

Disclaimer

1. The Customer hereby acknowledges and agrees that all product documents (including but not limited to data sheet, user guide, user manual) supplied by HOLTEK SEMICONDUCTOR INC. (hereinafter referred to as “HOLTEK”) are the proprietary and confidential intellectual property of HOLTEK and are protected by copyright law and other intellectual property laws.
2. The term “edition or modification” as used herein shall mean any of the following conditions:
 - (a) For product documents that HOLTEK has already released original versions: the edition or modification which is made by HOLTEK, the customer, or any third party appointed by the customer according to the customer’s instructions and requests, including but not limited to the edition or modification of the format or any part of the content of released documents.
 - (b) For product documents that HOLTEK has not yet released original versions or that simplified versions provided to customers during the development of products: the edition or modification which is made by HOLTEK, the customer, or any third party appointed by the customer according to customer's instructions and requests, including but not limited to the edition or modification of the format or any part of the content of unreleased and/or simplified documents.
3. The customer hereby acknowledges and agrees that product documents provided by HOLTEK, including but not limited to those edited or modified according to the customer's instructions and requests mentioned above, are provided “as-is” at the time of delivery. After delivery of product documents, the customer shall use them at its own risk. HOLTEK disclaims any expressed, implied or statutory warranties, including suitability for commercialisation, satisfactory quality, specifications, characteristics, functions, fitness for a particular purpose, and non-infringement of any third-party’s rights. If the content of the product documents (including but not limited to any inconsistency between it and the products regarding specifications, features and functions) causes losses to the customer or any third party, HOLTEK will not be responsible for any compensation and liabilities.

Principal Office: No.3, Creation Road. II, SBIP Hsinchu, 30077, Taiwan, R.O.C.

Tel: +886-3-563-1999

Fax: +886-3-563-1189

Taipei Office: 4F-2, No.3-2 BLD H, St.Yuan-Qu, NKSP Taipei, 11503, Taiwan R.O.C.

Tel: +886-2-2655-7070

Fax: +886-2-2655-7383

Printed at: October 16, 2024



Continuous Glucose Monitoring Flash MCU

BH66F2455

Revision: V0.00 Date: October 16, 2024

www.holtek.com

Table of Contents

| | |
|--|-----------|
| Features | 7 |
| CPU Features | 7 |
| Peripheral Features..... | 7 |
| General Description..... | 8 |
| Block Diagram..... | 9 |
| Pin Assignment..... | 9 |
| Pin Descriptions | 10 |
| Absolute Maximum Ratings..... | 12 |
| D.C. Characteristics..... | 13 |
| Operating Voltage Characteristics..... | 13 |
| Operating Current Characteristics..... | 13 |
| Standby Current Characteristics | 14 |
| A.C. Characteristics..... | 14 |
| Internal High Speed Oscillator – HIRC – Frequency Accuracy | 14 |
| Internal Medium Speed Oscillator Characteristics – MIRC | 15 |
| Internal Low Speed Oscillator Characteristics – LIRC | 15 |
| Operating Frequency Characteristic Curves | 15 |
| System Start Up Time Characteristics..... | 16 |
| Input/Output Characteristics | 16 |
| Memory Characteristics | 17 |
| LVR Electrical Characteristics | 18 |
| Analog Front End Circuit Characteristics | 18 |
| Operational Voltage Electrical Characteristics | 18 |
| Operational Amplifier Electrical Characteristics | 18 |
| Internal Reference Voltage Characteristics..... | 19 |
| 12-bit D/A Converter Electrical Characteristics | 19 |
| A/D Converter Electrical Characteristics..... | 19 |
| PGA Electrical Characteristics | 20 |
| Power-on Reset Characteristics..... | 20 |
| System Architecture | 21 |
| Clocking and Pipelining..... | 21 |
| Program Counter..... | 22 |
| Stack | 23 |
| Arithmetic and Logic Unit – ALU | 24 |
| Flash Program Memory..... | 25 |
| Structure..... | 25 |
| Special Vectors | 26 |
| Look-up Table..... | 26 |
| Table Program Example..... | 26 |
| Unique Identifier – UID..... | 27 |

| | |
|--|-----------|
| In Circuit Programming – ICP | 28 |
| On-Chip Debug Support – OCDS | 29 |
| In Application Programming – IAP | 29 |
| Data Memory | 45 |
| Structure..... | 45 |
| Data Memory Addressing..... | 46 |
| General Purpose Data Memory | 46 |
| Special Purpose Data Memory | 46 |
| Special Function Register Description..... | 48 |
| Indirect Addressing Registers – IAR0, IAR1, IAR2 | 48 |
| Memory Pointers – MP0, MP1H/MP1L, MP2H/MP2L..... | 48 |
| Accumulator – ACC..... | 49 |
| Program Counter Low Byte Register – PCL..... | 50 |
| Look-up Table Registers – TBLP, TBHP, TBLH..... | 50 |
| Option Memory Mapping Register – ORMC | 50 |
| Status Register – STATUS..... | 51 |
| EEPROM Data Memory..... | 53 |
| EEPROM Data Memory Structure | 53 |
| EEPROM Registers | 53 |
| Read Operation from the EEPROM..... | 55 |
| Page Erase Operation to the EEPROM..... | 56 |
| Write Operation to the EEPROM | 56 |
| Write Protection..... | 58 |
| EEPROM Interrupt | 58 |
| Programming Considerations..... | 58 |
| Oscillators | 61 |
| Oscillator Overview | 61 |
| System Clock Configurations..... | 61 |
| Internal High Speed RC Oscillator – HIRC | 62 |
| Internal Medium Speed RC Oscillator – MIRC..... | 62 |
| Internal 32.768kHz Oscillator – LIRC..... | 62 |
| Operating Modes and System Clocks | 63 |
| System Clocks | 63 |
| System Operation Modes..... | 63 |
| Control Registers | 65 |
| Operating Mode Switching..... | 66 |
| Standby Current Considerations..... | 70 |
| Wake-up | 70 |
| Watchdog Timer..... | 71 |
| Watchdog Timer Clock Source..... | 71 |
| Watchdog Timer Control Register | 71 |
| Watchdog Timer Operation | 72 |

| | |
|---|------------|
| Reset and Initialisation..... | 73 |
| Reset Functions | 73 |
| Reset Initial Conditions | 76 |
| Input/Output Ports | 80 |
| Pull-high Resistors | 80 |
| Port A Wake-up | 80 |
| I/O Port Control Registers | 81 |
| Pin-shared Functions | 81 |
| I/O Pin Structures..... | 84 |
| READ PORT function..... | 84 |
| Programming Considerations..... | 85 |
| Timer Modules – TM | 86 |
| Introduction | 86 |
| TM Operation | 86 |
| TM Clock Source..... | 87 |
| TM Interrupts..... | 87 |
| TM External Pins..... | 87 |
| Programming Considerations..... | 88 |
| Compact Type TM – CTM | 89 |
| Compact Type TM Operation | 89 |
| Compact Type TM Register Description..... | 89 |
| Compact Type TM Operating Modes | 93 |
| Periodic Type TM – PTM..... | 99 |
| Periodic TM Operation | 99 |
| Periodic Type TM Register Description | 99 |
| Periodic Type TM Operation Modes..... | 103 |
| Internal Reference Voltage Generator | 112 |
| Internal Reference Voltage Register Description | 112 |
| Digital to Analog Converter – DAC..... | 113 |
| D/A Converter Register Description | 113 |
| Operational Amplifiers | 115 |
| Operational Amplifier Register Description | 115 |
| Analog to Digital Converter – ADC..... | 117 |
| A/D Converter Overview | 117 |
| A/D Converter Register Description | 118 |
| CTRL Unit..... | 125 |
| A/D Data Accumulation and Average | 126 |
| A/D Converter Auto Mode | 127 |
| A/D Converter Data Rate Definition | 129 |
| A/D Converter Operation..... | 130 |
| Summary of A/D Conversion Steps..... | 130 |
| Programming Considerations..... | 131 |
| A/D Converter Transfer Function | 131 |

| | |
|---|------------|
| A/D Converted Data | 132 |
| A/D Converted Data to Voltage | 132 |
| Serial Peripheral Interface – SPI..... | 133 |
| SPI Interface Operation..... | 133 |
| SPI Registers | 134 |
| SPI Communication | 136 |
| SPI Bus Enable/Disable | 138 |
| SPI Operation..... | 138 |
| Error Detection | 140 |
| UART Interface | 141 |
| UART External Pins | 142 |
| UART Single Wire Mode | 142 |
| UART Data Transfer Scheme..... | 142 |
| UART Status and Control Registers..... | 143 |
| Baud Rate Generator..... | 150 |
| UART Setup and Control..... | 151 |
| UART Transmitter..... | 153 |
| UART Receiver | 154 |
| Managing Receiver Errors | 156 |
| UART Interrupt Structure..... | 157 |
| UART Power Down and Wake-up..... | 159 |
| Cyclic Redundancy Check – CRC | 159 |
| CRC Registers | 159 |
| CRC Operation..... | 160 |
| Interrupts | 162 |
| Interrupt Registers..... | 163 |
| Interrupt Operation | 168 |
| External Interrupts..... | 168 |
| A/D Converter Interrupt | 169 |
| A/D Converter Auto Mode Interrupt..... | 169 |
| A/D Converter Average Interrupt..... | 169 |
| Multi-function Interrupts..... | 169 |
| Timer Module Interrupts | 170 |
| EEPROM Interrupt | 170 |
| Time Base Interrupts | 170 |
| SPI Interrupt | 172 |
| UART Interrupt | 172 |
| Interrupt Wake-up Function..... | 172 |
| Programming Considerations..... | 173 |
| Application Circuits | 174 |
| Instruction Set..... | 175 |
| Introduction | 175 |
| Instruction Timing | 175 |
| Moving and Transferring Data | 175 |

| | |
|---|------------|
| Arithmetic Operations..... | 175 |
| Logical and Rotate Operation | 176 |
| Branches and Control Transfer | 176 |
| Bit Operations | 176 |
| Table Read Operations | 176 |
| Other Operations..... | 176 |
| Instruction Set Summary | 177 |
| Table Conventions..... | 177 |
| Extended Instruction Set..... | 179 |
| Instruction Definition..... | 181 |
| Extended Instruction Definition | 191 |
| Package Information | 200 |
| SAW Type 16-pin QFN (3mm×3mm×0.75mm, FP0.25mm) Outline Dimensions | 201 |
| SAW Type 24-pin QFN (3mm×3mm×0.55mm) Outline Dimensions | 202 |

Features

CPU Features

- Operating Voltage
 - ♦ MIRC
 - $f_{SYS}=400\text{kHz}$: 2.2V~5.5V
 - ♦ HIRC
 - $f_{SYS}=4\text{MHz}$: 2.2V~5.5V
- Up to 1 μs instruction cycle with 4MHz system clock at $V_{DD}=5\text{V}$
- Power down and wake-up functions to reduce power consumption
- Oscillator Types
 - ♦ Internal High Speed 4MHz RC – HIRC
 - ♦ Internal Medium Speed 400kHz RC – MIRC
 - ♦ Internal Low Speed 32.768kHz RC – LIRC
- Fully integrated internal oscillators require no external components
- Multi-mode operation: FAST, SLOW, IDLE and SLEEP
- All instructions executed in 1~3 instruction cycles
- Table read instructions
- 115 powerful instructions
- 16-level subroutine nesting
- Bit manipulation instruction

Peripheral Features

- Flash Program Memory: 8K \times 16
- Data Memory: 512 \times 8
- True EEPROM Memory: 512 \times 8
- In Application Programming function – IAP
- 128-bit unique ID
- Watchdog Timer function
- 9 bidirectional I/O lines
- 4 external interrupt lines shared with I/O pins
- Two Timer Modules for time measure, input capture, compare match output, PWM output function or single pulse output function
- Dual Time Base functions for generation of fixed time interrupt signals
- Glucose Meter AFE circuit
 - ♦ 2 external channel 24-bit resolution Delta Sigma A/D Converter
 - ♦ Internal Reference Voltage Generator
 - ♦ Three 12-bit D/A Converters
 - ♦ 3 internal Operational Amplifiers
- Fully-duplex / Half-duplex Universal Asynchronous Receiver and Transmitter Interface – UART
- Single Serial Peripheral Interface – SPI

- Integrated 16-bit Cyclic Redundancy Check function – CRC
- Low voltage reset function
- Package types: 16/24-pin QFN

General Description

The device is a Flash Memory 8-bit high performance RISC architecture microcontroller device with Continuous Glucose Monitoring (CGM) AFE module that specifically designed for CGM applications.

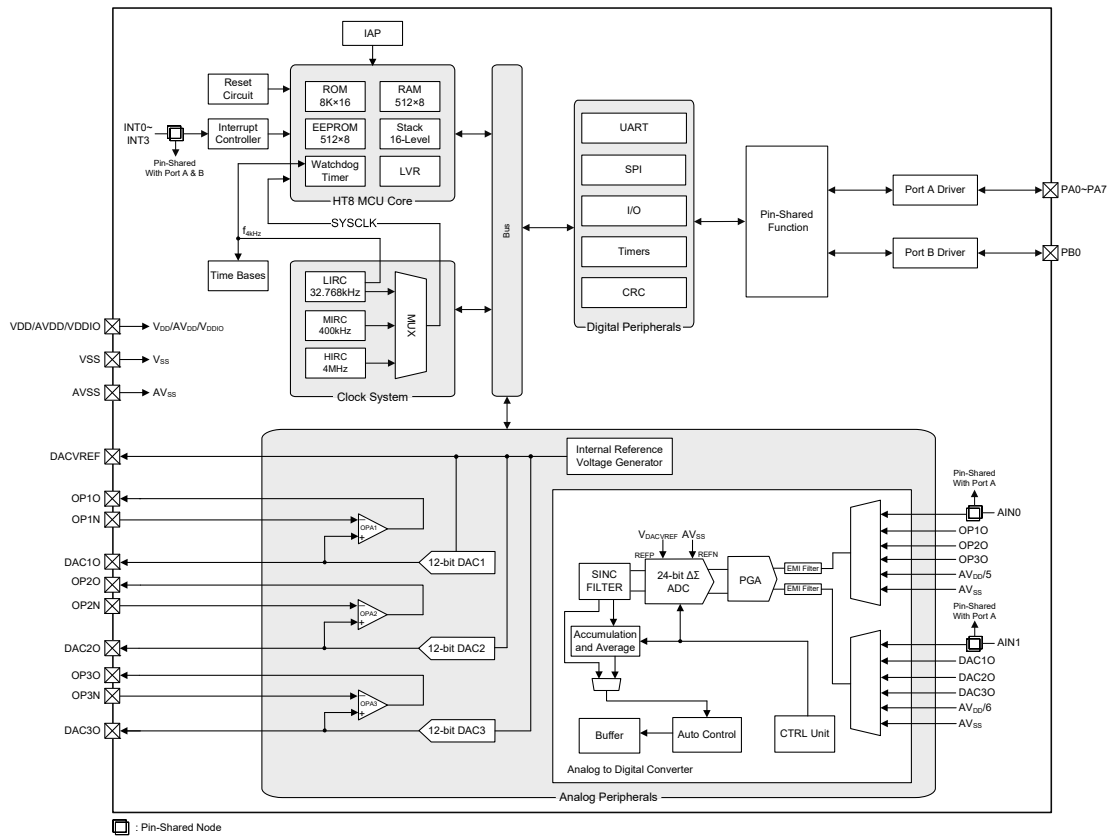
For memory features, the Flash Memory offers users the convenience of Flash Memory multi-programming features. Other memory includes an area of RAM Data Memory as well as an area of true EEPROM memory for storage of non-volatile data such as serial numbers, calibration data etc. By using the In Application Programming technology, users have a convenient means to directly store their measured data in the Flash Program Memory as well as having the ability to easily update their application programs.

Analog features include a multi-channel 24-bit Delta Sigma A/D converter, three 12-bit D/A converters and three internal operational amplifiers. Multiple and extremely flexible Timer Modules provide timing, pulse generation and PWM generation functions. Communication with the outside world is catered for by including fully integrated SPI and UART interface functions, two popular interfaces which provide designers with a means of easy communication with external peripheral hardware. 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.

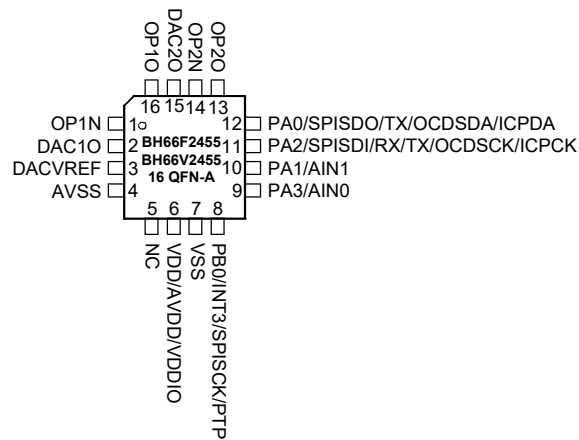
A full choice of internal high and low oscillator functions are provided including three fully integrated system oscillators which require no external components for their implementation. The ability to operate and switch dynamically between a range of operating modes using different clock sources gives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

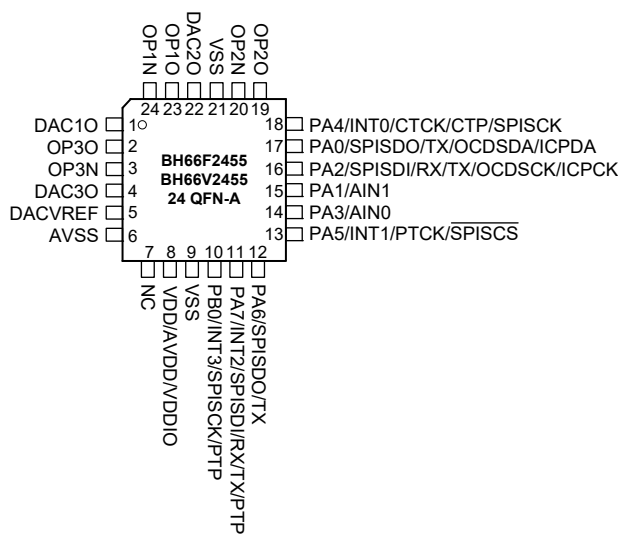
With regard to CGM applications, the device has integrated many of the functions required by these products. These include functions such as Internal Reference Voltage generator, 24-bit Delta Sigma A/D converter, 12-bit D/A Converters and operational amplifiers, etc. The inclusion of flexible I/O programming features, 16-bit Cyclic Redundancy Check function, Time Base functions along with many other features enhance the versatility of the device to suit for CGM applications.

Block Diagram



Pin Assignment





- 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 OCSDA and OCDSCK pins are the OCDS dedicated pins and only available for the BH66V2455 device which is the OCDS EV chip for the BH66F2455 device.
3. For the less pin-count package type there will be unbounded pins which should be properly configured to avoid unwanted power consumption resulting from floating input conditions. Refer to the “Standby Current Considerations” and “Input/Output Ports” sections.

Pin Descriptions

The function of each pin is listed in the following 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 table will be available on smaller package sizes.

| Pin Name | Function | OPT | I/T | O/T | Description |
|---------------------------|----------|----------------------|-----|------|---|
| PA0/SPISDO/TX/OCSDA/ICPDA | PA0 | PAPU PAWU PAS0 | ST | CMOS | General purpose I/O. Register enabled pull-high and wake-up |
| | SPISDO | PAS0 | — | CMOS | SPI serial data output |
| | TX | PAS0 | — | CMOS | UART serial data output |
| | OCSDA | — | ST | CMOS | OCDS data/address pin, for EV chip only |
| | ICPDA | — | ST | CMOS | ICP data/address pin |
| PA1/AIN1 | PA1 | PAPU PAWU PAS0 | ST | CMOS | General purpose I/O. Register enabled pull-high and wake-up |
| | AIN1 | PAS0 | AN | — | A/D Converter analog input |

| Pin Name | Function | OPT | I/T | O/T | Description |
|-----------------------------------|----------|------------------------|-----|------|--|
| PA2/SPISDI/RX/TX/ OCDSCK/ICPCK | PA2 | PAPU PAWU PAS0 | ST | CMOS | General purpose I/O. Register enabled pull-high and wake-up |
| | SPISDI | PAS0 IFS | ST | — | SPI serial data input |
| | RX/TX | PAS0 IFS | ST | CMOS | UART serial data input in full-duplex communication or UART serial data input/output in single wire mode communication |
| | OCDSCK | — | ST | — | OCDS clock pin, for EV chip only |
| | ICPCK | — | ST | — | ICP clock pin |
| PA3/AIN0 | PA3 | PAPU PAWU PAS0 | ST | CMOS | General purpose I/O. Register enabled pull-high and wake-up |
| | AIN0 | PAS0 | AN | — | A/D Converter analog input |
| PA4/INT0/CTCK/CTP/ SPISCK | PA4 | PAPU PAWU PAS1 | ST | CMOS | General purpose I/O. Register enabled pull-high and wake-up |
| | INT0 | PAS1 INTEG INTC0 | ST | — | External interrupt |
| | CTCK | PAS1 | ST | — | CTM clock input |
| | CTP | PAS1 | — | CMOS | CTM output |
| | SPISCK | PAS1 IFS | ST | CMOS | SPI serial clock |
| PA5/INT1/PTCK/SPISCS | PA5 | PAPU PAWU PAS1 | ST | CMOS | General purpose I/O. Register enabled pull-high and wake-up |
| | INT1 | PAS1 INTEG INTC0 | ST | — | External interrupt |
| | PTCK | PAS1 | ST | — | PTM clock input or capture input |
| | SPISCS | PAS1 | ST | CMOS | SPI slave select pin |
| PA6/SPISDO/TX | PA6 | PAPU PAWU PAS1 | ST | CMOS | General purpose I/O. Register enabled pull-high and wake-up |
| | SPISDO | PAS1 | — | CMOS | SPI serial data output |
| | TX | PAS1 | — | CMOS | UART serial data output |
| PA7/INT2/SPISDI/RX/TX/ PTPI | PA7 | PAPU PAWU PAS1 | ST | CMOS | General purpose I/O. Register enabled pull-high and wake-up |
| | INT2 | PAS1 INTEG INTC1 | ST | — | External interrupt |
| | SPISDI | PAS1 IFS | ST | — | SPI serial data input |
| | RX/TX | PAS1 IFS | ST | CMOS | UART serial data input in full-duplex communication or UART serial data input/output in single wire mode communication |
| | PTPI | PAS1 | ST | — | PTM capture input |

| Pin Name | Function | OPT | I/T | O/T | Description |
|---------------------|----------|------------------------|-----|------|--|
| PB0/INT3/SPISCK/PTP | PB0 | PBPU PBS0 | ST | CMOS | General purpose I/O. Register enabled pull-high |
| | INT3 | PBS0 INTEG INTC2 | ST | — | External interrupt |
| | SPISCK | PBS0 IFS | ST | CMOS | SPI serial clock |
| | PTP | PBS0 | — | CMOS | PTM output |
| OP1N | OP1N | — | AN | — | OPA1 negative input |
| OP1O | OP1O | — | — | AN | OPA1 output |
| OP2N | OP2N | — | AN | — | OPA2 negative input |
| OP2O | OP2O | — | — | AN | OPA2 output |
| OP3N | OP3N | — | AN | — | OPA3 negative input |
| OP3O | OP3O | — | — | AN | OPA3 output |
| DACVREF | DACVREF | — | — | AN | D/A Converter reference voltage output |
| DAC1O | DAC1O | — | — | AN | D/A Converter output |
| DAC2O | DAC2O | — | — | AN | D/A Converter output |
| DAC3O | DAC3O | — | — | AN | D/A Converter output |
| VDD/AVDD/VDDIO | VDD | — | PWR | — | Digital positive power supply |
| | AVDD | — | PWR | — | Analog positive power supply |
| | VDDIO | — | PWR | — | Power supply for digital input/output pin function |
| VSS | VSS | — | PWR | — | Digital negative power supply |
| AVSS | AVSS | — | PWR | — | Analog negative power supply |
| NC | NC | — | — | — | Not connect |

Legend: I/T: Input type;

OPT: Optional by register option;

CMOS: CMOS output;

PWR: Power

O/T: Output type;

ST: Schmitt Trigger input;

AN: Analog signal;

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..... | $-60^{\circ}C$ to $150^{\circ}C$ |
| Operating Temperature..... | $-40^{\circ}C$ to $85^{\circ}C$ |
| 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=-40°C~85°C

| Symbol | Parameter | Test Conditions | Min. | Typ. | Max. | Unit |
|-----------------|--------------------------|-----------------------------|------|------|------|------|
| V _{DD} | Operating Voltage – HIRC | f _{sys} =4MHz | 2.2 | — | 5.5 | V |
| | Operating Voltage – MIRC | f _{sys} =400kHz | 2.2 | — | 5.5 | V |
| | Operating Voltage – LIRC | f _{sys} =32.768kHz | 2.2 | — | 5.5 | V |

Operating Current Characteristics

Ta=-40°C~85°C

| Symbol | Operating Mode | Test Conditions | | Min. | Typ. | Max. | Unit |
|-----------------|------------------|-----------------|---|------|------|------|------|
| | | V _{DD} | Conditions | | | | |
| I _{DD} | SLOW Mode – LIRC | 2.2V | f _{sys} =32.768kHz | — | 1.5 | 2.0 | μA |
| | | 3V | | — | 3.8 | 5.0 | |
| | | 5V | | — | 11.8 | 15.3 | |
| | | 2.2V | f _{sys} =32.768kHz, LVR enable | — | 4.7 | 6.1 | |
| | | 3V | | — | 8.0 | 10.4 | |
| | | 5V | | — | 18 | 24 | |
| | FAST Mode – MIRC | 2.2V | f _{sys} =400kHz | — | 17 | 22 | μA |
| | | 3V | | — | 36 | 47 | |
| | | 5V | | — | 74 | 96 | |
| | FAST Mode – HIRC | 2.2V | f _{sys} =4MHz | — | 0.16 | 0.19 | mA |
| | | 3V | | — | 0.37 | 0.45 | |
| | | 5V | | — | 0.75 | 0.90 | |

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

1. Any digital inputs are setup in a non floating 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.

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 | 2.2V | WDT off | — | 0.45 | 0.80 | 7.00 | μA |
| | | 3V | | — | 0.45 | 0.90 | 8.00 | |
| | | 5V | | — | 0.5 | 2.0 | 10.0 | |
| | | 2.2V | WDT off, Time Base on, LIRC on | — | 1.5 | 3.0 | 3.6 | μA |
| | | 3V | | — | 1.5 | 3.0 | 3.7 | |
| | | 5V | | — | 3 | 5 | 6 | |
| | | 2.2V | WDT on, LIRC on | — | 1.5 | 3.0 | 3.6 | μA |
| | | 3V | | — | 1.5 | 3.0 | 3.7 | |
| | | 5V | | — | 3 | 5 | 6 | |
| | IDLE0 Mode– LIRC | 2.2V | f _{SUB} on | — | 3.0 | 5.0 | 5.7 | μA |
| | | 3V | | — | 3.0 | 5.0 | 5.7 | |
| | | 5V | | — | 5 | 10 | 11 | |
| | IDLE1 Mode – MIRC | 2.2V | f _{SUB} on, f _{SYS} =400kHz | — | 8.5 | 11.0 | 19.0 | μA |
| | | 3V | | — | 13.6 | 17.7 | 30.0 | |
| | | 5V | | — | 26.4 | 34.3 | 56.0 | |
| | IDLE1 Mode – HIRC | 2.2V | f _{SUB} on, f _{SYS} =4MHz | — | 144 | 180 | 200 | μA |
| | | 3V | | — | 180 | 220 | 230 | |
| | | 5V | | — | 350 | 420 | 450 | |

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

1. Any digital inputs are setup in a non floating 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.

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.

Internal High Speed Oscillator – HIRC – Frequency Accuracy

During the program writing operation the writer will trim the HIRC oscillator at a user selected HIRC frequency and user selected voltage of either 3V or 5V.

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|-------------------|------------------------------------|-----------------|------------|-------|------|-------|------|
| | | V _{DD} | Temp. | | | | |
| f _{HIRC} | 4MHz Writer Trimmed HIRC Frequency | 3V/5V | 25°C | -1% | 4 | +1% | MHz |
| | | | -40°C~85°C | -2% | 4 | +2% | |
| | | 2.2V~5.5V | 25°C | -2.5% | 4 | +2.5% | |
| | | | -40°C~85°C | -3% | 4 | +3% | |

Note: 1. The 3V/5V values for V_{DD} are provided as these are the two selectable fixed voltages at which the HIRC frequency is trimmed by the writer.

2. The row below the 3V/5V trim voltage row is provided to show the values for the full V_{DD} range operating voltage. It is recommended that the trim voltage is fixed at 3V for application voltage ranges from 2.2V to 3.6V and fixed at 5V for application voltage ranges from 3.3V to 5.5V.

Internal Medium Speed Oscillator Characteristics – MIRC

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|--------------------|--------------------|-----------------|------------|------|------|------|------|
| | | V _{DD} | Temp. | | | | |
| f _{MIRC} | MIRC Frequency | 3V/5V | 25°C | -2% | 400 | +2% | kHz |
| | | 2.2V~5.5V | -40°C~85°C | -7% | 400 | +7% | |
| t _{START} | MIRC Start Up Time | — | -40°C~85°C | — | — | 100 | μs |

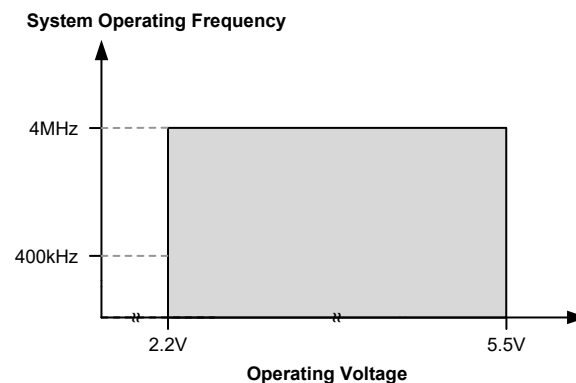
Internal Low Speed Oscillator Characteristics – LIRC

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|--------------------|--------------------|-----------------|------------|-------|--------|-------|------|
| | | V _{DD} | Temp. | | | | |
| f _{LIRC} | LIRC Frequency | 3V/5V | -10°C~50°C | -0.5% | 32.768 | +0.5% | kHz |
| | | | -40°C~85°C | -3% | 32.768 | +3% | |
| | | 2.2V~5.5V | -40°C~85°C | -10% | 32.768 | +10% | |
| t _{START} | LIRC Start Up Time | — | -40°C~85°C | — | — | 100 | μs |

Note: 1. The 3V/5V values for V_{DD} are provided as these are the two selectable fixed voltages at which the LIRC frequency is trimmed by the writer.

2. The row below the 3V/5V trim voltage row is provided to show the values for the full V_{DD} range operating voltage. It is recommended that the trim voltage is fixed at 3V for application voltage ranges from 2.2V to 3.6V and fixed at 5V for application voltage ranges from 3.3V to 5.5V.

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 Conditions where f _{SYS} is off) | — | f _{SYS} =f _H ~f _H /64, f _H =f _{HIRC} OR f _{MIRC} f _{SYS} =f _{SUB} =f _{LIRC} | — | 16 | — | t _{SYS} |
| | System Start-up Time (Wake-up from Conditions where f _{SYS} is on) | — | f _{SYS} =f _H ~f _H /64, f _H =f _{HIRC} OR f _{MIRC} f _{SYS} =f _{SUB} =f _{LIRC} | — | 2 | — | t _{SYS} |
| | System Speed Switch Time (FAST to Slow Mode or SLOW to FAST Mode) | — | f _{HIRC} OR f _{MIRC} switches from off → on | — | 16 | — | t _{HIRC} OR t _{MIRC} |
| t _{RSTD} | System Reset Delay Time (Reset Source from Power-on Reset or LVR Hardware Reset) | — | RR _{POR} =5V/ms | 14 | 16 | 18 | ms |
| | System Reset Delay Time (LVRC/WDTC/RSTC Software Reset) | — | — | | | | |
| | System Reset Delay Time (Reset Source from WDT Overflow) | — | — | 14 | 16 | 18 | |
| 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_{SYS}=1/f_{SYS} 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

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|-------------------|---|-----------------|--|----------------------|------|----------------------|------|
| | | V _{DD} | Conditions | | | | |
| V _{DD} | MCU Power Supply | — | — | 2.2 | — | 5.5 | V |
| V _{DDIO} | I/O Ports Power Supply | — | — | 2.2 | — | V _{DD} | V |
| V _{IL} | Input Low Voltage for I/O Ports | 5V | — | 0 | — | 1.5 | V |
| | | — | — | 0 | — | 0.2V _{DDIO} | |
| V _{IH} | Input High Voltage for I/O Ports | 5V | — | 3.5 | — | 5.0 | V |
| | | — | — | 0.8V _{DDIO} | — | V _{DDIO} | |
| I _{OL} | Sink Current for I/O Ports | 3V | V _{OL} =0.1V _{DDIO} | 16 | 32 | — | mA |
| | | 5V | | 32 | 65 | — | |
| I _{OH} | Source Current for I/O Ports | 3V | V _{OH} =0.9V _{DDIO} | -4 | -8 | — | mA |
| | | 5V | | -8 | -16 | — | |
| R _{PH} | Pull-high Resistance for I/O Ports ⁽¹⁾ | 3V | — | 20 | 60 | 100 | kΩ |
| | | 5V | — | 10 | 30 | 50 | |
| I _{LEAK} | Input leakage current for I/O Ports | 5V | V _{IN} =V _{DDIO} OR V _{IN} =V _{SS} | — | — | ±1 | μA |
| t _{INT} | External Interrupt Input Minimum Pulse Width | — | — | 10 | — | — | μs |
| t _{TPI} | PTPI Input Pin Minimum Pulse Width | — | — | 0.3 | — | — | μs |

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|--------------------|--|-----------------|------------|---------------------------------|------|------|------------------|
| | | V _{DD} | Conditions | | | | |
| t _{TCK} | xTCK Input Pin Minimum Pulse Width | — | — | 0.3 | — | — | μs |
| f _{TMCLK} | PTM Maximum Timer Clock Source Frequency | 5V | — | — | — | 1 | f _{sys} |
| t _{CPW} | PTM Minimum Capture Pulse Width | — | — | t _{CPW} ⁽²⁾ | — | — | μs |

Note: 1. 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.

2. t_{TMCLK}=1/f_{TMCLK}

If PTCAPTS=0, then t_{CPW}=max(2×t_{TMCLK}, t_{TPI})

If PTCAPTS=1, then t_{CPW}=max(2×t_{TMCLK}, t_{TCK})

Ex1: If PTCAPTS=0, f_{TMCLK}=4MHz, t_{TPI}=0.3μs, then t_{CPW}=max(0.5μs, 0.3μs)=0.5μs

Ex2: If PTCAPTS=1, f_{TMCLK}=4MHz, t_{TCK}=0.3μs, then t_{CPW}=max(0.5μs, 0.3μs)=0.5μs

Memory Characteristics

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

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|----------------------|--|-----------------|------------|------|------|------|------------------|
| | | V _{DD} | Conditions | | | | |
| Flash Program Memory | | | | | | | |
| t _{FER} | Erase Time | — | FWERTS=0 | — | 3.2 | 3.9 | ms |
| | | — | FWERTS=1 | — | 3.7 | 4.5 | |
| t _{FWR} | Write Time | — | FWERTS=0 | — | 2.2 | 2.7 | ms |
| | | — | FWERTS=1 | — | 3.0 | 3.6 | |
| E _P | Cell Endurance | — | — | 100K | — | — | E/W |
| t _{RETD} | ROM Data Retention Time | — | Ta=25°C | — | 40 | — | Year |
| t _{ACTV} | ROM Activation Time – Wake-up from IDLE/SLEEP Mode | — | — | 32 | — | 64 | μs |
| Data EEPROM Memory | | | | | | | |
| V _{DD} | V _{DD} for Read / Write | — | — | 2.2 | — | 5.5 | V |
| t _{EERD} | EEPROM Read Time | — | — | — | — | 4 | t _{sys} |
| t _{EEER} | EEPROM Erase Time | — | EWERTS=0 | — | 3.2 | 3.9 | ms |
| | | — | EWERTS=1 | — | 3.7 | 4.5 | |
| t _{EEWR} | EEPROM Write Time (Byte Mode) | — | EWERTS=0 | — | 5.4 | 6.6 | ms |
| | | — | EWERTS=1 | — | 6.7 | 8.1 | |
| | EEPROM Write Time (Page Mode) | — | EWERTS=0 | — | 2.2 | 2.7 | |
| | | — | EWERTS=1 | — | 3.0 | 3.6 | |
| E _P | Cell Endurance | — | — | 100K | — | — | E/W |
| t _{RETD} | Data Retention Time | — | Ta=25°C | — | 40 | — | Year |
| RAM Data Memory | | | | | | | |
| V _{DR} | RAM Data Retention Voltage | — | — | 1.0 | — | — | V |

Note: 1. “E/W” means Erase/Write times.

2. 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.

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 1.7V | -5% | 1.7 | +5% | V |
| | | — | LVR enable, voltage select 1.9V | -3% | 1.9 | +3% | |
| | | — | LVR enable, voltage select 2.1V | | 2.1 | | |
| | | — | LVR enable, voltage select 2.55V | | 2.55 | | |
| | | — | LVR enable, voltage select 3.15V | | 3.15 | | |
| t _{LVR} | Minimum Low Voltage Width to Reset | — | TLVR[1:0]=00B | 120 | 240 | 480 | μs |
| | | | TLVR[1:0]=01B | 0.5 | 1.0 | 2.0 | ms |
| | | | TLVR[1:0]=10B | 1 | 2 | 4 | |
| | | | TLVR[1:0]=11B | 2 | 4 | 8 | |
| I _{LVR} | Additional Current for LVR Enable | 3V | — | — | 4.5 | 6.0 | μA |
| | | 5V | — | — | 6.5 | 8.0 | |

Analog Front End Circuit Characteristics

Operational Voltage Electrical Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|------------------|-------------------|-----------------|------------|------|------|------|------|
| | | V _{DD} | Conditions | | | | |
| AV _{DD} | Operating Voltage | — | — | 2.2 | — | 5.5 | V |

Operational Amplifier Electrical Characteristics

Ta=25°C, unless otherwise specified

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|---------------------|-----------------------------------|-----------------|-------------------------------------|------------------|------|------------------|------|
| | | V _{DD} | Conditions | | | | |
| I _{OPA} | Additional Current for OPA Enable | — | I _{OUT} =0 | — | 0.6 | 1.0 | μA |
| V _{OS} | Input Offset Voltage | — | — | -3 | — | +3 | mV |
| V _{CM_OPA} | OPA Common Mode Voltage Range | — | — | AV _{SS} | — | AV _{DD} | V |
| PSRR | Power Supply Rejection Ratio | — | — | — | 80 | — | dB |
| CMRR | Common Mode Rejection Ratio | — | — | — | 80 | — | dB |
| R _{FB} | Feedback Resistor | — | R _{FB} =0.5MΩ, Ta=0°C~50°C | -25% | 0.5 | +25% | MΩ |
| | | — | R _{FB} =1MΩ, Ta=0°C~50°C | | 1 | | |
| | | — | R _{FB} =2MΩ, Ta=0°C~50°C | | 2 | | |
| | | — | R _{FB} =4MΩ, Ta=0°C~50°C | | 4 | | |
| | | — | R _{FB} =8MΩ, Ta=0°C~50°C | | 8 | | |
| C _{FB} | Feedback Capacitor | — | Ta=0°C~50°C | -16% | 100 | +16% | pF |

Internal Reference Voltage Characteristics

Ta=25°C, unless otherwise specified

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|--------------------|--|-----------------|--|------|------|------|--------|
| | | V _{DD} | Conditions | | | | |
| V _{IREF} | Internal Reference Voltage with Voltage Buffer | 3V | Trim V _{IREF} =1.25V±3% @ V _{IREF} =1.25V, IREFEN=1, PVREF=10000000B | -3% | 1.25 | +3% | V |
| | | 3V | Trim V _{IREF} =1.83V±3% @ V _{IREF} =1.83V, IREFEN=1, PVREF=10000000B | -4% | 1.83 | +4% | V |
| I _{IREF} | Additional Current for Internal Reference Voltage Enable | — | IREFEN=1 | — | 2 | 5 | μA |
| TC _{IREF} | Temperature Coefficient of Internal Reference Voltage | 3V | Ta=10°C~40°C | — | ±40 | — | ppm/°C |

12-bit D/A Converter Electrical Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|-------------------|-----------------------------------|-----------------|-----------------|------|------|------|----------------------|
| | | V _{DD} | Conditions | | | | |
| DNL | Differential Non-linearity | 3V | DACVRS[1:0]=00B | — | — | ±8 | LSB |
| I _{DAC} | Additional Current for DAC Enable | — | — | — | 0.4 | — | μA |
| INL | Integral Non-linearity | 3V | DACVRS[1:0]=00B | — | — | ±20 | LSB |
| Ro | R2R Output Resistor | 3V | — | — | 3480 | — | kΩ |
| V _{DACO} | Output Voltage Range | — | — | 0.00 | — | 1.00 | V _{DACVREF} |

A/D Converter Electrical Characteristics

Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|--------------------|-----------------------------------|-----------------|--|------|-------|-------|------|
| | | V _{DD} | Conditions | | | | |
| I _{ADC} | Additional Current for ADC Enable | 3V | — | — | 400 | 650 | μA |
| I _{ADSTB} | Standby Current | 3V | MCU enter SLEEP mode, no load | — | — | 1 | μA |
| N _R | Resolution | — | — | — | — | 24 | Bit |
| INL | Integral Non-linearity | — | AV _{DD} =2.6V, V _{REF} =1.25V, ΔSI=±450mV, PGAGN=1 | — | ±50 | — | ppm |
| I _{bias} | Input Bias Current | — | Ta=25°C | — | ±100 | — | nA |
| NFB | Noise Free Bits | — | f _{MCLK} =400kHz, FLMSPS=1, AV _{DD} =2.6V, V _{REF} =1.25V, PGAGN=128, OSR=16384 | — | 16 | — | Bit |
| ENOB | Effective Number of Bits | — | f _{MCLK} =400kHz, FLMSPS=1, AV _{DD} =2.6V, V _{REF} =1.25V, PGAGN=128, OSR=16384 | — | 18.7 | — | Bit |
| f _{ADCK} | ADC Clock Frequency | — | — | 40.0 | 409.6 | 440.0 | kHz |

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|-------------------|-------------------------|-----------------|--|---------------------------|------|---------------------------|------|
| | | V _{DD} | Conditions | | | | |
| f _{ADO} | ADC Output Data Rate | — | f _{MCLK} =4MHz, FLMSPS=0, FLMS[2:0]=000B, SINC3[1:0]=11B | 4 | — | 1042 | Hz |
| | | — | f _{MCLK} =4MHz, FLMSPS=0, FLMS[2:0]=010B, SINC3[1:0]=11B | 10 | — | 2604 | Hz |
| | | — | f _{MCLK} =400kHz, FLMSPS=1, SINC3[1:0]=11B | 12 | — | 3125 | Hz |
| V _{REFP} | Reference Input Voltage | — | — | V _{REFN} +0.8 | — | V _{DACVREF} | V |
| V _{REFN} | | — | — | 0 | — | V _{REFP} -0.8 | V |
| V _{REF} | | — | V _{REF} =(V _{REFP} - V _{REFN}) | 0.8 | — | 1.83 | V |

PGA Electrical Characteristics

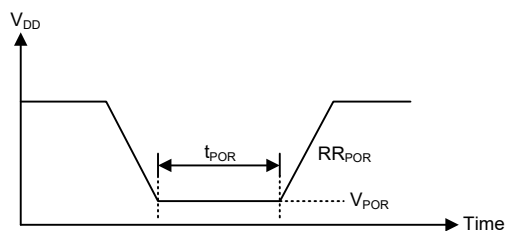
Ta=25°C

| Symbol | Parameter | Test Conditions | | Min. | Typ. | Max. | Unit |
|-----------------|----------------------------------|-----------------|-----------------|----------------------------|------|----------------------------|------|
| | | V _{DD} | Conditions | | | | |
| ΔD _I | Differential Input Voltage Range | — | Gain=PGAGN×ADGN | -V _{REF} /Gain | — | +V _{REF} /Gain | V |

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 |



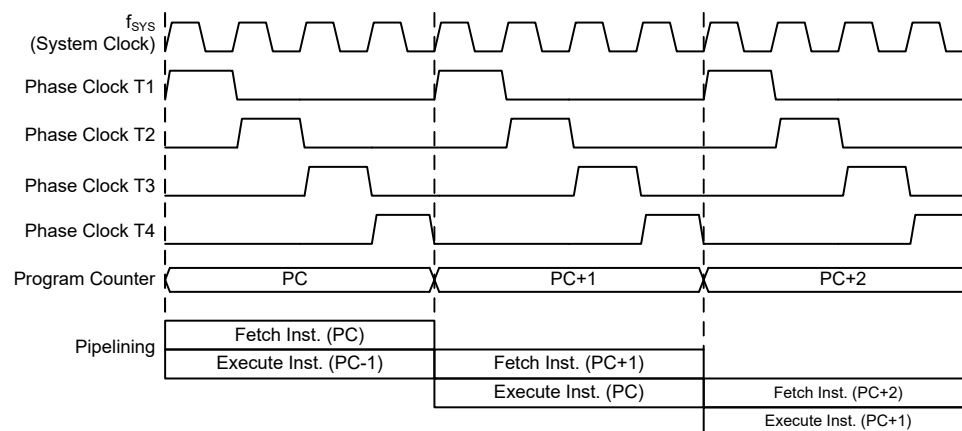
System Architecture

A key factor in the high-performance features of the Holtek range of 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 or two cycles for most of the standard or extended instructions respectively. The exceptions to these are 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 and A/D control system with maximum reliability and flexibility. This makes the device suitable for affordable and high-volume production for controller applications.

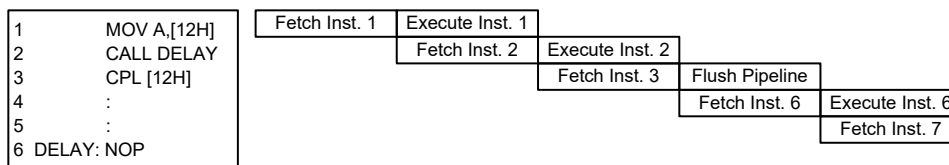
Clocking and Pipelining

The main system clock, derived from either an HIRC, MIRC 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 | |
|-----------------|----------------|
| High Byte | Low Byte (PCL) |
| PC12~PC8 | PCL7~PCL0 |

Program Counter

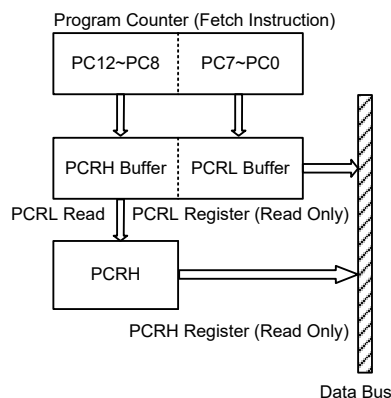
The lower byte of the Program Counter, known as the Program Counter Low register or PCL, is available for program control and is a readable and writeable register. By transferring data directly into this register, a short program jump can be executed directly; however, as only this low byte is available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

Program Counter Read Registers

The Program Counter Read registers are a read only register pair for reading the program counter value which indicates the current program execution address. Read the low byte register first then the high byte register. Reading the low byte register, PCRL, will read the low byte data of the current program execution address, and place the high byte data of the program counter into the 8-bit PCRH buffer. Then reading the PCRH register will read the corresponding data from the 8-bit PCRH buffer.

The following example shows how to read the current program execution address. When the current program execution address is 123H, the steps to execute the instructions are as follows:

- (1) MOV A, PCRL → the ACC value is 23H, and the PCRH value is 01H;
MOV A, PCRH → the ACC value is 01H.
- (2) LMOV A, PCRL → the ACC value is 23H, and the PCRH value is 01H;
LMOV A, PCRH → the ACC value is 01H.



