low period of the carrier output.

Ensure input values in the range of 001H to 1FFH to CARL and CARH.

**Example**:

```
MOV A, xxH            ; xxH=00H~FFH
MOV CARL0,A
MOV A, xxH            ; xxH 01H, CL8 (CARL1.0)
MOV CARL1,A
MOV A, xxH            ; xxH=00H~FFH
MOV CARH0,A
MOV A, xxH            ; xxH 02H, CH8 (CARH1.0)
MOV CARH1,A
CLR CARH1.1           ; The carrier is started by clearing CARY(CARH1.1)="0"
```

The values of CARH and CARL can be calculated from the following expressions.

CARL (CARL1.0, CARL0.7~CARL0.0)=($f_H$×(1 - D)×T) - 1……(1)

CARH (CARH1.0, CARH0.7~CARH0.0)=($f_H$×D×T) - 1…….(2)

(1)+(2)⇒CARL+CARH=($f_H$×T) - 2⇒Actual Carrier Frequency=$f_H$/(CARL+CARH+2)

Where D: Carrier duty ratio (0<D<1)

$f_H$: Input clock (4MHz)

T: Carrier cycle (μs)

**Example**:

If $f_H$=4MHz, Target $f_C$=38kHz, T=1/$f_C$=26.3157μs=$t_L$+$t_H$, duty=1/3

CARL=(4.00MHz×(1 - 1/3)×26.3157μs) - 1=69.1752

select 69, xxH=45H, actual $t_L$=(69+1)/4.00MHz=17.5μs

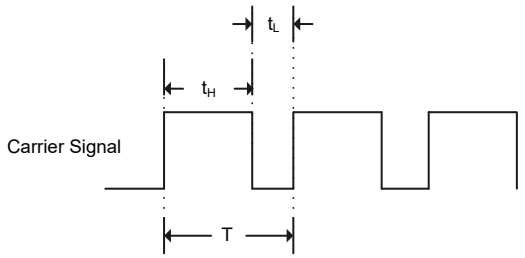
CARH=(4.00MHz×1/3×26.3157μs) - 1=34.087

select 34, xxH=22H , actual $t_H$=(34+1)/4.00MHz=8.75μs

For actual Carrier Frequency=$f_H$/(CARL+CARH+2)

So, actual $f_C$=$f_H$/(CARL+CARH+2)=4000kHz/(69+34+2)=38.09kHz

```
MOV A,045H
MOV CARL0,A
MOV A,022H
MOV CARH0,A
CLR CARH1.1           ; The carrier is started by clearing CARY(CARH1.1)="0"
```




**Carrier High and Low Periods**

| Target | | Setting | | Actual | | | |
|---|---|---|---|---|---|---|---|
| $f_C$(kHz) | Duty | CARH (CH[8:0] bits) | CARL (CL[8:0] bits) | $t_H$(μs) | $t_L$(μs) | T(μs) | $f_C$(kHz) |
| 36 | 1/3 | 24H | 49H | 9.25 | 18.50 | 27.75 | 36.04 |
| 38 | 1/3 | 22H | 45H | 8.75 | 17.50 | 26.25 | 38.10 |
| 56 | 1/3 | 17H | 2EH | 6.00 | 11.75 | 17.75 | 56.34 |
| 56 | 1/2 | 23H | 22H | 9.00 | 8.75 | 17.75 | 56.34 |

**Carrier Frequency Setting ($f_H$=4MHz)**

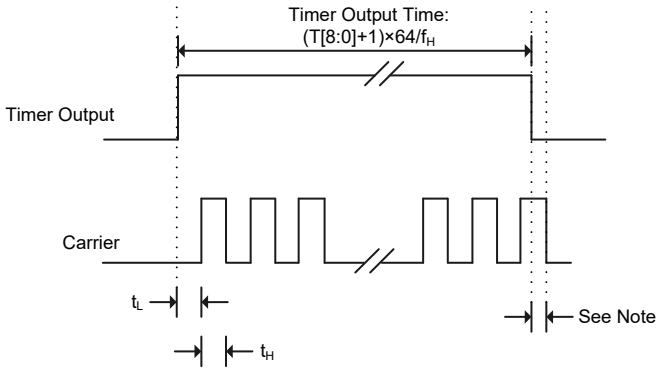
### Carrier Output Generator

The remote controller carrier can be output from the REM pin by clearing the CH9 (CARY) bit in the CARH1 register to zero for setting the high-level period (CARH).