• PCRL Register

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

Bit 7~0 **D7~D0**: Program Counter Read Low byte register bit 7 ~ bit 0

• PCRH Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|---|-----|-----|-----|----|----|
| Name | — | — | — | D12 | D11 | D10 | D9 | D8 |
| R/W | — | — | — | R | R | R | R | R |
| POR | — | — | — | 0 | 0 | 0 | 0 | 0 |

Bit 7~5 Unimplemented, read as “0”

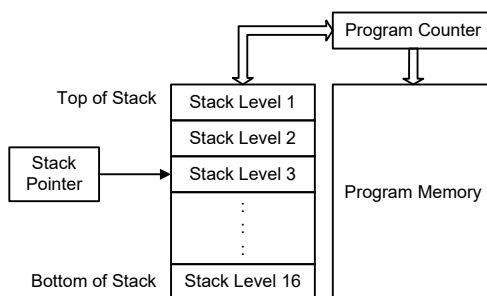
Bit 4~0 **D12~D8**: Program Counter Read High byte register bit 4 ~ bit 0

Stack

This is a special part of the memory which is used to save the contents of the Program Counter only. The stack is organized into 16 levels and is neither part of the data nor part of the program space, and is neither readable nor writeable. The activated level is indexed by the Stack Pointer, STKPTR[3:0]. 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.



• STKPTR Register

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

Bit 7 **OSF**: Stack overflow flag
 0: No stack overflow occurred
 1: Stack overflow occurred

When the stack is full and a CALL instruction is executed or when the stack is empty and a RET instruction is executed, the OSF bit will be set high. The OSF bit is cleared only by software and cannot be reset automatically by hardware.

Bit 6~4 Unimplemented, read as “0”

Bit 3~0 **D3~D0**: Stack pointer register bit 3 ~ bit 0

The following example shows how the Stack Pointer and Stack Overflow Flag change when program branching conditions occur.

- (1) When the CALL subroutine instruction is executed 17 times continuously and the RET instruction is not executed during the period, the corresponding changes of the STKPTR[3:0] and OSF bits are as follows:

| CALL Execution Times | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|-----------------------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|
| STKPTR[3:0] Bit Value | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 0 | 1 |
| OSF Bit Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

- (2) When the OSF bit is set high and not cleared, it will remain high no matter how many times the RET instruction is executed.

- (3) When the stack is empty, the RET instruction is executed 16 times continuously, the corresponding changes of the STKPTR[3:0] and OSF bits are as follows:

| RET Execution Times | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|-----------------------|---|----|----|----|----|----|----|---|---|---|----|----|----|----|----|----|----|
| STKPTR[3:0] Bit Value | 0 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| OSF Bit Value | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

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:

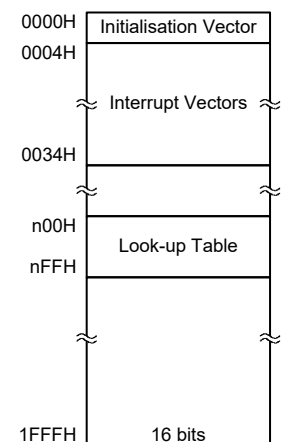
- Arithmetic operations:
ADD, ADDM, ADC, ADCM, SUB, SUBM, SBC, SBCM, DAA,
LADD, LADDM, LADC, LADCM, LSUB, LSUBM, LSBC, LSBCM, LDAA
- Logic operations:
AND, OR, XOR, ANDM, ORM, XORM, CPL, CPLA,
LAND, LOR, LXOR, LANDM, LORM, LXORM, LCPL, LCPLA
- Rotation:
RRA, RR, RRCA, RRC, RLA, RL, RLCA, RLC,
LRRR, LRR, LRRCA, LRRC, LRLA, LRL, LRLCA, LRLC
- Increment and Decrement:
INCA, INC, DECA, DEC,
LINCA, LINC, LDECA, LDEC
- Branch decision:
JMP, SZ, SZA, SNZ, SIZ, SDZ, SIZA, SDZA, CALL, RET, RETI,
LSZ, LSZA, LSNZ, LSIZ, LSDZ, LSIZA, LSDZA

Flash Program Memory

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

Structure

The Program Memory has a capacity of 8K×16 bits. The Program Memory is addressed by the Program Counter and also contains data, table information and interrupt entries. Table data, which can be setup in any location within the Program Memory, is addressed by a separate table pointer registers.



Program Memory Structure

Special Vectors

Within the Program Memory, certain locations are reserved for the reset and interrupts. The location 0000H 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 setup by placing the address of the look up data to be retrieved in the table pointer registers, TBLP and TBHP. These registers define the total address of the look-up table.

After setting up the table pointer, the table data can be retrieved from the Program Memory using the “TABRD [m]” or “TABRDL [m]” instructions respectively when the memory [m] is located in sector 0. If the memory [m] is located in other sectors except sector 0, the data can be retrieved from the program memory using the corresponding extended table read instruction such as “LTABRD [m]” or “LTABRDL [m]” 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.

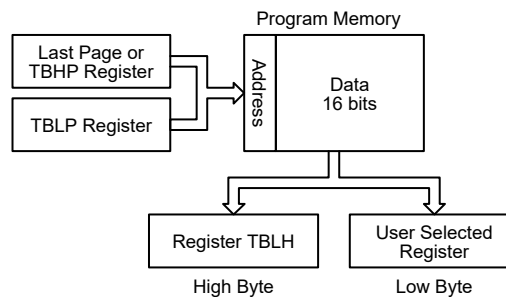


Table Program Example

The accompanying example shows how the table pointer and table data is defined and retrieved from the device. This example uses raw table data located in the Program Memory which is stored there using the ORG statement. The value at this ORG statement is “1F00H” which refers to the start address of the last page within the 8K words Program Memory. The table pointer low byte register is setup here to have an initial value of “06H”. This will ensure that the first data read from the data table will be at the Program Memory address “1F06H” or 6 locations after the start of the last page. Note that the value for the table pointer is referenced to the specific address pointed by TBHP and TBLP if the “TABRD [m]” or “LTABRD [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]” or “LTABRD [m]” instruction is executed.

Because the TBLH register is a read/write register and can 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 table pointer - note that this address is referenced
mov tblp,a         ; to the last page or the page that tbhp pointed
mov a,1FH          ; initialise high table pointer
mov tbhp,a         ; it is not necessary to set tbhp if executing tabrdl or ltabrdl
:
:
tabrd tempreg1     ; transfers value in table referenced by table pointer data at
                  ; program memory address "1F06H" 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 "1F05H" transferred to tempreg2 and TBLH
                  ; in this example the data "1AH" is transferred to tempreg1 and
                  ; data "0FH" to tempreg2 the value "00H" will be
                  ; transferred to the high byte register TBLH
:
:
org 1F00H          ; sets initial address of last page
dc 00Ah,00Bh,00Ch,00Dh,00Eh,00Fh,01Ah,01Bh

```

Unique Identifier – UID

The device contains a 128-bit unique ID. It is unchangeable and determined by MCU manufacturer. The UID format is shown below:

| Mapped Address in Program Memory Last Page | | UID No. |
|--|-----------|-------------|
| 0xE8 | Low byte | UID0 |
| | High byte | UID1 |
| 0xE9 | Low byte | UID2 |
| | High byte | UID3 |
| 0xEA | Low byte | UID4 |
| | High byte | UID5 |
| 0xEB | Low byte | UID6 |
| | High byte | UID7 |
| 0xEC | Low byte | UID8 |
| | High byte | UID9 |
| 0xED | Low byte | UID10 |
| | High byte | UID11 |
| 0xEE | Low byte | UID12 |
| | High byte | UID13 |
| 0xEF | Low byte | CRC16[7:0] |
| | High byte | CRC16[15:8] |

The UID is located at the Option Memory addresses 28H~2FH which will be mapped to the Program Memory last page addresses E8H~EFH. The UID can be read from the Program Memory last page using the table read instruction when the Option Memory mapping function is enabled via the ORMC register. For more details, refer to the “Option Memory Mapping Register – ORMC” in the Special Function Register Description section.

CRC Description

Calculation sequence: UID0→UID1→UID2→...→UID11→UID12→UID13.

CRC Specification:

CRC-CCITT: 1021H, $X^{16}+X^{12}+X^5+1$

Initial Value=0000H

Final Xor Value=0000H

Input reflected=No

Output reflected=No

• CRC Calculation Example

Write 4 bytes input data sequentially and the CRC checksum are sequentially listed in the following table.

| CRC Polynomial | CRC Data Input | 78H→56H→34H→12H |
|-------------------------------------|----------------|-------------------------|
| CRC-CCITT ($X^{16}+X^{12}+X^5+1$) | | FF9FH→BBC3H→A367H→D0FAH |

In Circuit Programming – ICP

The provision of Flash type Program Memory provides the user with a means of convenient and easy upgrades and modifications to their programs on the same device.

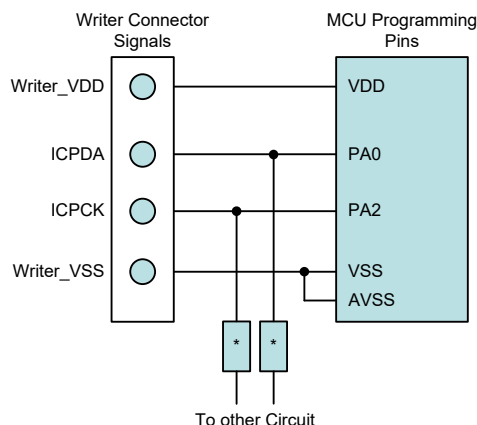
As an additional convenience, Holtek has provided a means of programming the microcontroller in-circuit using a 4-pin interface. This provides manufacturers with the possibility of manufacturing their circuit boards complete with a programmed or un-programmed microcontroller, and then programming or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

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

| Holtek Writer Pins | MCU Programming Pins | Pin Description |
|--------------------|----------------------|---------------------------------|
| ICPDA | PA0 | Programming Serial Data/Address |
| ICPCK | PA2 | Programming Clock |
| VDD | VDD | Power Supply |
| VSS | VSS&AVSS | Ground |

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

During the programming process, the 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: * may be resistor or capacitor. The resistance of * must be greater than 1k Ω or the capacitance of * must be less than 1nF.

On-Chip Debug Support – OCDS

There is an EV chip named BH66V2455 which is used to emulate the BH66F2455 device. 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 OCDS function to emulate the real chip device behavior by connecting the OCSDA and OCDSCK pins to the Holtek HT-IDE development tools. The OCSDA 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 OCSDA and OCDSCK pins in the device will have no effect in the EV chip. However, the two OCDS pins which are pin-shared with the ICP programming pins are still used as the Flash Memory programming pins for ICP. For more detailed OCDS information, refer to the corresponding document named “Holtek e-Link for 8-bit MCU OCDS User’s Guide”.

| Holtek e-Link Pins | EV Chip Pins | Pin Description |
|--------------------|--------------|---|
| OCSDA | OCSDA | On-Chip Debug Support Data/Address input/output |
| OCDSCK | OCDSCK | On-Chip Debug Support Clock input |
| VDD | VDD | Power Supply |
| VSS | VSS&AVSS | Ground |

In Application Programming – IAP

Flash type Program Memory provides the user with a means of convenient and easy upgrades and modifications to their programs on the same device. The provision of the IAP function offers users the convenience of Flash Memory multi-programming features. The convenience of the IAP function is that it can execute the updated program procedure using its internal firmware, without requiring an external Program Writer or PC. In addition, the IAP interface can also be any type of communication protocol, such as UART, using I/O pins. Regarding the internal firmware, the user can select versions provided by Holtek or create their own. The following section illustrates the procedures regarding how to implement the IAP firmware.

Flash Memory Read/Write Size

The Flash memory Erase and Write operations are carried out in a page format while the Read operation is carried out in a word format. The page size and write buffer size are both assigned with a capacity of 32 words. Note that the Erase operation should be executed before the Write operation is executed.

When the Flash Memory Erase/Write Function is successfully enabled, the CFWEN bit will be set high. When the CFWEN bit is set high, the data can be written into the write buffer. The FWT bit is used to initiate the write process and then indicate the write operation status. This bit is set high by application program to initiate a write process and will be cleared by hardware if the write process is finished.

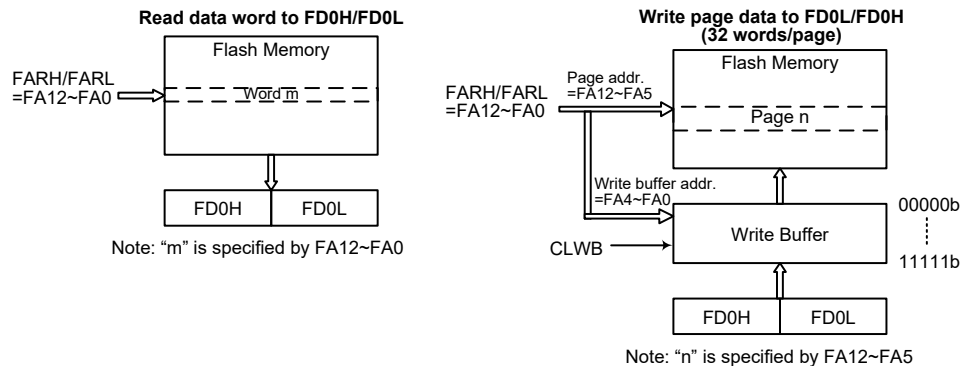
The Read operation can be carried out by executing a specific read procedure. The FRDEN bit is used to enable the read function and the FRD bit is used to initiate the read process by application programs and then indicate the read operation status. When the read process is finished, this bit will be cleared by hardware.

| Operations | Format |
|---|---------------|
| Erase | 32 words/time |
| Write | 32 words/time |
| Read | 1 word/time |
| Note: Page size=Write buffer size=32 words. | |

IAP Operation Format

| Page | FARH | FARL[7:5] | FARL[4:0] |
|------|-----------|-----------|-------------|
| 0 | 0000 0000 | 000 | Tag Address |
| 1 | 0000 0000 | 001 | |
| 2 | 0000 0000 | 010 | |
| 3 | 0000 0000 | 011 | |
| 4 | 0000 0000 | 100 | |
| : | : | : | |
| : | : | : | |
| 254 | 0001 1111 | 110 | |
| 255 | 0001 1111 | 111 | |

Page Number and Address Selection



Flash Memory IAP Read/Write Structure

Write Buffer

The write buffer is used to store the written data temporarily when executing the write operation. The Write Buffer can be filled with written data after the Flash Memory Erase/Write Function has been successfully enabled by executing the Flash Memory Erase/Write Function Enable procedure. The write buffer can be cleared by configuring the CLWB bit in the FC2 register. The CLWB bit can be set high to enable the Clear Write Buffer procedure. When the procedure is finished this bit will be cleared to zero by hardware. It is recommended that the write buffer should be cleared by setting the CLWB bit high before the write buffer is used for the first time or when the data in the write buffer is updated.

The write buffer size is 32 words corresponding to a page. The write buffer address is mapped to a specific Flash memory page specified by the memory address bits, FA12~FA5. The data written into the FD0L and FD0H registers will be loaded into the write buffer. When data is written into the high byte data register, FD0H, it will result in the data stored in the high and low byte data registers both being written into the write buffer. It will also cause the Flash memory address to be incremented by one, after which the new address will be loaded into the FARH and FARL address registers. When the Flash memory address reaches the page boundary, 11111b of a page with 32 words, the address will now not be incremented but stop at the last address of the page. At this point a new page address should be specified for any other erase/write operations.

After a write process is finished, the write buffer will automatically be cleared by hardware. Note that the write buffer should be cleared manually by the application program when the data written into the Flash memory is incorrect in the data verification step. The data should again be written into the write buffer after the write buffer has been cleared when the data is found to be incorrect during the data verification step.

IAP Flash Program Memory Registers

There are two address registers, four pairs of 16-bit data registers and three control registers, which are all located in Sector 1. Read and Write operations to the Flash memory are carried out using 16-bit data operations using the address and data registers and the control registers. Several registers control the overall operation of the internal Flash Program Memory. The address registers are named FARL and FARH, the data registers are named FDnL and FDnH, where n is equal to 0~3, and the control registers are named FC0, FC1 and FC2. As all the IAP related registers are located in Sector 1, they can be addressed directly only using the corresponding extended instructions or can be read from or written to indirectly using the MP1H/MP1L or MP2H/MP2L Memory Pointer pairs and Indirect Addressing Register, IAR1 or IAR2.

| Register Name | Bit | | | | | | | |
|---------------|-------|-------|-------|-------|-------|------|--------|------|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| FC0 | CFWEN | FMOD2 | FMOD1 | FMOD0 | FWPEN | FWT | FRDEN | FRD |
| FC1 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| FC2 | — | — | — | — | — | — | FWERTS | CLWB |
| FARL | FA7 | FA6 | FA5 | FA4 | FA3 | FA2 | FA1 | FA0 |
| FARH | — | — | — | FA12 | FA11 | FA10 | FA9 | FA8 |
| FD0L | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| FD0H | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| FD1L | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| FD1H | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| FD2L | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| FD2H | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| FD3L | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| FD3H | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |

IAP Register List

• FC0 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|-------|-------|-------|-------|-----|-------|-----|
| Name | CFWEN | FMOD2 | FMOD1 | FMOD0 | FWPEN | FWT | FRDEN | FRD |
| 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 CFWEN:** Flash Memory Erase/Write function enable control
 0: Flash memory erase/write function is disabled
 1: Flash memory erase/write function has been successfully enabled
 When this bit is cleared to 0 by application program, the Flash memory erase/write function is disabled. Note that writing a “1” into this bit results in no action. This bit is used to indicate the Flash memory erase/write function status. When this bit is set to 1 by hardware, it means that the Flash memory erase/write function is enabled successfully. Otherwise, the Flash memory erase/write function is disabled as the bit is zero.
- Bit 6~4 FMOD2~FMOD0:** Flash memory Mode selection
 000: Write Mode
 001: Page Erase Mode
 011: Read Mode
 110: Flash memory Erase/Write function Enable Mode
 Other values: Reserved
 These bits are used to select the Flash Memory operation modes. Note that the “Flash memory Erase/Write function Enable Mode” should first be successfully enabled before the Erase or Write Flash memory operation is executed.
- Bit 3 FWPEN:** Flash memory Erase/Write function enable procedure trigger control
 0: Erase/Write function enable procedure is not triggered or procedure timer times out
 1: Erase/Write function enable procedure is triggered and procedure timer starts to count
 This bit is used to activate the Flash memory Erase/Write function enable procedure and an internal timer. It is set by the application programs and then cleared by hardware when the internal timer times out. The correct patterns must be written into the FD1L/FD1H, FD2L/FD2H and FD3L/FD3H register pairs respectively as soon as possible after the FWPEN bit is set high.
- Bit 2 FWT:** Flash memory write initiate control
 0: Do not initiate Flash memory write or indicating that a Flash memory write process has completed
 1: Initiate Flash memory write process
 This bit is set by software and cleared by hardware when the Flash memory write process has completed.
- Bit 1 FRDEN:** Flash memory read enable control
 0: Flash memory read disable
 1: Flash memory read enable
 This is the Flash memory Read Enable Bit which must be set high before any Flash memory read operations are carried out. Clearing this bit to zero will inhibit Flash memory read operations.
- Bit 0 FRD:** Flash memory read initiate control
 0: Do not initiate Flash memory read or indicating that a Flash memory read process has completed
 1: Initiate Flash memory read process
 This bit is set by software and cleared by hardware when the Flash memory read process has completed.
- Note: 1. The FWT, FRDEN and FRD bits cannot be set to “1” at the same time with a single instruction.
 2. Ensure that the f_{SUB} clock is stable before executing the erase or write operation.

3. Note that the CPU will be stopped when a read, write or erase operation is successfully activated.
4. Ensure that the read, erase or write operation is totally complete before executing other operations.

• **FC1 Register**

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

Bit 7~0 **D7~D0: Chip Reset Pattern**
 When a specific value of “55H” is written into this register, a reset signal will be generated to reset the whole chip.

• **FC2 Register**

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

Bit 7~2 Unimplemented, read as “0”
 Bit 1 **FWERTS: Erase time and Write time selection**
 0: Erase time is 3.2ms (t_{FER}) / Write time is 2.2ms (t_{FWR})
 1: Erase time is 3.7ms (t_{FER}) / Write time is 3.0ms (t_{FWR})
 Bit 0 **CLWB: Flash memory Write Buffer Clear control**
 0: Do not initiate a Write Buffer Clear process or indicating that a Write Buffer Clear process has completed
 1: Initiate Write Buffer Clear process
 This bit is set by software and cleared by hardware when the Write Buffer Clear process has completed.

• **FARL Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| Name | FA7 | FA6 | FA5 | FA4 | FA3 | FA2 | FA1 | FA0 |
| 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 **FA7~FA0: Flash Memory Address bit 7 ~ bit 0**

• **FARH Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|---|------|------|------|-----|-----|
| Name | — | — | — | FA12 | FA11 | FA10 | FA9 | FA8 |
| R/W | — | — | — | R/W | R/W | R/W | R/W | R/W |
| POR | — | — | — | 0 | 0 | 0 | 0 | 0 |

Bit 7~5 Unimplemented, read as “0”
 Bit 4~0 **FA12~FA8: Flash Memory Address bit 12 ~ bit 8**

• FD0L Register

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

Bit 7~0 **D7~D0**: The first Flash Memory data word bit 7 ~ bit 0

Note that data written into the low byte data register FD0L will only be stored in the FD0L register and not loaded into the lower 8-bit write buffer.

• FD0H Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| Name | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| 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 **D15~D8**: The first Flash Memory data word bit 15 ~ bit 8

Note that when 8-bit data is written into the high byte data register FD0H, the whole 16 bits of data stored in the FD0H and FD0L registers will simultaneously be loaded into the 16-bit write buffer after which the contents of the Flash memory address register pair, FARH and FARL, will be incremented by one.

• FD1L Register

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

Bit 7~0 **D7~D0**: The second Flash Memory data word bit 7 ~ bit 0

• FD1H Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| Name | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| 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 **D15~D8**: The second Flash Memory data word bit 15 ~ bit 8

• FD2L Register

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

Bit 7~0 **D7~D0**: The third Flash Memory data word bit 7 ~ bit 0

• FD2H Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| Name | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| 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 **D15~D8**: The third Flash Memory data word bit 15 ~ bit 8

• **FD3L Register**

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

Bit 7~0 **D7~D0**: The fourth Flash Memory data word bit 7 ~ bit 0

• **FD3H Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| Name | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| 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 **D15~D8**: The fourth Flash Memory data word bit 15 ~ bit 8

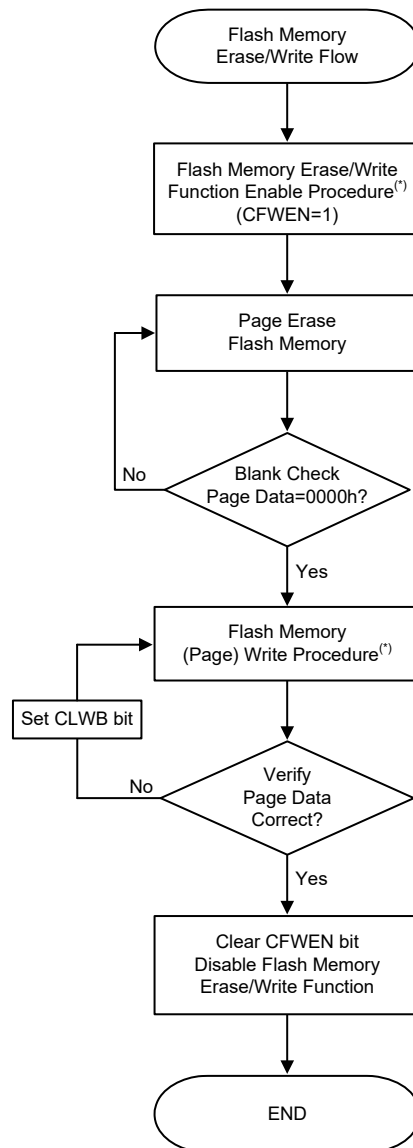
Flash Memory Erase/Write Flow

It is important to understand the Flash memory Erase/Write flow before the Flash memory contents are updated. Users can refer to the corresponding operation procedures when developing their IAP program to ensure that the Flash memory contents are correctly updated.

Flash Memory Erase/Write Flow Descriptions

1. Activate the “Flash Memory Erase/Write function enable procedure” first. When the Flash Memory Erase/Write function is successfully enabled, the CFWEN bit in the FC0 register will automatically be set high by hardware. After this, Erase or Write operations can be executed on the Flash memory. Refer to the “Flash Memory Erase/Write Function Enable Procedure” for details.
2. Configure the Flash memory address to select the desired erase page, tag address and then erase this page.

For a page erase operation, set the FARL and FARH registers to specify the start address of the erase page, then write dummy data into the FD0H register to tag address. The current address will be internally incremented by one after each dummy data is written into the FD0H register. When the address reaches the page boundary, 11111b, the address will not be further incremented but stop at the last address of the page. Note that the write operation to the FD0H register is used to tag address, it must be implemented to determine which addresses to be erased.
3. Execute a Blank Check operation to ensure whether the page erase operation is successful or not. The “TABRD” instruction should be executed to read the Flash memory contents and to check if the contents is 0000h or not. If the Flash memory page erase operation fails, users should go back to Step 2 and execute the page erase operation again.
4. Write data into the specific page. Refer to the “Flash Memory Write Procedure” for details.
5. Execute the “TABRD” instruction to read the Flash memory contents and check if the written data is correct or not. If the data read from the Flash memory is different from the written data, it means that the page write operation has failed. The CLWB bit should be set high to clear the write buffer and then write the data into the specific page again if the write operation has failed.
6. Clear the CFWEN bit to disable the Flash Memory Erase/Write function enable mode if the current page Erase and Write operations are complete if no more pages need to be erased or written.



Flash Memory Erase/Write Flow

Note: *The Flash Memory Erase/Write Function Enable procedure and Flash Memory Write procedure will be described in the following sections.

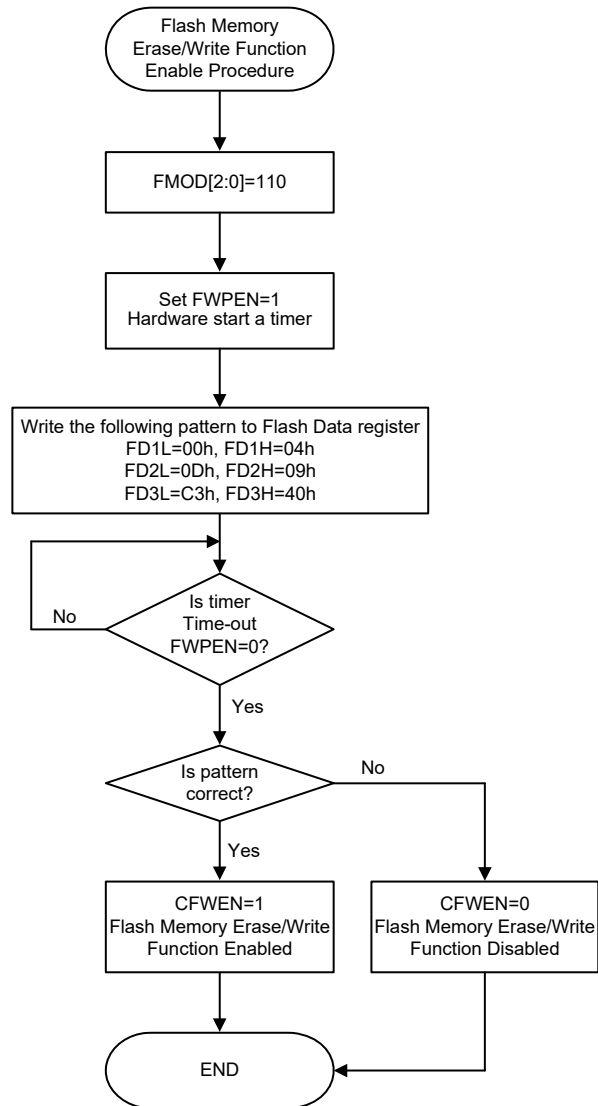
Flash Memory Erase/Write Function Enable Procedure

The Flash Memory Erase/Write Function Enable Mode is specially designed to prevent the Flash memory contents from being wrongly modified. In order to allow users to change the Flash memory data using the IAP control registers, users must first enable the Flash memory Erase/Write function.

Flash Memory Erase/Write Function Enable Procedure Description

1. Write data “110” to the FMOD [2:0] bits in the FC0 register to select the Flash Memory Erase/Write Function Enable Mode.
2. Set the FWPEN bit in the FC0 register to “1” to activate the Flash Memory Erase/Write Enable Function. This will also activate an internal timer.
3. Write the correct data pattern into the Flash data registers of FD1L, FD2L, FD3L, FD1H, FD2H and FD3H, successively and as soon as possible after the FWPEN bit is set high. The enable Flash memory erase/write function data pattern is 00H, 0DH, C3H, 04H, 09H and 40H corresponding to the FD1L~FD3L and FD1H~FD3H registers respectively.
4. Once the timer has timed out, the FWPEN bit will automatically be cleared to 0 by hardware regardless of the input data pattern.
5. If the written data pattern is incorrect, the Flash memory erase/write function will not be enabled successfully and the above steps should be repeated. If the written data pattern is correct, the Flash memory erase/write function will be enabled successfully.
6. Once the Flash memory erase/write function is enabled, the Flash memory contents can be updated by executing the page erase and write operations using the IAP control registers.

To disable the Flash memory erase/write function, the CFWEN bit in the FC0 register can be cleared. There is no need to execute the above procedure.



Flash Memory Erase/Write Function Enable Procedure

Flash Memory Write Procedure

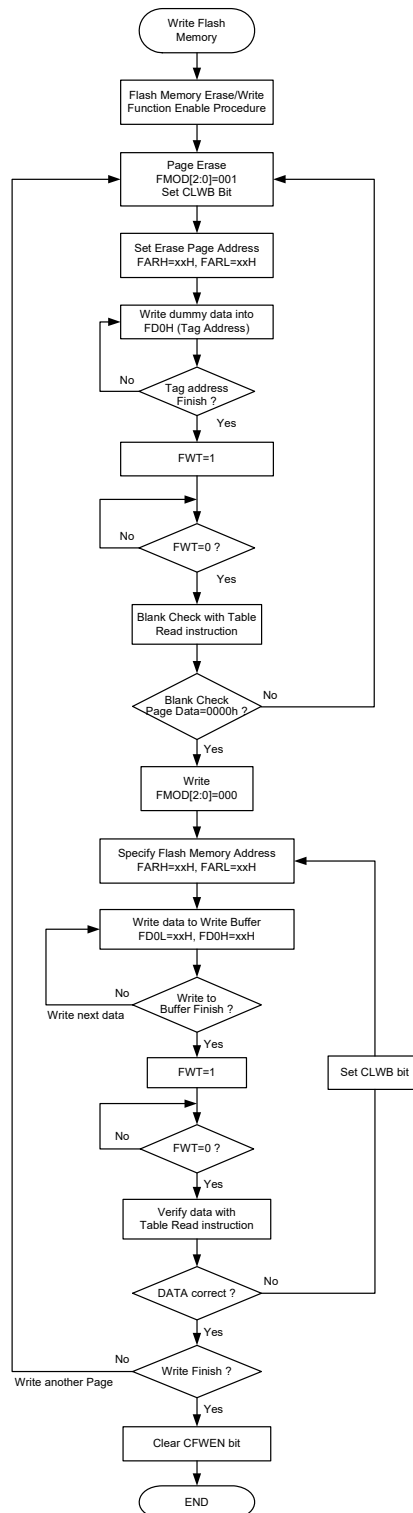
After the Flash memory erase/write function has been successfully enabled as the CFWEN bit is set high, the data to be written into the Flash memory can be loaded into the write buffer. The selected Flash memory page data should be erased by properly configuring the IAP control registers before the data write procedure is executed.

The write buffer size is 32 words, known as a page, whose address is mapped to a specific Flash memory page specified by the memory address bits, FA12~FA5. It is important to ensure that the page where the write buffer data is located is the same one which the memory address bits, FA12~FA5, specify.

Flash Memory Consecutive Write Description

The maximum amount of write data is 32 words for each write operation. The write buffer address will be automatically incremented by one when consecutive write operations are executed. The start address of a specific page should first be written into the FARL and FARH registers. Then the data word should be written into the FD0L register and then the FD0H register. At the same time the write buffer address will be incremented by one and then the next data word can be written into the FD0L and FD0H registers for the next address without modifying the address register pair, FARH and FARL. When the write buffer address reaches the page boundary the address will not be further incremented but will stop at the last address of the page.

1. Activate the “Flash Memory Erase/Write function enable procedure”. Check the CFWEN bit value and then execute the erase/write operations if the CFWEN bit is set high. Refer to the “Flash Memory Erase/Write function enable procedure” for more details.
2. Set the FMOD2~FMOD0 to “001” to select the erase operation and set the CLWB bit high to clear the write buffer. Set the FWT bit high to erase the desired page which is specified by the FARH and FARL registers and has been tagged address. Wait until the FWT bit goes low.
3. Execute a Blank Check operation using the table read instruction to ensure that the erase operation has successfully completed.
Go to step 2 if the erase operation is not successful.
Go to step 4 if the erase operation is successful.
4. Set the FMOD2~FMOD0 to “000” to select the write operation.
5. Setup the desired start address in the FARH and FARL registers. Write the desired data words consecutively into the FD0L and FD0H registers within a page as specified by their consecutive addresses. The maximum written data number is 32 words.
6. Set the FWT bit high to write the data words from the write buffer to the Flash memory. Wait until the FWT bit goes low.
7. Verify the data using the table read instruction to ensure that the write operation has successfully completed.
If the write operation has not successfully completed, set the CLWB bit high to clear the write buffer and then go to step 5.
Go to step 8 if the write operation is successful.
8. Clear the CFWEN bit low to disable the Flash memory erase/write function.



Flash Memory Consecutive Write Procedure

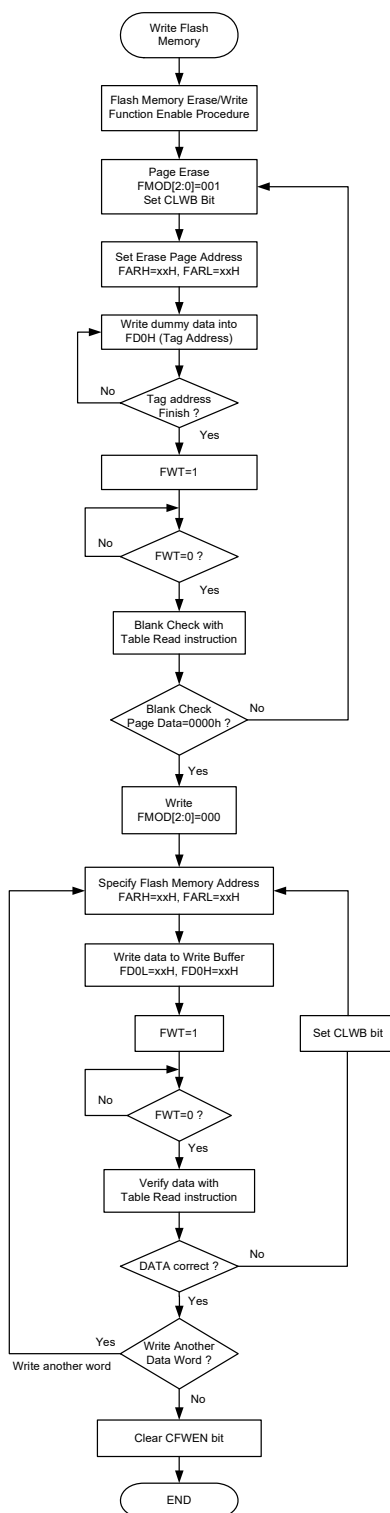
- Note: 1. When the erase or write operation is successfully activated, all CPU operations will temporarily cease.
2. It will take certain time for the FWT bit state changing from high to low in the erase or write operation, which can be selected by the FWERTS bit in the FC2 register.

Flash Memory Non-consecutive Write Description

The main difference between Flash Memory Consecutive and Non-Consecutive Write operations is whether the data words to be written are located in consecutive addresses or not. If the data to be written is not located in consecutive addresses the desired address should be re-assigned after a data word is successfully written into the Flash Memory.

A two data word non-consecutive write operation is taken as an example here and described as follows:

1. Activate the “Flash Memory Erase/Write function enable procedure”. Check the CFWEN bit value and then execute the erase/write operation if the CFWEN bit is set high. Refer to the “Flash Memory Erase/Write function enable procedure” for more details.
2. Set the FMOD2~FMOD0 to “001” to select the erase operation and set the CLWB bit high to clear the write buffer. Set the FWT bit high to erase the desired page which is specified by the FARH and FARL registers and has been tagged address. Wait until the FWT bit goes low.
3. Execute a Blank Check operation using the table read instruction to ensure that the erase operation has successfully completed.
Go to step 2 if the erase operation is not successful.
Go to step 4 if the erase operation is successful.
4. Set the FMOD2~FMOD0 to “000” to select the write operation.
5. Setup the desired address ADDR1 in the FARH and FRARL registers. Write the desired data word DATA1 first into the FD0L register and then into the FD0H register.
6. Set the FWT bit high to transfer the data word from the write buffer to the Flash memory. Wait until the FWT bit goes low.
7. Verify the data using the table read instruction to ensure that the write operation has successfully completed.
If the write operation has not successfully completed, set the CLWB bit high to clear the write buffer and then go to step 5.
Go to step 8 if the write operation is successful.
8. Setup the desired address ADDR2 in the FARH and FARL registers. Write the desired data word DATA2 first into the FD0L register and then into the FD0H register.
9. Set the FWT bit high to transfer the data word from the write buffer to the Flash memory. Wait until the FWT bit goes low.
10. Verify the data using the table read instruction to ensure that the write operation has successfully completed.
If the write operation has not successfully completed, set the CLWB bit high to clear the write buffer and then go to step 8.
Go to step 11 if the write operation is successful.
11. Clear the CFWEN bit low to disable the Flash memory erase/write function.



Flash Memory Non-consecutive Write Procedure

Note: 1. When the erase or write operation is successfully activated, all CPU operations will temporarily cease.

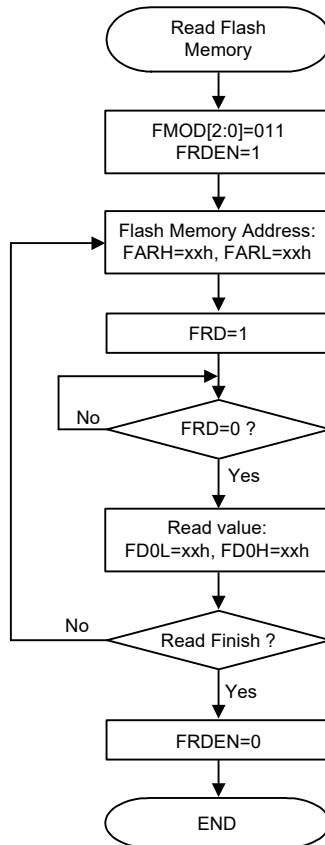
2. It will take certain time for the FWT bit state changing from high to low in the erase or write operation, which can be selected by the FWERTS bit in the FC2 register.

Important Points to Note for Flash Memory Write Operations

1. The “Flash Memory Erase/Write Function Enable Procedure” must be successfully activated before the Flash Memory erase/write operation is executed.
2. The Flash Memory erase operation is executed to erase a whole page.
3. The whole write buffer data will be written into the Flash memory in a page format. The corresponding address cannot exceed the page boundary.
4. After the data is written into the Flash memory the Flash memory contents must be read out using the table read instruction, TABRD, and checked if it is correct or not. If the data written into the Flash memory is incorrect, the write buffer should be cleared by setting the CLWB bit high and then writing the data again into the write buffer. Then activate a write operation on the same Flash memory page without erasing it. The data check, buffer clear and data re-write steps should be repeatedly executed until the data written into the Flash memory is correct.
5. The system frequency should be setup to the maximum application frequency when data write and data check operations are executed using the IAP function.

Flash Memory Read Procedure

To activate the Flash Memory Read procedure, the FMOD2~FMOD0 should be set to “011” to select the Flash memory read mode and the FRDEN bit should be set high to enable the read function. The desired Flash memory address should be written into the FARH and FARL registers and then the FRD bit should be set high. After this the Flash memory read operation will be activated. The data stored in the specified address can be read from the data registers, FD0H and FD0L, when the FRD bit goes low. There is no need to first activate the Flash Memory Erase/Write Function Enable Procedure before the Flash memory read operation is executed.



Flash Memory Read Procedure

- Note: 1. When the read operation is successfully activated, all CPU operations will temporarily cease.
2. It will take a typical time of three instruction cycles for the FRD bit state changing from high to low.

Data Memory

The Data Memory is an 8-bit wide RAM internal memory and is the location where temporary information is stored.

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 device also provides dedicated memory areas for the A/D Converter Auto Mode data storage.

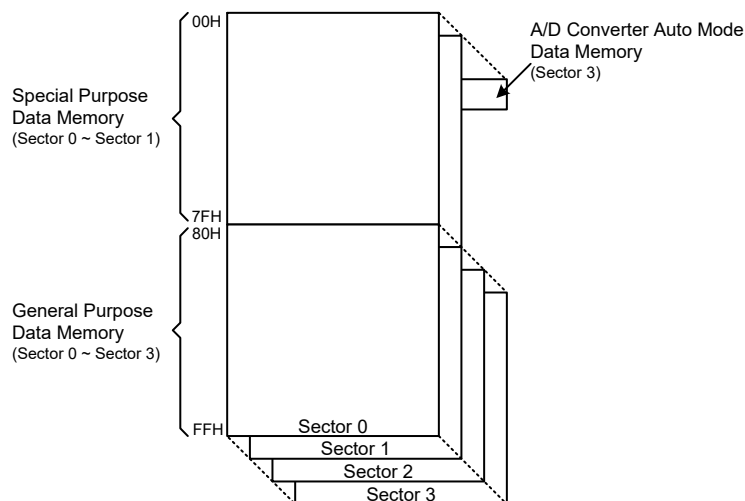
Switching between the different Data Memory sectors is achieved by properly setting the Memory Pointers to correct value when using the indirectly accessing method.

Structure

The Data Memory is subdivided into several sectors, all of which are implemented in 8-bit wide Memory. Each of the Data Memory Sector is categorized into two types, the Special Purpose Data Memory and the General Purpose Data Memory. The address range of the Special Purpose Data Memory for the device is from 00H to 7FH while the General Purpose Data Memory address range is from 80H to FFH. The A/D Converter Auto Mode Data Memory is located from 00H to 3FH in the Sector 3.

| Special Purpose Data Memory | General Purpose Data Memory | | A/D Converter Auto Mode Data Memory |
|--|-----------------------------|--|-------------------------------------|
| Located Sectors | Capacity | Sector: Address | Sector: Address |
| Sector 0: 00H~7FH Sector 1: 00H~7FH | 512×8 | 0: 80H~FFH 1: 80H~FFH 2: 80H~FFH 3: 80H~FFH | Sector 3: 00H~3FH |

Data Memory Summary



Data Memory Structure

Data Memory Addressing

For device that supports the extended instructions, there is no Bank Pointer for Data Memory. For Data Memory the desired Sector is pointed by the MP1H or MP2H register and the certain Data Memory address in the selected sector is specified by the MP1L or MP2L register when using indirect addressing access.

Direct Addressing can be used in all sectors using the corresponding instruction which can address all available data memory space. For the accessed data memory which is located in any data memory sectors except sector 0, the extended instructions can be used to access the data memory instead of using the indirect addressing access. The main difference between standard instructions and extended instructions is that the data memory address “m” in the extended instructions has 10 valid bits for this device, the high byte indicates a sector and the low byte indicates a specific address.


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”.

| Sector 0 | | Sector 1 | Sector 0 | | Sector 1 |
|----------|---------|----------|----------|----------|----------|
| 00H | IAR0 | | 40H | | EEC |
| 01H | MP0 | | 41H | EEAL | |
| 02H | IAR1 | | 42H | EEAH | |
| 03H | MP1L | | 43H | EED | |
| 04H | MP1H | | 44H | PWRC | |
| 05H | ACC | | 45H | IREFC | |
| 06H | PCL | | 46H | PVREF | |
| 07H | TBLP | | 47H | OPAC | |
| 08H | TBLH | | 48H | GSC1 | |
| 09H | TBHP | | 49H | AFEDA1C | |
| 0AH | STATUS | | 4AH | AFEDA1L | |
| 0BH | | | 4BH | AFEDA1H | |
| 0CH | IAR2 | | 4CH | AFEDA2C | |
| 0DH | MP2L | | 4DH | AFEDA2L | |
| 0EH | MP2H | | 4EH | AFEDA2H | |
| 0FH | RSTFC | | 4FH | AFEDA3C | |
| 10H | SCC | | 50H | AFEDA3L | |
| 11H | IRCC | | 51H | AFEDA3H | |
| 12H | STKPTR | | 52H | FBRC | |
| 13H | IECC | | 53H | PGAC | |
| 14H | PA | | 54H | PGACS | |
| 15H | PAC | | 55H | MODC | |
| 16H | PAPU | | 56H | ADRL | |
| 17H | PAWU | | 57H | ADRM | |
| 18H | RSTC | | 58H | ADRH | |
| 19H | LVRC | | 59H | ADCR0 | |
| 1AH | TLVRC | | 5AH | ADCR1 | |
| 1BH | MF10 | | 5BH | | |
| 1CH | MF11 | | 5CH | | |
| 1DH | | | 5DH | ADCS | |
| 1EH | WDTC | | 5EH | | |
| 1FH | INTEG | | 5FH | SINC3 | |
| 20H | INTC0 | | 60H | ADACCTRL | |
| 21H | INTC1 | | 61H | ADRL_AVG | |
| 22H | INTC2 | | 62H | ADRM_AVG | |
| 23H | INTC3 | | 63H | ADRH_AVG | |
| 24H | PB | | 64H | CTRL | |
| 25H | PBC | | 65H | AUTOADCC | |
| 26H | PBPU | | 66H | ERRCHKC | |
| 27H | PCRL | | 67H | ERRCHKR | |
| 28H | PCRH | | 68H | | |
| 29H | | | 69H | | |
| 2AH | | | 6AH | | |
| 2BH | | | 6BH | SPIC0 | |
| 2CH | PSCR | | 6CH | SPIC1 | |
| 2DH | TB0C | | 6DH | SPID | |
| 2EH | TB1C | | 6EH | ORMC | |
| 2FH | USR | | 6FH | CTMC0 | |
| 30H | UCR1 | FC0 | 70H | CTMC1 | |
| 31H | UCR2 | FC1 | 71H | CTMDL | |
| 32H | UCR3 | FC2 | 72H | CTMDH | |
| 33H | BRDH | FARL | 73H | CTMAL | |
| 34H | BRDL | FARH | 74H | CTMAH | |
| 35H | UFCR | FD0L | 75H | | |
| 36H | TXR_RXR | FD0H | 76H | | |
| 37H | RxCNT | FD1L | 77H | CRCCR | |
| 38H | PTMC0 | FD1H | 78H | CRCIN | |
| 39H | PTMC1 | FD2L | 79H | CRCDL | |
| 3AH | PTMDL | FD2H | 7AH | CRCDH | |
| 3BH | PTMDH | FD3L | 7BH | PAS0 | |
| 3CH | PTMAL | FD3H | 7CH | PAS1 | |
| 3DH | PTMAH | IFS | 7DH | PBS0 | |
| 3EH | PTMRPL | | 7EH | | |
| 3FH | PTMRPH | | 7FH | | |

 : Unused, read as 00H

 : Reserved, cannot be changed

Special Purpose Data Memory Structure

Special Function Register Description

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

Indirect Addressing Registers – IAR0, IAR1, IAR2

The Indirect Addressing Registers, IAR0, IAR1 and IAR2, although having their locations in normal RAM register space, do not actually physically exist as normal registers. The method of indirect addressing for RAM data manipulation uses these Indirect Addressing Registers and Memory Pointers, in contrast to direct memory addressing, where the actual memory address is specified. Actions on the IAR0, IAR1 and IAR2 registers will result in no actual read or write operation to these registers but rather to the memory location specified by their corresponding Memory Pointers, MP0, MP1L/MP1H or MP2L/MP2H. Acting as a pair, IAR0 and MP0 can together access data only from Sector 0 while the IAR1 register together with MP1L/MP1H register pair and IAR2 register together with MP2L/MP2H register pair can access data from any Data Memory sector. As the Indirect Addressing Registers are not physically implemented, reading the Indirect Addressing Registers indirectly will return a result of “00H” and writing to the registers indirectly will result in no operation.

Memory Pointers – MP0, MP1H/MP1L, MP2H/MP2L

Five Memory Pointers, known as MP0, MP1L, MP1H, MP2L and MP2H, are provided. These Memory Pointers are physically implemented in the Data Memory and can be manipulated in the same way as normal registers providing a convenient way with which to address and track data. When any operation to the relevant Indirect Addressing Registers is carried out, the actual address that the microcontroller is directed to is the address specified by the related Memory Pointer. MP0, together with Indirect Addressing Register, IAR0, are used to access data from Sector 0, while MP1L/MP1H together with IAR1 and MP2L/MP2H together with IAR2 are used to access data from all data sectors according to the corresponding MP1H or MP2H register. Direct Addressing can be used in all data sectors using the extended instructions which can address all available data memory space.

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

Example 1

```
data .section 'data'
adres1 db ?
adres2 db ?
adres3 db ?
adres4 db ?
block db ?
code .section at 0 code
org 00h
start:
    mov a,04h                ; setup size of block
    mov block,a
    mov a,offset adres1      ; Accumulator loaded with first RAM address
    mov mp0,a                ; setup memory pointer with first RAM address
loop:
    clr IAR0                 ; clear the data at address defined by MP0
    inc mp0                  ; increment memory pointer
```

```
        sdz block                ; check if last memory location has been cleared
        jmp loop
continue:
```

Example 2

```
data .section 'data'
adres1 db ?
adres2 db ?
adres3 db ?
adres4 db ?
block db ?
code .section at 0 'code'
org 00h
start:
    mov a,04h                    ; setup size of block
    mov block,a
    mov a,01h                    ; setup the memory sector
    mov mplh,a
    mov a,offset adres1          ; Accumulator loaded with first RAM address
    mov mpll,a                   ; setup memory pointer with first RAM address
loop:
    clr IAR1                     ; clear the data at address defined by MP1L
    inc mpll                     ; increment memory pointer MP1L
    sdz block                    ; check if last memory location has been cleared
    jmp loop
continue:
:
```

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

Direct Addressing Program Example using extended instructions

```
data .section 'data'
temp db ?
code .section at 0 code
org 00h
start:
    lmov a,[m]                   ; move [m] data to acc
    lsub a, [m+1]                ; compare [m] and [m+1] data
    snz c                        ; [m]>[m+1]?
    jmp continue                ; no
    lmov a,[m]                   ; yes, exchange [m] and [m+1] data
    mov temp,a
    lmov a,[m+1]
    lmov [m],a
    mov a,temp
    lmov [m+1],a
continue:
:
```

Note: Here “m” is a data memory address located in any data memory sectors. For example, m=1F0H, it indicates address 0F0H in Sector 1.

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, TBHP, TBLH

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

Option Memory Mapping Register – ORMC

The ORMC register is used to enable Option Memory Mapping function. The Option Memory capacity is 64 words. When a specific pattern of 55H and AAH is consecutively written into this register, the Option Memory Mapping function will be enabled and then the Option Memory code can be read by using the table read instruction. The Option Memory addresses 00H~3FH will be mapped to the Program Memory last page addresses C0H~FFH.

To successfully enable the Option Memory Mapping function, the specific pattern of 55H and AAH must be written into the ORMC register in two consecutive instruction cycles. It is therefore recommended that the global interrupt bit EMI should first be cleared before writing the specific pattern, and then set high again at a proper time according to users' requirements after the pattern is successfully written. An internal timer will be activated when the pattern is successfully written. The mapping operation will be automatically finished after a period of $4 \times t_{LIRC}$. Therefore, users should read the data in time, otherwise the Option Memory Mapping function needs to be restarted. After the completion of each consecutive write operation to the ORMC register, the timer will recount.

When the table read instructions are used to read the Option Memory code, both “TABRD [m]” and “TABRDL [m]” instructions can be used. However, care must be taken if the “TABRD [m]” instruction is used, the table pointer defined by the TBHP register must be referenced to the last page. Refer to corresponding sections about the table read instruction for more details.

• **ORMC Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|-------|-------|-------|-------|-------|-------|-------|
| Name | ORMC7 | ORMC6 | ORMC5 | ORMC4 | ORMC3 | ORMC2 | ORMC1 | ORMC0 |
| 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 **ORMC7~ORMC0**: Option Memory Mapping specific pattern

When a specific pattern of 55H and AAH is written into this register, the Option Memory Mapping function will be enabled. Note that the register content will be cleared after the MCU is woken up from the IDLE/SLEEP mode.

Status Register – STATUS

This 8-bit register contains the zero flag (Z), carry flag (C), auxiliary carry flag (AC), overflow flag (OV), SC flag, CZ flag, 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, C, SC and CZ 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.
- SC is the result of the “XOR” operation which is performed by the OV flag and the MSB of the current instruction operation result.
- CZ is the operational result of different flags for different instructions. Refer to register definitions for more details.

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 content of the status register is 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 | SC | CZ | TO | PDF | OV | Z | AC | C |
| R/W | R/W | R/W | R | R | R/W | R/W | R/W | R/W |
| POR | x | x | 0 | 0 | x | x | x | x |

"x": unknown

- Bit 7 **SC**: The result of the "XOR" operation which is performed by the OV flag and the MSB of the instruction operation result
- Bit 6 **CZ**: The operational result of different flags for different instructions
For SUB/SUBM/LSUB/LSUBM instructions, the CZ flag is equal to the Z flag.
For SBC/SBCM/LSBC/LSBCM instructions, the CZ flag is the "AND" operation result which is performed by the previous operation CZ flag and current operation Z flag. For other instructions, the CZ flag will not be affected.
- 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.

EEPROM Data Memory

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

EEPROM Data Memory Structure

The EEPROM Data Memory capacity is 512×8 bits for this device. Unlike the Program Memory and RAM Data Memory, the EEPROM Data Memory is not directly mapped into memory space and is therefore not directly addressable in the same way as the other types of memory. Read and Write operations to the EEPROM are carried out in either the byte mode or page mode determined by the mode selection bit, MODE, in the control register, EEC.

EEPROM Registers

Four registers control the overall operation of the internal EEPROM Data Memory. These are the address registers, EEAL and EEAH, the data register, EED and a single control register, EEC. As the EEAL, EEAH and EED registers are located in Sector 0, they can be directly accessed in the same way as any other Special Function Register. The EEC register, however, being located in Sector 1, can only be read from or written to indirectly using the MP1H/MP1L or MP2H/MP2L Memory Pointer pair and Indirect Addressing Register, IAR1 or IAR2. Because the EEC control register is located at address 40H in Sector 1, the Memory Pointer low byte register, MP1L or MP2L, must first be set to the value 40H and the Memory Pointer high byte register, MP1H or MP2H, set to the value, 01H, before any operations on the EEC register are executed.

| Register Name | Bit | | | | | | | |
|---------------|--------|-------|-------|-------|-------|-------|-------|-------|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| EEAL | EEAL7 | EEAL6 | EEAL5 | EEAL4 | EEAL3 | EEAL2 | EEAL1 | EEAL0 |
| EEAH | — | — | — | — | — | — | — | EEAH0 |
| EED | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| EEC | EWERTS | EREN | ER | MODE | WREN | WR | RDEN | RD |

EEPROM Register List

• EEAL Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|-------|-------|-------|-------|-------|-------|-------|
| Name | EEAL7 | EEAL6 | EEAL5 | EEAL4 | EEAL3 | EEAL2 | EEAL1 | EEAL0 |
| 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 **EEAL7~EEAL0**: Data EEPROM address low byte bit 7 ~ bit 0

• EEAH Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|---|---|---|---|---|-------|
| Name | — | — | — | — | — | — | — | EEAH0 |
| R/W | — | — | — | — | — | — | — | R/W |
| POR | — | — | — | — | — | — | — | 0 |

Bit 7~1 Unimplemented, read as “0”

Bit 0 **EEAH0**: Data EEPROM address high byte bit 0

• EED Register

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

Bit 7~0 **D7~D0**: Data EEPROM data bit 7 ~ bit 0

• EEC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|--------|------|-----|------|------|-----|------|-----|
| Name | EWERTS | EREN | ER | MODE | WREN | WR | RDEN | RD |
| 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 **EWERTS**: Data EEPROM Erase time and Write time select

0: Erase time is 3.2ms (t_{EEER}) / Write time is 2.2ms (t_{EEWR})

1: Erase time is 3.7ms (t_{EEER}) / Write time is 3.0ms (t_{EEWR})

Bit 6 **EREN**: Data EEPROM erase enable

0: Disable

1: Enable

This bit is used to enable Data EEPROM erase function and must be set high before Data EEPROM erase operations are carried out. This bit will be automatically reset to zero by the hardware after the erase cycle has finished. Clearing this bit to zero will inhibit data EEPROM erase operations.

Bit 5 **ER**: Data EEPROM erase control

0: Erase cycle has finished

1: Activate an erase cycle

This is the Data EEPROM Erase Control Bit. When this bit is set high by the application program, an erase cycle will be activated. This bit will be automatically reset to zero by hardware after the erase cycle has finished. Setting this bit high will have no effect if the EREN bit has not first been set high.

Bit 4 **MODE**: Data EEPROM operation mode selection

0: Byte operation mode

1: Page operation mode

This is the EEPROM operation mode selection bit. When the bit is set high by the application program, the Page write, erase or read function will be selected. Otherwise, the byte write or read function will be selected. The EEPROM page buffer size is 16-byte.

Bit 3 **WREN**: Data EEPROM write enable

0: Disable

1: Enable

This is the Data EEPROM Write Enable bit which must be set high before Data EEPROM write operations are carried out. Clearing this bit to zero will inhibit Data EEPROM write operations. Note that the WREN bit will automatically be cleared to zero after the write operation is finished.

Bit 2 **WR**: Data EEPROM write control

0: Write cycle has finished

1: Activate a write cycle

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

Bit 1 **RDEN**: Data EEPROM read enable

0: Disable

1: Enable

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

Bit 0 **RD**: Data EEPROM read control

0: Read cycle has finished

1: Activate a read cycle

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

Note: 1. The EREN, ER, WREN, WR, RDEN and RD cannot be set to “1” at the same time in one instruction.

2. Ensure that the f_{SUB} clock is stable before executing the erase or write operation.

3. Ensure that the erase or write operation is totally complete before changing contents of the EEPROM related registers or activating the IAP function.

Read Operation from the EEPROM

Reading data from the EEPROM can be implemented by two modes for this device, byte read mode or page read mode, which is controlled by the EEPROM operation mode selection bit, MODE, in the EEC register.

Byte Read Mode

The EEPROM byte read operation can be executed when the mode selection bit, MODE, is cleared to zero. For a byte read operation the desired EEPROM address should first be placed in the EEAH and EEAL registers, as well as the read enable bit, RDEN, in the EEC register should be set high to enable the read function. Then setting the RD bit high will initiate the EEPROM byte read operation. Note that setting only the RD bit high will not initiate a read operation if the RDEN bit is not set high. When the read cycle terminates, the RD bit will automatically be cleared to zero and the EEPROM data can be read from the EED register. The data will remain in the EED register until another read or write operation is executed. The application program can poll the RD bit to determine when the data is valid for reading.

Page Read Mode

The EEPROM page read operation can be executed when the mode selection bit, MODE, is set high. The page size can be up to 16 bytes for the page read operation. For a page read operation the start address of the desired EEPROM page should first be placed in the EEAH and EEAL registers, as well as the read enable bit, RDEN, in the EEC register should be set high to enable the read function. Then setting the RD bit high will initiate the EEPROM page read operation. Note that setting only the RD bit high will not initiate a read operation if the RDEN bit is not set high. When the current byte read cycle terminates, the RD bit will automatically be cleared to zero indicating that the EEPROM data can be read from the EED register and then the current address will be incremented by one by hardware. The data which is stored in the next EEPROM address can continuously be read out when the RD bit is set high again without reconfiguring the EEPROM address and RDEN control bit. The application program can poll the RD bit to determine when the data is valid for reading.

The EEPROM address higher 5 bits are used to specify the desired page location while the lower 4 bits are used to point to the actual address. In the page read operation mode the lower 4-bit address value will automatically be incremented by one. However, the higher 5-bit address value will not be incremented by hardware. When the EEPROM address lower 4-bit value which is internally incremented by one in the page mode reaches the page boundary, known as 0FH, the EEPROM address lower 4-bit value will stop at 0FH. The EEPROM address will not “roll over”.

Page Erase Operation to the EEPROM

The EEPROM page erase operation can be executed when the mode selection bit, MODE, is set high. The EEPROM is capable of a 16-byte page erase. The internal page buffer will be cleared by hardware after power on reset. When the EEPROM erase enable control bit, namely EREN, is changed from “1” to “0”, the internal page buffer will also be cleared. Note that when the EREN bit is changed from “0” to “1”, the internal page buffer will not be cleared. The EEPROM address higher 5 bits are used to specify the desired page location while the lower 4 bits are used to point to the actual address. In the page erase operation mode the lower 4-bit address value will automatically be incremented by one after each dummy data byte is written into the EED register. However, the higher 5-bit address value will not be incremented by hardware. When the EEPROM address lower 4-bit value which is internally incremented by one in the page mode reaches the page boundary, known as 0FH, the EEPROM address lower 4-bit value will stop at 0FH. The EEPROM address will not “roll over”.

For page erase operations the start address of the desired EEPROM page should first be placed in the EEAH and EEAL registers, then the dummy data to be written should be placed in the EED register. The maximum data length for a page is 16 bytes. Note that the write operation to the EED register is used to tag address, it must be implemented to determine which addresses to be erased. When the page dummy data is completely written, then the EREN bit in the EEC register should be set high to enable erase operations and the ER bit must be immediately set high to initiate the EEPROM erase process. These two instructions must be executed in two consecutive instruction cycles to activate an erase operation successfully. The global interrupt enable bit EMI should also first be cleared before implementing an erase operation and then set again after a valid erase activation procedure has completed.

Note: The above steps must be executed sequentially to successfully complete the page erase operation, refer to the corresponding programming example.

As the EEPROM erase cycle is controlled using an internal timer whose operation is asynchronous to microcontroller system clock, a certain time will elapse before the data will have been erased from the EEPROM. Detecting when the erase cycle has finished can be implemented either by polling the ER bit in the EEC register or by using the EEPROM interrupt. When the erase cycle terminates, the ER bit will be automatically cleared to zero by the microcontroller, indicating that the page data has been erased. The application program can therefore poll the ER bit to determine when the erase cycle has ended. After the erase operation is finished, the EREN bit will be cleared to zero by hardware. The Data EEPROM erased page content will all be zero after a page erase operation.

Write Operation to the EEPROM

Writing data to the EEPROM can be implemented by two modes for this device, byte write mode or page write mode, which is controlled by the EEPROM operation mode selection bit, MODE, in the EEC register.

Byte Write Mode

The EEPROM byte write operation can be executed when the mode selection bit, MODE, is cleared to zero. For byte write operations the desired EEPROM address should first be placed in the EEAH and EEAL registers, then the data to be written should be placed in the EED register. To write data to the EEPROM, the write enable bit, WREN, in the EEC register must first be set high to enable the write function. After this, the WR bit in the EEC register must be immediately set high to initiate a write cycle. These two instructions must be executed in two consecutive instruction cycles to activate a write operation successfully. The global interrupt bit EMI should also first be cleared before implementing any write operations, and then set high again after a valid write activation

procedure has completed. Note that setting the WR bit high only will not initiate a write cycle if the WREN bit is not set.

Note: The above steps must be executed sequentially to successfully complete the byte write operation, refer to the corresponding programming example.

As the EEPROM write cycle is controlled using an internal timer whose operation is asynchronous to microcontroller system clock, a certain time will elapse before the data will have been written into the EEPROM. Detecting when the write cycle has finished can be implemented either by polling the WR bit in the EEC register or by using the EEPROM interrupt. When the write cycle terminates, the WR bit will be automatically cleared to zero by the microcontroller, indicating that the data has been written to the EEPROM. The application program can therefore poll the WR bit to determine when the write cycle has ended. After the write operation is finished, the WREN bit will be cleared to zero by hardware. Note that a byte erase operation will automatically be executed before a byte write operation is successfully activated.

Page Write Mode

Before a page write operation is executed, it is important to ensure that a relevant page erase operation has been successfully executed. The EEPROM page write operation can be executed when the mode selection bit, MODE, is set high. The EEPROM is capable of a 16-byte page write. The internal page buffer will be cleared by hardware after power on reset. When the EEPROM write enable control bit, namely WREN, is changed from “1” to “0”, the internal page buffer will also be cleared. Note that when the WREN bit is changed from “0” to “1”, the internal page buffer will not be cleared. A page write is initiated in the same way as a byte write initiation except that the EEPROM data can be written up to 16 bytes. The EEPROM address higher 5 bits are used to specify the desired page location while the lower 4 bits are used to point to the actual address. In the page write operation mode the lower 4-bit address value will automatically be incremented by one after each data byte is written into the EED register. However, the higher 5-bit address value will not be incremented by hardware. When the EEPROM address lower 4-bit value which is internally incremented by one in the page mode reaches the page boundary, known as 0FH, the EEPROM address lower 4-bit value will stop at 0FH. The EEPROM address will not “roll over”. At this point any data write operations to the EED register will be invalid.

For page write operations the start address of the desired EEPROM page should first be placed in the EEAH and EEAL registers, then the data to be written should be placed in the EED register. The maximum data length for a page is 16 bytes. Note that when a data byte is written into the EED register, then the data in the EED register will be loaded into the internal page buffer and the current address value will automatically be incremented by one. When the page data is completely written into the page buffer, then the WREN bit in the EEC register should be set high to enable write operations and the WR bit must be immediately set high to initiate the EEPROM write process. These two instructions must be executed in two consecutive instruction cycles to activate a write operation successfully. The global interrupt enable bit EMI should also first be cleared before implementing any write operations, and then set high again after a valid write activation procedure has completed. Note that setting the WR bit high only will not initiate a write cycle if the WREN bit is not set.

Note: The above steps must be executed sequentially to successfully complete the page write operation, refer to the corresponding programming example.

As the EEPROM write cycle is controlled using an internal timer whose operation is asynchronous to microcontroller system clock, a certain time will elapse before the data will have been written into the EEPROM. Detecting when the write cycle has finished can be implemented either by polling the WR bit in the EEC register or by using the EEPROM interrupt. When the write cycle terminates, the WR bit will be automatically cleared to zero by the microcontroller, indicating that the data has been

written to the EEPROM. The application program can therefore poll the WR bit to determine when the write cycle has ended. After the write operation is finished, the WREN bit will be cleared to zero by hardware.

Write Protection

Protection against inadvertent write operation is provided in several ways. After the device is powered-on the Write Enable bit in the control register will be cleared preventing any write operations. Also at power-on the Memory Pointer high byte register, MP1H or MP2H, will be reset to zero, which means that Data Memory Sector 0 will be selected. As the EEPROM control register is located in Sector 1, this adds a further measure of protection against spurious write operations. During normal program operation, ensuring that the Write Enable bit in the control register is cleared will safeguard against incorrect write operations.

EEPROM Interrupt

The EEPROM interrupt is generated when an EEPROM erase or write cycle has ended. The EEPROM interrupt must first be enabled by setting the DEE bit in the relevant interrupt register. However as the EEPROM interrupt is contained within a Multi-function Interrupt, the associated multi-function interrupt enable bit must also be set. When an EEPROM erase or write cycle ends, the DEF request flag and its associated multi-function interrupt request flag will both be set. If the global, EEPROM and Multi-function interrupts are enabled and the stack is not full, a jump to the associated Multi-function Interrupt vector will take place. When the interrupt is serviced only the Multi-function interrupt flag will be automatically reset, the EEPROM interrupt flag must be manually reset by the application program. More details can be obtained in the Interrupt section.

Programming Considerations

Care must be taken that data is not inadvertently written to the EEPROM. Protection can be enhanced by ensuring that the Write Enable bit is normally cleared to zero when not writing. Also the Memory Pointer high byte register, MP1H or MP2H, could be normally cleared to zero as this would inhibit access to Sector 1 where the EEPROM control register exists. Although certainly not necessary, consideration might be given in the application program to the checking of the validity of new write data by a simple read back process.

When writing data the WR bit must be set high immediately after the WREN bit has been set high, to ensure the write cycle executes correctly. When erasing data the ER bit must be set high immediately after the EREN bit has been set high, to ensure the erase cycle executes correctly. The global interrupt bit EMI should also be cleared before a write or erase cycle is executed and then set again after a valid write or erase activation procedure has completed. Note that the device should not enter the IDLE or SLEEP mode until the EEPROM read/write/erase operation is totally complete. Otherwise, the EEPROM read/write/erase operation will fail.

Programming Examples

Reading a Data Byte from the EEPROM – polling method

```
MOV A, 40H          ; setup memory pointer lower byte MP1L
MOV MP1L, A         ; MP1L points to EEC register
MOV A, 01H          ; setup memory pointer high byte MP1H
MOV MP1H, A
CLR IAR1.4          ; clear MODE bit, select byte operation mode
MOV A, EEPROM_ADRES_H ; user defined high byte address
MOV EEAH, A
MOV A, EEPROM_ADRES_L ; user defined low byte address
```

```
MOV EEAL, A
SET IAR1.1          ; set RDEN bit, enable read operations
SET IAR1.0          ; start Read Cycle - set RD bit
BACK:
SZ IAR1.0           ; check for read cycle end
JMP BACK
CLR IAR1            ; disable EEPROM read function
CLR MP1H
MOV A, EED          ; move read data to register
MOV READ_DATA, A
```

Reading a Data Page from the EEPROM – polling method

```
MOV A, 40H          ; set memory pointer low byte MP1L
MOV MP1L, A         ; MP1L points to EEC register
MOV A, 01H          ; set memory pointer high byte MP1H
MOV MP1H, A
SET IAR1.4          ; set MODE bit, select page operation mode
MOV A, EEPROM_ADRES_H ; user defined high byte address
MOV EEAH, A
MOV A, EEPROM_ADRES_L ; user defined low byte address
MOV EEAL, A
SET IAR1.1          ; set RDEN bit, enable read operations
; ~~~~ The data length can be up to 16 bytes (Start) ~~~~
CALL READ
CALL READ
:
:
JMP PAGE_READ_FINISH
; ~~~~ The data length can be up to 16 bytes (End) ~~~~
READ:
SET IAR1.0          ; start Read Cycle - set RD bit
BACK:
SZ IAR1.0           ; check for read cycle end
JMP BACK
MOV A, EED          ; move read data to register
MOV READ_DATA, A
RET
:
PAGE_READ_FINISH:
CLR IAR1            ; disable EEPROM read function
CLR MP1H
```

Erasing a Data Page to the EEPROM – polling method

```
MOV A, 40H          ; set memory pointer low byte MP1L
MOV MP1L, A         ; MP1L points to EEC register
MOV A, 01H          ; set memory pointer high byte MP1H
MOV MP1H, A
SET IAR1.4          ; set MODE bit, select page operation mode
MOV A, EEPROM_ADRES_H ; user defined high byte address
MOV EEAH, A
MOV A, EEPROM_ADRES_L ; user defined low byte address
MOV EEAL, A
; ~~~~ The data length can be up to 16 bytes (Start) ~~~~
CALL WRITE_BUF
CALL WRITE_BUF
:
:
```

```

JMP Erase_START
; ~~~~ The data length can be up to 16 bytes (End) ~~~~
WRITE_BUF:
MOV A, EEPROM_DATA      ; user defined data, erase mode don't care data value
MOV EED, A
RET
:
Erase_START:
CLR EMI
SET IAR1.6               ; set EREN bit, enable write operations
SET IAR1.5               ; start Write Cycle - set ER bit - executed immediately
                        ; after setting EREN bit

SET EMI
BACK:
SZ IAR1.5                ; check for write cycle end
JMP BACK
CLR MP1H

```

Writing a Data Byte to the EEPROM – polling method

```

MOV A, 40H               ; set memory pointer low byte MP1L
MOV MP1L, A              ; MP1L points to EEC register
MOV A, 01H               ; set memory pointer high byte MP1H
MOV MP1H, A
CLR IAR1.4               ; clear MODE bit, select byte write mode
MOV A, EEPROM_ADRES      ; user defined high byte address
MOV EEAH, A
MOV A, EEPROM_ADRES      ; user defined low byte address
MOV EEAL, A
MOV A, EEPROM_DATA       ; user defined data
MOV EED, A
CLR EMI
SET IAR1.3               ; set WREN bit, enable write operations
SET IAR1.2               ; start Write Cycle - set WR bit - executed immediately
                        ; after setting WREN bit

SET EMI
BACK:
SZ IAR1.2                ; check for write cycle end
JMP BACK
CLR MP1H

```

Writing a Data Page to the EEPROM – polling method

```

MOV A, 40H               ; set memory pointer low byte MP1L
MOV MP1L, A              ; MP1L points to EEC register
MOV A, 01H               ; set memory pointer high byte MP1H
MOV MP1H, A
SET IAR1.4               ; set MODE bit, select page operation mode
MOV A, EEPROM_ADRES_H    ; user defined high byte address
MOV EEAH, A
MOV A, EEPROM_ADRES_L    ; user defined low byte address
MOV EEAL, A
; ~~~~ The data length can be up to 16 bytes (Start) ~~~~
CALL WRITE_BUF
CALL WRITE_BUF
:
:
JMP WRITE_START
; ~~~~ The data length can be up to 16 bytes (End) ~~~~

```

```
WRITE_BUF:
MOV  A, EEPROM_DATA      ; user define data
MOV  EED, A
RET
:
WRITE_START:
CLR  EMI
SET  IAR1.3              ; set WREN bit, enable write operations
SET  IAR1.2              ; start Write Cycle - set WR bit - executed immediately
                                ; after setting WREN bit