When performing a carrier output, be sure to set the timer operation enabled after setting the CARH (CH[8:0] bits) and CARL (CL[8:0] bits) values.

Note that a malfunction may occur if the values of CARH and CARL are changed while the carrier is being output on the REM pin.

Enabling the timer starts the carrier output from the low level.



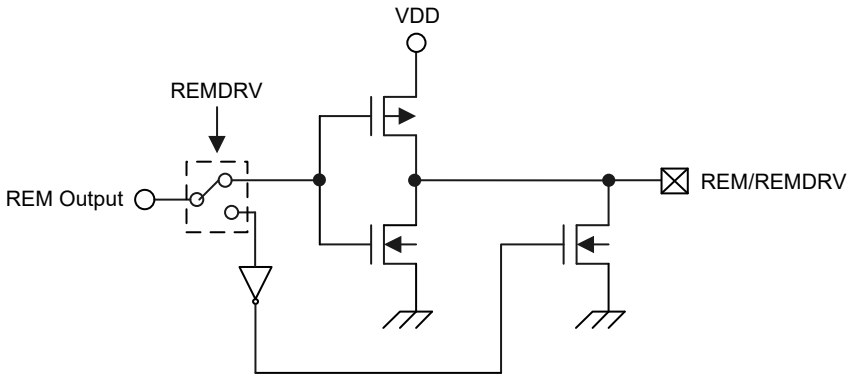

**Timer Output With Carrier Output**

Note: When the carrier signal is active and during the time when the signal is high, if the timer output should go low, the carrier signal will first complete its high level period before going low.

## Carrier Output Pins

There is a dual function remote controller carrier output pin named REM/REMDRV. The selection of REM or REMDRV is determined by the REMDRV bit in the TSR1 register. After a reset, the REM carrier output pin will have a low level while the REMDRV carrier output pin will be in a floating condition.

The generic structures of the REM or REMDRV function are illustrated in the accompanying diagram. As the exact construction of the carrier output pin will differ from these drawings, they are supplied as a guide only to assist with the functional understanding of the remote carrier output pins.




**REM/REMDRV Pin Structure**

The output from the REM pin is in accordance with the value of CH9 (CARY) bit in the CARH1 register and the timer output enable bit T9 in the TSR1 register, and the value of the timer 9-bit down counter T[8:0].

| CH9 Bit (CARY) | T9 Bit | T[8:0] Bits | REM Function (CMOS Output) | REMDRV Function (NMOS Output) |
|---|---|---|---|---|
| 0 | 0 | 0 | Low-level output | Floating output |
| 0 | 0 | Other than 0 | | |
| 0 | 1 | 0 | $64/f_H$ (with carrier output) | $64/f_H$ (with carrier output) |
| 0 | 1 | Other than 0 | Carrier output (Note) | Carrier output |
| 1 | 0 | — | Low-level output | Floating point |
| 1 | 1 | — | High-level output | Low-level point |

**REM Pin Output Control**

Note: The values of the CARH (CH[8:0]) and CARL (CL[8:0]) must be set while the REM pin is at a low level (T9=0 or T[8:0]=0).

# Interrupts

Interrupts are an important part of any microcontroller system. When an external event or an internal function such as a Timer requires microcontroller attention, its corresponding interrupt will enforce a temporary suspension of the main program allowing the microcontroller to direct attention to its needs. The device only contains one internal interrupt function which is generated by the Time Base.

## Interrupt Register

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 the INTC register, located in the Special Purpose Data Memory.

The register contains two enable bits to enable or disable the global interrupt and the Time base interrupt as well as an interrupt flag to indicate the presence of a time base interrupt request.

- **INTC Register**

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

Bit 7~6 Unimplemented, read as “0”

Bit 5 **TBF**: Time Base interrupt request flag
0: No request
1: Interrupt request

Bit 4~3 Unimplemented, read as “0”

Bit 2 **TBE**: Time Base interrupt control
0: Disable
1: Enable

Bit 1 Unimplemented, read as “0”

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

## Interrupt Operation

When the conditions for an interrupt event occur, 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.

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.