SET  EMI
BACK:
SZ   IAR1.2              ; check for write cycle end
JMP  BACK
CLR  MPlH
```

Oscillators

Various oscillator types 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 selections and operation are selected through the application program by using some 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 Interrupts. Fully integrated internal oscillators, requiring no external components, are provided to form a wide range of both fast and slow system oscillators. The higher frequency oscillators provide higher performance but carry with it the disadvantage of higher power requirements, while the opposite is of course true for the lower frequency 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.

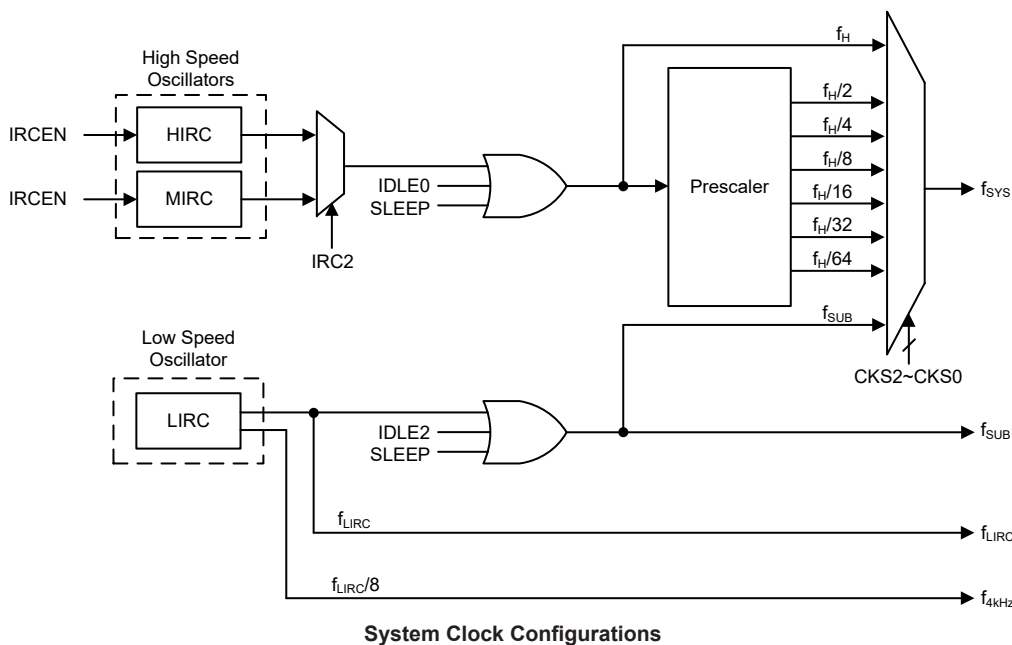
| Type | Name | Frequency |
|--------------------------|------|-----------|
| Internal High Speed RC | HIRC | 4MHz |
| Internal Medium Speed RC | MIRC | 400kHz |
| Internal Low Speed RC | LIRC | 32.768kHz |

Oscillator Types

System Clock Configurations

There are three oscillator sources, two high speed oscillators and one low speed oscillator. The high speed oscillators are the internal 4MHz RC oscillator, HIRC, and the internal 400kHz RC oscillator, MIRC. The low speed oscillator is the internal 32.768kHz 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 as the system clock can be dynamically selected.

The actual source clock used for the high speed oscillator the clock source is selected by the IRC2 bit in the IRCC register. The frequency of the slow speed or high speed system clock is determined using the CKS2~CKS0 bits in the SCC register. Note that two oscillator selections must be made namely one high speed and one low speed system oscillators.



System Clock Configurations

Internal High Speed RC Oscillator – HIRC

The high speed internal RC oscillator is one of the high frequency oscillator choices, which is selected by the IRC2 bit in the IRCC register. It is a fully integrated system oscillator requiring no external components. The internal high RC oscillator has a 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 Medium Speed RC Oscillator – MIRC

The medium speed internal RC oscillator is one of the high frequency oscillator choices, which is selected by the the IRC2 bit in the IRCC register. It is a fully integrated system oscillator requiring no external components. The internal medium RC oscillator has a fixed frequency of 400kHz. 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 32.768kHz Oscillator – LIRC

The Internal 32.768kHz System Oscillator is a fully integrated RC oscillator with a typical frequency of 32.768kHz, 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.

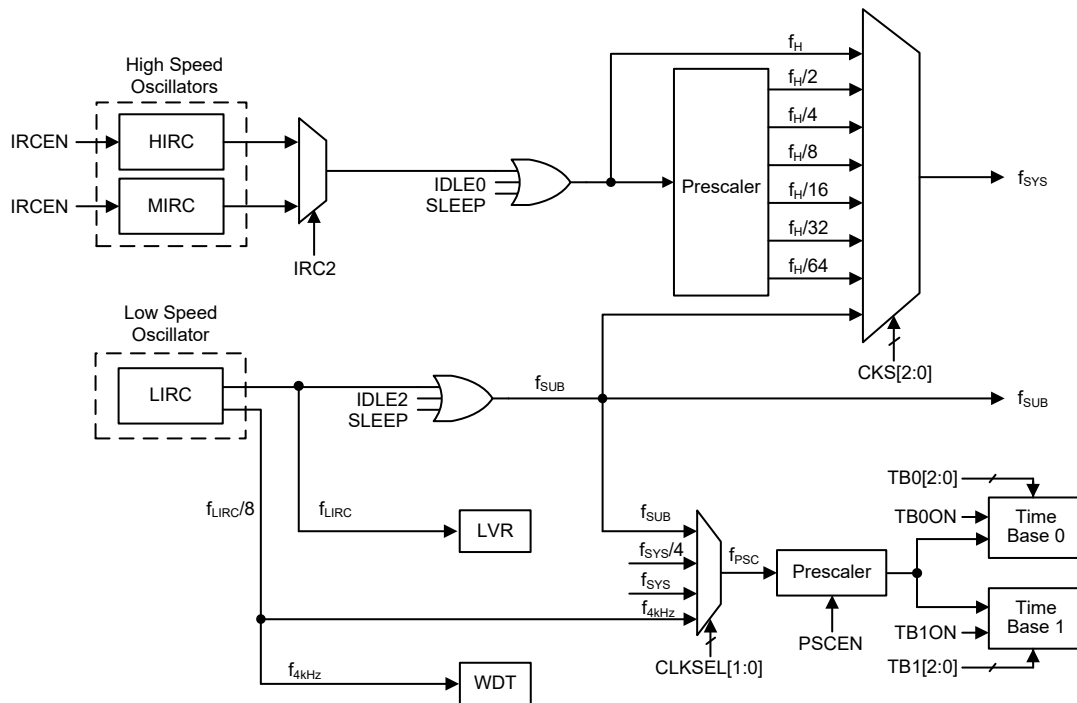
Operating Modes and System Clocks

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

System Clocks

The device has many different clock sources for both the CPU and peripheral function operation. By providing the user with a wide range of clock selections 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 can be sourced from either the HIRC or the MIRC oscillator, selected via configuring the IRC2 bit in the IRCC register. The low speed system clock source can be sourced from the internal clock f_{SUB} . If f_{SUB} is selected then it can be sourced by the LIRC oscillator. The other choice, which is a divided version of the high speed system oscillator has a range of $f_H/2 \sim f_H/64$.



Device Clock Configurations

Note: When the system clock source f_{SYS} is switched to f_{SUB} from f_H , the high speed oscillation can be stopped to conserve the power or continue to oscillate to provide the clock source, $f_H \sim f_H/64$, for peripheral circuits 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 microcontroller, each one with its own special characteristics and which can be chosen according to the specific performance and

power requirements of the application. There are two modes allowing normal operation of the microcontroller, the 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} | f_{4kHz} |
|----------------|-----|------------------|--------|-----------|-------------------|-----------------------|-----------|------------|-----------------------|
| | | FHIDEN | FSIDEN | CKS2~CKS0 | | | | | |
| FAST | On | x | x | 000~110 | $f_H \sim f_H/64$ | On | On | On | On/Off ⁽²⁾ |
| SLOW | On | x | x | 111 | f_{SUB} | On/Off ⁽¹⁾ | On | On | On/Off ⁽²⁾ |
| IDLE0 | Off | 0 | 1 | 000~110 | Off | Off | On | On | On/Off ⁽²⁾ |
| | | | | 111 | On | | | | |
| IDLE1 | Off | 1 | 1 | xxx | On | On | On | On | On/Off ⁽²⁾ |
| IDLE2 | Off | 1 | 0 | 000~110 | On | On | Off | On | On/Off ⁽²⁾ |
| | | | | 111 | Off | | | | |
| SLEEP | Off | 0 | 0 | xxx | Off | 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_{4kHz} clock will be switched on when the Time Base prescaler or WDT is enabled; the f_{4kHz} clock will be switched off when both the Time Base prescaler and the WDT are disabled.

FAST Mode

This is one of the main operating modes where the microcontroller has all of its functions operational and where the system clock is provided by the internal high speed oscillator. This mode operates allowing the microcontroller to operate normally with a clock source which will come from one of the high speed oscillators, either the HIRC or MIRC oscillator, selected via configuring the IRC2 bit in the IRCC register. 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 microcontroller at a divided clock ratio reduces the operating current.

SLOW Mode

This is also a mode where the microcontroller operates normally although now with a slower speed clock source. The clock source used will be from f_{SUB} . The f_{SUB} clock is derived from the LIRC oscillator.

SLEEP Mode

The SLEEP Mode is entered when a HALT instruction is executed and when the FHIDEN and FSIDEN bits in the SCC register are both low. In the SLEEP mode the CPU will be stopped. The f_{SUB} clock provided to the peripheral function will also be stopped. However, the f_{4kHz} clock will continue to operate if the Time Base prescaler or WDT function is enabled. The LIRC will be turned on to provide the clock source for Time Base prescaler and WDT functions.

IDLE0 Mode

The IDLE0 Mode is entered when a 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 on to drive some peripheral functions.

IDLE1 Mode

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

IDLE2 Mode

The IDLE2 Mode is entered when a 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 on to provide a clock source to keep some peripheral functions operational.

Control Registers

The SCC and IRCC registers are used to control the system clock and the corresponding oscillator configurations.

| Register Name | Bit | | | | | | | |
|---------------|------|------|------|------|---|---|--------|--------|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SCC | CKS2 | CKS1 | CKS0 | — | — | — | FHIDEN | FSIDEN |
| IRCC | — | — | — | IRC2 | — | — | IRCF | IRCEN |

System Operating Mode Control Register List

• SCC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|------|------|------|---|---|---|--------|--------|
| Name | CKS2 | CKS1 | CKS0 | — | — | — | FHIDEN | FSIDEN |
| R/W | R/W | R/W | R/W | — | — | — | R/W | R/W |
| POR | 1 | 1 | 1 | — | — | — | 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 4~2 Unimplemented, read as “0”

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 a “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 a “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 t_{SYS} indicates the current system clock period.

• IRCC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|---|------|---|---|------|-------|
| Name | — | — | — | IRC2 | — | — | IRCF | IRCEN |
| R/W | — | — | — | R/W | — | — | R | R/W |
| POR | — | — | — | 0 | — | — | 0 | 0 |

Bit 7~5 Unimplemented, read as “0”

Bit 4 **IRC2**: HIRC or MIRC frequency selection

0: 4MHz

1: 400kHz

When the HIRC or MIRC oscillator is enabled or the HIRC or MIRC frequency selection is changed by application program, the clock frequency will automatically be changed after the IRCF flag is set high.

Bit 3~2 Unimplemented, read as “0”

Bit 1 **IRCF**: HIRC or MIRC oscillator stable flag

0: Unstable

1: Stable

This bit is used to indicate whether the HIRC or MIRC oscillator is stable or not. When the IRCEN bit is set to 1 to enable the HIRC or MIRC oscillator or MIRC frequency selection is changed by application program, the IRCF bit will first be cleared to 0 and then set to 1 after the HIRC or MIRC oscillator is stable.

Bit 0 **IRCEN**: HIRC or MIRC oscillator enable control

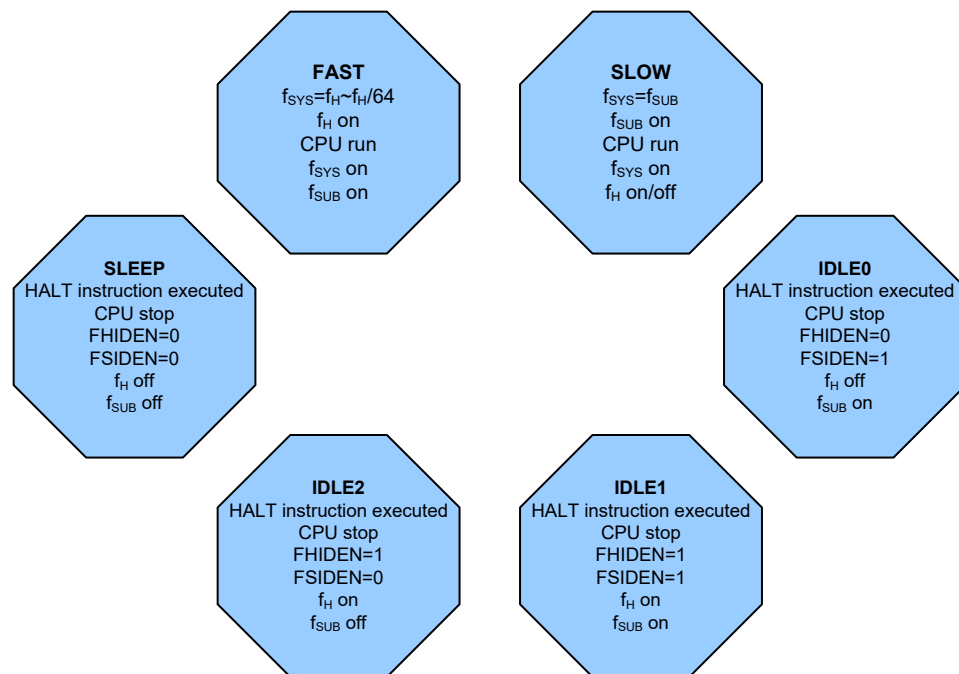
0: Disable

1: Enable

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.

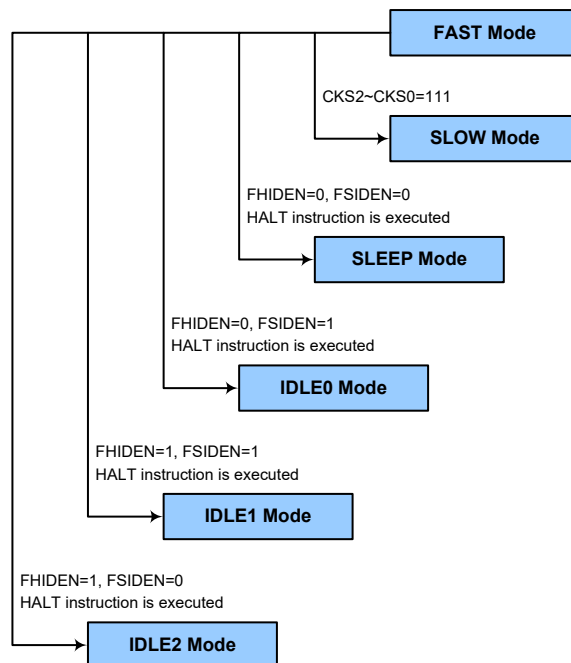
In simple terms, Mode Switching between the FAST Mode and SLOW Mode is executed using the CKS2~CKS0 bits in the SCC register while Mode Switching from the FAST/SLOW Modes to the SLEEP/IDLE Modes is executed via the HALT instruction. When a HALT instruction is executed, whether the device enters the IDLE 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 setting 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.

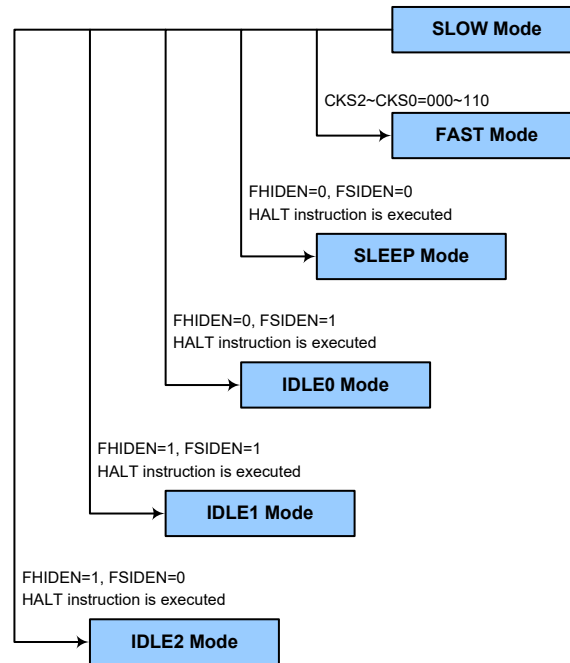
The SLOW Mode is sourced from the LIRC oscillator and therefore requires this oscillator to be stable before full mode switching occurs.



SLOW Mode to FAST Mode Switching

In 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 $f_H \sim f_H/64$.

However, if f_H is not used in 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 IRCF bit in the IRCC register. The time duration required for the high speed system oscillator stabilization 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 or Time Base 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.
- All the clock will be stopped except the f_{4kHz} clock which will be on if the WDT or Time Base function is enabled. The LIRC oscillator will be turned on and provides the clock source.
- 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 WDT timeout 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 f_{4kHz} clock will be on if the WDT or Time Base function is enabled. The LIRC oscillator will be turned on and provides the clock source.
- 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 WDT timeout 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 f_{4kHz} clock will be on if the WDT or Time Base function is enabled. The LIRC oscillator will be turned on and provides the clock source.
- 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 WDT timeout 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 f_{4kHz} clock will be on if the WDT or Time Base function is enabled. The LIRC oscillator will be turned on and provides the clock source.
- 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 WDT timeout 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 setup as outputs or if setup as inputs must have pull-high resistors connected.

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

When the device executes the “HALT” instruction, it will enter the SLEEP or IDLE mode and the PDF flag will be set high. The PDF flag will be cleared to zero if the device experiences a system power-up or executes the clear Watchdog Timer instruction. If the device 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, the other flags remain in their original status.

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

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_{4kHz} , which is sourced from the LIRC oscillator. The LIRC internal oscillator has an approximate frequency of 32.768kHz 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 to 2^{18} 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 Watchdog Timer the enable/disable and the MCU reset operation.

• WDTC Register

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

Bit 7~3 **WE4~WE0**: WDT function enable control

10101: Disable

01010: Enable

Other values: Reset MCU

When these bits are changed to any other values due to environmental noise the microcontroller will be reset; this reset operation will be activated after a delay time, t_{SRESET} , and the WRF bit in the RSTFC register will be set high.

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

000: $2^8/f_{4kHz}$

001: $2^{10}/f_{4kHz}$

010: $2^{12}/f_{4kHz}$

011: $2^{14}/f_{4kHz}$

100: $2^{15}/f_{4kHz}$

101: $2^{16}/f_{4kHz}$

110: $2^{17}/f_{4kHz}$

111: $2^{18}/f_{4kHz}$

These three bits determine the division ratio of the watchdog timer source clock, which in turn determines the time-out period.

• RSTFC Register

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

“x”: unknown

Bit 7~4 Unimplemented, read as “0”

Bit 3 **RSTF**: Reset control register software reset flag

Refer to the Internal Reset Control section.

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 **WRF**: WDT control register software reset flag
0: Not occurred
1: Occurred
This bit is set to 1 by the WDT control register software reset and cleared by the application program. This bit can only be cleared to zero by application program.

Watchdog Timer Operation

The Watchdog Timer operates by providing a device reset when its timer overflows. This means that in the application program and during normal operation the user has to strategically clear the Watchdog Timer before it overflows to prevent the Watchdog Timer from executing a reset. This is done using the clear watchdog 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. There are five bits, WE4~WE0, in the WDTC register to offer the Watchdog Timer enable/disable control and the MCU reset. The WDT function will be enabled when the WE4~WE0 bits are set to a value of 01010B while the WDT function will be disabled if the WE4~WE0 bits are equal to 10101B. If the WE4~WE0 bits are set to any other values rather than 01010B and 10101B, it will reset the device after a delay time, t_{SRESET} . After power on these bits will have a value of 01010B.

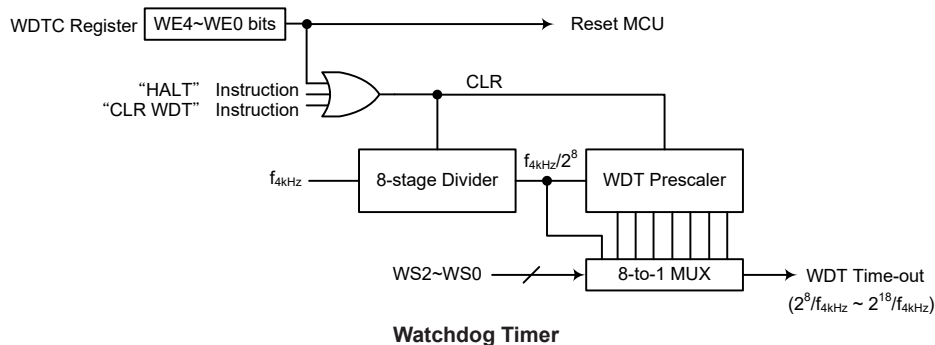
| WE4~WE0 Bits | WDT Function |
|-----------------|--------------|
| 10101B | Disable |
| 01010B | Enable |
| Any other value | Reset MCU |

Watchdog Timer Function Control

Under normal program operation, a Watchdog Timer time-out will initialise a device reset and set the status bit TO. However, if the system is in the SLEEP or IDLE Mode, when a Watchdog Timer time-out occurs, the TO bit in the status register will be set and only the Program Counter and Stack Pointer will be reset. Three methods can be adopted to clear the contents of the Watchdog Timer. The first is a WDTC register software reset, which means a certain value except 01010B and 10101B written into the WE4~WE0 bits, the second is using the Watchdog Timer software clear instruction and the third is via a HALT instruction.

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

The maximum time out period is when the 2^{18} division ratio is selected. As an example, with a 32.768kHz LIRC oscillator as its source clock, this will give a maximum watchdog period of 64 seconds for the 2^{18} division ratio and a minimum timeout of 62.5ms for the 2^8 division ration.



Reset and Initialisation

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

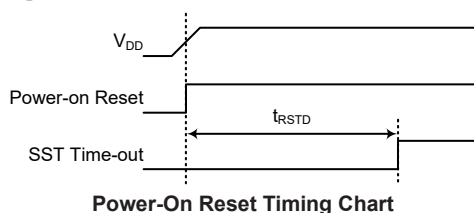
In addition to the power-on reset, another reset exists in the form of a Low Voltage Reset, LVR, where a full reset is implemented in situations where the power supply voltage falls below a certain threshold. Another type of reset is when the Watchdog Timer overflows and resets the microcontroller. All 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 internally.

Power-on Reset

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



Internal Reset Control

There is an internal reset control register, RSTC, which is used to provide a reset when the device operates abnormally due to the environmental noise interference. If the content of the RSTC register is set to any value other than 01010101B or 10101010B, it will reset the device after a delay time, t_{SRESET} . After power-on the register will have a value of 01010101B.

| RSTC7~RSTC0 Bits | Reset Function |
|------------------|----------------|
| 01010101B | No operation |
| 10101010B | No operation |
| Any other value | Reset MCU |

Internal Reset Function Control

• RSTC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|-------|-------|-------|-------|-------|-------|-------|
| Name | RSTC7 | RSTC6 | RSTC5 | RSTC4 | RSTC3 | RSTC2 | RSTC1 | RSTC0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

Bit 7~0 **RSTC7~RSTC0**: Reset function control

01010101: No operation

10101010: No operation

Other values: Reset MCU

If these bits are changed due to adverse environmental conditions, the microcontroller will be reset. The reset operation will be activated after a delay time, t_{SRESET} and the RSTF bit in the RSTFC register will be set to 1. All resets will reset this register to POR value except the WDT time-out hardware warm reset.

• RSTFC Register

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

“x”: unknown

Bit 7~4 Unimplemented, read as “0”

Bit 3 **RSTF**: Reset control register software reset flag

0: Not occurred

1: Occurred

This bit is set to 1 by the RSTC control register software reset and cleared by the application program. This bit can only be cleared to zero by application program.

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.

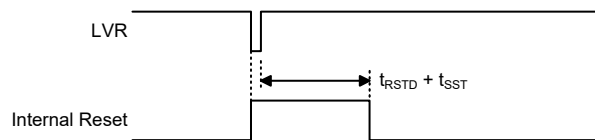
Bit 0 **WRF**: WDT control register software reset flag

Refer to the Watchdog Timer Control Register section.

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 when the value falls below a certain predefined level.

The LVR function can be enabled or disabled by the LVRC control register. If the supply voltage of the device drops to within a range of $0.9V \sim V_{LVR}$ such as might occur when changing the battery in battery powered applications, the LVR will automatically reset the device internally and the LVRF bit in the RSTFC register will also be set to 1. For a valid LVR signal, a low supply voltage, i.e., a voltage in the range between $0.9V \sim V_{LVR}$ 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. The actual t_{LVR} value can be selected by the TLVR1~TLVR0 bits in the TLVRC register. 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 any other 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 to 1. 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 SLEEP or IDLE 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

01100110: 1.7V

01010101: 1.9V

00110011: 2.1V

10011001: 2.55V

10101010: 3.15V

11110000: LVR disable

Other values: Generates an MCU reset

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 six defined register 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.

• **TLVRC Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|---|---|---|---|-------|-------|
| Name | — | — | — | — | — | — | TLVR1 | TLVR0 |
| R/W | — | — | — | — | — | — | R/W | R/W |
| POR | — | — | — | — | — | — | 0 | 1 |

Bit 7~2 Unimplemented, read as “0”

Bit 1~0 **TLVR1~TLVR0**: Minimum low voltage width to reset time (t_{LVR})

00: $(7 \sim 8) \times t_{LIRC}$

01: $(31 \sim 32) \times t_{LIRC}$

10: $(63 \sim 64) \times t_{LIRC}$

11: $(127 \sim 128) \times t_{LIRC}$

• **RSTFC Register**

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

“x”: unknown

Bit 7~4 Unimplemented, read as “0”

Bit 3 **RSTF**: Reset control register software reset flag

Refer to the Internal Reset Control section.

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

0: Not occurred

1: Occurred

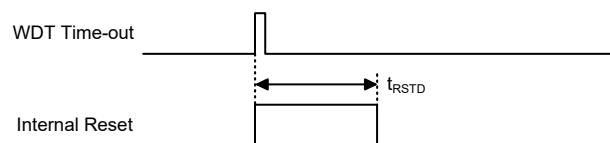
- This bit is set to 1 when a specific low voltage reset condition occurs. This bit can only be cleared to zero by application program.
- Bit 1 **LRF**: LVR control register software reset flag
0: Not occurred
1: Occurred
- This bit is set high if the LVRC register contains any non-defined LVR voltage register values. This in effect acts like a software reset function. This bit can only be cleared to zero by application program.
- Bit 0 **WRF**: WDT control register software reset flag
Refer to the Watchdog Timer Control Register section.

IAP Reset

When a specific value of “55H” is written into the FC1 register, a reset signal will be generated to reset the whole device. Refer to the In Application Programming section for more associated details.

Watchdog Time-out Reset during Normal Operation

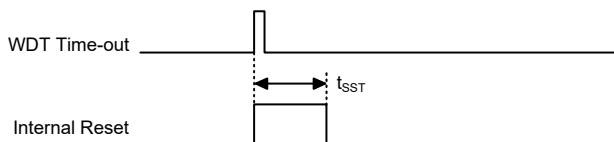
The Watchdog time-out Reset during normal operation in the FAST or SLOW Mode is the same as the LVR Reset except that the Watchdog time-out flag TO will be set to “1”.



WDT Time-out Reset during Normal Operation Timing Chart

Watchdog Time-out Reset during SLEEP or IDLE Mode

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



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

Reset Initial Conditions

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

| TO | PDF | Reset Conditions |
|----|-----|--|
| 0 | 0 | Power-on reset |
| u | u | LVR reset during 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”: 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 Bases | Cleared after reset, WDT begins counting |
| Timer Modules | Timer Modules will be turned off |
| Input/Output Ports | I/O ports will be setup as inputs |
| Stack Pointer | Stack Pointer will point to the top of the stack |

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

| Register | Power-On Reset | WDT Time-out (Normal Operation) | WDT Time-out (IDLE/SLEEP) |
|----------|----------------|------------------------------------|------------------------------|
| IAR0 | 0000 0000 | 0000 0000 | uuuu uuuu |
| MP0 | 0000 0000 | 0000 0000 | uuuu uuuu |
| IAR1 | 0000 0000 | 0000 0000 | uuuu uuuu |
| MP1L | 0000 0000 | 0000 0000 | uuuu uuuu |
| MP1H | 0000 0000 | 0000 0000 | uuuu uuuu |
| ACC | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| PCL | 0000 0000 | 0000 0000 | 0000 0000 |
| TBLP | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| TBLH | xxxx xxxx | uuuu uuuu | uuuu uuuu |
| TBHP | ---x xxxx | ---u uuuu | ---u uuuu |
| STATUS | xx00 xxxx | uu1u uuuu | uu11 uuuu |
| IAR2 | 0000 0000 | 0000 0000 | uuuu uuuu |
| MP2L | 0000 0000 | 0000 0000 | uuuu uuuu |
| MP2H | 0000 0000 | 0000 0000 | uuuu uuuu |
| RSTFC | ---- 0x00 | ---- uuuu | ---- uuuu |
| SCC | 111- --00 | 111- --00 | uuu- --uu |
| IRCC | ---0 --00 | ---0 --00 | ---u --uu |
| STKPTR | 0--- 0000 | 0--- 0000 | u--- 0000 |
| IECC | 0000 0000 | 0000 0000 | uuuu uuuu |
| PA | 1111 1111 | 1111 1111 | uuuu uuuu |
| PAC | 1111 1111 | 1111 1111 | uuuu uuuu |
| PAPU | 0000 0000 | 0000 0000 | uuuu uuuu |
| PAWU | 0000 0000 | 0000 0000 | uuuu uuuu |
| RSTC | 0101 0101 | 0101 0101 | uuuu uuuu |
| LVRC | 0110 0110 | 0110 0110 | uuuu uuuu |
| TLVRC | ---- --01 | ---- --01 | ---- --uu |
| MFIO | --00 --00 | --00 --00 | --uu --uu |
| MF11 | -000 -000 | -000 -000 | -uuu -uuu |
| WDTC | 0101 0010 | 0101 0010 | uuuu uuuu |
| INTEG | 0000 0000 | 0000 0000 | uuuu uuuu |
| INTC0 | -000 0000 | -000 0000 | -uuu uuuu |
| INTC1 | 0000 0000 | 0000 0000 | uuuu uuuu |
| INTC2 | 0000 0000 | 0000 0000 | uuuu uuuu |
| INTC3 | --00 --00 | --00 --00 | --uu --uu |
| PB | ---- ---1 | ---- ---1 | ---- ---u |

| Register | Power-On Reset | WDT Time-out (Normal Operation) | WDT Time-out (IDLE/SLEEP) |
|----------|----------------|------------------------------------|------------------------------|
| PBC | ---- --- 1 | ---- --- 1 | ---- --- u |
| PBPU | ---- --- 0 | ---- --- 0 | ---- --- u |
| PCRL | 0000 0000 | 0000 0000 | uuuu uuuu |
| PCRH | ---0 0000 | ---0 0000 | ---u uuuu |
| PSCR | ---- -000 | ---- -000 | ---- -uuu |
| TB0C | 0--- -000 | 0--- -000 | u--- -uuu |
| TB1C | 0--- -000 | 0--- -000 | u--- -uuu |
| USR | 0000 1011 | 0000 1011 | uuuu uuuu |
| UCR1 | 0000 00x0 | 0000 00x0 | uuuu uuuu |
| UCR2 | 0000 0000 | 0000 0000 | uuuu uuuu |
| UCR3 | ---- --- 0 | ---- --- 0 | ---- --- u |
| BRDH | 0000 0000 | 0000 0000 | uuuu uuuu |
| BRDL | 0000 0000 | 0000 0000 | uuuu uuuu |
| UFCR | --00 0000 | --00 0000 | --uu uuuu |
| TXR_RXR | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| RxCNT | ---- -000 | ---- -000 | ---- -uuu |
| PTMC0 | 0000 0--- | 0000 0--- | uuuu u--- |
| PTMC1 | 0000 0000 | 0000 0000 | uuuu uuuu |
| PTMDL | 0000 0000 | 0000 0000 | uuuu uuuu |
| PTMDH | 0000 0000 | 0000 0000 | uuuu uuuu |
| PTMAL | 0000 0000 | 0000 0000 | uuuu uuuu |
| PTMAH | 0000 0000 | 0000 0000 | uuuu uuuu |
| PTMRPL | 0000 0000 | 0000 0000 | uuuu uuuu |
| PTMRPH | 0000 0000 | 0000 0000 | uuuu uuuu |
| EEAL | 0000 0000 | 0000 0000 | uuuu uuuu |
| EEAH | ---- --- 0 | ---- --- 0 | ---- --- u |
| EED | 0000 0000 | 0000 0000 | uuuu uuuu |
| PWRC | ---- 0--- | ---- 0--- | ---- u--- |
| IREFC | ---0 -000 | ---0 -000 | ---u -uuu |
| PVREF | 0000 0000 | 0000 0000 | uuuu uuuu |
| OPAC | 000- ---- | 000- ---- | uuu- ---- |
| GSC1 | ---- -000 | ---- -000 | ---- -uuu |
| AFEDA1C | ---- --00 | ---- --00 | ---- --uu |
| AFEDA1L | 0000 ---- | 0000 ---- | uuuu ---- |
| AFEDA1H | 0000 0000 | 0000 0000 | uuuu uuuu |
| AFEDA2C | ---- --00 | ---- --00 | ---- --uu |
| AFEDA2L | 0000 ---- | 0000 ---- | uuuu ---- |
| AFEDA2H | 0000 0000 | 0000 0000 | uuuu uuuu |
| AFEDA3C | ---- --00 | ---- --00 | ---- --uu |
| AFEDA3L | 0000 ---- | 0000 ---- | uuuu ---- |
| AFEDA3H | 0000 0000 | 0000 0000 | uuuu uuuu |
| FBRC | 0000 0000 | 0000 0000 | uuuu uuuu |
| PGAC | -000 0000 | -000 0000 | -uuu uuuu |
| PGACS | 0000 0000 | 0000 0000 | uuuu uuuu |
| MODC | ---- 0000 | ---- 0000 | ---- uuuu |
| ADRL | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| ADRM | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| ADRH | xxxx xxxx | xxxx xxxx | uuuu uuuu |

| Register | Power-On Reset | WDT Time-out (Normal Operation) | WDT Time-out (IDLE/SLEEP) |
|----------|----------------|------------------------------------|------------------------------|
| ADCR0 | ---0 000- | ---0 000- | ---u uuu- |
| ADCR1 | 0000 -00- | 0000 -00- | uuuu -uu- |
| ADCS | ---0 0000 | ---0 0000 | ---u uuuu |
| SINC3 | 1001 0011 | 1001 0011 | uuuu uuuu |
| ADACCTRL | 00-- -000 | 00-- -000 | uu-- -uuu |
| ADRL_AVG | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| ADRM_AVG | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| ADRH_AVG | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| CTRL | 000- ---- | 000- ---- | uuu- ---- |
| AUTOADCC | 0-00 0000 | 0-00 0000 | u-uu uuuu |
| ERRCHKC | 00-- ---- | 00-- ---- | uu-- ---- |
| ERRCHKR | 0000 0000 | 0000 0000 | uuuu uuuu |
| SPIC0 | 111- --00 | 111- --00 | uuu- --uu |
| SPIC1 | --00 0000 | --00 0000 | --uu uuuu |
| SPID | xxxx xxxx | xxxx xxxx | uuuu uuuu |
| ORMC | 0000 0000 | 0000 0000 | 0000 0000 |
| CTMC0 | 0000 0000 | 0000 0000 | uuuu uuuu |
| CTMC1 | 0000 0000 | 0000 0000 | uuuu uuuu |
| CTMDL | 0000 0000 | 0000 0000 | uuuu uuuu |
| CTMDH | ---- --00 | ---- --00 | ---- --uu |
| CTMAL | 0000 0000 | 0000 0000 | uuuu uuuu |
| CTMAH | ---- --00 | ---- --00 | ---- --uu |
| CRCCR | ---- --0 | ---- --0 | ---- --u |
| CRCIN | 0000 0000 | 0000 0000 | uuuu uuuu |
| CRCDL | 0000 0000 | 0000 0000 | uuuu uuuu |
| CRCDH | 0000 0000 | 0000 0000 | uuuu uuuu |
| PAS0 | 0000 0000 | 0000 0000 | uuuu uuuu |
| PAS1 | 0000 0000 | 0000 0000 | uuuu uuuu |
| PBS0 | ---- --00 | ---- --00 | ---- --uu |
| FC0 | 0000 0000 | 0000 0000 | uuuu uuuu |
| FC1 | 0000 0000 | 0000 0000 | uuuu uuuu |
| FC2 | ---- --00 | ---- --00 | ---- --uu |
| FARL | 0000 0000 | 0000 0000 | uuuu uuuu |
| FARH | ---0 0000 | ---0 0000 | ---u uuuu |
| FD0L | 0000 0000 | 0000 0000 | uuuu uuuu |
| FD0H | 0000 0000 | 0000 0000 | uuuu uuuu |
| FD1L | 0000 0000 | 0000 0000 | uuuu uuuu |
| FD1H | 0000 0000 | 0000 0000 | uuuu uuuu |
| FD2L | 0000 0000 | 0000 0000 | uuuu uuuu |
| FD2H | 0000 0000 | 0000 0000 | uuuu uuuu |
| FD3L | 0000 0000 | 0000 0000 | uuuu uuuu |
| FD3H | 0000 0000 | 0000 0000 | uuuu uuuu |
| IFS | ---- -000 | ---- -000 | ---- -uuu |
| EEC | 0000 0000 | 0000 0000 | uuuu uuuu |

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

Input/Output Ports

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

The device provides bidirectional input/output lines labeled with port names PA~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 | — | — | — | — | — | — | — | PB0 |
| PBC | — | — | — | — | — | — | — | PBC0 |
| PBPU | — | — | — | — | — | — | — | PBPU0 |

“—”: Unimplemented, read as “0”

I/O Logic Function Registers 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 an 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 Pin pull-high function control

0: Disable

1: Enable

The PxPUn bit is used to control the pin pull-high function. Here the “x” is the Port name which can be A or B. However, the actual available bits for each I/O Port may be different.

Port A Wake-up

The HALT instruction forces the microcontroller into the SLEEP or IDLE Mode which preserves power, a feature that is important for battery and other low-power applications. Various methods exist to wake up the microcontroller, one of which is to change the logic condition on one of the Port

A pins from high to low. This function is especially suitable for applications that can be woken up via external switches. Each pin on Port A can be selected individually to have this wake-up feature using the PAWU register.

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

• PAWU Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|-------|-------|-------|-------|-------|-------|-------|
| Name | PAWU7 | PAWU6 | PAWU5 | PAWU4 | PAWU3 | PAWU2 | PAWU1 | PAWU0 |
| 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 PAWU7~PAWU0: PA7~PA0 pin Wake-up function control

0: Disable

1: Enable

I/O Port Control Registers

Each I/O port has its own control register which controls the input/output configuration. With this control register, each CMOS output or input can be reconfigured dynamically under software control. Each pin of the I/O ports is directly mapped to a bit in its associated port control register. For the I/O pin to function as an input, the corresponding bit of the control register must be written as a “1”. This will then allow the logic state of the input pin to be directly read by instructions. When the corresponding bit of the control register is written as a “0”, the I/O pin will be setup as a CMOS output. If the pin is currently setup as an output, instructions can still be used to read the output register. However, it should be noted that the program will in fact only read the status of the output data latch and not the actual logic status of the output pin when the IECM is set to “0”.

• 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 Port x Pin type selection

0: Output

1: Input

The PxCn bit is used to control the pin type selection. Here the “x” is the Port name which can be A or B. However, the actual available bits for each I/O Port may be different.

Pin-shared Functions

The flexibility of the microcontroller range is greatly enhanced by the use of pins that have more than one function. Limited numbers of pins can force serious design constraints on designers but by supplying pins with multi-functions, many of these difficulties can be overcome. For these pins, the desired function of the multi-function I/O pins is selected by a series of registers via the application program control.

Pin-shared Function Selection Registers

The limited number of supplied pins in a package can impose restrictions on the amount of functions a certain device can contain. However by allowing the same pins to share several different functions and providing a means of function selection, a wide range of different functions can be incorporated into even relatively small package sizes. The device includes Port “x” output function selection

register “n”, labeled as PxSn, and Input Function source pin selection register, labeled as IFS, which can select the desired functions of the multi-function pin-shared pins.

The most important point to note is to make sure that the desired pin-shared function is properly selected and also deselected. For most pin-shared functions, to select the desired pin-shared function, the pin-shared function should first be correctly selected using the corresponding pin-shared control register. After that the corresponding peripheral functional setting should be configured and then the peripheral function can be enabled. However, a special point must be noted for some digital input pins, such as INTn, xTCK, PTPI, etc., which share the same pin-shared control configuration with their corresponding general purpose I/O functions when setting the relevant pin-shared control bit fields. To select these pin functions, in addition to the necessary pin-shared control and peripheral functional setup aforementioned, they must also be setup as input by setting the corresponding bit in the I/O port control register. To correctly deselect the pin-shared function, the peripheral function should first be disabled and then the corresponding pin-shared function control register can be modified to select other pin-shared functions.

| Register Name | Bit | | | | | | | |
|---------------|-------|-------|-------|-------|-------|--------|----------|----------|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IFS | — | — | — | — | — | RXTXPS | SPISDIPS | SPISCKPS |
| PAS0 | PAS07 | PAS06 | PAS05 | PAS04 | PAS03 | PAS02 | PAS01 | PAS00 |
| PAS1 | PAS17 | PAS16 | PAS15 | PAS14 | PAS13 | PAS12 | PAS11 | PAS10 |
| PBS0 | — | — | — | — | — | — | PBS01 | PBS00 |

Pin-shared Function Selection Register List

• IFS Register

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

Bit 7~3 Unimplemented, read as “0”

Bit 2 **RXTXPS**: RX/TX input source pin selection

0: PA7

1: PA2

Bit 1 **SPISDIPS**: SPISDI input source pin selection

0: PA7

1: PA2

Bit 0 **SPISCKPS**: SPISCK input source pin selection

0: PB0

1: PA4

• PAS0 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|-------|-------|-------|-------|-------|-------|-------|
| Name | PAS07 | PAS06 | PAS05 | PAS04 | PAS03 | PAS02 | PAS01 | PAS00 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7~6 **PAS07~PAS06**: PA3 pin-shared function selection

00: PA3

01: PA3

10: PA3

11: AIN0

- Bit 5~4 **PAS05~PAS04**: PA2 pin-shared function selection
 00: PA2
 01: SPISDI
 10: RX/TX
 11: PA2
- Bit 3~2 **PAS03~PAS02**: PA1 pin-shared function selection
 00: PA1
 01: PA1
 10: PA1
 11: AIN1
- Bit 1~0 **PAS01~PAS00**: PA0 pin-shared function selection
 00: PA0
 01: SPISDO
 10: TX
 11: PA0

• **PAS1 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|-------|-------|-------|-------|-------|-------|-------|
| Name | PAS17 | PAS16 | PAS15 | PAS14 | PAS13 | PAS12 | PAS11 | PAS10 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

- Bit 7~6 **PAS17~PAS16**: PA7 pin-shared function selection
 00: PA7/INT2/PTPI
 01: SPISDI
 10: RX/TX
 11: PA7/INT2/PTPI
- Bit 5~4 **PAS15~PAS14**: PA6 pin-shared function selection
 00: PA6
 01: PA6
 10: SPISDO
 11: TX
- Bit 3~2 **PAS13~PAS12**: PA5 pin-shared function selection
 00: PA5/INT1/PTCK
 01: PA5/INT1/PTCK
 10: PA5/INT1/PTCK
 11: SPISCS
- Bit 1~0 **PAS11~PAS10**: PA4 pin-shared function selection
 00: PA4/INT0/CTCK
 01: CTP
 10: SPISCK
 11: PA4/INT0/CTCK

• **PBS0 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|---|---|---|---|-------|-------|
| Name | — | — | — | — | — | — | PBS01 | PBS00 |
| R/W | — | — | — | — | — | — | R/W | R/W |
| POR | — | — | — | — | — | — | 0 | 0 |

- Bit 7~2 Unimplemented, read as “0”
- Bit 1~0 **PBS01~PBS00**: PB0 pin-shared function selection
 00: PB0/INT3
 01: SPISCK
 10: PTP
 11: PB0/INT3

• IECC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|-------|-------|-------|-------|-------|-------|-------|
| Name | IECS7 | IECS6 | IECS5 | IECS4 | IECS3 | IECS2 | IECS1 | IECS0 |
| 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 |

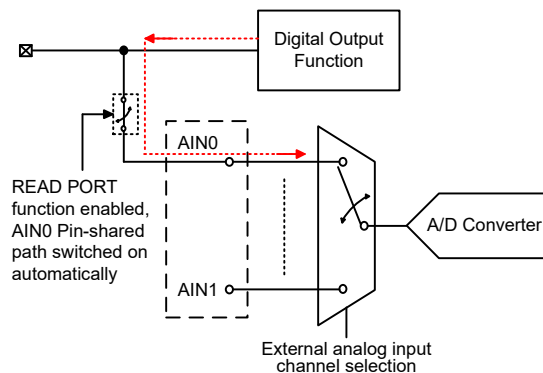
Bit 7~0 **IECS7~IECS0**: READ PORT function enable control bit 7~ bit 0
11001010: IECM=1 – READ PORT function is enabled
Others: IECM=0 – READ PORT function is disabled

| READ PORT Function | Disabled | | Enabled | |
|---------------------------------------|-----------|------------------|-----------|---|
| Port Control Register Bit – Px.C.n | 1 | 0 | 1 | 0 |
| I/O Function | Pin value | Data latch value | Pin value | |
| Digital Input Function | | | | |
| Digital Output Function (except UART) | | | | |
| UART: RX/TX | | | | |
| Analog Function | 0 | | | |

Note: The value in the above table is the content of the ACC register after “mov a, Px” instruction is executed where “x” means the relevant port name.

The additional function of the READ PORT mode is to check the A/D path. When the READ PORT function is disabled, the A/D path from the external pin to the internal analog input will be switched off if the A/D input pin function is not selected by the corresponding selection bits. For the MCU with A/D converter channels, such as A/D AIN1~AIN0, the desired A/D channel can be switched on by properly configuring the external analog input channel selection bits in the A/D Control Register together with the corresponding analog input pin function is selected. However, the additional function of the READ PORT mode is to force the A/D path to be switched on. For example, when the AIN0 is selected as the analog input channel as the READ PORT function is enabled, the AIN0 analog input path will be switched on even if the AIN0 analog input pin function is not selected. In this way, the AIN0 analog input path can be examined by internally connecting the digital output on this shared pin with the AIN0 analog input pin switch and then converting the corresponding digital data without any external analog input voltage connected.

Note that the A/D converter reference voltage should be equal to the I/O power supply voltage when examining the A/D path using the READ PORT function.



A/D Channel Input Path Internally Connection

Programming Considerations

Within the user program, one of the things first 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 setup 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.

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

Timer Modules – TM

One of the most fundamental functions in any microcontroller device is the ability to control and measure time. To implement time related functions the device includes several Timer Modules, generally abbreviated to the name TM. The TMs are multi-purpose timing units and serve to provide operations such as Timer/Counter, Input Capture, Compare Match Output and Single Pulse Output as well as being the functional unit for the generation of PWM signals. Each of the TMs has two interrupts. The addition of input and output pins for each TM ensures that users are provided with timing units with a wide and flexible range of features.

The common features of the different TM types are described here with more detailed information provided in the individual Compact and Periodic Type TM sections.

Introduction

The device contains two TMs and each individual TM is categorised as a certain type, namely Compact Type TM or Periodic Type TM. Although similar in nature, the different TM types vary in their feature complexity. The common features to all of the Compact and Periodic TMs will be described in this section and the detailed operation regarding each of the TM types will be described in separate sections. The main features and differences between the two types of TMs are summarised in the accompanying table.

| TM Function | CTM | PTM |
|------------------------------|----------------|----------------|
| Timer/Counter | √ | √ |
| Input Capture | — | √ |
| Compare Match Output | √ | √ |
| PWM Output | √ | √ |
| Single Pulse Output | — | √ |
| PWM Alignment | Edge | Edge |
| PWM Adjustment Period & Duty | Duty or Period | Duty or Period |

TM Function Summary

TM Operation

The different types of TM offer a diverse range of functions, from simple timing operations to PWM signal generation. The key to understanding how the TM operates is to see it in terms of a free running counter whose value is then compared with the value of pre-programmed internal comparators. When the free running counter has the same value as the pre-programmed comparator, known as a compare match situation, a TM interrupt signal will be generated which can clear the

counter and perhaps also change the condition of the TM output pin. The internal TM counter is driven by a user selectable clock source, which can be an internal clock or an external pin.

TM Clock Source

The clock source which drives the main counter in each TM can originate from various sources. The selection of the required clock source is implemented using the xTCK2~xTCK0 bits in the xTM control registers, where “x” stands for C or P type TM. The clock source can be a ratio of the system clock, f_{SYS} , or the internal high clock, f_H , the f_{SUB} clock source or the external xTCK pin. The xTCK pin clock source is used to allow an external signal to drive the TM as an external clock source for event counting.

TM Interrupts

Each Compact or Periodic type TMs has two internal interrupts, one for each of the internal comparator A or comparator P, which generate a TM interrupt when a compare match condition occurs. When a TM interrupt is generated, it can be used to clear the counter and also to change the state of the TM output pins.

TM External Pins

Each of the TMs, irrespective of what type, has one TM input pin, with the label xTCK while the PTM has another input pin with the label PTPI. The xTMn input pin, xTCK, is essentially a clock source for the xTM and is selected using the xTCK2~xTCK0 bits in the xTMC0 register. This external TM input pin allows an external clock source to drive the internal TM. The xTCK input pin can be chosen to have either a rising or falling active edge. The PTCK pin is also used as the external trigger input pin in single pulse output mode for the PTM.

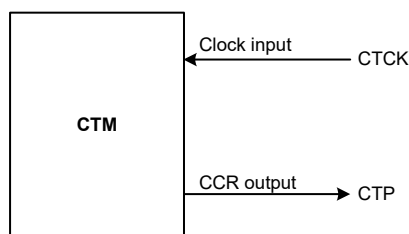
The PTM has another input pin, PTPI, which is the capture input whose active edge can be a rising edge, a falling edge or both rising and falling edges and the active edge transition type is selected using the PTIO1~PTIO0 bits in the PTMC1 register. There is another capture input, PTCK, for PTM capture input mode, which can be used as the external trigger input source except the PTPI pin.

The TMs each have one output pin with the label xTP. When the TM is in the Compare Match Output Mode, this pin can be controlled by the TM to switch to a high or low level or to toggle when a compare match situation occurs. The external xTP output pin is also the pin where the TM generates the PWM output waveform.

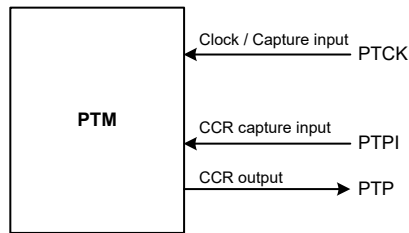
As the TM input and output pins are pin-shared with other functions, the TM input and output functions must first be selected using relevant pin-shared function selection. The details of the pin-shared function selection are described in the pin-shared function section.

| CTM | | PTM | |
|-------|--------|------------|--------|
| Input | Output | Input | Output |
| CTCK | CTP | PTCK, PTPI | PTP |

TM External Pins



CTM Function Pin Block Diagram

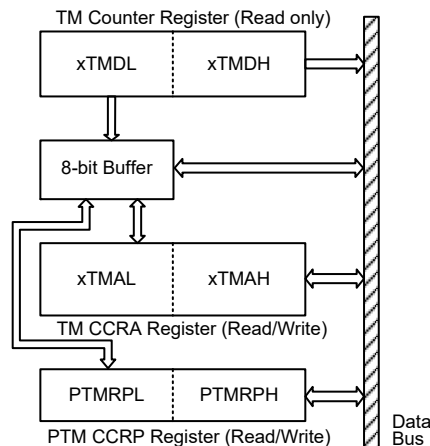


PTM Function Pin Block Diagram

Programming Considerations

The TM Counter Registers and the Capture/Compare CCRA and CCRP registers all have a low and high byte structure. The high bytes can be directly accessed, but as the low bytes can only be accessed via an internal 8-bit buffer, reading or writing to these register pairs must be carried out in a specific way. The important point to note is that data transfer to and from the 8-bit buffer and its related low byte only takes place when a write or read operation to its corresponding high byte is executed.

As the CCRA and CCRP registers are implemented in the way shown in the following diagram and accessing these register pairs is carried out in a specific way as described above, it is recommended to use the “MOV” instruction to access the CCRA and CCRP low byte registers, named xTMAL and PTMRPL, using the following access procedures. Accessing the CCRA and CCRP low byte registers without following these access procedures will result in unpredictable values.



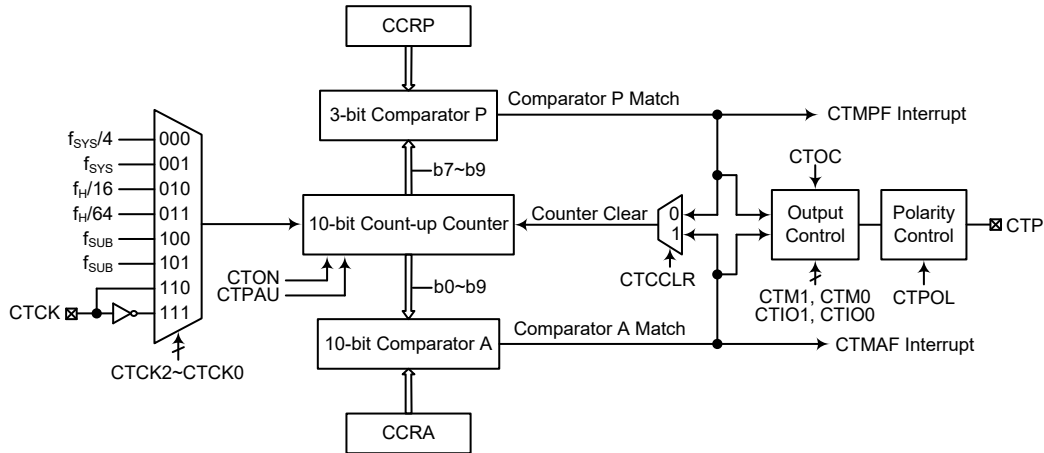
The following steps show the read and write procedures:

- Writing Data to CCRA or CCRP
 - ♦ Step 1. Write data to Low Byte xTMAL or PTMRPL
 - Note that here data is only written to the 8-bit buffer.
 - ♦ Step 2. Write data to High Byte xTMAH or PTMRPH
 - Here data is written directly to the high byte registers and simultaneously data is latched from the 8-bit buffer to the Low Byte registers.
- Reading Data from the Counter Registers, CCRA or CCRP
 - ♦ Step 1. Read data from the High Byte xTMDH, xTMAH or PTMRPH
 - Here data is read directly from the High Byte registers and simultaneously data is latched from the Low Byte register into the 8-bit buffer.

- ♦ Step 2. Read data from the Low Byte xTMDL, xTMAL or PTMRPL
 - This step reads data from the 8-bit buffer.

Compact Type TM – CTM

The Compact TM type contains three operating modes, which are Compare Match Output, Timer/Event Counter and PWM Output modes. The Compact TMs can also be controlled with an external input pin and can drive one external output pin.



Note: As the CTM external pins are pin-shared with other functions, so before using the CTM function the relevant pin-shared function registers must be set properly to enable the CTM pin function. The CTCK pin, if used, must also be set as an input by setting the corresponding bits in the port control register.

10-bit Compact Type TM Block Diagram

Compact Type TM Operation

The size of Compact TM is 10-bit wide and its core is a 10-bit count-up counter which is driven by a user selectable internal or external clock source. There are also two internal comparators with the names, Comparator A and Comparator P. These comparators will compare the value in the counter with CCRP and CCRA registers. The CCRP is three bits wide whose value is compared with the highest 3 bits in the counter while the CCRA is the 10 bits and therefore compares with all counter bits.

The only way of changing the value of the 10-bit counter using the application program, is to clear the counter by changing the CTON bit from low to high. The counter will also be cleared automatically by a counter overflow or a compare match with one of its associated comparators. When these conditions occur, a CTM interrupt signal will also usually be generated. The Compact Type TM can operate in a number of different operational modes, can be driven by different clock sources including an input pin and can also control one output pin. All operating setup conditions are selected using relevant internal registers.

Compact Type TM Register Description

Overall operation of the Compact TM is controlled using several registers. A read only register pair exists to store the internal counter 10-bit value, while a read/write register pair exists to store the internal 10-bit CCRA value. The remaining two registers are control registers which setup the different operating and control modes as well as the 3 CCRP bits.

| Register Name | Bit | | | | | | | |
|---------------|-------|-------|-------|-------|------|-------|-------|--------|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CTMC0 | CTPAU | CTCK2 | CTCK1 | CTCK0 | CTON | CTRP2 | CTRP1 | CTRP0 |
| CTMC1 | CTM1 | CTM0 | CTIO1 | CTIO0 | CTOC | CTPOL | CTDPX | CTCCLR |
| CTMDL | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| CTMDH | — | — | — | — | — | — | D9 | D8 |
| CTMAL | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| CTMAH | — | — | — | — | — | — | D9 | D8 |

10-bit Compact TM Register List

• CTMC0 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|-------|-------|-------|------|-------|-------|-------|
| Name | CTPAU | CTCK2 | CTCK1 | CTCK0 | CTON | CTRP2 | CTRP1 | CTRP0 |
| 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 **CTPAU**: CTM Counter Pause Control

0: Run

1: Pause

The counter can be paused by setting this bit high. Clearing the bit to zero restores normal counter operation. When in a Pause condition the CTM will remain powered up and continue to consume power. The counter will retain its residual value when this bit changes from low to high and resume counting from this value when the bit changes to a low value again.

Bit 6~4 **CTCK2~CTCK0**: Select CTM Counter clock

000: $f_{SYS}/4$

001: f_{SYS}

010: $f_H/16$

011: $f_H/64$

100: f_{SUB}

101: f_{SUB}

110: CTCK rising edge clock

111: CTCK falling edge clock

These three bits are used to select the clock source for the CTM. The external pin clock source can be chosen to be active on the rising or falling edge. The clock source f_{SYS} is the system clock, while f_H and f_{SUB} are other internal clocks, the details of which can be found in the oscillator section.

Bit 3 **CTON**: CTM Counter On/Off Control

0: Off

1: On

This bit controls the overall on/off function of the CTM. Setting the bit high enables the counter to run, clearing the bit disables the CTM. Clearing this bit to zero will stop the counter from counting and turn off the CTM which will reduce its power consumption. When the bit changes state from low to high the internal counter value will be reset to zero, however when the bit changes from high to low, the internal counter will retain its residual value until the bit returns high again.

If the CTM is in the Compare Match Output Mode or the PWM Output Mode then the CTM output pin will be reset to its initial condition, as specified by the CTOC bit, when the CTON bit changes from low to high.

Bit 2~0 **CTRP2~CTRP0**: CTM CCRP 3-bit register, compared with the CTM Counter bit 9 ~ bit 7

Comparator P Match Period=

000: 1024 CTM clocks

001: 128 CTM clocks

010: 256 CTM clocks
 011: 384 CTM clocks
 100: 512 CTM clocks
 101: 640 CTM clocks
 110: 768 CTM clocks
 111: 896 CTM clocks

These three bits are used to setup the value on the internal CCRP 3-bit register, which are then compared with the internal counter's highest three bits. The result of this comparison can be selected to clear the internal counter if the CTCCLR bit is set to zero. Setting the CTCCLR bit to zero ensures that a compare match with the CCRP values will reset the internal counter. As the CCRP bits are only compared with the highest three counter bits, the compare values exist in 128 clock cycle multiples. Clearing all three bits to zero is in effect allowing the counter to overflow at its maximum value.

• **CTMC1 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|------|------|-------|-------|------|-------|-------|--------|
| Name | CTM1 | CTM0 | CTIO1 | CTIO0 | CTOC | CTPOL | CTDPX | CTCCLR |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7~6 **CTM1~CTM0**: Select CTM Operating Mode
 00: Compare Match Output Mode
 01: Undefined
 10: PWM Output Mode
 11: Timer/Counter Mode

These bits setup the required operating mode for the CTM. To ensure reliable operation the CTM should be switched off before any changes are made to the CTM1 and CTM0 bits. In the Timer/Counter Mode, the CTM output pin state is undefined.

Bit 5~4 **CTIO1~CTIO0**: Select CTM external pin function
 Compare Match Output Mode
 00: No change
 01: Output low
 10: Output high
 11: Toggle output
 PWM Output Mode
 00: PWM output inactive state
 01: PWM output active state
 10: PWM output
 11: Undefined
 Timer/Counter Mode
 Unused

These two bits are used to determine how the CTM external pin changes state when a certain condition is reached. The function that these bits select depends upon in which mode the CTM is running.

In the Compare Match Output Mode, the CTIO1 and CTIO0 bits determine how the CTM output pin changes state when a compare match occurs from the Comparator A. The CTM output pin can be setup to switch high, switch low or to toggle its present state when a compare match occurs from the Comparator A. When the bits are both zero, then no change will take place on the output. The initial value of the CTM output pin should be setup using the CTOC bit in the CTMC1 register. Note that the output level requested by the CTIO1 and CTIO0 bits must be different from the initial value setup using the CTOC bit otherwise no change will occur on the CTM output pin when a compare match occurs. After the CTM output pin changes state, it can be reset to its initial level by changing the level of the CTON bit from low to high.

In the PWM Output Mode, the CTIO1 and CTIO0 bits determine how the CTM output pin changes state when a certain compare match condition occurs. The PWM output function is modified by changing these two bits. It is necessary to change the values of the CTIO1 and CTIO0 bits only after the CTM has been switched off. Unpredictable PWM outputs will occur if the CTIO1 and CTIO0 bits are changed when the CTM is running.

Bit 3 **CTOC**: CTM CTP Output control

Compare Match Output Mode

0: Initial low

1: Initial high

PWM Output Mode

0: Active low

1: Active high

This is the output control bit for the CTM output pin. Its operation depends upon whether CTM is being used in the Compare Match Output Mode or in the PWM Output Mode. It has no effect if the CTM is in the Timer/Counter Mode. In the Compare Match Output Mode it determines the logic level of the CTM output pin before a compare match occurs. In the PWM Output Mode it determines if the PWM signal is active high or active low.

Bit 2 **CTPOL**: CTP Output polarity control

0: Non-invert

1: Invert

This bit controls the polarity of the CTP output pin. When the bit is set high the CTM output pin will be inverted and not inverted when the bit is zero. It has no effect if the CTM is in the Timer/Counter Mode.

Bit 1 **CTDPX**: CTM PWM duty/period control

0: CCRP – period; CCRA – duty

1: CCRP – duty; CCRA – period

This bit determines which of the CCRA and CCRP registers are used for period and duty control of the PWM waveform.

Bit 0 **CTCCLR**: Select CTM Counter Clear condition

0: Comparator P match

1: Comparator A match

This bit is used to select the method which clears the counter. Remember that the Compact TM contains two comparators, Comparator A and Comparator P, either of which can be selected to clear the internal counter. With the CTCCLR bit set high, the counter will be cleared when a compare match occurs from the Comparator A. When the bit is low, the counter will be cleared when a compare match occurs from the Comparator P or with a counter overflow. A counter overflow clearing method can only be implemented if the CCRP bits are all cleared to zero. The CTCCLR bit is not used in the PWM Output Mode.

• CTMDL Register

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

Bit 7~0 **D7~D0**: CTM Counter Low Byte Register bit 7 ~ bit 0

CTM 10-bit Counter bit 7 ~ bit 0

• **CTMDH Register**

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

Bit 7~2 Unimplemented, read as “0”

Bit 1~0 **D9~D8**: CTM Counter High Byte Register bit 1 ~ bit 0

CTM 10-bit Counter bit 9 ~ bit 8

• **CTMAL Register**

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

Bit 7~0 **D7~D0**: CTM CCRA Low Byte Register bit 7 ~ bit 0

CTM 10-bit CCRA bit 7 ~ bit 0

• **CTMAH Register**

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

Bit 7~2 Unimplemented, read as “0”

Bit 1~0 **D9~D8**: CTM CCRA High Byte Register bit 1 ~ bit 0

CTM 10-bit CCRA bit 9 ~ bit 8

Compact Type TM Operating Modes

The Compact Type TM can operate in one of three operating modes, Compare Match Output Mode, PWM Output Mode or Timer/Counter Mode. The operating mode is selected using the CTM1 and CTM0 bits in the CTMC1 register.

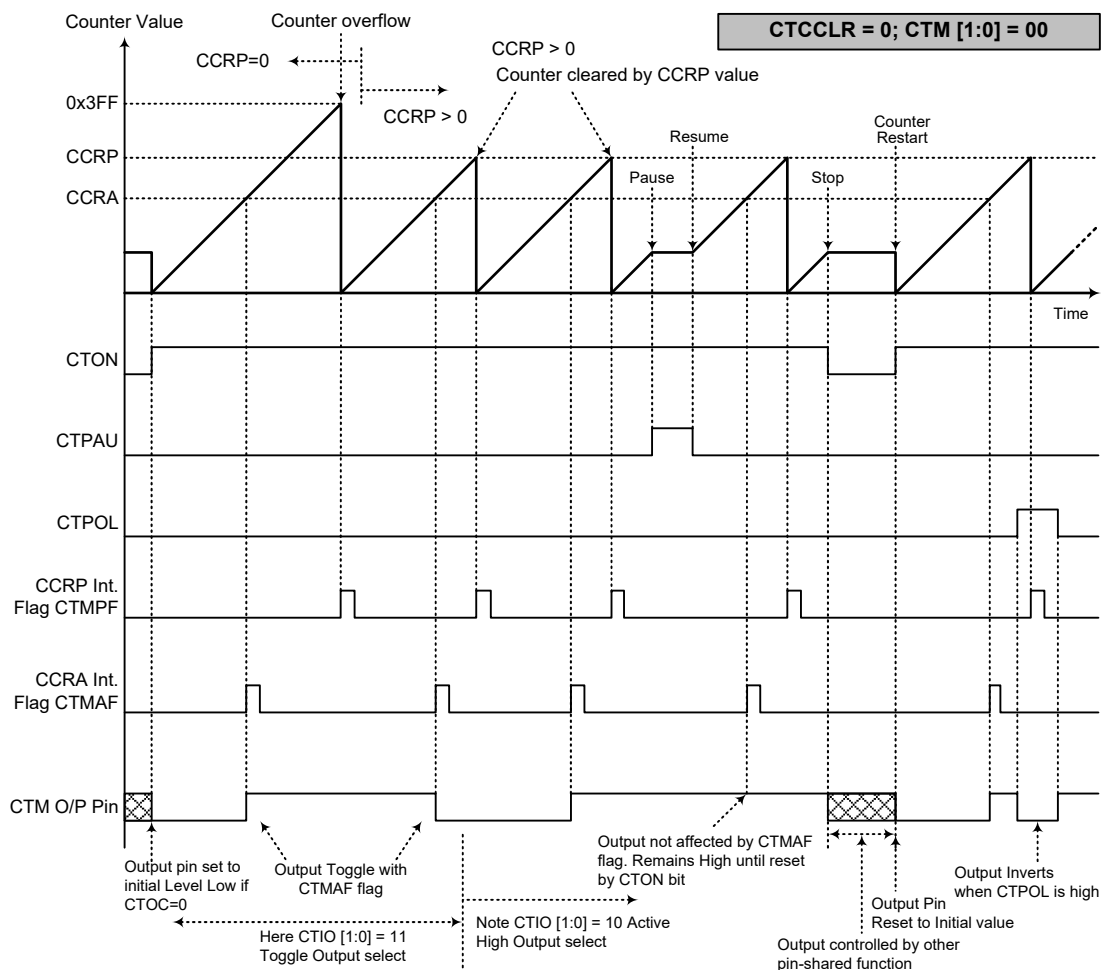
Compare Match Output Mode

To select this mode, bits CTM1 and CTM0 in the CTMC1 register, should be set to 00 respectively. In this mode once the counter is enabled and running it can be cleared by three methods. These are a counter overflow, a compare match from Comparator A and a compare match from Comparator P. When the CTCCLR bit is low, there are two ways in which the counter can be cleared. One is when a compare match from Comparator P, the other is when the CCRP bits are all zero which allows the counter to overflow. Here both CTMAF and CTMPF interrupt request flags for Comparator A and Comparator P respectively, will both be generated.

If the CTCCLR bit in the CTMC1 register is high then the counter will be cleared when a compare match occurs from Comparator A. However, here only the CTMAF interrupt request flag will be generated even if the value of the CCRP bits is less than that of the CCRA registers. Therefore when CTCCLR is high no CTMPF interrupt request flag will be generated. If the CCRA bits are all zero, the counter will overflow when it reaches its maximum 10-bit, 3FF Hex, value, however here the CTMAF interrupt request flag will not be generated.

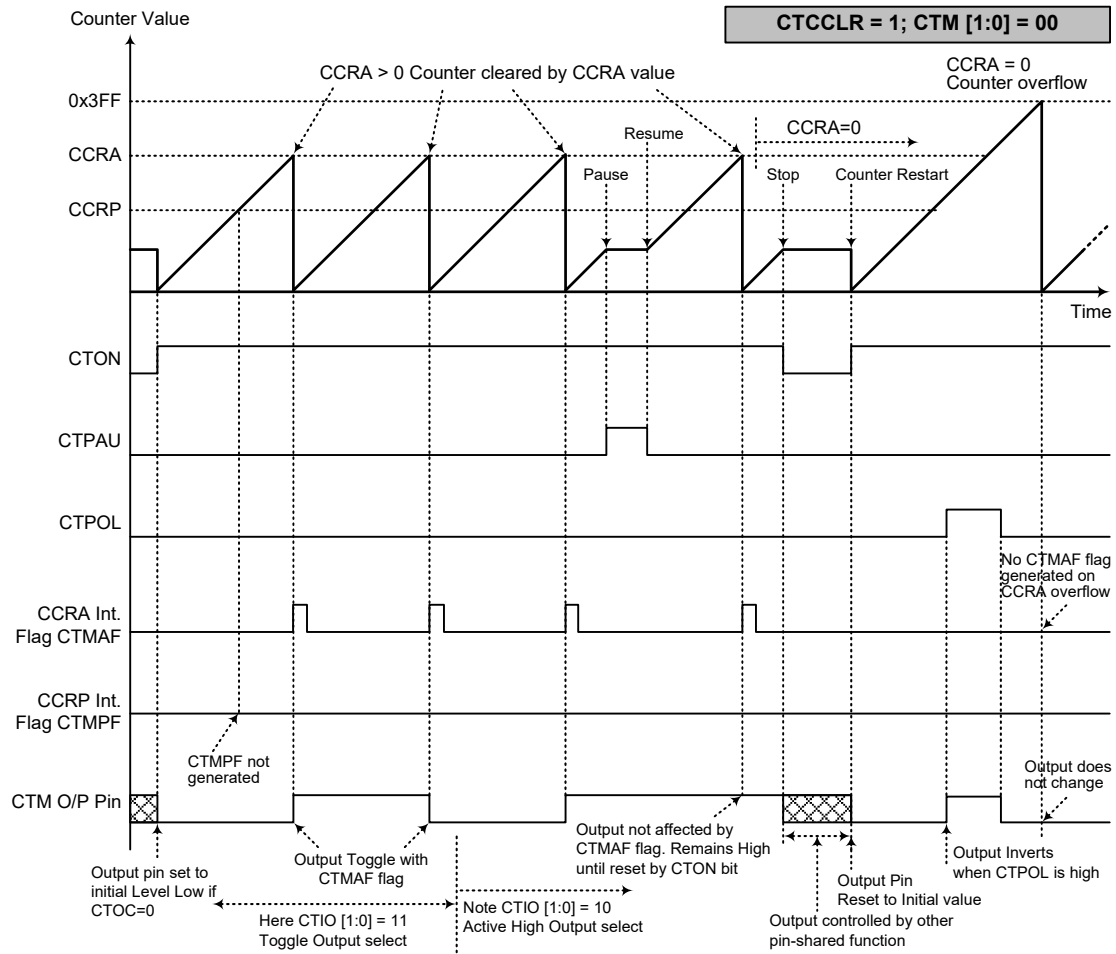
As the name of the mode suggests, after a comparison is made, the CTM output pin will change state. The CTM output pin condition however only changes state when a CTMAF interrupt request flag is generated after a compare match occurs from Comparator A. The CTMPF interrupt request

flag, generated from a compare match occurs from Comparator P, will have no effect on the CTM output pin. The way in which the CTM output pin changes state are determined by the condition of the CTIO1 and CTIO0 bits in the CTMC1 register. The CTM output pin can be selected using the CTIO1 and CTIO0 bits to go high, to go low or to toggle from its present condition when a compare match occurs from Comparator A. The initial condition of the CTM output pin, which is setup after the CTON bit changes from low to high, is setup using the CTOC bit. Note that if the CTIO1 and CTIO0 bits are zero then no pin change will take place.



Compare Match Output Mode – CTCCLR=0

- Note: 1. With CTCCLR=0, a Comparator P match will clear the counter
2. The CTM output pin controlled only by the CTMAF flag
3. The output pin reset to initial state by a CTON bit rising edge



Compare Match Output Mode – CTCCLR=1

- Note: 1. With CTCCLR=1, a Comparator A match will clear the counter
2. The CTM output pin controlled only by the CTMAF flag
3. The output pin reset to initial state by a CTON rising edge
4. The CTMPF flags is not generated when CTCCLR=1

Timer/Counter Mode

To select this mode, bits CTM1 and CTM0 in the CTMC1 register should be set to 11 respectively. The Timer/Counter Mode operates in an identical way to the Compare Match Output Mode generating the same interrupt flags. The exception is that in the Timer/Counter Mode the CTM output pin is not used. Therefore the above description and Timing Diagrams for the Compare Match Output Mode can be used to understand its function. As the CTM output pin is not used in this mode, the pin can be used as a normal I/O pin or other pin-shared function.

PWM Output Mode

To select this mode, bits CTM1 and CTM0 in the CTMC1 register should be set to 10 respectively. The PWM function within the CTM is useful for applications which require functions such as motor control, heating control, illumination control etc. By providing a signal of fixed frequency but of varying duty cycle on the CTM output pin, a square wave AC waveform can be generated with varying equivalent DC RMS values.

As both the period and duty cycle of the PWM waveform can be controlled, the choice of generated waveform is extremely flexible. In the PWM Output Mode, the CTCCLR bit has no effect on the PWM operation. Both of the CCRA and CCRP registers are used to generate the PWM waveform, one register is used to clear the internal counter and thus control the PWM waveform frequency, while the other one is used to control the duty cycle. Which register is used to control either frequency or duty cycle is determined using the CTD PX bit in the CTMC1 register. The PWM waveform frequency and duty cycle can therefore be controlled by the values in the CCRA and CCRP registers.

An interrupt flag, one for each of the CCRA and CCRP, will be generated when a compare match occurs from either Comparator A or Comparator P. The CTOC bit In the CTMC1 register is used to select the required polarity of the PWM waveform while the two CTIO1 and CTIO0 bits are used to enable the PWM output or to force the CTM output pin to a fixed high or low level. The CTPOL bit is used to reverse the polarity of the PWM output waveform.

• CTM, PWM Output Mode, Edge-aligned Mode, CTD PX=0

| CCRP | 1~7 | 0 |
|--------|----------|------|
| Period | CCRP×128 | 1024 |
| Duty | CCRA | |

If $f_{SYS}=4\text{MHz}$, CTMn clock source is $f_{SYS}/4$, CCRP=100b, CCRA=128,

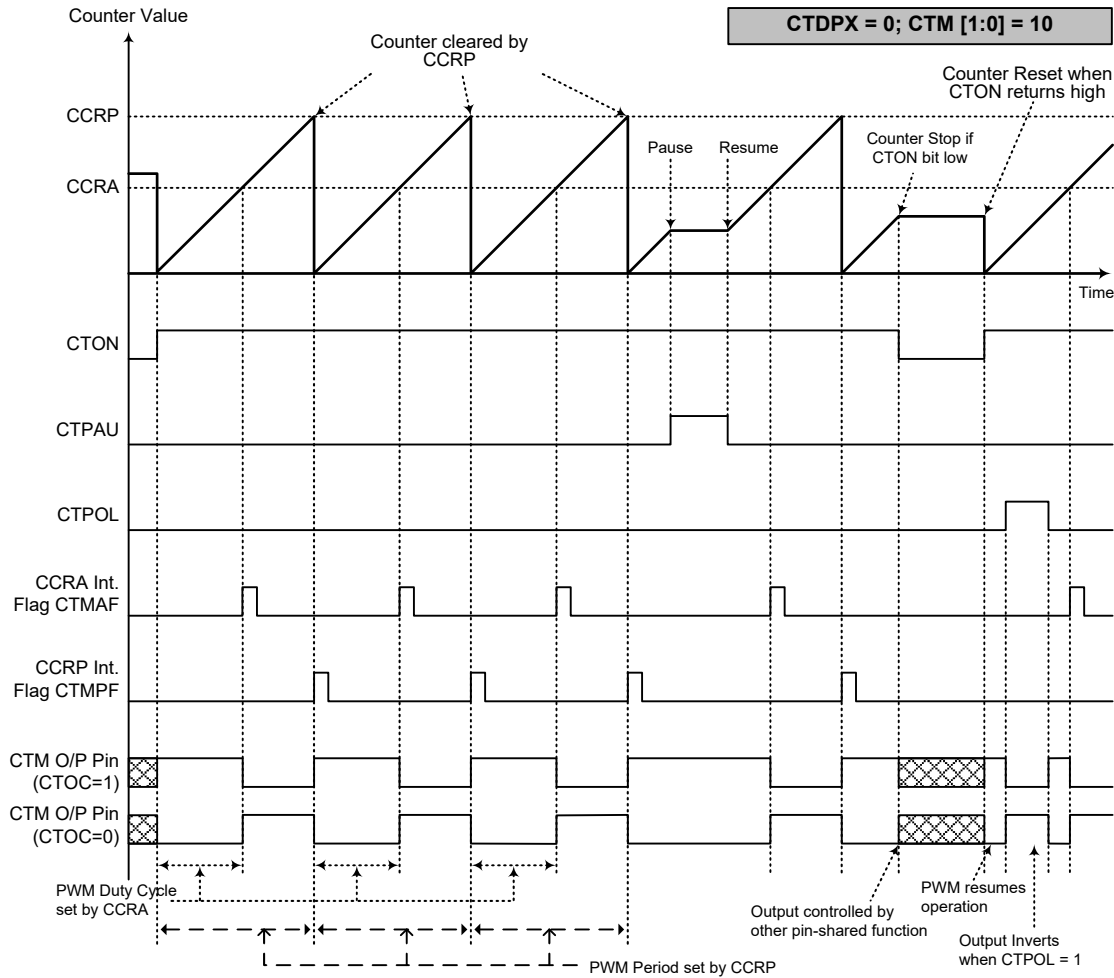
The CTM PWM output frequency= $(f_{SYS}/4)/512=f_{SYS}/2048=1.9531\text{kHz}$, duty=128/512=25%.

If the Duty value defined by the CCRA register is equal to or greater than the Period value, then the PWM output duty is 100%.

• CTM, PWM Output Mode, Edge-aligned Mode, CTD PX=1

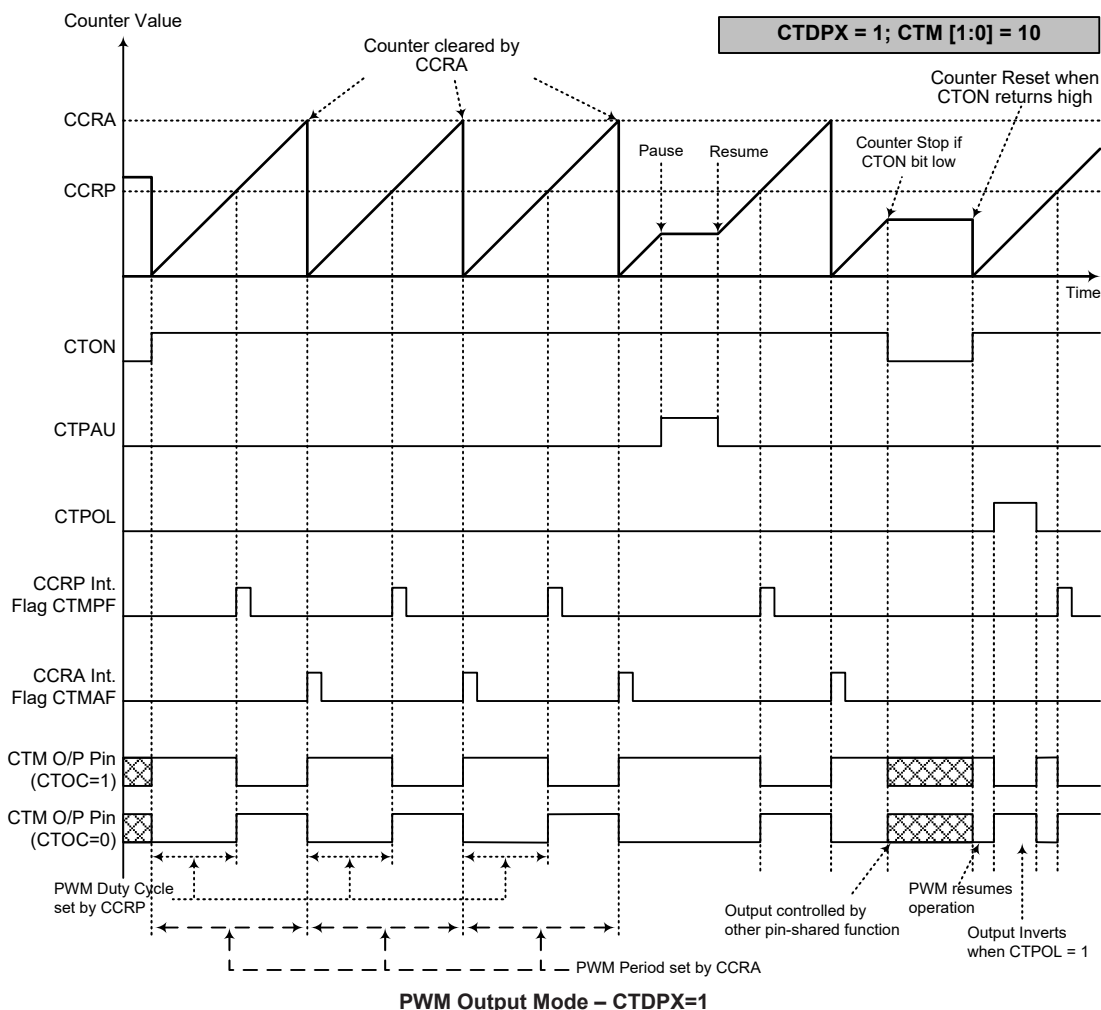
| CCRP | 1~7 | 0 |
|--------|----------|------|
| Period | CCRA | |
| Duty | CCRP×128 | 1024 |

The PWM output period is determined by the CCRA register value together with the CTMn clock while the PWM duty cycle is defined by the CCRP register value.



PWM Output Mode – CTD PX=0

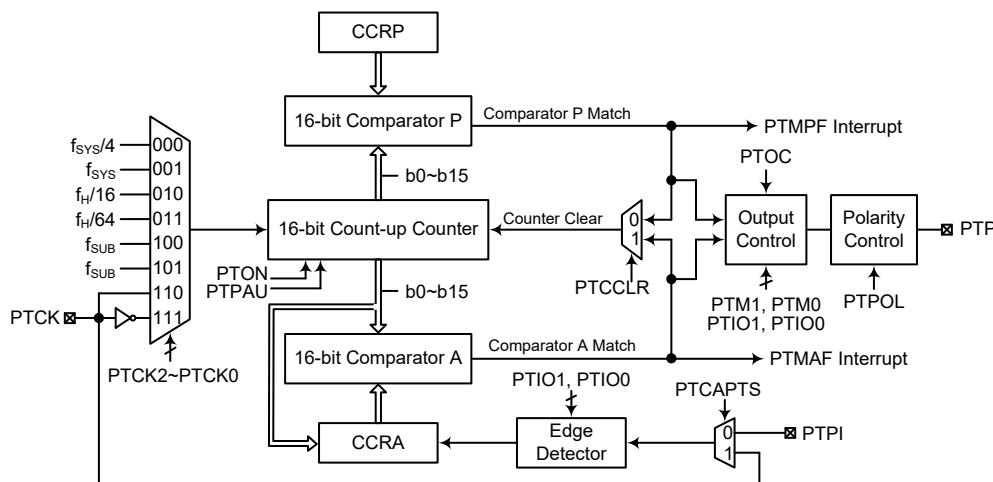
- Note: 1. Here CTD PX=0 – Counter cleared by CCRP
 2. A counter clear sets PWM Period
 3. The internal PWM function continues running even when CTIO[1:0]=00 or 01
 4. The CTCCLR bit has no influence on PWM operation



- Note: 1. Here CTD PX=1 – Counter cleared by CCRA
2. A counter clear sets PWM Period
3. The internal PWM function continues even when CTIO[1:0]=00 or 01
4. The CTCCLR bit has no influence on PWM operation

Periodic Type TM – PTM

The Periodic Type TM contains five operating modes, which are Compare Match Output, Timer/Event Counter, Capture Input, Single Pulse Output and PWM Output modes. The Periodic TM can also be controlled with two external input pins and can drive one external output pin.



Note: The PTM external pins are pin-shared with other functions, therefore before using the PTM function, ensure that the pin-shared function registers have been set properly to enable the PTM pin function. The PTCK and PTPI pins, if used, must also be set as an input by setting the corresponding bits in the port control register.

16-bit Periodic Type TM Block Diagram

Periodic TM Operation

The size of Periodic Type TM is 16-bit wide and its core is a 16-bit count-up counter which is driven by a user selectable internal or external clock source. There are also two internal comparators with the names, Comparator A and Comparator P. These comparators will compare the value in the counter with CCRP and CCRA registers. The CCRP and CCRA comparators are 16-bit wide whose value is respectively compared with all counter bits.

The only way of changing the value of the 16-bit counter using the application program is to clear the counter by changing the PTON bit from low to high. The counter will also be cleared automatically by a counter overflow or a compare match with one of its associated comparators. When these conditions occur, a PTM interrupt signal will also usually be generated. The Periodic Type TM can operate in a number of different operational modes, can be driven by different clock sources including an input pin and can also control an output pin. All operating setup conditions are selected using relevant internal registers.

Periodic Type TM Register Description

Overall operation of the Periodic TM is controlled using a series of registers. A read only register pair exists to store the internal counter 16-bit value, while two read/write register pairs exist to store the internal 16-bit CCRA and CCRP value. The remaining two registers are control registers which setup the different operating and control modes.

| Register Name | Bit | | | | | | | |
|---------------|-------|-------|-------|-------|------|-------|---------|--------|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PTMC0 | PTPAU | PTCK2 | PTCK1 | PTCK0 | PTON | — | — | — |
| PTMC1 | PTM1 | PTM0 | PTIO1 | PTIO0 | PTOC | PTPOL | PTCAPTS | PTCCLR |
| PTMDL | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| PTMDH | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| PTMAL | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| PTMAH | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| PTMRPL | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| PTMRPH | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |

16-bit Periodic TM Register List

• PTMC0 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|-------|-------|-------|------|---|---|---|
| Name | PTPAU | PTCK2 | PTCK1 | PTCK0 | PTON | — | — | — |
| R/W | R/W | R/W | R/W | R/W | R/W | — | — | — |
| POR | 0 | 0 | 0 | 0 | 0 | — | — | — |

Bit 7 **PTPAU**: PTM Counter Pause control

0: Run
1: Pause

The counter can be paused by setting this bit high. Clearing the bit to zero restores normal counter operation. When in a Pause condition the PTM will remain powered up and continue to consume power. The counter will retain its residual value when this bit changes from low to high and resume counting from this value when the bit changes to a low value again.

Bit 6~4 **PTCK2~PTCK0**: Select PTM Counter clock

000: $f_{SYS}/4$
001: f_{SYS}
010: $f_H/16$
011: $f_H/64$
100: f_{SUB}
101: f_{SUB}
110: PTCK rising edge clock
111: PTCK falling edge clock

These three bits are used to select the clock source for the PTM. The external pin clock source can be chosen to be active on the rising or falling edge. The clock source f_{SYS} is the system clock, while f_H and f_{SUB} are other internal clocks, the details of which can be found in the oscillator section.

Bit 3 **PTON**: PTM Counter On/Off control

0: Off
1: On

This bit controls the overall on/off function of the PTM. Setting the bit high enables the counter to run while clearing the bit disables the PTM. Clearing this bit to zero will stop the counter from counting and turn off the PTM which will reduce its power consumption. When the bit changes state from low to high the internal counter value will be reset to zero, however when the bit changes from high to low, the internal counter will retain its residual value.

If the PTM is in the Compare Match Output Mode, PWM Output Mode or Single Pulse Output Mode then the PTM output pin will be reset to its initial condition, as specified by the PTOC bit, when the PTON bit changes from low to high.

Bit 2~0 Unimplemented, read as “0”

• **PTMC1 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|------|------|-------|-------|------|-------|---------|--------|
| Name | PTM1 | PTM0 | PTIO1 | PTIO0 | PTOC | PTPOL | PTCAPTS | PTCCLR |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7~6 **PTM1~PTM0**: Select PTM Operating Mode
00: Compare Match Output Mode
01: Capture Input Mode
10: PWM Output Mode or Single Pulse Output Mode
11: Timer/Counter Mode

These bits setup the required operating mode for the PTM. To ensure reliable operation the PTM should be switched off before any changes are made to the PTM1 and PTM0 bits. In the Timer/Counter Mode, the PTM output pin state is undefined.

Bit 5~4 **PTIO1~PTIO0**: Select PTM external pin function selection

Compare Match Output Mode

00: No change
01: Output low
10: Output high
11: Toggle output

PWM Output Mode/Single Pulse Output Mode

00: PWM output inactive state
01: PWM output active state
10: PWM output
11: Single Pulse Output

Capture Input Mode

00: Input capture at rising edge of PTPI or PTCK
01: Input capture at falling edge of PTPI or PTCK
10: Input capture at rising/falling edge of PTPI or PTCK
11: Input capture disabled

Timer/Counter Mode

Unused

These two bits are used to determine how the PTM external pin changes state when a certain condition is reached. The function that these bits select depends upon in which mode the PTM is running. These two bits have no effect if the PTM is in the Timer/Counter Mode.

In the Compare Match Output Mode, the PTIO1 and PTIO0 bits determine how the PTM output pin changes state when a compare match occurs from Comparator A. The PTM output pin can be setup to switch high, switch low or to toggle its present state when a compare match occurs from Comparator A. When the bits are both zero, then no change will take place on the output. The initial value of the PTM output pin should be setup using the PTOC bit in the PTMC1 register. Note that the output level requested by the PTIO1 and PTIO0 bits must be different from the initial value setup using the PTOC bit otherwise no change will occur on the PTM output pin when a compare match occurs. After the PTM output pin changes state, it can be reset to its initial level by changing the level of the PTON bit from low to high.

In the PWM Output Mode, the PTIO1 and PTIO0 bits determine how the PTM output pin changes state when a certain compare match condition occurs. The PTM output function is modified by changing these two bits. It is necessary to only change the values of the PTIO1 and PTIO0 bits only after the PTM has been switched off. Unpredictable PWM outputs will occur if the PTIO1 and PTIO0 bits are changed when the PTM is running.

Bit 3 **PTOC**: PTM PTP Output control

Compare Match Output Mode

0: Initial low
1: Initial high

PWM Output Mode/Single Pulse Output Mode

0: Active low

1: Active high

This is the output control bit for the PTM output pin. Its operation depends upon whether PTM is being used in the Compare Match Output Mode or in the PWM Output Mode/Single Pulse Output Mode. It has no effect if the PTM is in the Timer/Counter Mode. In the Compare Match Output Mode it determines the logic level of the PTM output pin before a compare match occurs. In the PWM Output Mode it determines if the PWM signal is active high or active low. In the Single Pulse Output Mode it determines the logic level of the PTM output pin when the PTON bit changes from low to high.

Bit 2 **PTPOL**: PTM PTP Output polarity control

0: Non-invert

1: Invert

This bit controls the polarity of the PTP output pin. When the bit is set high the PTM output pin will be inverted and not inverted when the bit is zero. It has no effect if the PTM is in the Timer/Counter Mode.

Bit 1 **PTCAPTS**: PTM Capture Trigger Source selection

0: From PTPI pin

1: From PTCK pin

Bit 0 **PTCCLR**: PTM Counter Clear condition selection

0: Comparator P match

1: Comparator A match

This bit is used to select the method which clears the counter. Remember that the Periodic TM contains two comparators, Comparator A and Comparator P, either of which can be selected to clear the internal counter. With the PTCCLR bit set high, the counter will be cleared when a compare match occurs from the Comparator A. When the bit is low, the counter will be cleared when a compare match occurs from the Comparator P or with a counter overflow. A counter overflow clearing method can only be implemented if the CCRP bits are all cleared to zero. The PTCCLR bit is not used in the PWM Output, Single Pulse Output or Capture Input Mode.

• PTMDL Register

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

Bit 7~0 **D7~D0**: PTM Counter Low Byte Register bit 7 ~ bit 0

PTM 16-bit Counter bit 7 ~ bit 0

• PTMDH Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-----|-----|-----|-----|-----|-----|----|----|
| Name | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| R/W | R | R | R | R | R | R | R | R |
| POR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7~0 **D15~D8**: PTM Counter High Byte Register bit 7 ~ bit 0

PTM 16-bit Counter bit 15 ~ bit 8

• **PTMAL Register**

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

Bit 7~0 **D7~D0**: PTM CCRA Low Byte Register bit 7 ~ bit 0
PTM 16-bit CCRA bit 7 ~ bit 0

• **PTMAH Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| Name | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| 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 **D15~D8**: PTM CCRA High Byte Register bit 7 ~ bit 0
PTM 16-bit CCRA bit 15 ~ bit 8

• **PTMRPL Register**

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

Bit 7~0 **D7~D0**: PTM CCRP Low Byte Register bit 7 ~ bit 0
PTM 16-bit CCRP bit 7 ~ bit 0

• **PTMRPH Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| Name | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| 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 **D15~D8**: PTM CCRP High Byte Register bit 7 ~ bit 0
PTM 16-bit CCRP bit 15 ~ bit 8

Periodic Type TM Operation Modes

The Periodic Type TM can operate in one of five operating modes, Compare Match Output Mode, PWM Output Mode, Single Pulse Output Mode, Capture Input Mode or Timer/Counter Mode. The operating mode is selected using the PTM1 and PTM0 bits in the PTMC1 register.

Compare Match Output Mode

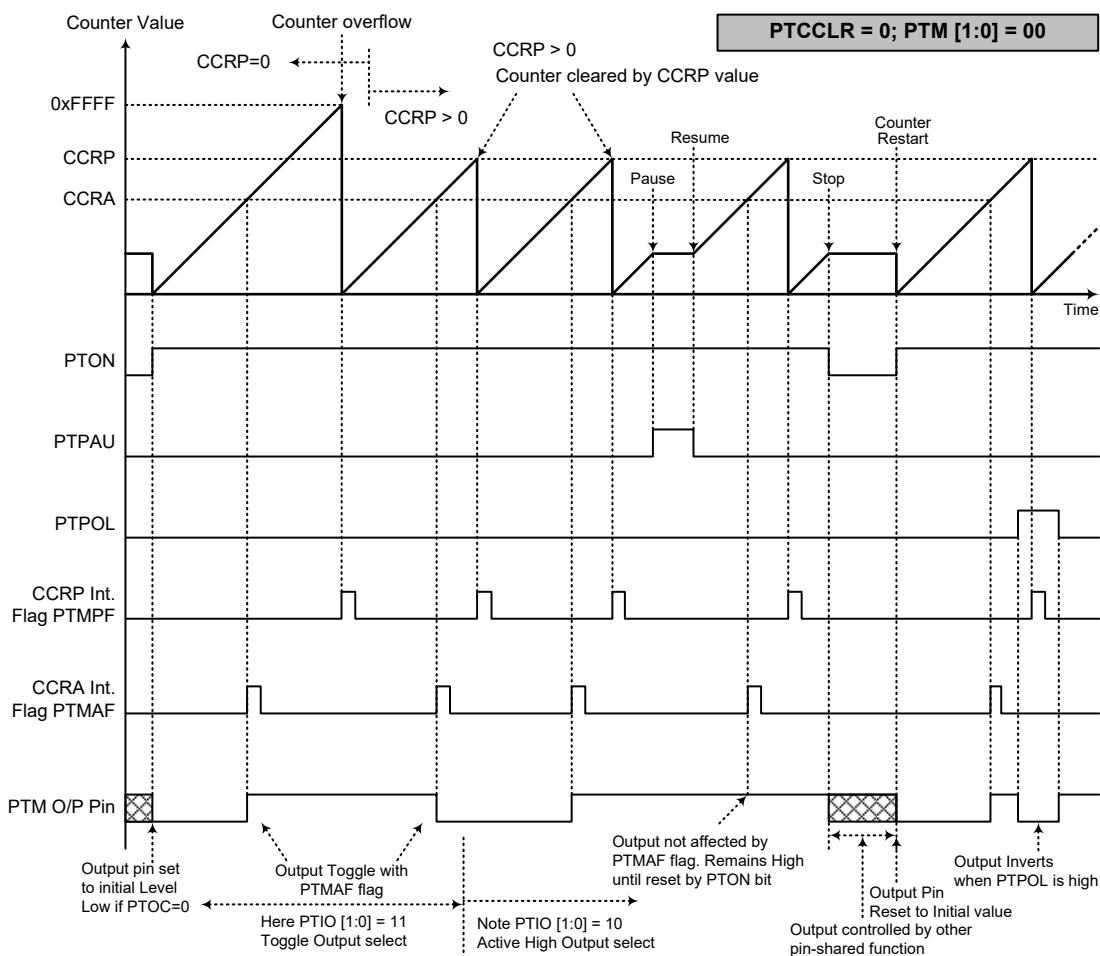
To select this mode, bits PTM1 and PTM0 in the PTMC1 register, should be set to 00 respectively. In this mode once the counter is enabled and running it can be cleared by three methods. These are a counter overflow, a compare match from Comparator A and a compare match from Comparator P. When the PTCCLR bit is low, there are two ways in which the counter can be cleared. One is when a compare match from Comparator P, the other is when the CCRP bits are all zero which allows the counter to overflow. Here both PTMAF and PTMPF interrupt request flags for Comparator A and Comparator P respectively, will both be generated.

If the PTCCLR bit in the PTMC1 register is high then the counter will be cleared when a compare match occurs from Comparator A. However, here only the PTMAF interrupt request flag will be generated even if the value of the CCRP bits is less than that of the CCRA registers. Therefore when

PTCCLR is high no PTMPF interrupt request flag will be generated. In the Compare Match Output Mode, the CCRA can not be cleared to “0”.

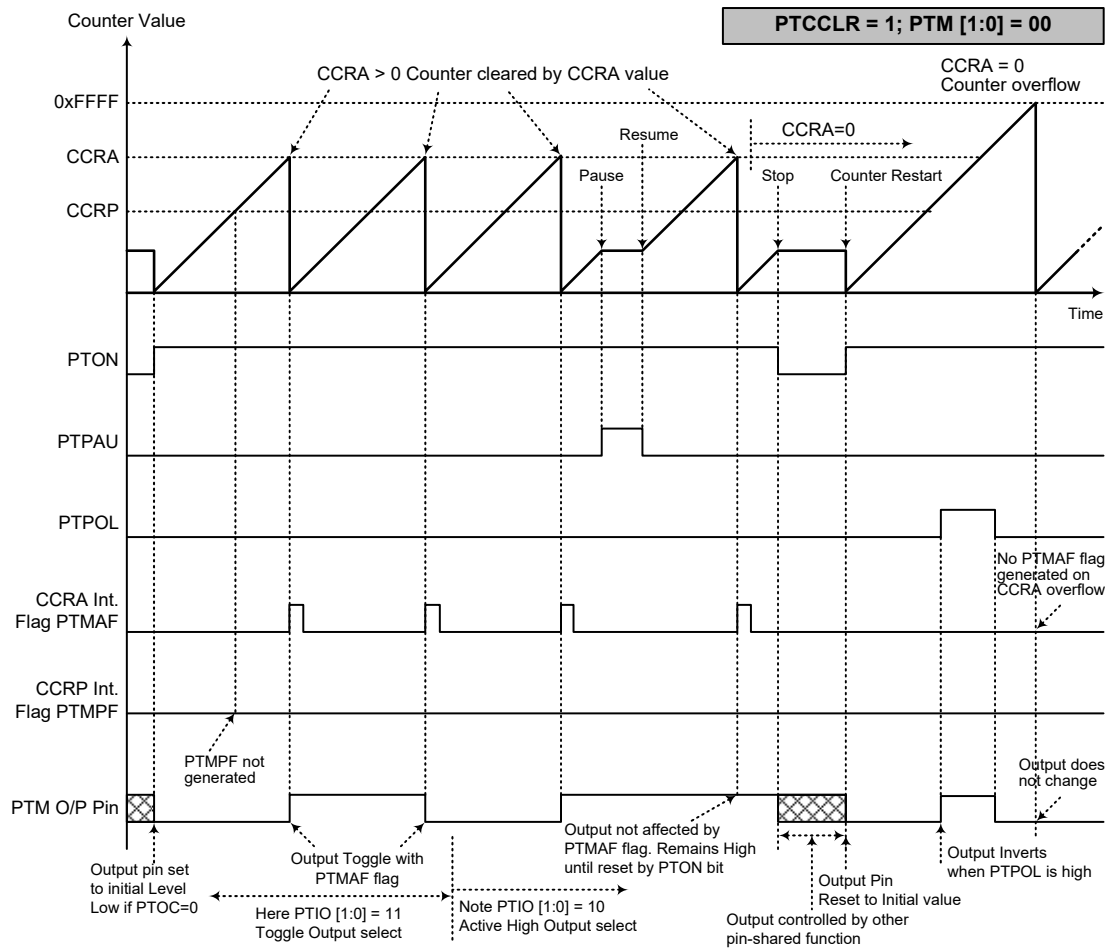
If the CCRA bits are all zero, the counter will overflow when it reaches its maximum 16-bit, FFFF Hex, value, however here the PTMAF interrupt request flag will not be generated.

As the name of the mode suggests, after a comparison is made, the PTM output pin will change state. The PTM output pin condition however only changes state when a PTMAF interrupt request flag is generated after a compare match occurs from Comparator A. The PTMPF interrupt request flag, generated from a compare match occurs from Comparator P, will have no effect on the PTM output pin. The way in which the PTM output pin changes state are determined by the condition of the PTIO1 and PTIO0 bits in the PTMC1 register. The PTM output pin can be selected using the PTIO1 and PTIO0 bits to go high, to go low or to toggle from its present condition when a compare match occurs from Comparator A. The initial condition of the PTM output pin, which is setup after the PTON bit changes from low to high, is setup using the PTOC bit. Note that if the PTIO1 and PTIO0 bits are zero then no pin change will take place.



Compare Match Output Mode – PTCCLR=0

- Note: 1. With PTCCLR=0, a Comparator P match will clear the counter
 2. The PTM output pin is controlled only by the PTMAF flag
 3. The output pin is reset to its initial state by a PTON bit rising edge



Compare Match Output Mode – PTCCLR=1

- Note: 1. With PTCCLR=1, a Comparator A match will clear the counter
2. The PTM output pin is controlled only by the PTMAF flag
3. The output pin is reset to its initial state by a PTON bit rising edge
4. A PTMPF flag is not generated when PTCCLR=1

Timer/Counter Mode

To select this mode, bits PTM1 and PTM0 in the PTMC1 register should be set to 11. The Timer/Counter Mode operates in an identical way to the Compare Match Output Mode generating the same interrupt flags. The exception is that in the Timer/Counter Mode the PTM output pin is not used. Therefore the above description and Timing Diagrams for the Compare Match Output Mode can be used to understand its function. As the PTM output pin is not used in this mode, the pin can be used as a normal I/O pin or other pin-shared function.

PWM Output Mode

To select this mode, bits PTM1 and PTM0 in the PTMC1 register should be set to 10. The PWM function within the PTM is useful for applications which require functions such as motor control, heating control, illumination control, etc. By providing a signal of fixed frequency but of varying duty cycle on the PTM output pin, a square wave AC waveform can be generated with varying equivalent DC RMS values.

As both the period and duty cycle of the PWM waveform can be controlled, the choice of generated waveform is extremely flexible. In the PWM output mode, the PTCCLR bit has no effect as the PWM period. Both of the CCRP and CCRA registers are used to generate the PWM waveform, the CCRP is used to clear the internal counter and thus control the PWM waveform frequency, while the CCRA is used to control the duty cycle. The PWM waveform frequency and duty cycle can therefore be controlled by the values in the CCRA and CCRP registers.

An interrupt flag, one for each of the CCRA and CCRP, will be generated when a compare match occurs from either Comparator A or Comparator P. The PTOC bit in the PTMC1 register is used to select the required polarity of the PWM waveform while the two PTIO1 and PTIO0 bits are used to enable the PWM output or to force the PTM output pin to a fixed high or low level. The PTPOL bit is used to reverse the polarity of the PWM output waveform.

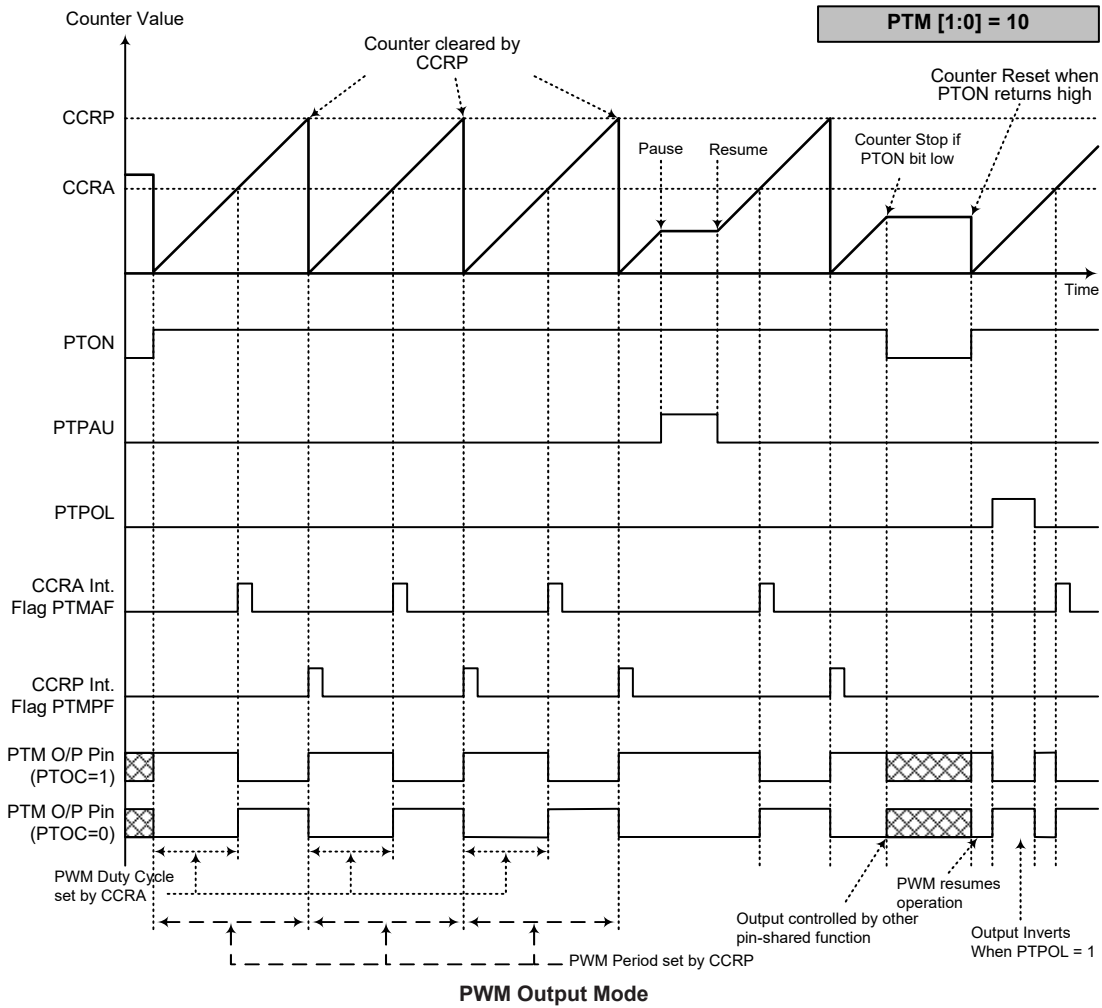
• 16-bit PTM, PWM Output Mode, Edge-aligned Mode

| CCRP | 1~65535 | 0 |
|--------|---------|-------|
| Period | 1~65535 | 65536 |
| Duty | CCRA | |

If $f_{SYS}=4\text{MHz}$, PTM clock source select $f_{SYS}/4$, CCRP=512 and CCRA=128,

The PTM PWM output frequency= $(f_{SYS}/4)/512=f_{SYS}/2048=1.9531\text{kHz}$, duty=128/512=25%,

If the Duty value defined by the CCRA register is equal to or greater than the Period value, then the PWM output duty is 100%.



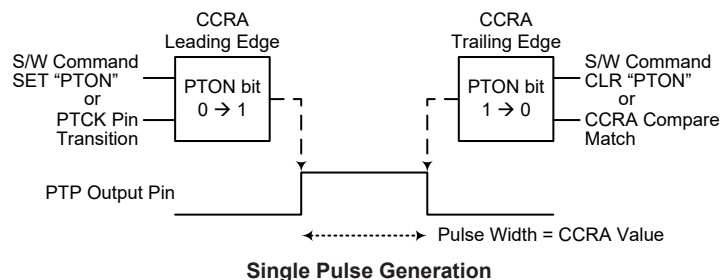
- Note: 1. The counter is cleared by CCRP
 2. A counter clear sets the PWM Period
 3. The internal PWM function continues running even when PTIO[1:0]=00 or 01
 4. The PTCCLR bit has no influence on PWM operation

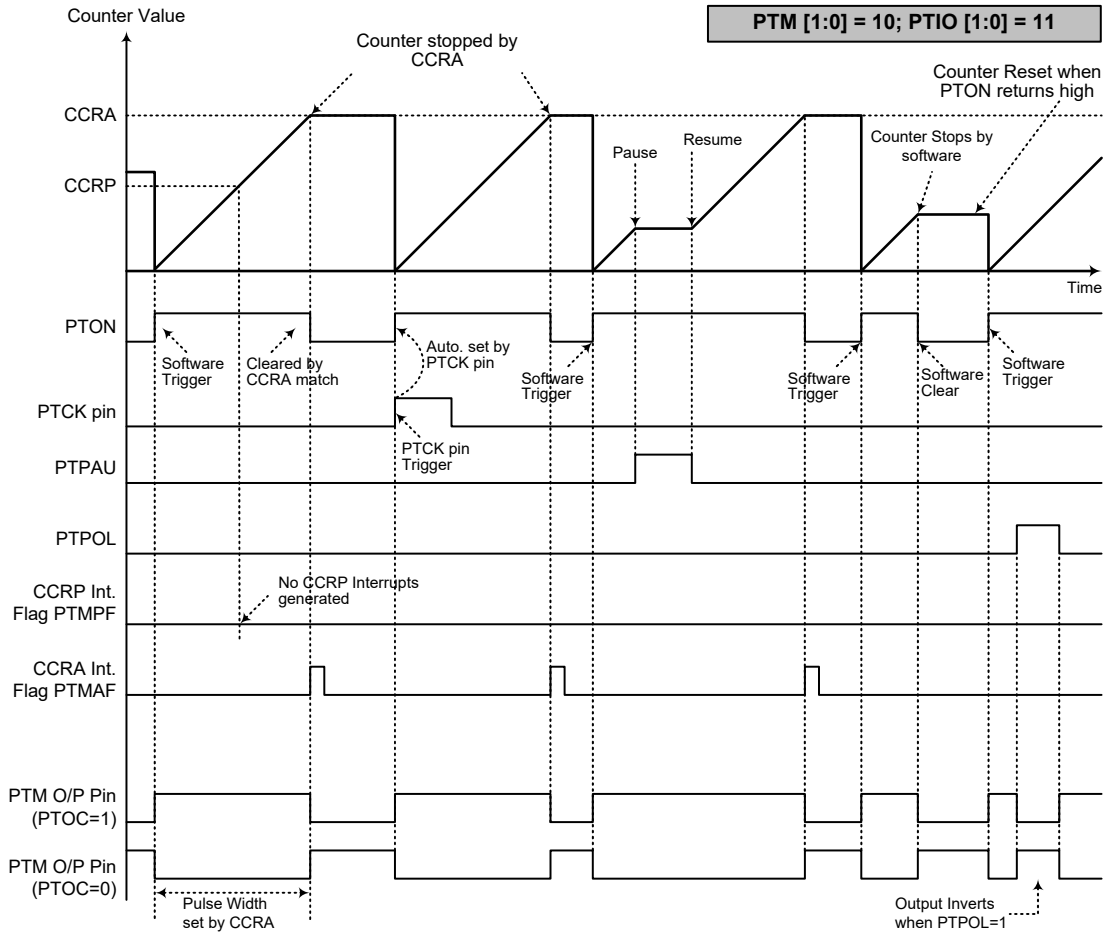
Single Pulse Output Mode

To select this mode, bits PTM1 and PTM0 in the PTMC1 register should be set to 10 and also the PTIO1 and PTIO0 bits should be set to 11. The Single Pulse Output Mode, as the name suggests, will generate a single shot pulse on the PTM output pin.

The trigger for the pulse output leading edge is a low to high transition of the PTON bit, which can be implemented using the application program. However in the Single Pulse Output Mode, the PTON bit can also be made to automatically change from low to high using the external PTCK pin, which will in turn initiate the Single Pulse output. When the PTON bit transitions to a high level, the counter will start running and the pulse leading edge will be generated. The PTON bit should remain high when the pulse is in its active state. The generated pulse trailing edge will be generated when the PTON bit is cleared to zero, which can be implemented using the application program or when a compare match occurs from Comparator A.

However a compare match from Comparator A will also automatically clear the PTON bit and thus generate the Single Pulse output trailing edge. In this way the CCRA value can be used to control the pulse width. A compare match from Comparator A will also generate a PTM interrupt. The counter can only be reset back to zero when the PTON bit changes from low to high when the counter restarts. In the Single Pulse Output Mode, CCRP is not used. The PTCCLR bit is not used in this mode.





Single Pulse Output Mode

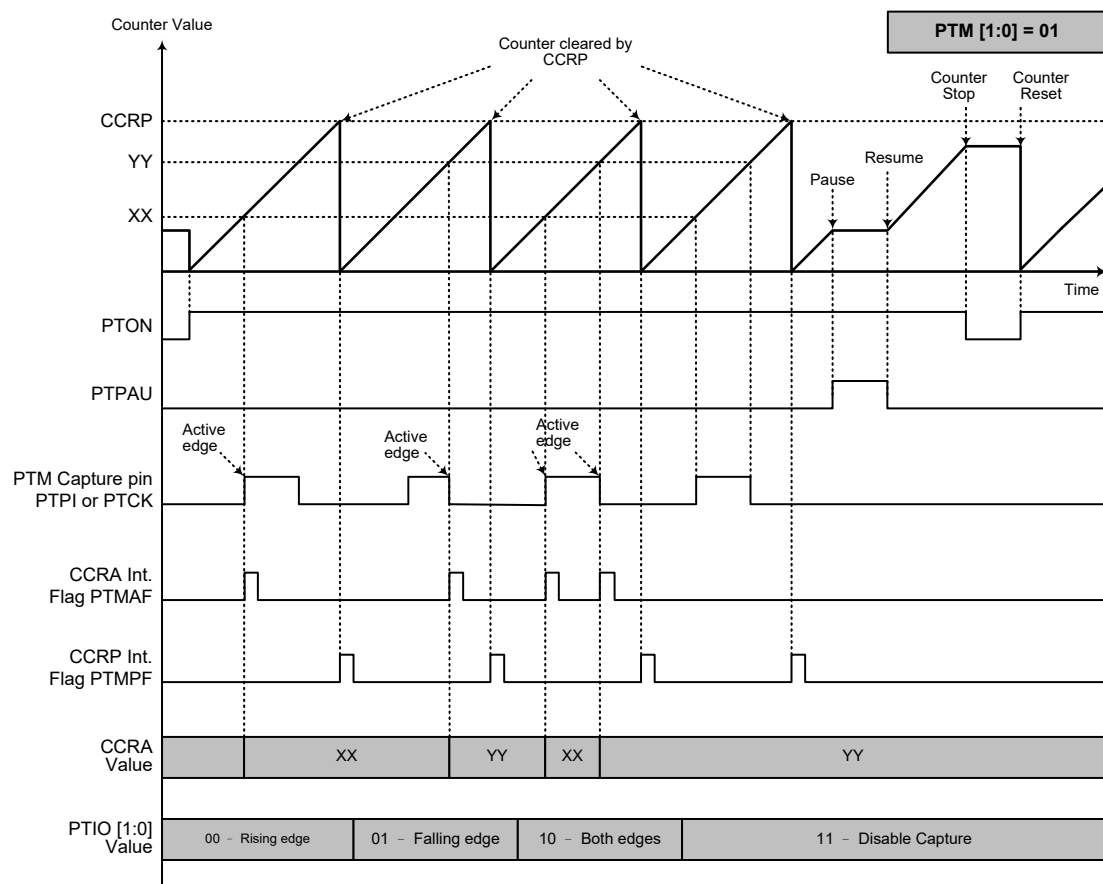
- Note:
1. Counter stopped by CCRA
 2. CCRP is not used
 3. The pulse triggered by the PTCK pin or by setting the PTON bit high
 4. A PTCK pin active edge will automatically set the PTON bit high
 5. In the Single Pulse Output Mode, PTIO [1:0] must be set to "11" and cannot be changed

Capture Input Mode

To select this mode bits PTM1 and PTM0 in the PTMC1 register should be set to 01. This mode enables external signals to capture and store the present value of the internal counter and can therefore be used for applications such as pulse width measurements. The external signal is supplied on the PTPI or PTCK pin, selected by the PTCAPTS bit in the PTMC1 register. The input pin active edge can be either a rising edge, a falling edge or both rising and falling edges; the active edge transition type is selected using the PTIO1 and PTIO0 bits in the PTMC1 register. The counter is started when the PTON bit changes from low to high which is initiated using the application program.