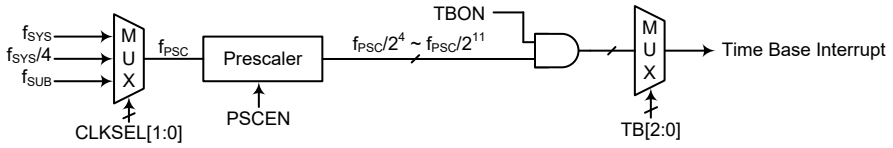
If an interrupt requires immediate servicing while the program is already in another interrupt service routine, the EMI bit should be set after entering the routine, to allow interrupt nesting. If the stack is full, the interrupt request will not be acknowledged, even if the related interrupt is enabled, until the Stack Pointer is decremented. If immediate service is desired, the stack must be prevented from becoming full. 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.

## Time Base Interrupt

The Time Base Interrupt vector is located at 008H in the program memory.

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 signals from the timer function. When this happens its interrupt request flag 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 interrupt 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, $f_{PSC}$, originates from the internal clock source $f_{SYS}$, $f_{SYS}/4$ or $f_{SUB}$ and then 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.




**Time Base Interrupt**

- **TBC Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Name | TBON | PSCEN | CLKSEL1 | CLKSEL0 | — | TB2 | TB1 | TB0 |
| 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 **TBON**: Time Base control
0: Disable
1: Enable

Bit 6 **PSCEN**: Prescaler clock enable control
0: Disable
1: Enable
The PSCEN bit is the Prescaler clock enable or disable control bit. When the Prescale clock is disabled, it can reduce extra power consumption

Bit 5~4 **CLKSEL1~CLKSEL0**: Prescaler clock source $f_{PSC}$ selection
00: $f_{SYS}$
01: $f_{SYS}/4$
1x: $f_{SUB}$

Bit 3 Unimplemented, read as "0"

Bit 2~0 **TB2~TB0**: Time Base Time-out period selection
000: $2^4/f_{PSC}$
001: $2^5/f_{PSC}$
010: $2^6/f_{PSC}$
011: $2^7/f_{PSC}$
100: $2^8/f_{PSC}$
101: $2^9/f_{PSC}$
110: $2^{10}/f_{PSC}$
111: $2^{11}/f_{PSC}$

## 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. 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 the 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.

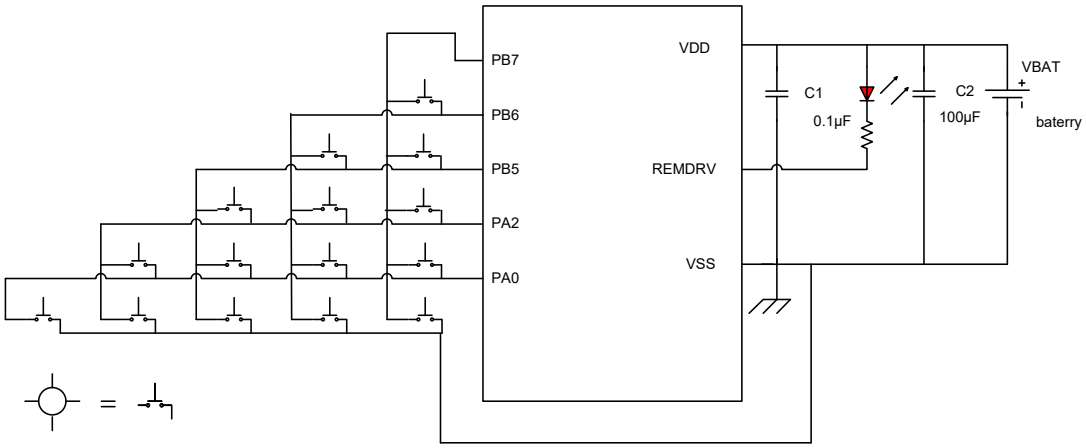
To return from an interrupt subroutine, either a RET or RETI instruction may be executed. The RETI instruction in addition to executing a return to the main program also automatically sets the EMI bit high to allow further interrupts. The RET instruction however only executes a return to the main program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