When the required edge transition appears on the PTPI or PTCK pin the present value in the counter will be latched into the CCRA registers and a PTM interrupt generated. Irrespective of what events occur on the PTPI or PTCK pin, the counter will continue to free run until the PTON bit changes from high to low. When a CCRP compare match occurs the counter will reset back to zero; in this way the CCRP value can be used to control the maximum counter value. When a CCRP compare match occurs from Comparator P, a PTM interrupt will also be generated. Counting the number of overflow interrupt signals from the CCRP can be a useful method in measuring long pulse widths. The PTIO1 and PTIO0 bits can select the active trigger edge on the PTPI or PTCK pin to be a rising edge, falling edge or both edge types. If the PTIO1 and PTIO0 bits are both set high, then no capture operation will take place irrespective of what happens on the PTPI or PTCK pin, however it must be noted that the counter will continue to run. The PTCCLR, PTOC and PTPOL bits are not used in this mode.

There are some considerations that should be noted. If PTCK is used as the capture input source, then it cannot be selected as the PTM clock source. If the captured pulse width is less than 2 timer clock periods, it may be ignored by hardware. After the counter value is latched to the CCRA registers by an active capture edge, the PTMAF flag will be set high after 0.5 timer clock period. The delay time from the active capture edge received to the action of latching counter value to CCRA registers is less than 1.5 timer clock periods.

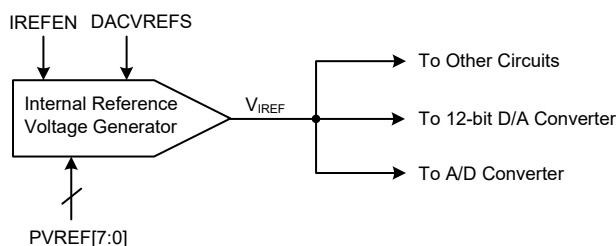


Capture Input Mode

- Note: 1. PTM [1:0]=01 and active edge set by the PTIO[1:0] bits
2. A PTM Capture input pin active edge transfers the counter value to CCRA
3. PTCCLR bit not used
4. No output function – PTOC and PTPOL bits are not used
5. CCRP determines the counter value and the counter has a maximum count value when CCRP is equal to zero
6. The capture input mode cannot be used if the selected PTM counter clock is not available

Internal Reference Voltage Generator

The device includes an internal reference voltage generator to provide an accurate reference voltage V_{REF} . This reference voltage can be used as the internal 12-bit D/A Converter or 24-bit Delta Sigma A/D Converter reference voltage or can be output through the DACVREF pin by properly settings to use for other circuits. Refer to the Internal Reference Voltage Characteristics section for more information.



Internal Reference Voltage Register Description

The internal reference voltage is controlled by two registers. The IREFC register is used for the enable/disable control while the PVREF register is used for fine tuning the internal reference voltage value.

| Register Name | Bit | | | | | | | |
|---------------|-----|----|----|--------|----|----------|---------|---------|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IREFC | — | — | — | IREFEN | — | DACVREFS | DACVRS1 | DACVRS0 |
| PVREF | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

Internal Reference Voltage Register List

• IREFC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|---|--------|---|----------|---------|---------|
| Name | — | — | — | IREFEN | — | DACVREFS | DACVRS1 | DACVRS0 |
| R/W | — | — | — | R/W | — | R/W | R/W | R/W |
| POR | — | — | — | 0 | — | 0 | 0 | 0 |

Bit 7~5 Unimplemented, read as “0”

Bit 4 **IREFEN**: Internal Reference Voltage Generator control

0: Disable

1: Enable

This bit controls the internal reference voltage generator on/off. When this bit is set high, the internal reference voltage V_{REF} can be generated and used as the reference voltage. If the internal reference voltage is not used by any circuits, the IREFEN bit should be cleared to zero to reduce power consumption.

Bit 3 Unimplemented, read as “0”

Bit 2 **DACVREFS**: V_{REF} voltage selection

0: 1.25V

1: 1.83V

Bit 1~0 **DACVRS1~DACVRS0**: 12-bit D/A Converter reference voltage V_{DACVREF} selection

Refer to the D/A Converter Registers section.

• **PVREF Register**

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

Bit 7~0

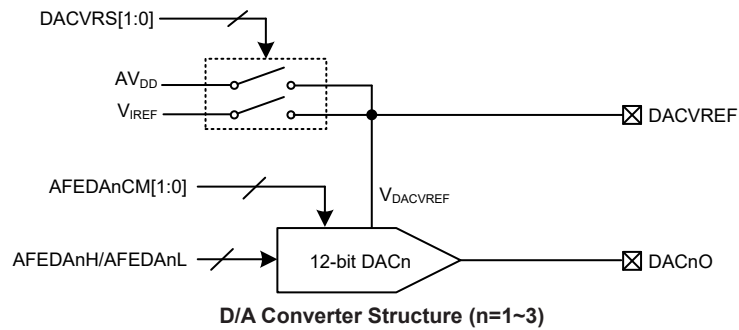
D7~D0: Internal Reference Voltage Generator fine tune control

If the V_{IREF} voltage is 1.25V, setting the register can fine tune the internal reference voltage with a range of -60mV~+60mV (based on PVREF=80H). When the PVREF register value is increased by one, the internal reference voltage will decrease around 500μV, vice verse.

If the V_{IREF} voltage is 1.83V, setting the register can fine tune the internal reference voltage with a range of -70mV~+70mV (based on PVREF=80H). When the PVREF register value is increased by one, the internal reference voltage will decrease around 1000μV, vice verse.

Digital to Analog Converter – DAC

This device includes three 12-bit D/A Converters which can provide a certain voltage ranging from 0 to $V_{DACVREF}$ to the positive input of the internal operational amplifiers. The D/A converter reference voltage, $V_{DACVREF}$, can also be used for the A/D converter reference voltage.



D/A Converter Structure (n=1~3)

D/A Converter Register Description

The D/A converter overall function is controlled by several registers. The DACVRS1~DACVRS0 bits in the IREFC register are used to select the D/A converter reference voltage. The DACVREFS bit in the IREFC register is used to select the V_{IREF} voltage value. The AFEDAnC registers are used for the D/A converter n function enable/disable control. Three 12-bit register pairs, AFEDAnH and AFEDAnL, are used for the D/A converter n output control.

| Register Name | Bit | | | | | | | |
|---------------|-----|-----|----|--------|----|----------|-----------|-----------|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IREFC | — | — | — | IREFEN | — | DACVREFS | DACVRS1 | DACVRS0 |
| AFEDAnC | — | — | — | — | — | — | AFEDAnCM1 | AFEDAnCM0 |
| AFEDAnL | D3 | D2 | D1 | D0 | — | — | — | — |
| AFEDAnH | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 |

D/A Converter Register List (n=1~3)

• IREFC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|---|--------|---|----------|---------|---------|
| Name | — | — | — | IREFEN | — | DACVREFS | DACVRS1 | DACVRS0 |
| R/W | — | — | — | R/W | — | R/W | R/W | R/W |
| POR | — | — | — | 0 | — | 0 | 0 | 0 |

Bit 7~5 Unimplemented, read as “0”

Bit 4 **IREFEN**: Internal Reference Voltage Generator control
Refer to the Internal Reference Voltage Generator section.

Bit 3 Unimplemented, read as “0”

Bit 2 **DACVREFS**: V_{IREF} voltage selection
Refer to the Internal Reference Voltage Generator section.

Bit 1~0 **DACVRS1~DACVRS0**: 12-bit D/A Converter Reference Voltage $V_{DACVREF}$ selection
00/01: V_{IREF}
10: AV_{DD}
11: Floating

• AFEDAnC Register (n=1~3)

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|---|---|---|---|-----------|-----------|
| Name | — | — | — | — | — | — | AFEDAnCM1 | AFEDAnCM0 |
| R/W | — | — | — | — | — | — | R/W | R/W |
| POR | — | — | — | — | — | — | 0 | 0 |

Bit 7~2 Unimplemented, read as “0”

Bit 1~0 **AFEDAnCM1~AFEDAnCM0**: 12-bit D/A Converter n mode control
00: Disable, output in a high impedance state
01/11: Enable
10: Disable, output in a ground state

• AFEDAnH & AFEDAnL Registers (n=1~3)

| Register | AFEDAnH | | | | | | | | AFEDAnL | | | | | | | |
|----------|---------|-----|-----|-----|-----|-----|-----|-----|---------|-----|-----|-----|---|---|---|---|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Name | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | — | — | — | — |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | — | — | — | — |
| POR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | — | — | — | — |

“—”: Unimplemented, read as “0”

D11~D0: 12-bit D/A Converter n output control bits

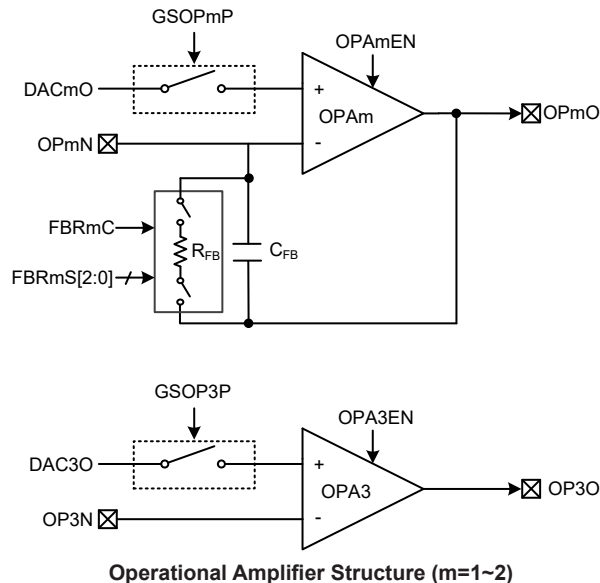
The bit 7 ~ bit 0 in the AFEDAnH register combine with the bit 7 ~ bit 4 in the AFEDAnL register to form a 12-bit DACn value of 0~4095.

DACn Output Voltage (V_{DACO})= $V_{DACVREF} \times (\text{DACn value}/4096)$

Note: It is necessary to firstly write into the AFEDAnL register followed by the AFEDAnH register.

Operational Amplifiers

This device includes three operational amplifiers for measurement applications. The 12-bit D/A Converters offer a voltage of $0 \sim V_{DACVREF}$ to the positive input terminals of the operational amplifiers. By proper setup, the OPAn ($n=1 \sim 3$) output voltage can be connected to the A/D converter internal input channel for measurement.



Operational Amplifier Register Description

The overall operation of the internal Operational Amplifiers is controlled by a series of registers. The OPAC and GSC1 registers are used for the OPAn enable/disable control. The FBRC register is used to control the feedback resistor connection of the OPA1 and OPA2 and to set the corresponding resistance.

| Register Name | Bit | | | | | | | |
|---------------|--------|--------|--------|--------|-------|--------|--------|--------|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| OPAC | OPA3EN | OPA2EN | OPA1EN | — | — | — | — | — |
| GSC1 | — | — | — | — | — | GSOP3P | GSOP2P | GSOP1P |
| FBRC | FBR1C | FBR1S2 | FBR1S1 | FBR1S0 | FBR2C | FBR2S2 | FBR2S1 | FBR2S0 |

Operational Amplifier Register List

• OPAC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|--------|--------|--------|---|---|---|---|---|
| Name | OPA3EN | OPA2EN | OPA1EN | — | — | — | — | — |
| R/W | R/W | R/W | R/W | — | — | — | — | — |
| POR | 0 | 0 | 0 | — | — | — | — | — |

Bit 7 **OPA3EN**: OPA3 enable/disable control
0: Disable
1: Enable

Bit 6 **OPA2EN**: OPA2 enable/disable control
0: Disable
1: Enable

Bit 5 **OPA1EN**: OPA1 enable/disable control
 0: Disable
 1: Enable
 Bit 4~0 Unimplemented, read as “0”

• GSC1 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|---|---|---|--------|--------|--------|
| Name | — | — | — | — | — | GSOP3P | GSOP2P | GSOP1P |
| R/W | — | — | — | — | — | R/W | R/W | R/W |
| POR | — | — | — | — | — | 0 | 0 | 0 |

Bit 7~3 Unimplemented, read as “0”
 Bit 2 **GSOP3P**: OP3P Selection switch for DAC3O
 0: Off
 1: On
 Bit 1 **GSOP2P**: OP2P Selection switch for DAC2O
 0: Off
 1: On
 Bit 0 **GSOP1P**: OP1P Selection switch for DAC1O
 0: Off
 1: On

Note: When the OPAnEN is enabled, the GSOPnP must be enabled.

• FBRC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|--------|--------|--------|-------|--------|--------|--------|
| Name | FBR1C | FBR1S2 | FBR1S1 | FBR1S0 | FBR2C | FBR2S2 | FBR2S1 | FBR2S0 |
| 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 **FBR1C**: OPA1 feedback resistor connection control
 0: Not connected
 1: Connected
 Bit 6~4 **FBR1S2~FBR1S0**: OPA1 feedback resistance setting
 000: 0.5MΩ
 001: 1MΩ
 010: 2MΩ
 011: 4MΩ
 100: 8MΩ
 Others: Reserved
 Bit 3 **FBR2C**: OPA2 feedback resistor connection control
 0: Not connected
 1: Connected
 Bit 2~0 **FBR2S2~FBR2S0**: OPA2 feedback resistance setting
 000: 0.5MΩ
 001: 1MΩ
 010: 2MΩ
 011: 4MΩ
 100: 8MΩ
 Others: Reserved

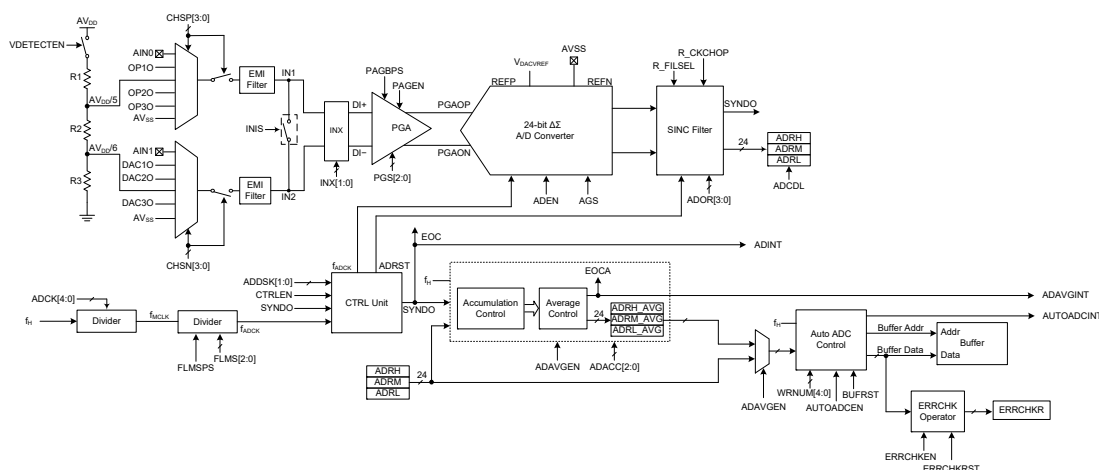
Analog to Digital Converter – ADC

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

A/D Converter Overview

This device contains a high-accuracy multi-channel 24-bit Delta Sigma analog to digital converter which can directly interface to external analog signals, such as that from sensors or other control signals and convert these signals directly into a 24-bit digital value.

In addition, PGA gain control and A/D converter gain control determine the amplification gain for A/D converter input signal. The designer can select the best gain combination for the desired amplification applied to the input signal. The following block diagram illustrates the A/D converter basic operational function. The A/D converter external input channels can be arranged as two single-ended A/D converter input channels or one differential external input channel. The input signal can be amplified by PGA before entering the 24-bit Delta Sigma A/D converter. The A/D converter module will output one bit converted data to SINC filter which can transform the converted one-bit data to 24 bits and store them into the specific data registers. With high accuracy and performance, the device is very suitable for differential output sensor applications such as weight measurement scales and other related products.



Note: The PGA and A/D converter are supplied by the V_{DD} and V_{SS} . The SINC Filter is supplied by the V_{DD} and V_{SS} .

A/D Converter Structure

A/D Converter Register Description

The overall operation of the A/D converter is controlled by a series of registers.

| Register Name | Bit | | | | | | | |
|---------------|-----------|-----------|--------|--------|-----------|--------|----------|----------|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PWRC | — | — | — | — | VDETECTEN | — | — | — |
| PGAC | — | INIS | INX1 | INX0 | AGS | PGS2 | PGS1 | PGS0 |
| PGACS | CHSN3 | CHSN2 | CHSN1 | CHSN0 | CHSP3 | CHSP2 | CHSP1 | CHSP0 |
| MODC | — | — | — | — | ADEN | ADRST | PGABPS | PGAEN |
| ADRL | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| ADRM | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| ADRH | D23 | D22 | D21 | D20 | D19 | D18 | D17 | D16 |
| ADCR0 | — | — | — | ADOR3 | ADOR2 | ADOR1 | ADOR0 | — |
| ADCR1 | FLMS2 | FLMS1 | FLMS0 | FLMSPS | — | ADCDL | EOC | — |
| ADCS | — | — | — | ADCK4 | ADCK3 | ADCK2 | ADCK1 | ADCK0 |
| SINC3 | D7 | D6 | D5 | D4 | D3 | D2 | R_FILSEL | R_CKCHOP |
| CTRL | CTRLLEN | ADDSK1 | ADDSK0 | — | — | — | — | — |
| ADACCTRL | ADAVGEN | EOCA | — | — | — | ADACC2 | ADACC1 | ADACC0 |
| ADRL_AVG | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| ADRM_AVG | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| ADRH_AVG | D23 | D22 | D21 | D20 | D19 | D18 | D17 | D16 |
| AUTOADCC | AUTOADCEN | — | WRNUM4 | WRNUM3 | WRNUM2 | WRNUM1 | WRNUM0 | BUFRST |
| ERRCHKC | ERRCHKEN | ERRCHKRST | — | — | — | — | — | — |
| ERRCHKR | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

A/D Converter Register List

Programmable Gain Amplifier Registers – PWRC, PGAC, PGACS, MODC

There are four registers related to the programmable gain control, PWRC, PGAC, PGACS and MODC. The PWRC register is used to control the $AV_{DD}/5$ and $AV_{DD}/6$ voltage divider. The PGAC register is used to select the PGA gain, A/D converter gain and to define the PGA input connection. The MODC register is used to define the A/D converter enable/disable control, PGA enable/disable control and PGA bypass control. In addition, the PGACS register is used to select the PGA inputs. Therefore, the input channels have to be determined by the CHSP3~CHSP0 and CHSN3~CHSN0 bits to determine which analog channel signals, OPA outputs, DAC outputs or internal power supply are actually connected to the internal differential A/D converter.

• PWRC Register

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

Bit 7~4 Unimplemented, read as “0”

Bit 3 **VDETECTEN**: Voltage divider $AV_{DD}/5$ and $AV_{DD}/6$ control
 0: Disable
 1: Enable

Bit 2~0 Unimplemented, read as “0”

• **PGAC Register**

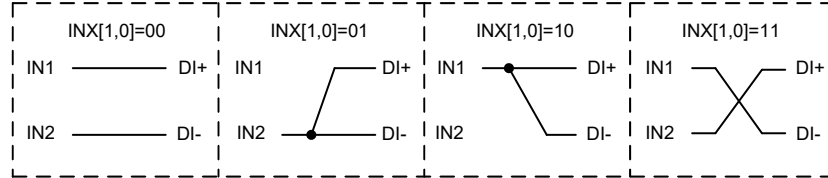
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|------|------|------|-----|------|------|------|
| Name | — | INIS | INX1 | INX0 | AGS | PGS2 | PGS1 | PGS0 |
| R/W | — | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | — | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7 Unimplemented, read as “0”

Bit 6 **INIS**: Selected inputs, IN1/IN2, internal connection control
 0: Not connected
 1: Connected

When PGAEN=0 and ADEN=0, the INIS bit setting is invalid.

Bit 5~4 **INX1~INX0**: Selected inputs, IN1/IN2, and the PGA differential input ends, DI+/DI- connection control bit



Bit 3 **AGS**: A/D converter PGAOP/PAGON differential input signal gain selection
 0: ADGN=1
 1: ADGN=2

Bit 2~0 **PGS2~PGS0**: PGA DI+/DI- differential channel input gain selection
 000: PGAGN=1
 001: PGAGN=2
 010: PGAGN=4
 011: PGAGN=8
 100: PGAGN=16
 101: PGAGN=32
 110: PGAGN=64
 111: PGAGN=128

• **PGACS Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|-------|-------|-------|-------|-------|-------|-------|
| Name | CHSN3 | CHSN2 | CHSN1 | CHSN0 | CHSP3 | CHSP2 | CHSP1 | CHSP0 |
| 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 7~4 **CHSN3~CHSN0**: Negative input end IN2 selection

0000~0111: Reserved
 1000: DAC1O
 1001: DAC2O
 1010: DAC3O
 1011: AV_{DD}/6
 1100: AV_{SS}
 1101: AIN1
 1110~1111: Reserved

Bit 3~0 **CHSP3~CHSP0**: Positive input end IN1 selection

0000~0111: Reserved
 1000: OP1O
 1001: OP2O
 1010: OP3O
 1011: AV_{SS}
 1100: AV_{DD}/5
 1101: AIN0
 1110~1111: Reserved

• MODC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|---|---|------|-------|--------|-------|
| Name | — | — | — | — | ADEN | ADRST | PGABPS | PGAEN |
| R/W | — | — | — | — | R/W | R/W | R/W | R/W |
| POR | — | — | — | — | 0 | 0 | 0 | 0 |

Bit 7~4 Unimplemented, read as “0”

Bit 3 **ADEN**: A/D enable/disable control

0: Disable

1: Enable

Bit 2 **ADRST**: A/D converter software reset control

0: Disable

1: Enable

This bit is used to reset the A/D converter internal digital SINC filter. This bit is set low for A/D normal operations. However, if this bit is set high, the internal digital SINC filter will be reset and the current A/D converted data will be aborted. A new A/D data conversion process will not be initiated until this bit is set low again.

Bit 1 **PGABPS**: PGA bypass control

0: Normal mode

1: Bypass PGA mode

When this bit is set high, the PGAEN bit setting does not matter.

Bit 0 **PGAEN**: PGA enable/disable control

0: Disable

1: Enable

The PGAEN bit must be enabled before using the PGA function.

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

The 24-bit Delta Sigma A/D converter requires three data registers to store the converted value. These are a high byte register, known as ADRH, a middle byte register, known as ADRM, and a low byte register, known as ADRL. After the conversion process is complete, these registers can be directly read by the microcontroller to obtain the digitised conversion value, D23~D0

• ADRL Register

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

“x”: unknown

Bit 7~0 **D7~D0**: A/D conversion data register bit 7 ~ bit 0

• ADRM Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-----|-----|-----|-----|-----|-----|----|----|
| Name | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| R/W | R | R | R | R | R | R | R | R |
| POR | x | x | x | x | x | x | x | x |

“x”: unknown

Bit 7~0 **D15~D8**: A/D conversion data register bit 15 ~ bit 8

• **ADRH Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| Name | D23 | D22 | D21 | D20 | D19 | D18 | D17 | D16 |
| R/W | R | R | R | R | R | R | R | R |
| POR | x | x | x | x | x | x | x | x |

“x”: unknown

Bit 7~0 **D23~D16**: A/D conversion data register bit 23 ~ bit 16

A/D Converter Control Registers – ADCR0, ADCR1, ADCS

To control the function and operation of the A/D converter, several control registers known as ADCR0, ADCR1 and ADCS are provided. These 8-bit registers define functions such as the selection of which oversampling rate is used by the internal A/D converter and set the A/D converter clock.

• **ADCR0 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|---|-------|-------|-------|-------|---|
| Name | — | — | — | ADOR3 | ADOR2 | ADOR1 | ADOR0 | — |
| R/W | — | — | — | R/W | R/W | R/W | R/W | — |
| POR | — | — | — | 0 | 0 | 0 | 0 | — |

Bit 7~5 Unimplemented, read as “0”

Bit 4~1 **ADOR3~ADOR0**: A/D conversion oversampling rate selection

0000: Oversampling rate OSR=32768
 0001: Oversampling rate OSR=16384
 0010: Oversampling rate OSR=8192
 0011: Oversampling rate OSR=4096
 0100: Oversampling rate OSR=2048
 0101: Oversampling rate OSR=1024
 0110: Oversampling rate OSR=512
 0111: Oversampling rate OSR=256
 1000: Oversampling rate OSR=128
 Others: Reserved

Bit 0 Unimplemented, read as “0”

• **ADCR1 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|-------|-------|--------|---|-------|-----|---|
| Name | FLMS2 | FLMS1 | FLMS0 | FLMSPS | — | ADCDL | EOC | — |
| R/W | R/W | R/W | R/W | R/W | — | R/W | R/W | — |
| POR | 0 | 0 | 0 | 0 | — | 0 | 0 | — |

Bit 7~5 **FLMS2~FLMS0**: A/D converter clock f_{ADCK} and divide factor (N) selection

000: $f_{ADCK}=f_{MCLK}/30$, N=30
 010: $f_{ADCK}=f_{MCLK}/12$, N=12
 Others: Reserved

Bit 4 **FLMSPS**: A/D converter clock division bypass control

0: Disable
 1: Enable, $f_{ADCK}=f_{MCLK}$

Bit 3 Unimplemented, read as “0”

Bit 2 **ADCDL**: A/D converted data latch function control

0: Disable data latch function
 1: Enable data latch function

If the A/D converted data latch function is enabled, the latest converted data value will be latched and will not be updated by any subsequent conversion results until this function is disabled. Although the converted data is latched into the data registers, the A/D converter circuits remain operational, but will not generate an interrupt and the EOC bit will not change. It is recommended that this bit should be set high before reading the converted data from the ADRL, ADRM and ADRH registers. After the converted data has been read out, the bit can then be cleared to zero to disable the A/D converter data latch function and allow further conversion values to be stored. In this way, the possibility of obtaining undesired data during A/D converter conversions can be prevented.

- Bit 1 **EOC**: End of A/D conversion flag
 0: A/D conversion in progress
 1: A/D conversion ended
 This bit must be cleared by software.
- Bit 0 Unimplemented, read as “0”

• ADCS Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|---|-------|-------|-------|-------|-------|
| Name | — | — | — | ADCK4 | ADCK3 | ADCK2 | ADCK1 | ADCK0 |
| R/W | — | — | — | R/W | R/W | R/W | R/W | R/W |
| POR | — | — | — | 0 | 0 | 0 | 0 | 0 |

- Bit 7~5 Unimplemented, read as “0”
- Bit 4~0 **ADCK4~ADCK0**: A/D converter clock source f_{MCLK} setup
 00000~11110: $f_{MCLK}=f_H/2/(ADCK[4:0]+1)$
 11111: $f_{MCLK}=f_H$

SINC Filter Register – SINC3

There is a register related to the SINC Filter control, SINC3. This register is used for the output digital filter selection and the chopper function control.

• SINC3 Register

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

- Bit 7~2 **D7~D2**: Reserved bits, cannot be changed
- Bit 1 **R_FILSEL**: Output digital filter selection
 0: SINC2 filter (R_CKCHOP should be logic low), Data latency=3 output data clocks
 1: SINC3 filter (R_CKCHOP should be logic high), Data latency=4 output data clocks
- Bit 0 **R_CKCHOP**: System chopper function selection
 0: Enable (R_FILSEL should be logic low)
 1: Disable (R_FILSEL should be logic high)

CTRL Unit Register – CTRL

There is a register related to the CTRL unit control, CTRL. This register is used to define the CTRL unit enable/disable control and to select the number of A/D converted data to skip.

• CTRL Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---------|--------|--------|---|---|---|---|---|
| Name | CTRLLEN | ADDSK1 | ADDSK0 | — | — | — | — | — |
| R/W | R/W | R/W | R/W | — | — | — | — | — |
| POR | 0 | 0 | 0 | — | — | — | — | — |

Bit 7 **CTRLLEN**: CTRL unit enable/disable control

0: Disable

1: Enable

If the CTRL unit is enabled, the problem of the initial state difference after A/D converter circuits reset can be solved. The first converted data value does not need to be discarded. And after the A/D conversion is completed, the CTRL unit will automatically skip the converted data value according to the ADDSK[1:0] setting. If the CTRL unit is disabled, the converted data values will not be skipped.

Bit 6~5 **ADDSK1~ADDSK0**: Select the number of A/D converted data to skip

00: Skip 1 A/D converted data

01: Skip 2 A/D converted data

10: Skip 3 A/D converted data

11: Skip 4 A/D converted data

Bit 4~0 Unimplemented, read as “0”

A/D Data Accumulation and Average Registers – ADACCTRL, ADRL_AVG, ADRM_AVG, ADRH_AVG

To control the function and operation of the A/D data accumulation and average function, several registers known as ADACCTRL, ADRL_AVG, ADRM_AVG and ADRH_AVG are provided. The ADACCTRL is provided to define the function enable/disable control and the accumulation and average times. The ADRL_AVG, ADRM_AVG and ADRH_AVG registers are used to store the averaged value after accumulation.

• ADACCTRL Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---------|------|---|---|---|--------|--------|--------|
| Name | ADAVGEN | EOCA | — | — | — | ADACC2 | ADACC1 | ADACC0 |
| R/W | R/W | R/W | — | — | — | R/W | R/W | R/W |
| POR | 0 | 0 | — | — | — | 0 | 0 | 0 |

Bit 7 **ADAVGEN**: A/D converted data accumulation and average enable/disable control

0: Disable

1: Enable

Bit 6 **EOCA**: End of A/D converted data accumulation and average flag

0: A/D converted data accumulation and average in progress

1: A/D converted data accumulation and average ended

This bit must be cleared by software.

Bit 5~3 Unimplemented, read as “0”

Bit 2~0 **ADACC2~ADACC0**: Select A/D converter output execution accumulation and average times

000: 2² times

001: 2³ times

010: 2⁴ times

011: 2⁵ times

Others: 2⁶ times

• ADRL_AVG Register

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

"x": unknown

Bit 7~0 **D7~D0**: A/D conversion data average register bit 7 ~ bit 0

• ADRM_AVG Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-----|-----|-----|-----|-----|-----|----|----|
| Name | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| R/W | R | R | R | R | R | R | R | R |
| POR | x | x | x | x | x | x | x | x |

"x": unknown

Bit 7~0 **D15~D8**: A/D conversion data average register bit 15 ~ bit 8

• ADRH_AVG Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| Name | D23 | D22 | D21 | D20 | D19 | D18 | D17 | D16 |
| R/W | R | R | R | R | R | R | R | R |
| POR | x | x | x | x | x | x | x | x |

"x": unknown

Bit 7~0 **D23~D16**: A/D conversion data average register bit 23 ~ bit 16

A/D Converter Auto Mode Register – AUTOADCC

There is a register related to the A/D converter auto mode control, AUTOADCC. This register is used to define the A/D converter auto mode enable/disable control and to select the number of A/D converted data to be written into the buffer.

• AUTOADCC Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-----------|---|--------|--------|--------|--------|--------|--------|
| Name | AUTOADCEN | — | WRNUM4 | WRNUM3 | WRNUM2 | WRNUM1 | WRNUM0 | BUFRST |
| 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 **AUTOADCEN**: A/D converter auto mode enable/disable control

0: Disable

1: Enable

If the A/D converter auto mode is enabled, the A/D converted data will be automatically sent into the buffer until the number of data written into the buffer reaches the WRNUM[4:0] bit setting, at this time an A/D converter auto mode interrupt will be generated, and the A/D converter auto mode will be automatically disabled.

Bit 6 Unimplemented, read as "0"

Bit 5~1 **WRNUM4~WRNUM0**: Set the number of A/D converted data to be written into the buffer

00000: 1

:

10011: 20

10100~11111: 21

The buffer size is 64 bytes, and up to 21 A/D converted data can be written into the buffer at one time.

Bit 0 **BUFRST**: Buffer address software reset enable/disable control
 0: Disable
 1: Enable
 This bit is used to reset the buffer write address. If this bit is set high, the buffer write address is reset to 00H. After the reset is completed, the BUFRST bit will be automatically cleared by the hardware.

ERRCHK Opeartor Registers – ERRCHKC, ERRCHKR

There are two registers related to the ERRCHK operator control, ERRCHKC and ERRCHKR. The ERRCHKC register is used to define the ERRCHK operator enable/disable control and ERRCHK operation result software reset enable/disable control. The ERRCHKR register is used to store the ERRCHK operation result.

• ERRCHKC Register

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

Bit 7 **ERRCHKEN**: ERRCHK operator enable/disable control
 0: Disable
 1: Enable
 If this bit is set high, a bitwise XOR operation will be executed on each data written into the buffer. The result is stored in the ERRCHKR register.

Bit 6 **ERRCHKRST**: ERRCHK operation result software reset enable/disable control
 0: Disable
 1: Enable
 This bit is used to clear the ERRCHKR register value. After clearing, this bit will be automatically cleared.

Bit 5~0 Unimplemented, read as “0”

• ERRCHKR Register

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

Bit 7~0 **D7~D0**: ERRCHK operation result register bit 7 ~ bit 0

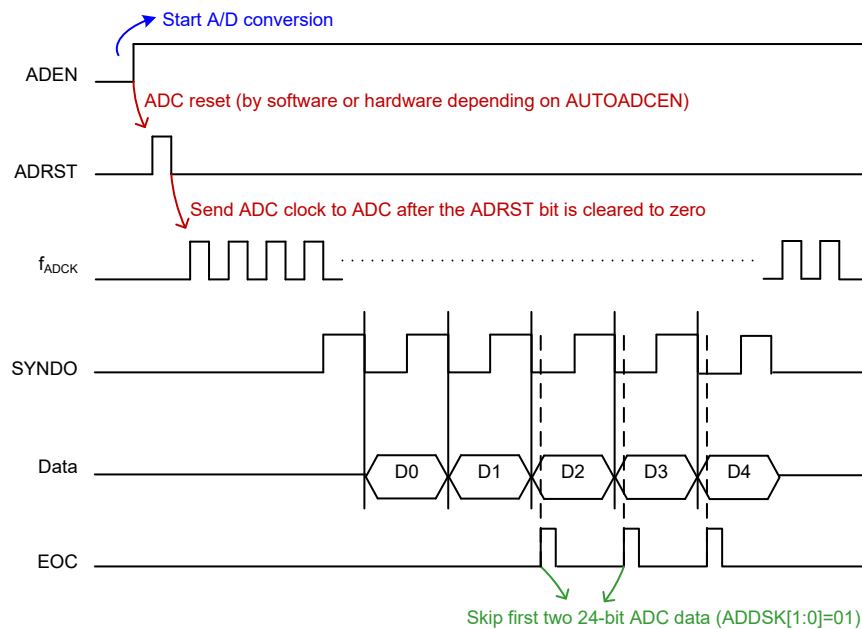
CTRL Unit

The CTRL unit is used to keep the MCU in the IDLE mode during the A/D conversion and optimise the A/D converter timing. It must be ensured that the CTRLLEN and ADEN bits have been set high before the MCU executing a HALT instruction to enter the IDLE mode.

If the AUTOADCEN bit is cleared to zero to select the normal mode, after the microcontroller changes the ADRST bit from low to high and then low again, the A/D converter clock f_{ADCK} will be sent to the A/D converter. If the AUTOADCEN bit is set high to select the auto mode, the PGAEN and ADEN bits will be set high by hardware to execute the A/D conversion. If it has been confirmed that the ADEN bit in the MODC register has been set high, an ADRST signal will be sent to the A/D converter by hardware, then the A/D converter clock f_{ADCK} will be sent to the A/D converter.

After the analog to digital conversion process is complete, the hardware will decide how many data to skip according to the ADDSK[1:0] bit setting. If the ADDSK[1:0] bits are set to 01, skip the first two SYND0 signals, the EOC bit will be set high after the third SYND0 signal is generated. Then

the converted data can be read from the ADRL, ADRM and ADRH registers. If the A/D converter auto mode is enabled, the value stored in the ADRL, ADRM and ADRH registers will be written into the buffer automatically.



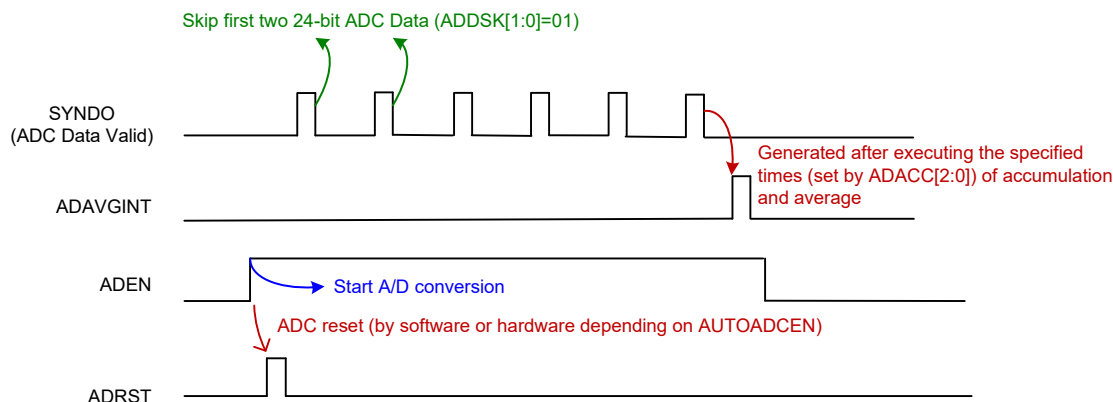
A/D Converter CTRL Unit Timing Chart – ADDSK[1:0]=01

A/D Data Accumulation and Average

The accumulation and average function is used to accumulate and average the 24-bit A/D converter output, which is implemented by hardware not by program. At this time, the MCU can execute a HALT instruction to reduce the average current consumption. It must be ensured that the ADEN bit has been set high before the MCU executing a HALT instruction to enter the IDLE mode.

The number of accumulation and average times is set by the ADACC[2:0] bits. An A/D converter average interrupt will be generated after the accumulation and average is completed. The averaged data can be read from the ADRL_AVG, ADRM_AVG and ADRH_AVG registers. Before the interrupt is generated, the data stored in the ADRL_AVG, ADRM_AVG and ADRH_AVG registers are invalid. If the A/D converter auto mode is enabled, the value stored in the ADRL_AVG, ADRM_AVG and ADRH_AVG registers will be written into the buffer automatically.

Set the ADAVGEN bit high to enable the accumulation and average function. When the ADEN bit is high, the accumulation and average function is started. If the ADEN bit is cleared to zero when the accumulation and average operation has not completed, this indicates that the accumulation and average function is forced to stop and the corresponding interrupt will not be generated.



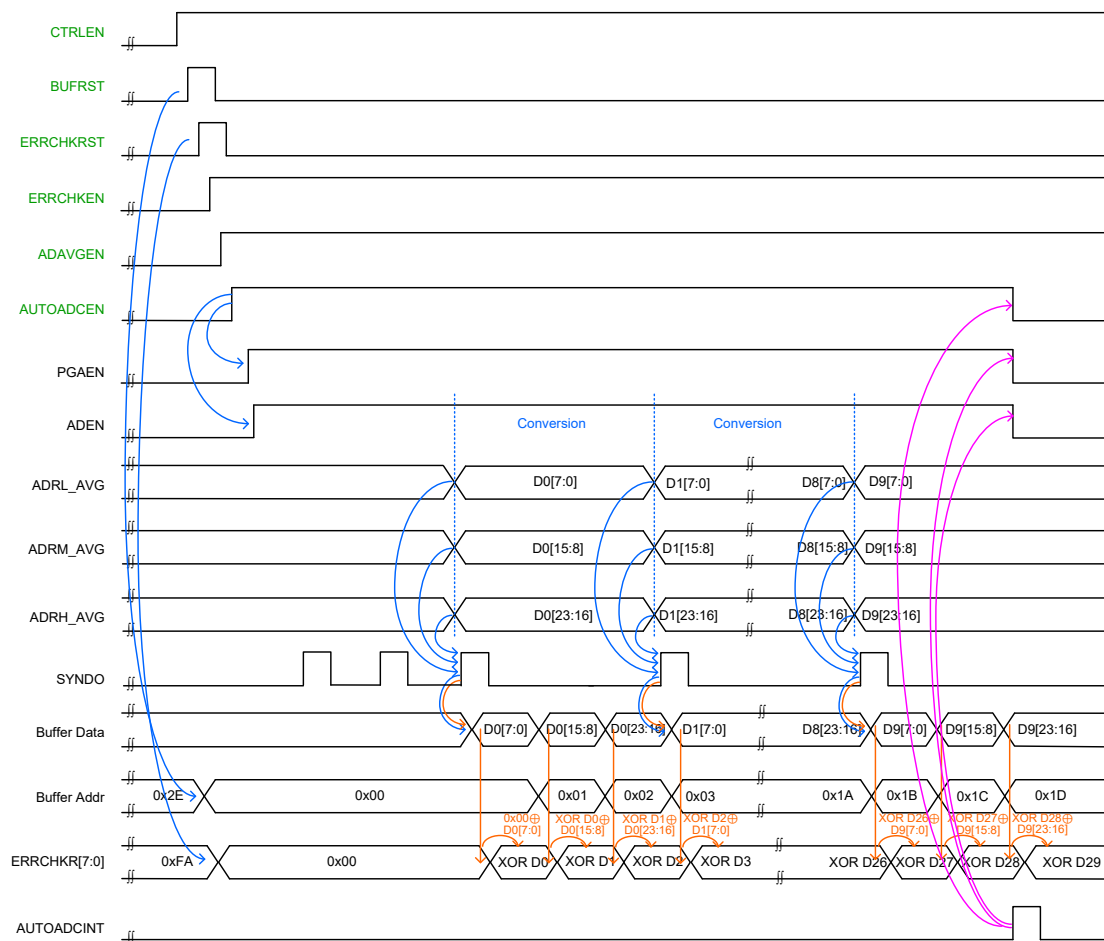
A/D Data Accumulation and Average Timing Chart – ADACC[2:0]=000, ADAVGEN=1

A/D Converter Auto Mode

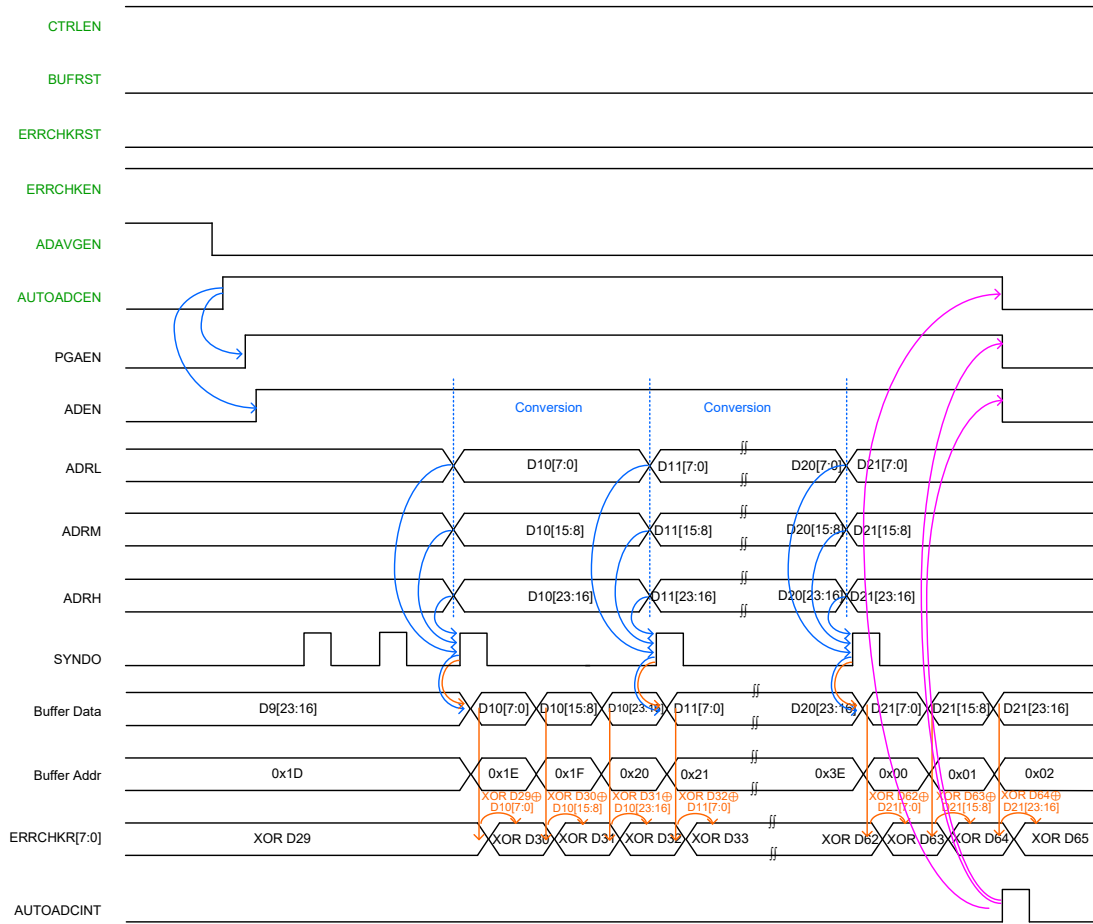
The A/D converter auto mode is used to write the converted data to the buffer automatically when the MCU is in the IDLE mode. When the number of data to be written into the buffer reaches the value defined by WRNUM[4:0] bits, an A/D converter auto mode interrupt will be generated to wake up the MCU. It must be ensured that the PGAEN and ADEN bits have been cleared to zero and the AUTOADCEN bit has been set high before the MCU executing a HALT instruction to enter the IDLE mode.

Set the AUTOADCEN bit from low to high to enable the A/D converter auto mode, the PGAEN and ADEN bits will be set high by hardware to start the A/D conversion. The auto control circuit will write the data stored in the ADRL, ADRM and ADRH registers or ADRL_AVG, ADRM_AVG and ADRH_AVG registers into the buffer, depending on the ADVGEN bit setting. The buffer is located from 00H to 3FH in the RAM Sector 3. Once the device enters the A/D converter auto mode, the CPU cannot read/write any data from/to this RAM area. When the number of data in the buffer reaches the value defined by WRNUM[4:0] bits, the PGAEN, ADEN and AUTOADCEN bits will be cleared to zero by hardware and an A/D converter auto mode interrupt will be generated to wake up the MCU and to disable the A/D converter auto mode.

The RAM address is reset by software. If the address is not reset, when the current address is 3EH, it will continue to write from 00H to overwrite the previous data. During the A/D converter auto mode, if the ERRCHKEN bit is high, a bitwise XOR operation will be executed on each data written into the buffer. The result can be read from the ERRCHKR register.



A/D Converter Auto Mode Timing Chart – ADDSK[1:0]=01, ADAVGEN=1, WRNUM[4:0]=01001



A/D Converter Auto Mode Timing Chart – ADDSK[1:0]=01, ADAVGEN=0, WRNUM[4:0]=01011

A/D Converter Data Rate Definition

The Delta Sigma A/D converter data rate can be calculated using the following equation:

Data Rate for SINC2

$$\begin{aligned}
 &= f_{ADCK} / (2 \times OSR) \\
 &= (f_{MCLK} / N) / (2 \times OSR) \\
 &= f_{MCLK} / (N \times 2 \times OSR)
 \end{aligned}$$

Data Rate for SINC3

$$\begin{aligned}
 &= f_{ADCK} / OSR \\
 &= (f_{MCLK} / N) / OSR \\
 &= f_{MCLK} / (N \times OSR)
 \end{aligned}$$

f_{ADCK} : A/D converter clock frequency, derived from f_{MCLK}/N .

f_{MCLK} : A/D converter clock source, derived from f_H or $f_H/2/(ADCK[4:0]+1)$ determined by the ADCK[4:0] bits.

N: A constant divide factor equal to 12 or 30 determined by the FLMS[2:0] bits.

OSR: Oversampling rate determined by the ADOR[3:0] bits.

For example, if a data rate of 8Hz is desired, an f_{CLK} clock source with a frequency of 4MHz can be selected. Then set the FLMS[2:0] bits to “000” to obtain an “N” equal to 30.

For the SINC2 filter, set the ADOR[3:0] bits to “0010” to select an oversampling rate equal to 8192. Therefore, the Data Rate= $4\text{MHz}/(30 \times 2 \times 8192)=8\text{Hz}$.

For the SINC3 filter, set the ADOR[3:0] bits to “0001” to select an oversampling rate equal to 16384. Therefore, the Data Rate= $4\text{MHz}/(30 \times 16384)=8\text{Hz}$.

A/D Converter Operation

To enable the A/D Converter, the first step is to disable the A/D converter power down by setting the ADEN bit in the MODC register from low to high to make sure that the A/D Converter is powered up. The ADRST bit in the MODC register is used to start and reset the A/D converter after power-on. When the microcontroller changes the ADRST bit from low to high and then low again, an analog to digital conversion cycle in the SINC filter will be initiated. After this setup is completed, the A/D Converter is ready for operation. These two bits are used to control the overall start operation of the internal analog to digital converter.

The EOC bit in the ADCR1 register is used to indicate when the analog to digital conversion process is complete. This bit will be automatically set high by the microcontroller after a conversion cycle has ended. In addition, the corresponding A/D converter interrupt request flag will be set in the interrupt control register, and if the A/D converter and global interrupts are enabled, an appropriate internal interrupt signal will be generated. This A/D converter internal interrupt signal will direct the program flow to the associated A/D converter internal interrupt address for processing. If the A/D converter internal interrupt is disabled, the microcontroller can poll the EOC bit in the ADCR1 register to check whether it has been set to “1” as an alternative method of detecting the end of an A/D conversion cycle. The A/D converted data will be updated continuously by the newly converted data. If the A/D converted data latch function is enabled, the latest converted data will be latched and the following newly converted data will be discarded until this data latch function is disabled.

The clock source for the A/D converter should be typically fixed at a value of 4MHz, which originates from the system clock f_H , and can be chosen to be either f_H or a subdivided version of f_H . The division ratio value is determined by the ADCK4~ADCK0 bits in the ADCS register to obtain a 4MHz clock source for the A/D Converter.

The differential reference voltage supply to the A/D Converter can be supplied from an internal reference source.

Summary of A/D Conversion Steps

The following summarises the individual steps that should be executed in order to implement a single A/D conversion process in normal mode.

- Step 1
Select the PGA, A/D converter gains and PGA input connection by the PGAC register.
- Step 2
Select the $V_{DACVREF}$ voltage by the IREFC register.
- Step 3
Select the required A/D conversion clock source by correctly programming bits ADCK4~ADCK0 in the ADCS register.
- Step 4
Select output data rate by configuring the ADOR3~ADOR0 bits in the ADCR0 register and FLMS2~FLMS0 bits in the ADCR1 register.

- Step 5
Select which channel is to be connected to the internal PGA by correctly programming the CHSP3~CHSP0 and CHSN3~CHSN0 bits which are contained in the PGACS register.
- Step 6
Enable the PGA and A/D converter by setting the PGAEN bit and ADEN bit in the MODC register.
- Step 7
Reset the A/D converter by setting the ADRST in the MODC register to high, and then release the reset status by clearing this bit to zero.
- Step 8
To check when the analog to digital conversion process is completed, the EOC bit in the ADCR1 register can be polled. The conversion process is completed when this bit goes high. When this occurs the A/D converter data registers ADRL, ADRM and ADRH can be read to obtain the conversion value.

Programming Considerations

During microcontroller operations where the A/D converter is not being used, the A/D internal circuitry can be switched off to reduce power consumption. When this happens, the internal A/D converter circuits will not consume power irrespective of what analog voltage is applied to their input lines.

A/D Converter Transfer Function

The device contains a 24-bit Delta Sigma A/D converter and its full-scale converted digitized value is from 8388607 to -8388608 in decimal value. The converted data format is formed using a two's complement binary value. The MSB of the converted data is the signed bit. Since the full-scale analog input value is equal to the value of the differential reference input voltage, ΔVR_{\pm} , which is equal to the $V_{DACVREF}$ voltage, this gives a single bit analog input value of ΔVR_{\pm} divided by 8388608.

$$1 \text{ LSB} = \Delta VR_{\pm} / 8388608$$

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

$$\Delta SI = PGAGN \times ADGN \times \Delta DI$$

$$ADC_Conversion_Data = (\Delta SI / \Delta VR_{\pm}) \times K$$

Where K is equal to 2^{23}

Note: 1. The PGAGN and ADGN values are decided by the PGS[2:0], AGS control bits.

2. ΔSI : Differential Input Signal after amplification
3. PGAGN: Programmable Gain Amplifier gain
4. ADGN: A/D Converter gain
5. ΔDI : Differential input signal derived from external channels or internal signals
6. ΔVR_{\pm} : Differential Reference voltage

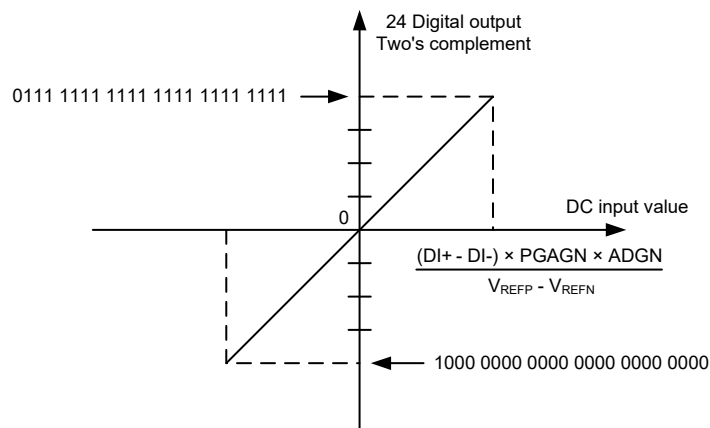
Due to the digital system design of the Delta Sigma A/D Converter, the maximum A/D converted value is 8388607 and the minimum value is -8388608. Therefore, there is a middle value of 0. The ADC_Conversion_Data equation illustrates this range of converted data variation.

| A/D Conversion Data (2's complement, Hexadecimal) | Decimal Value |
|--|---------------|
| 0x7FFFFFFF | 8388607 |
| 0x800000 | -8388608 |

A/D Conversion Data Range

The above A/D conversion data table illustrates the range of A/D conversion data.

The following diagram shows the relationship between the DC input value and the A/D converted data which is presented using Two's Complement.



A/D Converted Data

The A/D converted data is related to the input voltage and the PGA selections. The format of the A/D Converter output is a two's complement binary code. The length of this output code is 24 bits and the MSB is a signed bit. When the MSB is "0", this represents a "positive" input. If the MSB is "1", this represents a "negative" input. The maximum value is 8388607 and the minimum value is -8388608. If the input signal is greater than the maximum value, the converted data is limited to 8388607, and if the input signal is less than the minimum value, the converted data is limited to -8388608.

A/D Converted Data to Voltage

The converted data can be recovered using the following equations:

If MSB=0 – Positive Converted data

$$\text{Input Voltage} = \frac{(\text{Converted_data}) \times \text{LSB}}{\text{PGAGN} \times \text{ADGN}}$$

If the MSB=1 – Negative Converted data

$$\text{Input voltage} = \frac{(\text{Two's_complement_of_Converted_data}) \times \text{LSB}}{\text{PGAGN} \times \text{ADGN}}$$

Note: Two's complement=One's complement+1

Serial Peripheral Interface – SPI

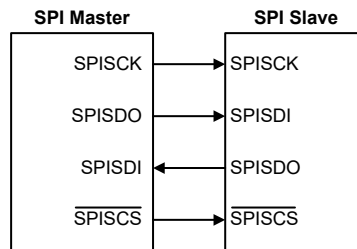
This SPI interface is often used to communicate with external peripheral devices such as sensors, Flash or EEPROM memory devices, etc. Originally developed by Motorola, the four line SPI interface is a synchronous serial data interface that has a relatively simple communication protocol simplifying the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

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

SPI Interface Operation

The SPI interface is a full duplex synchronous serial data link. It is a four line interface with pin names SPISDI, SPISDO, SPISCK and $\overline{\text{SPISCS}}$. Pins SPISDI and SPISDO are the Serial Data Input and Serial Data Output lines, SPISCK is the Serial Clock line and $\overline{\text{SPISCS}}$ is the Slave Select line. As the SPI interface pins are pin-shared with other functions, the SPI interface pins must first be selected by configuring the corresponding selection bits in the pin-shared function selection registers. The SPI interface function is disabled or enabled using the SPIEN bit in the SPIC0 register. Communication between devices connected to the SPI interface is carried out in a slave/master mode with all data transfer initiations being implemented by the master. The master also controls the clock/signal. As the device only contains a single $\overline{\text{SPISCS}}$ pin only one slave device can be utilized. The pull-high resistors of the SPI pin-shared I/O are selected using pull-high control registers when the SPI function is enabled and the corresponding pins are used as SPI input pins.

The $\overline{\text{SPISCS}}$ pin is controlled by software, set SPICSEN bit to 1 to enable the $\overline{\text{SPISCS}}$ pin function, and clear SPICSEN bit to 0, the $\overline{\text{SPISCS}}$ pin will be floating state.

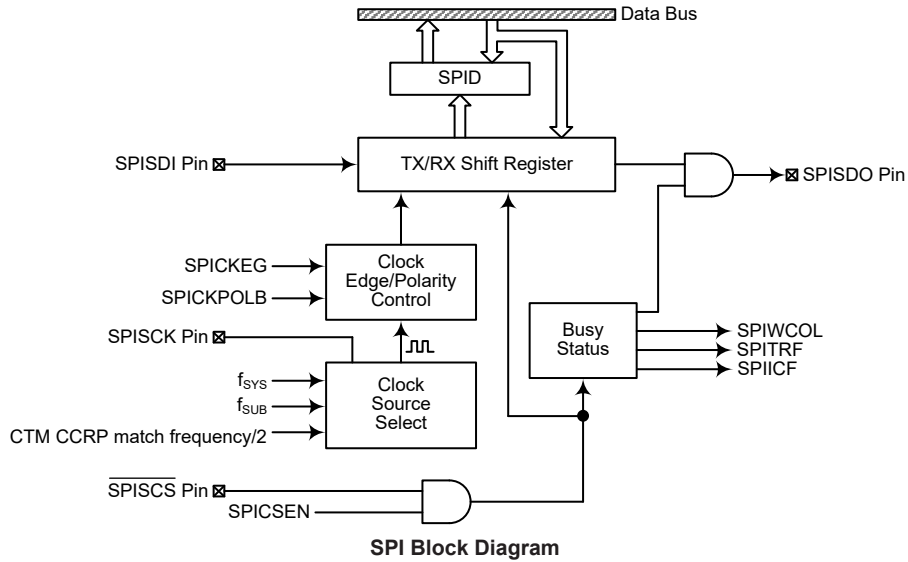


SPI Master/Slave Connection

The SPI Serial Interface function includes the following features:

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

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



SPI Registers

There are three internal registers which control the overall operation of the SPI interface. These are the SPID data register and two registers SPIC0 and SPIC1.

| Register Name | Bit | | | | | | | |
|---------------|-------|-------|-----------|---------|--------|---------|---------|--------|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SPIC0 | SPIM2 | SPIM1 | SPIM0 | — | — | — | SPIEN | SPIICF |
| SPIC1 | — | — | SPICKPOLB | SPICKEG | SPIMLS | SPICSEN | SPIWCOL | SPITRF |
| SPID | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

SPI Register List

SPI Data Register

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

• SPID Register

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

"x": unknown

Bit 7~0 **D7~D0**: SPI data register bit 7 ~ bit 0

SPI Control Registers

There are also two control registers for the SPI interface, SPIC0 and SPIC1. Register SPIC0 is used to control the enable/disable function and to set the data transmission clock frequency. Register SPIC1 is used for other control functions such as LSB/MSB selection, write collision flag, etc.

• **SPIC0 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|-------|-------|---|---|---|-------|--------|
| Name | SPIM2 | SPIM1 | SPIM0 | — | — | — | SPIEN | SPIICF |
| R/W | R/W | R/W | R/W | — | — | — | R/W | R/W |
| POR | 1 | 1 | 1 | — | — | — | 0 | 0 |

Bit 7~5 **SPIM2~SPIM0**: SPI Master/Slave clock select
 000: SPI master mode with clock $f_{SYS}/4$
 001: SPI master mode with clock $f_{SYS}/16$
 010: SPI master mode with clock $f_{SYS}/64$
 011: SPI master mode with clock f_{SUB}
 100: SPI master mode with clock CTM CCRP match frequency/2
 101: SPI slave mode
 11x: SPI disable

These bits are used to control the SPI Master/Slave selection and the SPI Master clock frequency. The SPI clock is a function of the system clock but can also be chosen to be sourced from CTM and f_{SUB} . If the SPI Slave Mode is selected then the clock will be supplied by an external Master device.

Bit 4~2 Unimplemented, read as “0”

Bit 1 **SPIEN**: SPI Enable Control

0: Disable
 1: Enable

The bit is the overall on/off control for the SPI interface. When the SPIEN bit is cleared to zero to disable the SPI interface, the SPISDI, SPISDO, SPISCK and SPISCS lines will lose the SPI function and the SPI operating current will be reduced to a minimum value. When the bit is high the SPI interface is enabled.

Bit 0 **SPIICF**: SPI Incomplete Flag

0: SPI incomplete condition not occurred
 1: SPI incomplete condition occurred

This bit is only available when the SPI is configured to operate in an SPI slave mode. If the SPI operates in the slave mode with the SPIEN and SPICSEN bits both being set to 1 but the SPISCS line is pulled high by the external master device before the SPI data transfer is completely finished, the SPIICF bit will be set to 1 together with the SPITRF bit. When this condition occurs, the corresponding interrupt will occur if the interrupt function is enabled. However, the SPITRF bit will not be set to 1 if the SPIICF bit is set to 1 by software application program.

• **SPIC1 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|-----------|---------|--------|---------|---------|--------|
| Name | — | — | SPICKPOLB | SPICKEG | SPIMLS | SPICSEN | SPIWCOL | SPITRF |
| R/W | — | — | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | — | — | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7~6 Unimplemented, read as “0”

Bit 5 **SPICKPOLB**: SPI clock line base condition selection

0: The SPISCK line will be high when the clock is inactive.
 1: The SPISCK line will be low when the clock is inactive.

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

Bit 4 **SPICKEG**: SPI SPISCK clock active edge type selection

SPICKPOLB=0

0: SPISCK is high base level and data capture at SPISCK rising edge
 1: SPISCK is high base level and data capture at SPISCK falling edge

SPICKPOLB=1

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

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

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

Bit 3 **SPIMLS**: SPI data shift order

0: LSB first

1: MSB first

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

Bit 2 **SPICSEN**: SPI $\overline{\text{SPISCS}}$ pin control

0: Disable

1: Enable

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

Bit 1 **SPIWCOL**: SPI write collision flag

0: No collision

1: Collision

The SPIWCOL flag is used to detect whether a data collision has occurred or not. If this bit is high, it means that data has been attempted to be written to the SPID register during a data transfer operation. This writing operation will be ignored if data is being transferred. This bit can be cleared to zero by the application program.

Bit 0 **SPITRF**: SPI Transmit/Receive complete flag

0: SPI data is being transferred

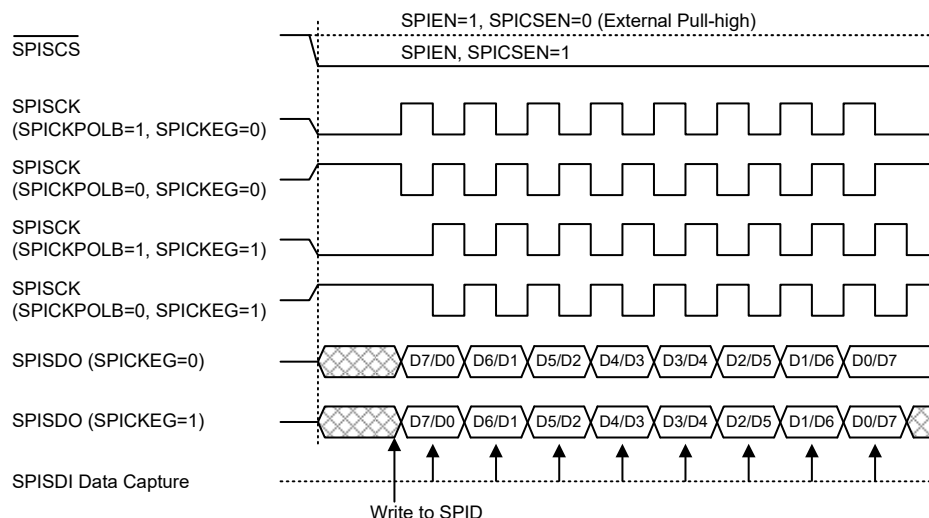
1: SPI data transfer is completed

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

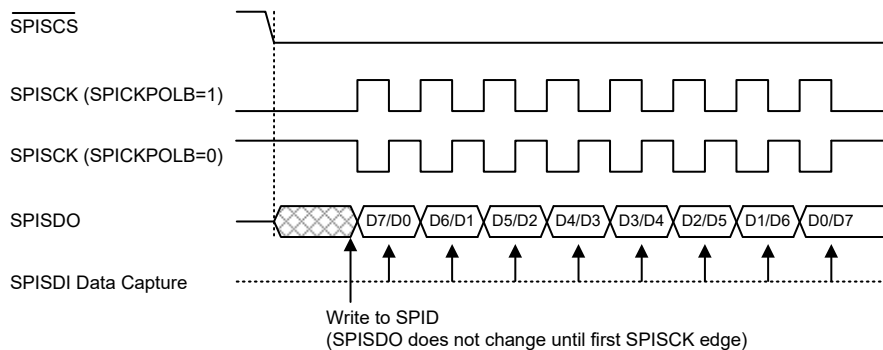
SPI Communication

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

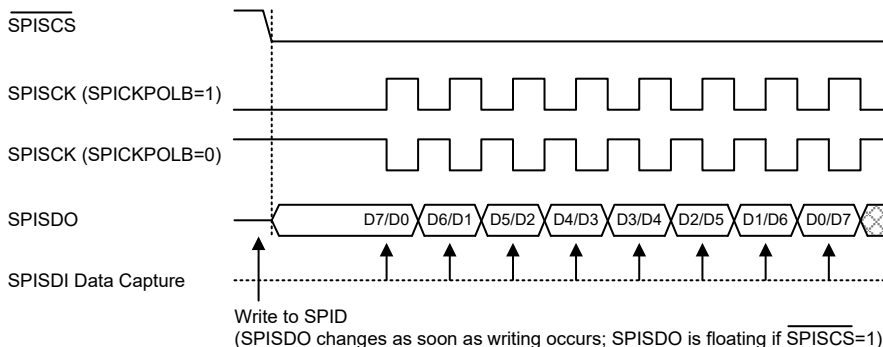
The SPI Master mode will continue to function if the SPI clock is running.



SPI Master Mode Timing

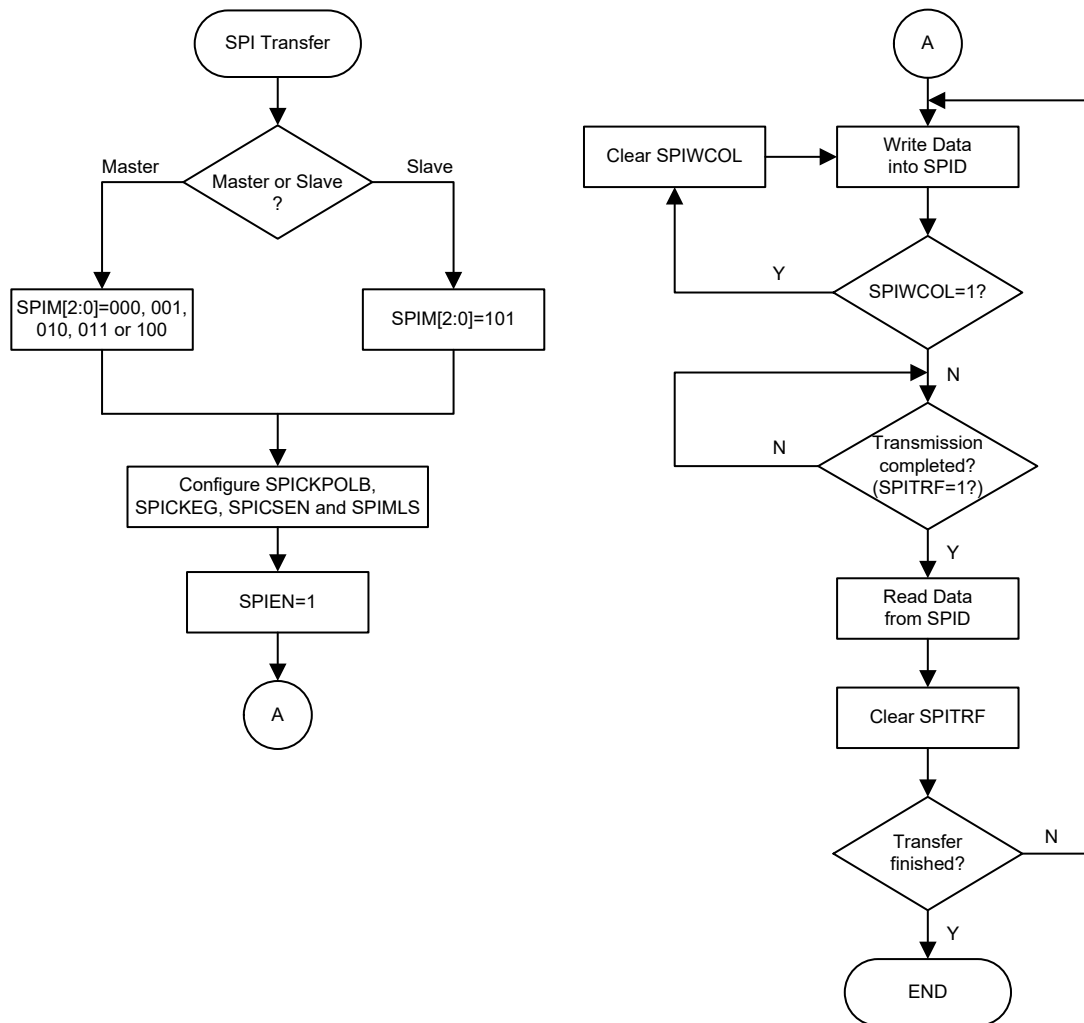


SPI Slave Mode Timing – SPICKEG=0



Note: For SPI slave mode, if SPIEN=1 and SPICSEN=0, SPI is always enabled and ignores the SPISCS level.

SPI Slave Mode Timing – SPICKEG=1



SPI Transfer Control Flow Chart

SPI Bus Enable/Disable

To enable the SPI bus, set $\overline{\text{SPICSEN}}=1$ and $\overline{\text{SPISCS}}=0$, then wait for data to be written into the SPID (TXRX buffer) register. For the Master Mode, after data has been written to the SPID (TXRX buffer) register, then transmission or reception will start automatically. When all the data has been transferred the SPITRF bit should be set. For the Slave Mode, when clock pulses are received on SPISCK, data in the TXRX buffer will be shifted out or data on SPISDI will be shifted in.

When the SPI bus is disabled, the SPISCK, SPISDI, SPISDO and $\overline{\text{SPISCS}}$ pins can become I/O pins or other pin-shared functions using the corresponding pin-shared function selection bits.

SPI Operation

All communication is carried out using the 4-line interface for either Master or Slave Mode.

The SPICSEN bit in the SPIC1 register controls the overall function of the SPI interface. Setting this bit high will enable the SPI interface by allowing the $\overline{\text{SPISCS}}$ line to be active, which can then be used to control the SPI interface. If the SPICSEN bit is low, the SPI interface will be disabled and the $\overline{\text{SPISCS}}$ line will be in a floating condition and can therefore not be used for control of the SPI interface. If the SPICSEN bit and the SPIEN bit in the SPIC0 register are set high, this will place the

SPISDI line in a floating condition and the SPISDO line high. If in Master Mode the SPISCK line will be either high or low depending upon the clock polarity selection bit SPICKPOLB in the SPIC1 register. If in Slave Mode the SPISCK line will be in a floating condition. If SPIEN is low, then the bus will be disabled and $\overline{\text{SPISCS}}$, SPISDI, SPISDO and SPISCK pins will all become I/O pins or other pin-shared functions using the corresponding pin-shared function selection bits. In the Master Mode the Master will always generate the clock signal. The clock and data transmission will be initiated after data has been written into the SPID register. In the Slave Mode, the clock signal will be received from an external master device for both data transmission and reception. The following sequences show the order to be followed for data transfer in both Master and Slave Mode.

Master Mode:

- Step 1
Select the clock source and Master mode using the SPIM2~ SPIM0 bits in the SPIC0 control register.
- Step 2
Setup the SPICSEN bit and setup the SPIMLS bit to choose if the data is MSB or LSB shifted first, this must be same as the Slave device.
- Step 3
Setup the SPIEN bit in the SPIC0 control register to enable the SPI interface.
- Step 4
For write operations: write the data to the SPID register, which will actually place the data into the TXRX buffer. Then use the SPISCK and SPISDO lines to output the data. After this go to step 5.
For read operations: the data transferred in on the SPISDI line will be stored in the TXRX buffer until all the data has been received at which point it will be latched into the SPID register.
- Step 5
Check the SPIWCOL bit if set high then a collision error has occurred so return to step 4. If equal to zero then go to the following step.
- Step 6
Check the SPITRF bit or wait for a SPI serial bus interrupt.
- Step 7
Read data from the SPID register.
- Step 8
Clear SPITRF.
- Step 9
Go to step 4.

Slave Mode:

- Step 1
Select the SPI Slave mode using the SPIM2~ SPIM0 bits in the SPIC0 control register
- Step 2
Setup the SPICSEN bit and setup the SPIMLS bit to choose if the data is MSB or LSB shifted first, this setting must be the same with the Master device.
- Step 3
Setup the SPIEN bit in the SPIC0 control register to enable the SPI interface.

- Step 4

For write operations: write the data to the SPID register, which will actually place the data into the TXRX buffer. Then wait for the master clock SPISCK and $\overline{\text{SPISCS}}$ signal. After this, go to step 5.

For read operations: the data transferred in on the SPISDI line will be stored in the TXRX buffer until all the data has been received at which point it will be latched into the SPID register.

- Step 5

Check the SPIWCOL bit if set high then a collision error has occurred so return to step 4. If equal to zero then go to the following step.

- Step 6

Check the SPITRF bit or wait for a SPI serial bus interrupt.

- Step 7

Read data from the SPID register.

- Step 8

Clear SPITRF.

- Step 9

Go to step 4.

Error Detection

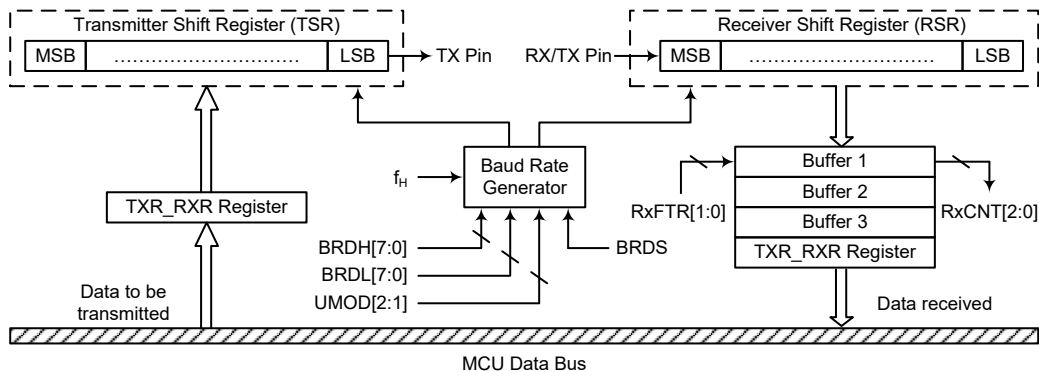
The SPIWCOL bit in the SPIC1 register is provided to indicate errors during data transfer. The bit is set by the SPI serial Interface but must be cleared by the application program. This bit indicates a data collision has occurred which happens if a write to the SPID register takes place during a data transfer operation and will prevent the write operation from continuing.

UART Interface

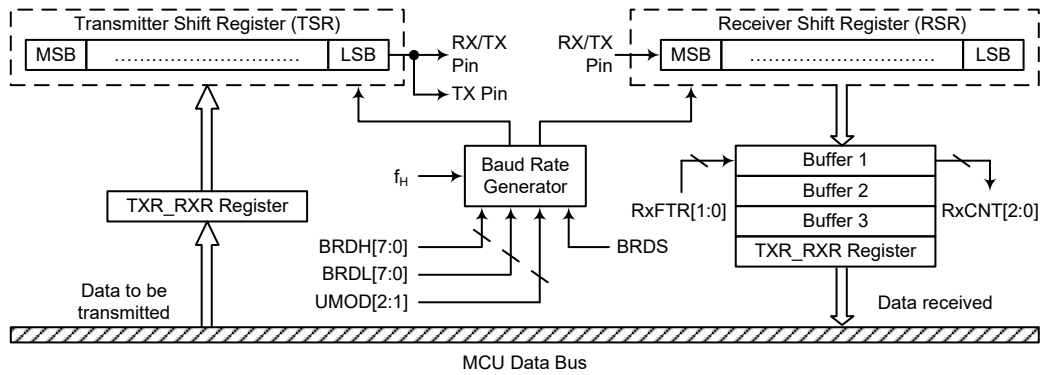
These devices contain an integrated full-duplex or half-duplex asynchronous serial communications UART interface that enables communication with external devices that contain a serial interface. The UART function has many features and can transmit and receive data serially by transferring a frame of data with eight or nine data bits per transmission as well as being able to detect errors when the data is overwritten or incorrectly framed. The UART function possesses its own internal interrupt which can be used to indicate when a reception occurs or when a transmission terminates.

The integrated UART function contains the following features:

- Full-duplex or half-duplex (single wire mode) asynchronous communication
- 8 or 9 bits character length
- Even, odd, mark, space or no parity options
- One or two stop bits configurable for receiver
- Two stop bits for transmitter
- Baud rate generator with 16-bit prescaler
- Parity, framing, noise and overrun error detection
- Support for interrupt on address detect (last character bit=1)
- Separately enabled transmitter and receiver
- 4-byte Deep FIFO Receive Data Buffer
- 1-byte Deep FIFO Transmit Data Buffer
- RX/TX pin wake-up function
- Transmit and receive interrupts
- Interrupts can be triggered by the following conditions:
 - ♦ Transmitter Empty
 - ♦ Transmitter Idle
 - ♦ Receiver reaching FIFO trigger level
 - ♦ Receiver Overrun
 - ♦ Address Mode Detect



UART Data Transfer Block Diagram – SWM=0



UART Data Transfer Block Diagram – SWM=1

UART External Pins

To communicate with an external serial interface, the internal UART has two external pins known as TX and RX/TX, which are pin-shared with I/O or other pin functions. The TX and RX/TX pin function should first be selected by the corresponding pin-shared function selection register before the UART function is used. Along with the UARTEN bit, the TXEN and RXEN bits, if set, will setup these pins to transmitter output and receiver input conditions. At this time the internal pull-high resistor related to the transmitter output pin will be disabled, while the internal pull-high resistor related to the receiver input pin is controlled by the corresponding I/O pull-high function control bit. When the TX or RX/TX pin function is disabled by clearing the UARTEN, TXEN or RXEN bit, the TX or RX/TX pin will be set to a floating state. At this time whether the internal pull-high resistor is connected to the TX or RX/TX pin or not is determined by the corresponding I/O pull-high function control bit.

UART Single Wire Mode

The UART function also supports the Single Wire Mode communication which is selected using the SWM bit in the UCR3 register. When the SWM bit is set high, the UART function will be in the single wire mode. In the single wire mode, a single RX/TX pin can be used to transmit and receive data depending upon the corresponding control bits. When the RXEN bit is set high, the RX/TX pin is used as a receiver pin. When the RXEN bit is cleared to zero and the TXEN bit is set high, the RX/TX pin will act as a transmitter pin.

It is recommended not to set both the RXEN and TXEN bits high in the single wire mode. If both the RXEN and TXEN bits are set high, the RXEN bit will have the priority and the UART will act as a receiver.

It is important to note that the functional description in this UART Interface chapter, which is described from the full-duplex communication standpoint, also applies to the half-duplex (single wire mode) communication except the pin usage. In the single wire mode, the TX pin mentioned in this chapter should be replaced by the RX/TX pin to understand the whole UART single wire mode function.

In the single wire mode, the data can also be transmitted on the TX pin in a transmission operation with proper software configurations. Therefore, the data will be output on the RX/TX and TX pins.

UART Data Transfer Scheme

The UART Data Transfer Block Diagram shows the overall data transfer structure arrangement for the UART interface. The actual data to be transmitted from the MCU is first transferred to the TXR_RXR register by the application program. The data will then be transferred to the Transmit Shift Register

from where it will be shifted out, LSB first, onto the TX pin at a rate controlled by the Baud Rate Generator. Only the TXR_RXR register is mapped onto the MCU Data Memory, the Transmit Shift Register is not mapped and is therefore inaccessible to the application program.

Data to be received by the UART is accepted on the external RX/TX pin, from where it is shifted in, LSB first, to the Receiver Shift Register at a rate controlled by the Baud Rate Generator. When the shift register is full, the data will then be transferred from the shift register to the internal TXR_RXR register, where it is buffered and can be manipulated by the application program. Only the TXR_RXR register is mapped onto the MCU Data Memory, the Receiver Shift Register is not mapped and is therefore inaccessible to the application program.

It should be noted that the actual register for data transmission and reception only exists as a single shared register in the Data Memory. This shared register known as the TXR_RXR register is used for both data transmission and data reception.

UART Status and Control Registers

There are nine control registers associated with the UART function. The SWM bit in the UCR3 register is used to enable/disable the UART Single Wire Mode. The USR, UCR1, UCR2, UFCR and RxCNT registers control the overall function of the UART, while the BRDH and BRDL registers control the Baud rate. The actual data to be transmitted and received on the serial interface is managed through the TXR_RXR data register.

| Register Name | Bit | | | | | | | |
|---------------|--------|-------|-------|-------|-------|-------|--------|--------|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| USR | PERR | NF | FERR | OERR | RIDLE | RXIF | TIDLE | TXIF |
| UCR1 | UARTEN | BNO | PREN | PRT1 | PRT0 | TXBRK | RX8 | TX8 |
| UCR2 | TXEN | RXEN | STOPS | ADDEN | WAKE | RIE | TIIE | TEIE |
| UCR3 | — | — | — | — | — | — | — | SWM |
| TXR_RXR | TXRX7 | TXRX6 | TXRX5 | TXRX4 | TXRX3 | TXRX2 | TXRX1 | TXRX0 |
| BRDH | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| BRDL | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| UFCR | — | — | UMOD2 | UMOD1 | UMOD0 | BRDS | RxFTR1 | RxFTR0 |
| RxCNT | — | — | — | — | — | D2 | D1 | D0 |

UART Register List

• USR Register

The USR register is the status register for the UART, which can be read by the program to determine the present status of the UART. All flags within the USR register are read only. Further explanation on each of the flags is given below:

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|------|----|------|------|-------|------|-------|------|
| Name | PERR | NF | FERR | OERR | RIDLE | RXIF | TIDLE | TXIF |
| R/W | R | R | R | R | R | R | R | R |
| POR | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |

Bit 7

PERR: Parity error flag

0: No parity error is detected

1: Parity error is detected

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

| | |
|-------|---|
| Bit 6 | <p>NF: Noise flag</p> <p>0: No noise is detected</p> <p>1: Noise is detected</p> <p>The NF flag is the noise flag. When this read only flag is “0”, it indicates no noise condition. When the flag is “1”, it indicates that the UART has detected noise on the receiver input. The NF flag is set during the same cycle as the RXIF flag but will not be set in the case of an overrun. The NF flag can be cleared by a software sequence which will involve a read to the status register USR followed by an access to the TXR_RXR data register.</p> |
| Bit 5 | <p>FERR: Framing error flag</p> <p>0: No framing error is detected</p> <p>1: Framing error is detected</p> <p>The FERR flag is the framing error flag. When this read only flag is “0”, it indicates that there is no framing error. When the flag is “1”, it indicates that a framing error has been detected for the current character. The flag can also be cleared by a software sequence which will involve a read to the status register USR followed by an access to the TXR_RXR data register.</p> |
| Bit 4 | <p>OERR: Overrun error flag</p> <p>0: No overrun error is detected</p> <p>1: Overrun error is detected</p> <p>The OERR flag is the overrun error flag which indicates when the receiver buffer has overflowed. When this read only flag is “0”, it indicates that there is no overrun error. When the flag is “1”, it indicates that an overrun error occurs which will inhibit further transfers to the TXR_RXR receive data register. The flag is cleared by a software sequence, which is a read to the status register USR followed by an access to the TXR_RXR data register.</p> |
| Bit 3 | <p>RIDLE: Receiver status</p> <p>0: Data reception is in progress (Data being received)</p> <p>1: No data reception is in progress (Receiver is idle)</p> <p>The RIDLE flag is the receiver status flag. When this read only flag is “0”, it indicates that the receiver is between the initial detection of the start bit and the completion of the stop bit. When the flag is “1”, it indicates that the receiver is idle. Between the completion of the stop bit and the detection of the next start bit, the RIDLE bit is “1” indicating that the UART receiver is idle and the RX/TX pin stays in logic high condition.</p> |
| Bit 2 | <p>RXIF: Receive TXR_RXR data register status</p> <p>0: TXR_RXR data register is empty</p> <p>1: TXR_RXR data register has available data and reaches receiver FIFO trigger level</p> <p>The RXIF flag is the receive data register status flag. When this read only flag is “0”, it indicates that the TXR_RXR read data register is empty. When the flag is “1”, it indicates that the TXR_RXR read data register contains new data and reaches the Receiver FIFO trigger level. When the contents of the shift register are transferred to the TXR_RXR register and reach receiver FIFO trigger level, an interrupt is generated if RIE=1 in the UCR2 register. If one or more errors are detected in the received word, the appropriate receive-related flags NF, FERR, and/or PERR are set within the same clock cycle. The RXIF flag will eventually be cleared when the USR register is read with RXIF set, followed by a read from the TXR_RXR register, and if the TXR_RXR register has no more new data available.</p> |
| Bit 1 | <p>TIDLE: Transmission status</p> <p>0: Data transmission is in progress (Data being transmitted)</p> <p>1: No data transmission is in progress (Transmitter is idle)</p> <p>The TIDLE flag is known as the transmission complete flag. When this read only flag is “0”, it indicates that a transmission is in progress. This flag will be set high when the TXIF flag is “1” and when there is no transmit data or break character being transmitted. When TIDLE is equal to “1”, the TX pin becomes idle with the pin state in logic high condition. The TIDLE flag is cleared by reading the USR register with</p> |

TIDLE set and then writing to the TXR_RXR register. The flag is not generated when a data character or a break is queued and ready to be sent.

Bit 0

TXIF: Transmit TXR_RXR data register status

0: Character is not transferred to the transmit shift register

1: Character has transferred to the transmit shift register (TXR_RXR data register is empty)

The TXIF flag is the transmit data register empty flag. When this read only flag is “0”, it indicates that the character is not transferred to the transmitter shift register. When the flag is “1”, it indicates that the transmitter shift register has received a character from the TXR_RXR data register. The TXIF flag is cleared by reading the UART status register (USR) with TXIF set and then writing to the TXR_RXR data register. Note that when the TXEN bit is set, the TXIF flag bit will also be set since the transmit data register is not yet full.

• UCR1 Register

The UCR1 register together with the UCR2 and UCR3 registers are the three UART control registers that are used to set the various options for the UART function, such as overall on/off control, parity control, data transfer bit length, single wire mode communication etc. Further explanation on each of the bits is given below:

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|--------|-----|------|------|------|-------|-----|-----|
| Name | UARTEN | BNO | PREN | PRT1 | PRT0 | TXBRK | RX8 | TX8 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R | W |
| POR | 0 | 0 | 0 | 0 | 0 | 0 | x | 0 |

“x”: unknown

Bit 7

UARTEN: UART function enable control

0: Disable UART. TX and RX/TX pins are in the floating state

1: Enable UART. TX and RX/TX pins function as UART pins

The UARTEN bit is the UART enable bit. When this bit is equal to “0”, the UART will be disabled and the RX/TX pin as well as the TX pin will be in the floating state. When the bit is equal to “1”, the UART will be enabled and the TX and RX/TX pins will function as defined by SWM mode selection bit together with the TXEN and RXEN enable control bits.

When the UART is disabled, it will empty the buffer so any character remaining in the buffer will be discarded. In addition, the value of the baud rate counter will be reset. If the UART is disabled, all error and status flags will be reset. Also the TXEN, RXEN, TXBRK, RXIF, OERR, FERR, PERR and NF bits as well as the RxCNT register will be cleared, while the TIDLE, TXIF and RIDLE bits will be set. Other control bits in UCR1, UCR2, UCR3, UFCR, BRDH and BRDL registers will remain unaffected. If the UART is active and the UARTEN bit is cleared, all pending transmissions and receptions will be terminated and the module will be reset as defined above. When the UART is re-enabled, it will restart in the same configuration.

Bit 6

BNO: Number of data transfer bits selection

0: 8-bit data transfer

1: 9-bit data transfer

This bit is used to select the data length format, which can have a choice of either 8-bit or 9-bit format. When this bit is equal to “1”, a 9-bit data length format will be selected. If the bit is equal to “0”, then an 8-bit data length format will be selected. If 9-bit data length format is selected, then bits RX8 and TX8 will be used to store the 9th bit of the received and transmitted data respectively.

Note that the 9th bit of data if BNO=1, or the 8th bit of data if BNO=0, which is used as the parity bit, does not transfer to RX8 or TXRX7 respectively when the parity function is enabled.

- Bit 5 **PREN**: Parity function enable control
 0: Parity function is disabled
 1: Parity function is enabled
 This is the parity enable bit. When this bit is equal to “1”, the parity function will be enabled. If the bit is equal to “0”, then the parity function will be disabled.
- Bit 4~3 **PRT1~PRT0**: Parity type selection bits
 00: Even parity for parity generator
 01: Odd parity for parity generator
 10: Mark parity for parity generator
 11: Space parity for parity generator
 These bits are the parity type selection bits. When these bits are equal to 00b, even parity type will be selected. If these bits are equal to 01b, then odd parity type will be selected. If these bits are equal to 10b, then a 1 (Mark) in the parity bit location will be selected. If these bits are equal to 11b, then a 0 (Space) in the parity bit location will be selected.
- Bit 2 **TXBRK**: Transmit break character
 0: No break character is transmitted
 1: Break characters transmit
 The TXBRK bit is the Transmit Break Character bit. When this bit is “0”, there are no break characters and the TX pin operates normally. When the bit is “1”, there are transmit break characters and the transmitter will send logic zeros. When this bit is equal to “1”, after the buffered data has been transmitted, the transmitter output is held low for a minimum of a 13-bit length and until the TXBRK bit is reset.
- Bit 1 **RX8**: Receive data bit 8 for 9-bit data transfer format (read only)
 This bit is only used if 9-bit data transfers are used, in which case this bit location will store the 9th bit of the received data known as RX8. The BNO bit is used to determine whether data transfers are in 8-bit or 9-bit format.
- Bit 0 **TX8**: Transmit data bit 8 for 9-bit data transfer format (write only)
 This bit is only used if 9-bit data transfers are used, in which case this bit location will store the 9th bit of the transmitted data known as TX8. The BNO bit is used to determine whether data transfers are in 8-bit or 9-bit format.

• UCR2 Register

The UCR2 register is the second of the UART control registers and serves several purposes. One of its main functions is to control the basic enable/disable operation of the UART Transmitter and Receiver as well as enabling the various UART interrupt sources. The register also serves to control the receiver STOP bit number selection, receiver wake-up enable and the address detect enable. Further explanation on each of the bits is given below:

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|------|------|-------|-------|------|-----|------|------|
| Name | TXEN | RXEN | STOPS | ADDEN | WAKE | RIE | TIIE | TEIE |
| 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 **TXEN**: UART Transmitter enabled control
 0: UART transmitter is disabled
 1: UART transmitter is enabled
 The bit named TXEN is the Transmitter Enable Bit. When this bit is equal to “0”, the transmitter will be disabled with any pending data transmissions being aborted. In addition the buffers will be reset. In this situation the TX pin will be in a floating state. If the TXEN bit is equal to “1” and the UARTEN bit is also equal to “1”, the transmitter will be enabled and the TX pin will be controlled by the UART. Clearing the TXEN bit during a transmission will cause the data transmission to be aborted and will reset the transmitter. If this situation occurs, the TX pin will be in a floating state.

| | |
|-------|--|
| Bit 6 | <p>RXEN: UART Receiver enabled control</p> <p>0: UART receiver is disabled</p> <p>1: UART receiver is enabled</p> <p>The bit named RXEN is the Receiver Enable Bit. When this bit is equal to “0”, the receiver will be disabled with any pending data receptions being aborted. In addition, the receive buffers will be reset. In this situation the RX/TX pin will be in a floating state. If the RXEN bit is equal to “1” and the UARTEN bit is also equal to “1”, the receiver will be enabled and the RX/TX pin will be controlled by the UART. Clearing the RXEN bit during a reception will cause the data reception to be aborted and will reset the receiver. If this situation occurs, the RX/TX pin will be in a floating state.</p> |
| Bit 5 | <p>STOPS: Number of stop bits selection for receiver</p> <p>0: One stop bit format is used</p> <p>1: Two stop bits format is used</p> <p>This bit determines if one or two stop bits are to be used for receiver. When this bit is equal to “1”, two stop bits are used. If this bit is equal to “0”, then only one stop bit is used. Two stop bits are used for transmitter.</p> |
| Bit 4 | <p>ADDEN: Address detect function enable control</p> <p>0: Address detect function is disabled</p> <p>1: Address detect function is enabled</p> <p>The bit named ADDEN is the address detect function enable control bit. When this bit is equal to “1”, the address detect function is enabled. When it occurs, if the 8th bit, which corresponds to TXRX7 if BNO=0 or the 9th bit, which corresponds to RX8 if BNO=1, has a value of “1”, then the received word will be identified as an address, rather than data. If the corresponding interrupt is enabled, an interrupt request will be generated each time the received word has the address bit set, which is the 8th or 9th bit depending on the value of BNO. If the address bit known as the 8th or 9th bit of the received word is “0” with the address detect function being enabled, an interrupt will not be generated and the received data will be discarded.</p> |
| Bit 3 | <p>WAKE: RX/TX pin wake-up UART function enable control</p> <p>0: RX/TX pin wake-up UART function is disabled</p> <p>1: RX/TX pin wake-up UART function is enabled</p> <p>This bit is used to control the wake-up UART function when a falling edge on the RX/TX pin occurs. Note that this bit is only available when the UART clock (f_{H1}) is switched off. There will be no RX/TX pin wake-up UART function if the UART clock (f_H) exists. If the WAKE bit is set to 1 as the UART clock (f_H) is switched off, a UART wake-up request will be initiated when a falling edge on the RX/TX pin occurs. When this request happens and the corresponding interrupt is enabled, an RX/TX pin wake-up UART interrupt will be generated to inform the MCU to wake up the UART function by switching on the UART clock (f_H) via the application program. Otherwise, the UART function cannot resume even if there is a falling edge on the RX/TX pin when the WAKE bit is cleared to 0.</p> |
| Bit 2 | <p>RIE: Receiver interrupt enable control</p> <p>0: Receiver related interrupt is disabled</p> <p>1: Receiver related interrupt is enabled</p> <p>This bit enables or disables the receiver interrupt. If this bit is equal to “1” and when the receiver overrun flag OERR or receive data available flag RXIF is set, the UART interrupt request flag will be set. If this bit is equal to “0”, the UART interrupt request flag will not be influenced by the condition of the OERR or RXIF flags.</p> |
| Bit 1 | <p>TIE: Transmitter Idle interrupt enable control</p> <p>0: Transmitter idle interrupt is disabled</p> <p>1: Transmitter idle interrupt is enabled</p> <p>This bit enables or disables the transmitter idle interrupt. If this bit is equal to “1” and when the transmitter idle flag TIDLE is set, due to a transmitter idle condition, the UART interrupt request flag will be set. If this bit is equal to “0”, the UART interrupt request flag will not be influenced by the condition of the TIDLE flag.</p> |

Bit 0 **TEIE**: Transmitter Empty interrupt enable control
 0: Transmitter empty interrupt is disabled
 1: Transmitter empty interrupt is enabled
 This bit enables or disables the transmitter empty interrupt. If this bit is equal to “1” and when the transmitter empty flag TXIF is set, due to a transmitter empty condition, the UART interrupt request flag will be set. If this bit is equal to “0”, the UART interrupt request flag will not be influenced by the condition of the TXIF flag.

• UCR3 Register

The UCR3 register is used to enable the UART Single Wire Mode communication. As the name suggests in the single wire mode the UART communication can be implemented in one single line, RX/TX, together with the control of the RXEN and TXEN bits in the UCR2 register.

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

Bit 7~1 Unimplemented, read as “0”

Bit 0 **SWM**: Single Wire Mode enable control
 0: Disable, the RX/TX pin is used as UART receiver function only
 1: Enable, the RX/TX pin can be used as UART receiver or transmitter function controlled by the RXEN and TXEN bits
 Note that when the Single Wire Mode is enabled, if both the RXEN and TXEN bits are high, the RX/TX pin will only be used as UART receiver input.

• TXR_RXR Register

The TXR_RXR register is the data register which is used to store the data to be transmitted on the TX pin or being received from the RX/TX pin.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|-------|-------|-------|-------|-------|-------|-------|
| Name | TXRX7 | TXRX6 | TXRX5 | TXRX4 | TXRX3 | TXRX2 | TXRX1 | TXRX0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | x | x | x | x | x | x | x | x |

“x”: unknown

Bit 7~0 **TXRX7~TXRX0**: UART Transmit/Receive Data bit 7 ~ bit 0

• BRDH Register

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

Bit 7~0 **D7~D0**: Baud rate divider high byte
 The baud rate divider BRD (BRDH/BRDL) defines the UART clock divider ratio.

$$\text{Baud Rate} = f_{\text{H}} / (\text{BRD} + \text{UMOD} / 8)$$

$$\text{BRD} = 16 \sim 65535 \text{ or } 8 \sim 65535 \text{ depending on BRDS}$$

 Note: 1. BRD value should not be set to less than 16 when BRDS=0 or less than 8 when BRDS=1, otherwise errors may occur.
 2. The BRDL must be written first and then BRDH, otherwise errors may occur.
 3. The BRDH register should not be modified during data transmission process.

• **BRDL Register**

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

- Bit 7~0 **D7~D0**: Baud rate divider low byte
The baud rate divider BRD (BRDH/BRDL) defines the UART clock divider ratio.
 $\text{Baud Rate} = f_{\text{H}} / (\text{BRD} + \text{UMOD} / 8)$
 $\text{BRD} = 16 \sim 65535$ or $8 \sim 65535$ depending on BRDS
Note: 1. BRD value should not be set to less than 16 when BRDS=0 or less than 8 when BRDS=1, otherwise errors may occur.
2. The BRDL must be written first and then BRDH, otherwise errors may occur.
3. The BRDL register should not be modified during data transmission process.

• **UFCR Register**

The UFCR register is the FIFO control register which is used for UART modulation control, BRD range selection and trigger level selection for RXIF and interrupt.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|-------|-------|-------|------|--------|--------|
| Name | — | — | UMOD2 | UMOD1 | UMOD0 | BRDS | RxFTR1 | RxFTR0 |
| R/W | — | — | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | — | — | 0 | 0 | 0 | 0 | 0 | 0 |

- Bit 7~6 Unimplemented, read as “0”
Bit 5~3 **UMOD2~UMOD0**: UART Modulation Control bits
The modulation control bits are used to correct the baud rate of the received or transmitted UART signal. These bits determine if the extra UART clock cycle should be added in a UART bit time. The UMOD2~UMOD0 will be added to internal accumulator for every UART bit time. Until a carry to bit 3, the corresponding UART bit time increases a UART clock cycle.
Bit 2 **BRDS**: BRD range selection
0: BRD range is from 16 to 65535
1: BRD range is from 8 to 65535
The BRDS is used to control the sampling point in a UART bit time. If the BRDS is cleared to zero, the sampling point will be $\text{BRD}/2$, $\text{BRD}/2 + 1 \times f_{\text{H}}$, and $\text{BRD}/2 + 2 \times f_{\text{H}}$ in a UART bit time. If the BRDS is set high, the sampling point will be $\text{BRD}/2 - 1 \times f_{\text{H}}$, $\text{BRD}/2$, and $\text{BRD}/2 + 2 \times f_{\text{H}}$ in a UART bit time.
Note that the BRDS bit should not be modified during data transmission process.
Bit 1~0 **RxFTR1~RxFTR0**: Receiver FIFO trigger level (bytes)
00: 4 bytes in Receiver FIFO
01: 1 or more bytes in Receiver FIFO
10: 2 or more bytes in Receiver FIFO
11: 3 or more bytes in Receiver FIFO
For the receiver these bits define the number of received data bytes in the Receiver FIFO that will trigger the RXIF bit being set high, an interrupt will also be generated if the RIE bit is enabled. To prevent OERR from being set high, the receiver FIFO trigger level can be set to 2 bytes, avoiding an overrun state that cannot be processed by the program in time when more than 4 data bytes are received. After the reset the receiver FIFO is empty.

• RxCNT Register