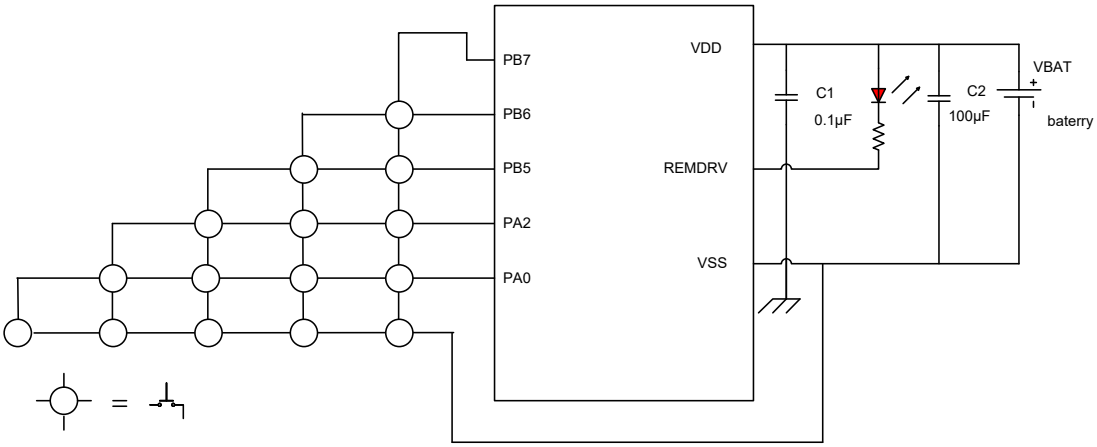
## Configuration Options

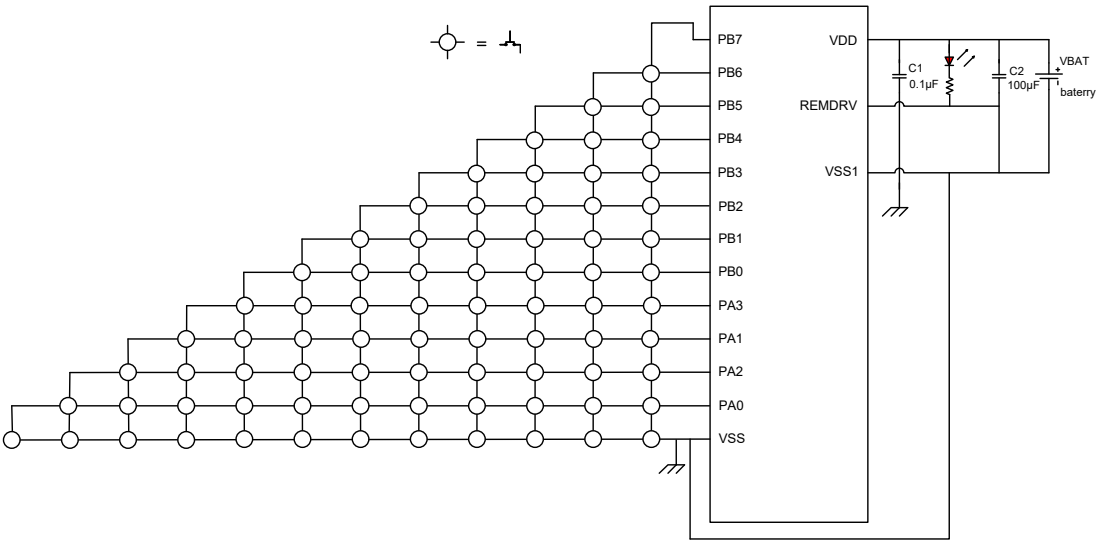
Configuration options refer to certain options within the MCU that are programmed into the device during the programming process. During the development process, these options are selected using the IDE software development tools. As these options are programmed into the device using the hardware programming tools, once they are selected they cannot be changed later using the application program.All options must be defined for proper system function, the details of which are shown in the table.

| No. | Options |
|---|---|
| **I/O Pin-Shared Options** | |
| 1 | $\overline{\text{RES}}$ pin reset function selection:<br>0: Always I/O or other function<br>1: $\overline{\text{RES}}$ pin |
| **HIRC Options** | |
| 2 | HIRC frequency & IR carrier frequency selection:<br>IR carrier frequency=40kHz ($f_{HIRC}$=4.000MHz – default value)<br>IR carrier frequency=36kHz ($f_{HIRC}$=3.996MHz)<br>IR carrier frequency=56kHz ($f_{HIRC}$=3.976MHz)<br>IR carrier frequency=38kHz ($f_{HIRC}$=3.990MHz) |

## 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 the 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 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 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 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, all microcontrollers allow an area of Program Memory to be set as a table where data can be directly stored. A set of easy to use instructions provides the means by which this fixed data can be referenced and retrieved from the Program Memory.

### Other Operations

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

## Instruction Set Summary

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

### Table Conventions

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

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

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

Note: 1. For skip instructions, if the result of the comparison involves a skip then two cycles are required, if no skip takes place only one cycle is required.
2. Any instruction which changes the contents of the PCL will also require 2 cycles for execution.

## 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<br>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<br>TO ← 0<br>PDF ← 0 |
| Affected flag(s) | TO, PDF |
| **CPL [m]** | Complement Data Memory |
| Description | Each bit of the specified Data Memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice versa. |
| Operation | [m] ← $\overline{[m]}$ |
| Affected flag(s) | Z |
| **CPLA [m]** | Complement Data Memory with result in ACC |
| Description | Each bit of the specified Data Memory is logically complemented (1′s complement). Bits which previously contained a 1 are changed to 0 and vice versa. The complemented result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | ACC ← $\overline{[m]}$ |
| Affected flag(s) | Z |
| **DAA [m]** | Decimal-Adjust ACC for addition with result in Data Memory |
| Description | Convert the contents of the Accumulator value to a BCD (Binary Coded Decimal) value resulting from the previous addition of two BCD variables. If the low nibble is greater than 9 or if AC flag is set, then a value of 6 will be added to the low nibble. Otherwise the low nibble remains unchanged. If the high nibble is greater than 9 or if the C flag is set, then a value of 6 will be added to the high nibble. Essentially, the decimal conversion is performed by adding 00H, 06H, 60H or 66H depending on the Accumulator and flag conditions. Only the C flag may be affected by this instruction which indicates that if the original BCD sum is greater than 100, it allows multiple precision decimal addition. |
| Operation | [m] ← ACC + 00H or<br>[m] ← ACC + 06H or<br>[m] ← ACC + 60H or<br>[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<br>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<br>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<br>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)<br>[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)<br>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)<br>[m].0 ← C<br>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)<br>ACC.0 ← C<br>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)<br>[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 is rotated right by 1 bit with bit 0 rotated into bit 7.<br>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)<br>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)<br>[m].7 ← C<br>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)<br>ACC.7 ← C<br>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] − $\overline{C}$ |
| Affected flag(s) | OV, Z, AC, C |

| | |
|---|---|
| **SBCM A,[m]** | Subtract Data Memory from ACC with Carry and result in Data Memory |
| Description | The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | [m] ← ACC − [m] − $\overline{C}$ |
| Affected flag(s) | OV, Z, AC, C |

| | |
|---|---|
| **SDZ [m]** | Skip if decrement Data Memory is 0 |
| Description | The contents of the specified Data Memory are first decremented by 1. If the result is 0 the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | [m] ← [m] − 1<br>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<br>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<br>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<br>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<br>ACC.7~ACC.4 ← [m].3~[m].0 |
| Affected flag(s) | None |
| **SZ [m]** | Skip if Data Memory is 0 |
| Description | The contents of the specified Data Memory are read out and then written to the specified Data Memory again. 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]<br>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)<br>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)<br>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 |

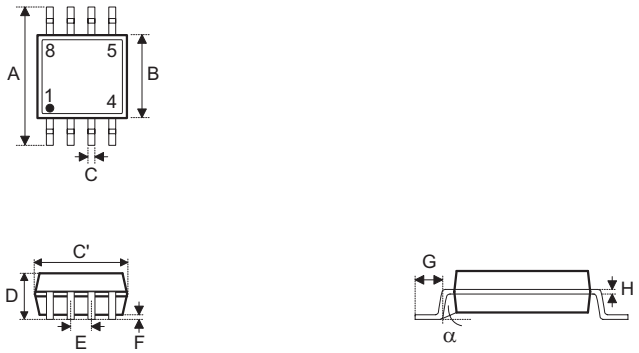
## Package Information

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 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

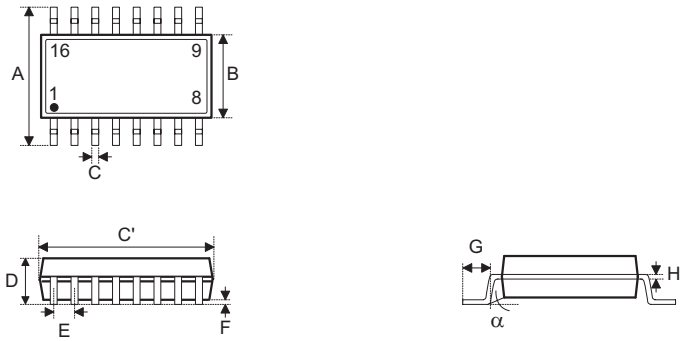
## 8-pin SOP (150mil) Outline Dimensions




| Symbol | Dimensions in inch | | |
|---|---|---|---|
| | Min. | Nom. | Max. |
| A | | 0.236 BSC | |
| B | | 0.154 BSC | |
| C | 0.012 | — | 0.020 |
| C′ | | 0.193 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 | | 6.00 BSC | |
| B | | 3.90 BSC | |
| C | 0.31 | — | 0.51 |
| C′ | | 4.90 BSC | |
| D | — | — | 1.75 |
| E | | 1.27 BSC | |
| F | 0.10 | — | 0.25 |
| G | 0.40 | — | 1.27 |
| H | 0.10 | — | 0.25 |
| α | 0° | — | 8° |

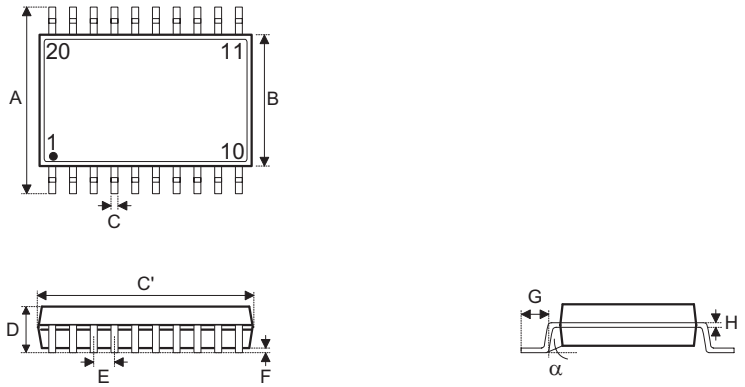
### 16-pin NSOP (150mil) Outline Dimensions




| 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 | | 6.00 BSC | |
| B | | 3.90 BSC | |
| C | 0.31 | — | 0.51 |
| C’ | | 9.90 BSC | |
| D | — | — | 1.75 |
| E | | 1.27 BSC | |
| F | 0.10 | — | 0.25 |
| G | 0.40 | — | 1.27 |
| H | 0.10 | — | 0.25 |
| α | 0° | — | 8° |

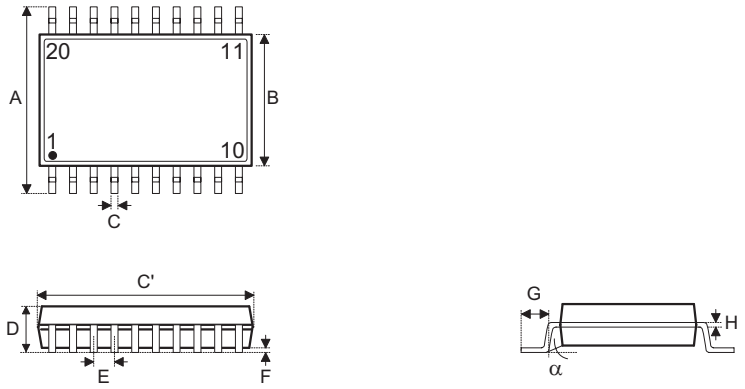
## 20-pin NSOP (150mil) Outline Dimensions




| 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 | | 0.154 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 | | 6.00 BSC | |
| B | | 3.90 BSC | |
| C | 0.20 | — | 0.30 |
| C’ | | 8.66 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° |

Copyright© 2026 by XINQUN SEMICONDUCTOR (XIAMEN) INC. All Rights Reserved.

The information provided in this document has been produced with reasonable care and attention before publication, however, XINQUN does not guarantee that the information is completely accurate. The information contained in this publication is provided for reference only and may be superseded by updates. XINQUN disclaims any expressed, implied or statutory warranties, including but not limited to suitability for commercialization, satisfactory quality, specifications, characteristics, functions, fitness for a particular purpose, and non-infringement of any third-party's rights. XINQUN disclaims all liability arising from the information and its application. In addition, XINQUN does not recommend the use of XINQUN's products where there is a risk of personal hazard due to malfunction or other reasons. XINQUN hereby declares that it does not authorise the use of these products in life-saving, life-sustaining or safety critical components. Any use of XINQUN's products in life-saving/sustaining or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold XINQUN harmless from any damages, claims, suits, or expenses resulting from such use. The information provided in this document, including but not limited to the content, data, examples, materials, graphs, and trademarks, is the intellectual property of XINQUN (and its licensors, where applicable) and is protected by copyright law and other intellectual property laws. No license, express or implied, to any intellectual property right, is granted by XINQUN herein. XINQUN reserves the right to revise the information described in the document at any time without prior notice. For the latest information, please contact us.