The RxCNT register is the counter used to indicate the number of received data bytes in the Receiver FIFO which have not been read by the MCU. This register is read only.

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

Bit 7~3 Unimplemented, read as “0”

Bit 2~0 **D2~D0**: Receiver FIFO counter

The RxCNT register is the counter used to indicate the number of receiver data bytes in Receiver FIFO which is not read by MCU. When Receiver FIFO receives one byte data, the RxCNT will increase by one; when the MCU reads one byte data from Receiver FIFO, the RxCNT will decrease by one. If there are 4 bytes of data in the Receiver FIFO, the 5th data will be saved in the shift register. If there is 6th data, the 6th data will be saved in the shift register. But the RxCNT remains the value of 4. The RxCNT will be cleared when reset occurs or UARTEN=1. This register is read only.

Baud Rate Generator

To setup the speed of the serial data communication, the UART function contains its own dedicated baud rate generator. The baud rate is controlled by its own internal free running 16-bit timer, the period of which is determined by two factors. The first of these is the value placed in BRDH/BRDL register and the second is the UART modulation control bits (UMOD2~UMOD0). To prevent accumulated error of the receiver baud rate frequency, it is recommended to use two stop bits for resynchronization after each byte is received. If a baud rate BR is required with UART clock f_H .

$$f_H/BR = \text{Integer Part} + \text{Fractional Part}$$

The integer part is loaded into BRD (BRDH/BRDL). The fractional part is multiplied by 8 and rounded, then loaded into UMOD bit field as following:

$$BRD = \text{TRUNC}(f_H/BR)$$

$$UMOD = \text{ROUND}[\text{MOD}(f_H/BR) \times 8]$$

Therefore, the actual baud rate is as following:

$$\text{Baud rate} = f_H / [BRD + (UMOD/8)]$$

Calculating the Baud Rate and Error Values

For a clock frequency of 4MHz, determine the BRDH/BRDL register value, the actual baud rate and the error value for a desired baud rate of 230400.

From the above formula, the $BRD = \text{TRUNC}(f_H/BR) = \text{TRUNC}(17.36111) = 17$

The $UMOD = \text{ROUND}[\text{MOD}(f_H/BR) \times 8] = \text{ROUND}(0.36111 \times 8) = \text{ROUND}(2.88888) = 3$

The actual Baud Rate $= f_H / [BRD + (UMOD/8)] = 230215.83$

Therefore the error is equal to $(230215.83 - 230400) / 230400 = -0.08\%$

Modulation Control Example

To get the best-fitting bit sequence for UART modulation control bits UMOD2~UMOD0, the following algorithm can be used: Firstly, the fractional part of the theoretical division factor is multiplied by 8. Then the product will be rounded and UMOD2~UMOD0 bits will be filled with the rounded value. The UMOD2~UMOD0 bits will be added to internal accumulator for every UART bit time. Until a carry to bit 3, the corresponding UART bit time increases a UART clock cycle. The

following is an example using the fraction 0.36111 previously calculated: $UMOD[2:0] = \text{ROUND}(0.36111 \times 8) = 011b$.

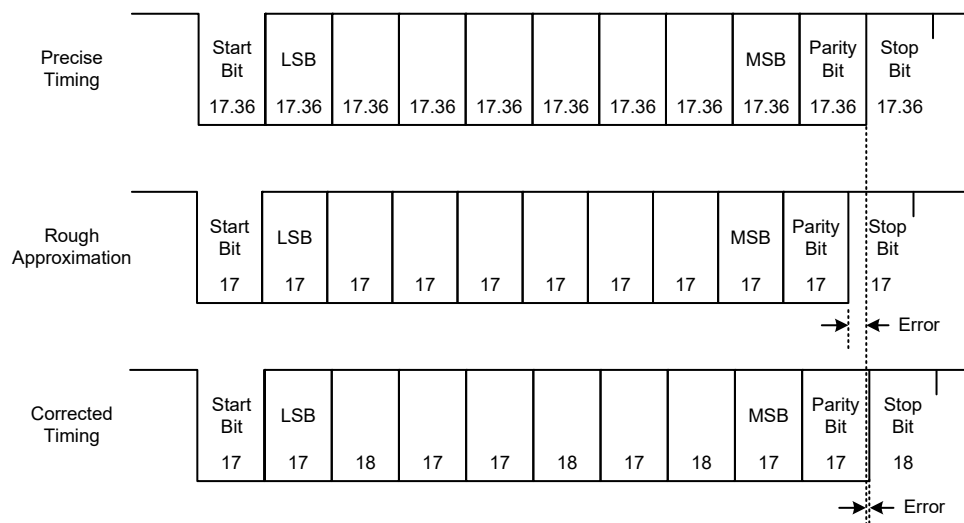
| Fraction Addition | Carry to Bit 3 | UART Bit Time Sequence | Extra UART Clock Cycle |
|-------------------|----------------|------------------------|------------------------|
| 0000b+0011b=0011b | No | Start bit | No |
| 0011b+0011b=0110b | No | D0 | No |
| 0110b+0011b=1001b | Yes | D1 | Yes |
| 1001b+0011b=1100b | No | D2 | No |
| 1100b+0011b=1111b | No | D3 | No |
| 1111b+0011b=0010b | Yes | D4 | Yes |
| 0010b+0011b=0101b | No | D5 | No |
| 0101b+0011b=1000b | Yes | D6 | Yes |
| 1000b+0011b=1011b | No | D7 | No |
| 1011b+0011b=1110b | No | Parity bit | No |
| 1110b+0011b=0001b | Yes | Stop bit | Yes |

Baud Rate Correction Example

The following figure presents an example using a baud rate of 230400 generated with UART clock f_H . The data format for the following figure is: eight data bits, parity enabled, no address bit; two stop bits.

The following figure shows three different frames:

- The upper frame is the correct one, with a bit-length of 17.36 f_H cycles ($4000000/230400=17.36$).
- The middle frame uses a rough estimate, with 17 f_H cycles for the bit length.
- The lower frame shows a corrected frame using the best fit for the UART modulation control bits $UMOD2 \sim UMOD0$.



UART Setup and Control

For data transfer, the UART function utilizes a non-return-to-zero, more commonly known as NRZ, format. This is composed of one start bit, eight or nine data bits, and one or two stop bits. Parity is supported by the UART hardware, and can be set to be even, odd, mark, space or no parity. For the most common data format, 8 data bits along with no parity and one stop bit, denoted as 8, N, 1, is used as the default setting, which is the setting at power-on. The number of data bits along with the parity are setup by programming the BNO, PRT1~PRT0 and PREN bits. The transmitter always uses

two stop bits while the receiver uses one or two stop bits which is determined by the STOPS bit. The baud rate used to transmit and receive data is set using the internal 16-bit baud rate generator, while the data is transmitted and received LSB first. Although the UART transmitter and receiver are functionally independent, they both use the same data format and baud rate. In all cases stop bits will be used for data transmission.

Enabling/Disabling the UART Interface

The basic on/off function of the internal UART function is controlled using the UARTEN bit in the UCR1 register. If the UARTEN, TXEN and RXEN bits are set, then these two UART pins will act as normal TX output pin and RX/TX input pin respectively. If no data is being transmitted on the TX pin, then it will default to a logic high value.

Clearing the UARTEN bit will disable the TX and RX/TX pins and allow these two pins to be used as normal I/O or other pin-shared functional pins by configuring the corresponding pin-shared control bits. When the UART function is disabled the buffer will be reset to an empty condition, at the same time discarding any remaining residual data. Disabling the UART will also reset the error and status flags with bits TXEN, RXEN, TXBRK, RXIF, OERR, FERR, PERR and NF as well as register RxCNT being cleared while bits TIDLE, TXIF and RIDLE will be set. The remaining control bits in the UCR1, UCR2, UCR3, UFCR, BRDH and BRDL registers will remain unaffected. If the UARTEN bit in the UCR1 register is cleared while the UART is active, then all pending transmissions and receptions will be immediately suspended and the UART will be reset to a condition as defined above. If the UART is then subsequently re-enabled, it will restart again in the same configuration.

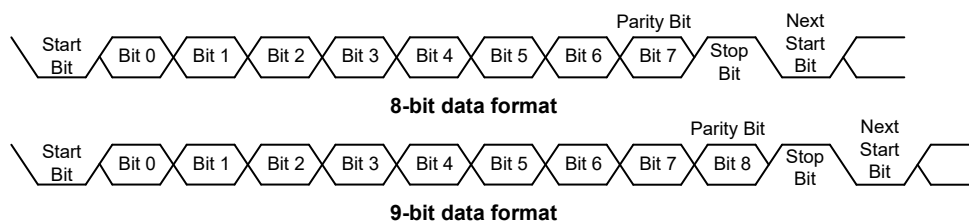
Data, Parity and Stop Bit Selection

The format of the data to be transferred is composed of various factors such as data bit length, parity on/off, parity type, address bits and the number of stop bits. These factors are determined by the setup of various bits within the UCR1 and UCR2 registers. The BNO bit controls the number of data bits which can be set to either 8 or 9, the PRT1~PRT0 bits control the choice of odd, even, mark or space parity, the PREN bit controls the parity on/off function and the STOPS bit decides whether one or two stop bits are to be used for the receiver, while the transmitter always uses two stop bits. The following table shows various formats for data transmission. The address bit, which is the MSB of the data byte, identifies the frame as an address character or data if the address detect function is enabled. The number of stop bits, which can be either one or two, is independent of the data length and is only configurable for the receiver. The transmitter uses two stop bits.

| Start Bit | Data Bits | Address Bit | Parity Bit | Stop Bit |
|--------------------------------------|-----------|-------------|------------|----------|
| Example of 8-bit Data Formats | | | | |
| 1 | 8 | 0 | 0 | 1 or 2 |
| 1 | 7 | 0 | 1 | 1 or 2 |
| 1 | 7 | 1 | 0 | 1 or 2 |
| Example of 9-bit Data Formats | | | | |
| 1 | 9 | 0 | 0 | 1 or 2 |
| 1 | 8 | 0 | 1 | 1 or 2 |
| 1 | 8 | 1 | 0 | 1 or 2 |

Transmitter Receiver Data Format

The following diagram shows the transmit and receive waveforms for both 8-bit and 9-bit data formats.



UART Transmitter

Data word lengths of either 8 or 9 bits can be selected by programming the BNO bit in the UCR1 register. When the BNO bit is set, the word length will be set to 9 bits. In this case the 9th bit, which is the MSB, needs to be stored in the TX8 bit in the UCR1 register. At the transmitter core lies the Transmitter Shift Register, more commonly known as the TSR, whose data is obtained from the transmit data register, which is known as the TXR_RXR register. The data to be transmitted is loaded into this TXR_RXR register by the application program. The TSR register is not written to with new data until the stop bit from the previous transmission has been sent out. As soon as this stop bit has been transmitted, the TSR can then be loaded with new data from the TXR_RXR register, if it is available. It should be noted that the TSR register, unlike many other registers, is not directly mapped into the Data Memory area and as such is not available to the application program for direct read/write operations. An actual transmission of data will normally be enabled when the TXEN bit is set, but the data will not be transmitted until the TXR_RXR register has been loaded with data and the baud rate generator has defined a shift clock source. However, the transmission can also be initiated by first loading data into the TXR_RXR register, after which the TXEN bit can be set. When a transmission of data begins, the TSR is normally empty, in which case a transfer to the TXR_RXR register will result in an immediate transfer to the TSR. If during a transmission the TXEN bit is cleared, the transmission will immediately cease and the transmitter will be reset. The TX output pin can then be configured as the I/O or other pin-shared function by configuring the corresponding pin-shared control bits.

Transmitting Data

When the UART is transmitting data, the data is shifted on the TX pin from the shift register, with the least significant bit LSB first. In the transmit mode, the TXR_RXR register forms a buffer between the internal bus and the transmitter shift register. It should be noted that if 9-bit data format has been selected, then the MSB will be taken from the TX8 bit in the UCR1 register. The steps to initiate a data transfer can be summarized as follows:

- Make the correct selection of the BNO, PRT1~PRT0 and PREN bits to define the required word length and parity type. Two stop bits are used for the transmitter.
- Setup the BRDH, BRDL registers and UMOD2~UMOD0 bits to select the desired baud rate.
- Set the TXEN bit to ensure that the UART transmitter is enabled and the TX pin is used as a UART transmitter pin.
- Access the USR register and write the data that is to be transmitted into the TXR_RXR register. Note that this step will clear the TXIF bit.

This sequence of events can now be repeated to send additional data.

It should be noted that when TXIF=0, data will be inhibited from being written to the TXR_RXR register. Clearing the TXIF flag is always achieved using the following software sequence:

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

The read-only TXIF flag is set by the UART hardware and if set indicates that the TXR_RXR register is empty and that other data can now be written into the TXR_RXR register without overwriting the previous data. If the TEIE bit is set then the TXIF flag will generate an interrupt.

During a data transmission, a write instruction to the TXR_RXR register will place the data into the TXR_RXR register, which will be copied to the shift register at the end of the present transmission. When there is no data transmission in progress, a write instruction to the TXR_RXR register will place the data directly into the shift register, resulting in the commencement of data transmission, and the TXIF bit being immediately set. When a frame transmission is complete, which happens after stop bits are sent or after the break frame, the TIDLE bit will be set. To clear the TIDLE bit the following software sequence is used:

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

Note that both the TXIF and TIDLE bits are cleared by the same software sequence.

Transmitting Break

If the TXBRK bit is set high and the state keeps for a time of greater than $[(BRD+1) \times t_{H}]$ while TIDLE=1, then break characters will be sent on the next transmission. Break character transmission consists of a start bit, followed by $13 \times N$ '0' bits and stop bits, where $N=1, 2$, etc. If a break character is to be transmitted then the TXBRK bit must be first set by the application program, and then cleared to generate the stop bits. Transmitting a break character will not generate a transmit interrupt. Note that a break condition length is at least 13 bits long. If the TXBRK bit is continually kept at a logic high level then the transmitter circuitry will transmit continuous break characters. After the application program has cleared the TXBRK bit, the transmitter will finish transmitting the last break character and subsequently send out two stop bits. The automatic logic highs at the end of the last break character will ensure that the start bit of the next frame is recognized.

UART Receiver

The UART is capable of receiving word lengths of either 8 or 9 bits. If the BNO bit is set, the word length will be set to 9 bits with the MSB being stored in the RX8 bit of the UCR1 register. At the receiver core lies the Receive Serial Shift Register, commonly known as the RSR. The data which is received on the RX/TX external input pin is sent to the data recovery block. The data recovery block operating speed is 16 times that of the baud rate, while the main receive serial shifter operates at the baud rate. After the RX/TX pin is sampled for the stop bit, the received data in RSR is transferred to the receive data register, if the register is empty. The data which is received on the external RX/TX pin is sampled three times by a majority detect circuit to determine the logic level that has been placed onto the RX/TX pin. It should be noted that the RSR register, unlike many other registers, is not directly mapped into the Data Memory area and as such is not available to the application program for direct read/write operations.

Receiving Data

When the UART receiver is receiving data, the data is serially shifted in on the external RX/TX pin, LSB first. In the read mode, the TXR_RXR register forms a buffer between the internal bus and the receiver shift register. The TXR_RXR register is a four-byte deep FIFO data buffer, where four bytes can be held in the FIFO while a fifth byte can continue to be received. Note that the application program must ensure that the data is read from TXR_RXR before the fifth byte has been completely shifted in, otherwise this fifth byte will be discarded and an overrun error OERR will be subsequently indicated. For continuous multi-byte data transmission, it is strongly recommended

that the receiver uses two stop bits to avoid a receiving error caused by the accumulated error of the receiver baud rate frequency.

The steps to initiate a data transfer can be summarized as follows:

- Make the correct selection of BNO, PRT1~PRT0, PREN and STOPS bits to define the word length, parity type and number of stop bits.
- Setup the BRDH, BRDL registers and the UMOD2~UMOD0 bits to select the desired baud rate.
- Set the RXEN bit to ensure that the RX/TX pin is used as a UART receiver pin.

At this point the receiver will be enabled which will begin to look for a start bit.

When a character is received the following sequence of events will occur:

- The RXIF bit in the USR register will be set when the TXR_RXR register has data available, the number of the available data bytes can be checked by polling the RxCNT register content.
- When the contents of the shift register have been transferred to the TXR_RXR register and reach receiver FIFO trigger level, if the RIE bit is set, an interrupt will be generated.
- If during reception, a frame error, noise error, parity error or an overrun error has been detected, then the error flags can be set.

The RXIF bit can be cleared using the following software sequence:

1. A USR register access
2. A TXR_RXR register read execution

Receiving Break

Any break character received by the UART will be managed as a framing error. The receiver will count and expect a certain number of bit times as specified by the values programmed into the BNO bit plus one or two stop bits. If the break is much longer than 13 bit times, the reception will be considered as complete after the number of bit times specified by BNO plus one or two stop bits. The RXIF bit is set, FERR is set, zeros are loaded into the receive data register, interrupts are generated if appropriate and the RIDLE bit is set. A break is regarded as a character that contains only zeros with the FERR flag set. If a long break signal has been detected, the receiver will regard it as a data frame including a start bit, data bits and the invalid stop bit and the FERR flag will be set. The receiver must wait for a valid stop bit before looking for the next start bit. The receiver will not make the assumption that the break condition on the line is the next start bit. The break character will be loaded into the buffer and no further data will be received until one or two stop bits are received. It should be noted that the RIDLE read only flag will go high when the stop bits have not yet been received. The reception of a break character on the UART registers will result in the following:

- The framing error flag, FERR, will be set.
- The receive data register, TXR_RXR, will be cleared.
- The OERR, NF, PERR, RIDLE or RXIF flags will possibly be set.

Idle Status

When the receiver is reading data, which means it will be in between the detection of a start bit and the reading of a stop bit, the receiver status flag in the USR register, otherwise known as the RIDLE flag, will have a zero value. In between the reception of a stop bit and the detection of the next start bit, the RIDLE flag will have a high value, which indicates the receiver is in an idle condition.

Receiver Interrupt

The read only receive interrupt flag RXIF in the USR register is set by an edge generated by the receiver. An interrupt is generated if RIE=1, when a word is transferred from the Receive Shift Register, RSR, to the Receive Data Register, TXR_RXR. An overrun error can also generate an interrupt if RIE=1.

When a subroutine will be called with an execution time longer than the time for UART to receive five data bytes, if the UART received data could not be read in time during the subroutine execution, clear the RXEN bit to zero in advance to suspend data reception. If the UART interrupt could not be served in time to process the overrun error during the subroutine execution, ensure that both EMI and RXEN bits are disabled during this period, and then enable EMI and RXEN again after the subroutine execution has been completed to continue the UART data reception.

Managing Receiver Errors

Several types of reception errors can occur within the UART module, the following section describes the various types and how they are managed by the UART.

Overrun Error – OERR

The TXR_RXR register is composed of a four-byte deep FIFO data buffer, where four bytes can be held in the FIFO register, while a fifth byte can continue to be received. Before this fifth byte has been entirely shifted in, the data should be read from the TXR_RXR register. If this is not done, the overrun error flag OERR will be consequently indicated.

In the event of an overrun error occurring, the following will happen:

- The OERR flag in the USR register will be set.
- The TXR_RXR contents will not be lost.
- The shift register will be overwritten.
- An interrupt will be generated if the RIE bit is set.

When the OERR flag is set to “1”, it is necessary to read five data bytes from the four-byte deep receiver FIFO and the shift register immediately to avoid unexpected errors, such as the UART is unable to receive data. If such an error occurs, clear the RXEN bit to “0” then set it to “1” again to continue data reception.

The OERR flag can be cleared by an access to the USR register followed by a read to the TXR_RXR register.

Noise Error – NF

Over-sampling is used for data recovery to identify valid incoming data and noise. If noise is detected within a frame, the following will occur:

- The read only noise flag, NF, in the USR register will be set on the rising edge of the RXIF bit.
- Data will be transferred from the Shift register to the TXR_RXR register.
- No interrupt will be generated. However this bit rises at the same time as the RXIF bit which itself generates an interrupt.

Note that the NF flag is reset by an USR register read operation followed by a TXR_RXR register read operation.

Framing Error – FERR

The read only framing error flag, FERR, in the USR register, is set if a zero is detected instead of stop bits. If two stop bits are selected, both stop bits must be high; otherwise the FERR flag will

be set. The FERR flag and the received data will be recorded in the USR and TXR_RXR registers respectively, and the flag is cleared in any reset.

Parity Error – PERR

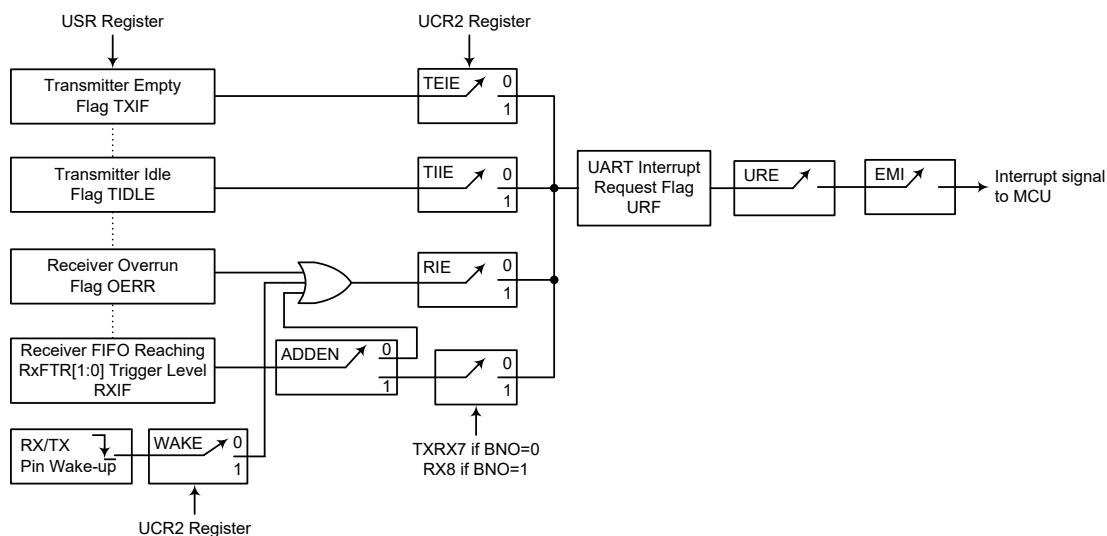
The read only parity error flag, PERR, in the USR register, is set if the parity of the received word is incorrect. This error flag is only applicable if the parity is enabled, PREN=1, and if the parity type, odd, even, mark or space is selected. The read only PERR flag and the received data will be recorded in the USR and TXR_RXR registers respectively. It is cleared on any reset, it should be noted that the flags, FERR and PERR, in the USR register should first be read by the application program before reading the data word.

UART Interrupt Structure

Several individual UART conditions can generate a UART interrupt. When these conditions exist, a low pulse will be generated to get the attention of the microcontroller. These conditions are a transmitter data register empty, transmitter idle, receiver reaching FIFO trigger level, receiver overrun, address detect and an RX/TX pin wake-up. When any of these conditions are created, if the global interrupt enable bit and its corresponding interrupt control bit are enabled and the stack is not full, the program will jump to its corresponding interrupt vector where it can be serviced before returning to the main program. Four of these conditions have the corresponding USR register flags which will generate a UART interrupt if its associated interrupt enable control bit in the UCR2 register is set. The two transmitter interrupt conditions have their own corresponding enable control bits, while the two receiver interrupt conditions have a shared enable control bit. These enable bits can be used to mask out individual UART interrupt sources.

The address detect condition, which is also a UART interrupt source, does not have an associated flag, but will generate a UART interrupt when an address detect condition occurs if its function is enabled by setting the ADDEN bit in the UCR2 register. An RX/TX pin wake-up, which is also a UART interrupt source, does not have an associated flag, but will generate a UART interrupt if the UART clock (f_H) source is switched off and the WAKE and RIE bits in the UCR2 register are set when a falling edge on the RX/TX pin occurs.

Note that the USR register flags are read only and cannot be cleared or set by the application program, neither will they be cleared when the program jumps to the corresponding interrupt servicing routine, as is the case for some of the other interrupts. The flags will be cleared automatically when certain actions are taken by the UART, the details of which are given in the UART register section. The overall UART interrupt can be disabled or enabled by the related interrupt enable control bits in the interrupt control registers of the microcontroller to decide whether the interrupt requested by the UART module is masked out or allowed.



UART Interrupt Structure

Address Detect Mode

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

| ADDEN | 9th Bit if BNO=1 8th Bit if BNO=0 | UART Interrupt Generated |
|-------|--------------------------------------|--------------------------|
| 0 | 0 | √ |
| | 1 | √ |
| 1 | 0 | × |
| | 1 | √ |

ADDEN Bit Function

UART Power Down and Wake-up

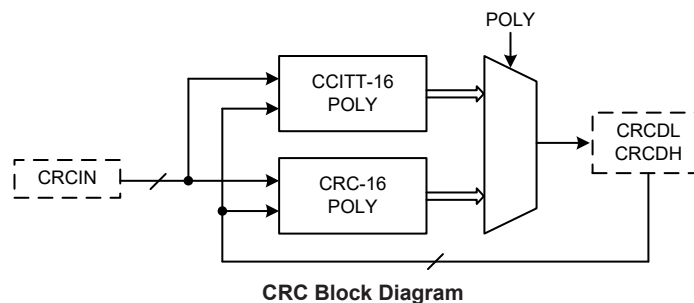
When the UART clock (f_{H}) is off, the UART will cease to function, and all clock sources to the module are shutdown. If the UART clock (f_{H}) is off while a transmission is still in progress, then the transmission will be paused until the UART clock source derived from the microcontroller is activated. In a similar way, if the MCU enters the IDLE or SLEEP Mode while receiving data, then the reception of data will likewise be paused. When the MCU enters the IDLE or SLEEP Mode, note that the USR, UCR1, UCR2, UCR3, UFCR, RxCNT and TXR_RXR as well as the BRDH and BRDL registers will not be affected. It is recommended to make sure first that the UART data transmission or reception has been finished before the microcontroller enters the IDLE or SLEEP mode.

The UART function contains a receiver RX/TX pin wake-up function, which is enabled or disabled by the WAKE bit in the UCR2 register. If this bit, along with the UART enable bit, UARTEN, the receiver enable bit, RXEN and the receiver interrupt bit, RIE, are all set when the UART clock (f_{H}) is off, then a falling edge on the RX/TX pin will trigger an RX/TX pin wake-up UART interrupt. Note that as it takes certain system clock cycles after a wake-up, before normal microcontroller operation resumes, any data received during this time on the RX/TX pin will be ignored.

For a UART wake-up interrupt to occur, in addition to the bits for the wake-up being set, the global interrupt enable bit, EMI, and the UART interrupt enable bit, URE, must be set. If the EMI and URE bits are not set then only a wake-up event will occur and no interrupt will be generated. Note also that as it takes certain system clock cycles after a wake-up before normal microcontroller resumes, the UART interrupt will not be generated until after this time has elapsed.

Cyclic Redundancy Check – CRC

The Cyclic Redundancy Check, CRC, calculation unit is an error detection technique test algorithm and uses to verify data transmission or storage data correctness. A CRC calculation takes a data stream or a block of data as input and generates a 16-bit output remainder. Ordinarily, a data stream is suffixed by a CRC code and used as a checksum when being sent or stored. Therefore, the received or restored data stream is calculated by the same generator polynomial as described in the following section.



CRC Registers

The CRC generator contains an 8-bit CRC data input register, CRCIN, and a CRC checksum register pair, CRCDH and CRCDL. The CRCIN register is used to input new data and the CRCDH and CRCDL registers are used to hold the previous CRC calculation result. A CRC control register, CRCCR, is used to select which CRC generating polynomial is used.

| Register Name | Bit | | | | | | | |
|---------------|-----|-----|-----|-----|-----|-----|----|------|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CRCIN | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| CRCDL | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
| CRCDH | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| CRCCR | — | — | — | — | — | — | — | POLY |

CRC Register List

• CRCIN Register

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

Bit 7~0 **D7~D0**: CRC input data register

• CRCDL Register

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

Bit 7~0 **D7~D0**: 16-bit CRC checksum low byte data register

• CRCDH Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-----|-----|-----|-----|-----|-----|-----|-----|
| Name | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |
| 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 **D15~D8**: 16-bit CRC checksum high byte data register

• CRCCR Register

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

Bit 7~1 Unimplemented, read as “0”

Bit 0 **POLY**: 16-bit CRC generating polynomial selection

0: CRC-CCITT: $X^{16}+X^{12}+X^5+1$

1: CRC-16: $X^{16}+X^{15}+X^2+1$

CRC Operation

The CRC generator provides the 16-bit CRC result calculation based on the CRC16 and CCITT CRC16 polynomials. In this CRC generator, there are only these two polynomials available for the numeric values calculation. It can not support the 16-bit CRC calculations based on any other polynomials.

The following two expressions can be used for the CRC generating polynomial which is determined using the POLY bit in the CRC control register, CRCCR. The CRC calculation result is called as the CRC checksum, CRCSUM, and stored in the CRC checksum register pair, CRCDH and CRCDL.

- CRC-CCITT: $X^{16}+X^{12}+X^5+1$
- CRC-16: $X^{16}+X^{15}+X^2+1$

CRC Computation

Each write operation to the CRCIN register creates a combination of the previous CRC value stored in the CRCDH and CRCDL registers and the new data input. The CRC unit calculates the CRC data register value is based on byte by byte. It will take one MCU instruction cycle to calculate the CRC checksum.

CRC Calculation Procedures:

1. Clear the checksum register pair, CRCDH and CRCDL.
2. Execute an “Exclusive OR” operation with the 8-bit input data byte and the 16-bit CRCSUM high byte. The result is called the temporary CRCSUM.
3. Shift the temporary CRCSUM value left by one bit and move a “0” into the LSB.
4. Check the shifted temporary CRCSUM value after procedure 3.

If the MSB is 0, then this shifted temporary CRCSUM will be considered as a new temporary CRCSUM.

Otherwise, execute an “Exclusive OR” operation with the shifted temporary CRCSUM in procedure 3 and a data “8005H”. Then the operation result will be regarded as the new temporary CRCSUM.

Note that the data to be perform an “Exclusive OR” operation is “8005H” for the CRC-16 polynomial while for the CRC-CCITT polynomial the data is “1021H”.

5. Repeat the procedure 3 ~ procedure 4 until all bits of the input data byte are completely calculated.
6. Repeat the procedure 2 ~ procedure 5 until all of the input data bytes are completely calculated. Then, the latest calculated result is the final CRC checksum, CRCSUM.

CRC Calculation Examples:

- Write 1 byte input data into the CRCIN register and the corresponding CRC checksum are individually calculated as the following table shown.

| CRC Data Input CRC Polynomial | 00H | 01H | 02H | 03H | 04H | 05H | 06H | 07H |
|-------------------------------------|-------|-------|-------|-------|-------|-------|-------|-------|
| CRC-CCITT ($X^{16}+X^{12}+X^5+1$) | 0000H | 1021H | 2042H | 3063H | 4084H | 50A5H | 60C6H | 70E7H |
| CRC-16 ($X^{16}+X^{15}+X^2+1$) | 0000H | 8005H | 800FH | 000AH | 801BH | 001EH | 0014H | 8011H |

Note: The initial value of the CRC checksum register pair, CRCDH and CRCDL, is zero before each CRC input data is written into the CRCIN register.

- Write 4 bytes input data into the CRCIN register sequentially and the CRC checksum are sequentially listed in the following table.

| CRC Data Input CRC Polynomial | CRCIN=78H→56H→34H→12H |
|-------------------------------------|--|
| CRC-CCITT ($X^{16}+X^{12}+X^5+1$) | (CRCDH, CRCDL)=FF9FH→BBC3H→A367H→D0FAH |
| CRC-16 ($X^{16}+X^{15}+X^2+1$) | (CRCDH, CRCDL)=0110H→91F1H→F2DEH→5C43H |

Note: The initial value of the CRC checksum register pair, CRCDH and CRCDL, is zero before the sequential CRC data input operation.

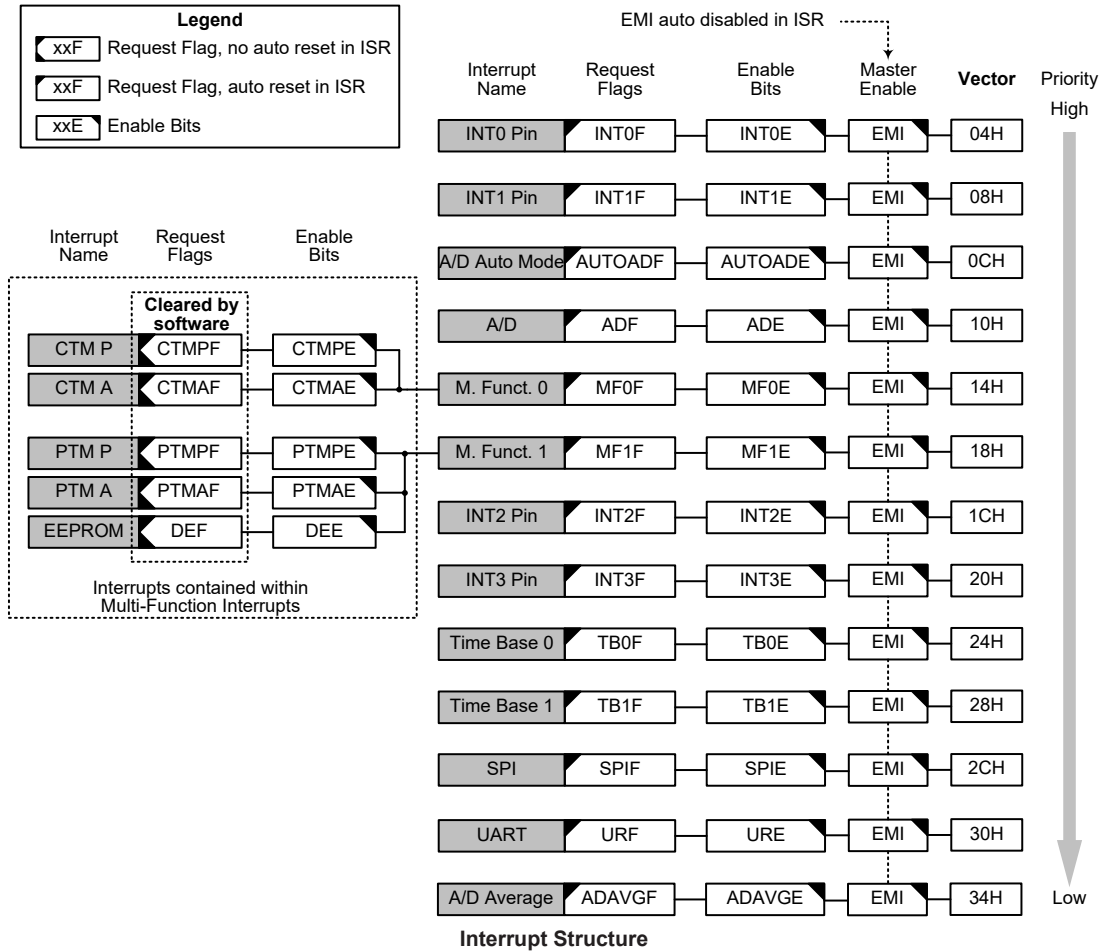
Program Memory CRC Checksum Calculation Example:

1. Clear the checksum register pair, CRCDH and CRCDL.
2. Select the CRC-CCITT or CRC-16 polynomial as the generating polynomial using the POLY bit in the CRCCR register.
3. Execute the table read instruction to read the program memory data value.
4. Write the table data low byte into the CRCIN register and execute the CRC calculation with the current CRCSUM value. Then a new CRCSUM result will be obtained and stored in the CRC checksum register pair, CRCDH and CRCDL.
5. Write the table data high byte into the CRCIN register and execute the CRC calculation with the current CRCSUM value. Then a new CRCSUM result will be obtained and stored in the CRC checksum register pair, CRCDH and CRCDL.
6. Repeat the procedure 3 ~ procedure 5 to read the next program memory data value and execute the CRC calculation until all program memory data are read followed by the sequential CRC calculation. Then the value in the CRC checksum register pair is the final CRC calculation result.

Interrupts

Interrupts are an important part of any microcontroller system. When an external event or an internal function such as a Timer Module or an A/D converter requires microcontroller attention, their corresponding interrupt will enforce a temporary suspension of the main program allowing the microcontroller to direct attention to their respective needs. The device contains several external interrupt and internal interrupts functions. The external interrupts are generated by the action of the external INT0~INT3 pins, while the internal interrupts are generated by various internal functions such as the TMs, Time Base, EEPROM, SPI, UART and the A/D converter, etc.

The various interrupt enable bits, together with their associated request flags, are shown in the accompanying diagrams with their order of priority. Some interrupt sources have their own individual vector while others share the same multi-function interrupt vector.



Interrupt Registers

Overall interrupt control, which basically means the setting of request flags when certain microcontroller conditions occur and the setting of interrupt enable bits by the application program, is controlled by a series of registers, located in the Special Purpose Data Memory, as shown in the accompanying table. The number of registers depends upon the device chosen but fall into three categories. The first is the INTC0~INTC3 registers which setup the primary interrupts, the second is the MFI0~MFI1 registers which setup the Multi-function interrupts. Finally there is an INTEG register to setup the external interrupt trigger edge type.

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

| Function | Enable Bit | Request Flag | Notes |
|-------------------------|------------|--------------|------------|
| Global | EMI | — | — |
| INTn Pins | INTnE | INTnF | n=0~3 |
| A/D Converter | ADE | ADF | ADINT |
| A/D Converter Auto Mode | AUTOADE | AUTOADF | AUTOADCINT |
| A/D Converter Average | ADAVGE | ADAVGF | ADAVGINT |
| Multi-function | MFnE | MFnF | n=0~1 |

| Function | Enable Bit | Request Flag | Notes |
|-----------|------------|--------------|-------|
| Time Base | TBnE | TBnF | n=0~1 |
| SPI | SPIE | SPIF | — |
| UART | URE | URF | — |
| EEPROM | DEE | DEF | — |
| CTM | CTMPE | CTMPF | — |
| | CTMAE | CTMAF | |
| PTM | PTMPE | PTMPF | — |
| | PTMAE | PTMAF | |

Interrupt Register Bit Naming Conventions

| Register Name | Bit | | | | | | | |
|---------------|--------|---------|--------|--------|---------|--------|--------|--------|
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| INTEG | INT3S1 | INT3S0 | INT2S1 | INT2S0 | INT1S1 | INT1S0 | INT0S1 | INT0S0 |
| INTC0 | — | AUTOADF | INT1F | INT0F | AUTOADE | INT1E | INT0E | EMI |
| INTC1 | INT2F | MF1F | MF0F | ADF | INT2E | MF1E | MF0E | ADE |
| INTC2 | SPIF | TB1F | TB0F | INT3F | SPIE | TB1E | TB0E | INT3E |
| INTC3 | — | — | ADAVGF | URF | — | — | ADAVGE | URE |
| MF10 | — | — | CTMAF | CTMPF | — | — | CTMAE | CTMPE |
| MF11 | — | DEF | PTMAF | PTMPF | — | DEE | PTMAE | PTMPE |

Interrupt Register List

• INTEG Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|--------|--------|--------|--------|--------|--------|--------|--------|
| Name | INT3S1 | INT3S0 | INT2S1 | INT2S0 | INT1S1 | INT1S0 | INT0S1 | INT0S0 |
| R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W |
| POR | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Bit 7~6 **INT3S1~INT3S0**: Interrupt edge control for INT3 pin

00: Disable
01: Rising edge
10: Falling edge
11: Rising and falling edges

Bit 5~4 **INT2S1~INT2S0**: Interrupt edge control for INT2 pin

00: Disable
01: Rising edge
10: Falling edge
11: Rising and falling edges

Bit 3~2 **INT1S1~INT1S0**: Interrupt edge control for INT1 pin

00: Disable
01: Rising edge
10: Falling edge
11: Rising and falling edges

Bit 1~0 **INT0S1~INT0S0**: Interrupt edge control for INT0 pin

00: Disable
01: Rising edge
10: Falling edge
11: Rising and falling edges

• **INTC0 Register**

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

- Bit 7 Unimplemented, read as “0”
- Bit 6 **AUTOADF**: A/D Converter auto mode interrupt request flag
0: No request
1: Interrupt request
- Bit 5 **INT1F**: INT1 interrupt request flag
0: No request
1: Interrupt request
- Bit 4 **INT0F**: INT0 interrupt request flag
0: No request
1: Interrupt request
- Bit 3 **AUTOADE**: A/D Converter auto mode interrupt control
0: Disable
1: Enable
- Bit 2 **INT1E**: INT1 interrupt control
0: Disable
1: Enable
- Bit 1 **INT0E**: INT0 interrupt control
0: Disable
1: Enable
- Bit 0 **EMI**: Global interrupt control
0: Disable
1: Enable

• **INTC1 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|------|------|-----|-------|------|------|-----|
| Name | INT2F | MF1F | MF0F | ADF | INT2E | MF1E | MF0E | ADE |
| 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 **INT2F**: INT2 interrupt request flag
0: No request
1: Interrupt request
- Bit 6 **MF1F**: Multi-function 1 interrupt request flag
0: No request
1: Interrupt request
- Bit 5 **MF0F**: Multi-function 0 interrupt request flag
0: No request
1: Interrupt request
- Bit 4 **ADF**: A/D Converter interrupt request flag
0: No request
1: Interrupt request
- Bit 3 **INT2E**: INT2 interrupt control
0: Disable
1: Enable
- Bit 2 **MF1E**: Multi-function 1 interrupt control
0: Disable
1: Enable

- Bit 1 **MF0E**: Multi-function 0 interrupt control
0: Disable
1: Enable
- Bit 0 **ADE**: A/D Converter interrupt control
0: Disable
1: Enable

• INTC2 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|------|------|------|-------|------|------|------|-------|
| Name | SPIF | TB1F | TB0F | INT3F | SPIE | TB1E | TB0E | INT3E |
| 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 **SPIF**: SPI interrupt request flag
0: No request
1: Interrupt request
- Bit 6 **TB1F**: Time Base 1 interrupt request flag
0: No request
1: Interrupt request
- Bit 5 **TB0F**: Time Base 0 interrupt request flag
0: No request
1: Interrupt request
- Bit 4 **INT3F**: INT3 interrupt request flag
0: No request
1: Interrupt request
- Bit 3 **SPIE**: SPI interrupt control
0: Disable
1: Enable
- Bit 2 **TB1E**: Time Base 1 interrupt control
0: Disable
1: Enable
- Bit 1 **TB0E**: Time Base 0 interrupt control
0: Disable
1: Enable
- Bit 0 **INT3E**: INT3 interrupt control
0: Disable
1: Enable

• INT3 Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|--------|-----|---|---|--------|-----|
| Name | — | — | ADAVGF | URF | — | — | ADAVGE | URE |
| R/W | — | — | R/W | R/W | — | — | R/W | R/W |
| POR | — | — | 0 | 0 | — | — | 0 | 0 |

- Bit 7~6 Unimplemented, read as “0”
- Bit 5 **ADAVGF**: A/D Converter average interrupt request flag
0: No request
1: Interrupt request
- Bit 4 **URF**: UART interrupt request flag
0: No request
1: Interrupt request
- Bit 3~2 Unimplemented, read as “0”
- Bit 1 **ADAVGE**: A/D Converter average interrupt control
0: Disable
1: Enable

Bit 0 **URE**: UART interrupt control
 0: Disable
 1: Enable

• **MF10 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|-------|-------|---|---|-------|-------|
| Name | — | — | CTMAF | CTMPF | — | — | CTMAE | CTMPE |
| R/W | — | — | R/W | R/W | — | — | R/W | R/W |
| POR | — | — | 0 | 0 | — | — | 0 | 0 |

Bit 7~6 Unimplemented, read as “0”

Bit 5 **CTMAF**: CTM Comparator A match interrupt request flag
 0: No request
 1: Interrupt request

Note that this bit must be cleared to zero by the application program when the interrupt is serviced.

Bit 4 **CTMPF**: CTM Comparator P match interrupt request flag
 0: No request
 1: Interrupt request

Note that this bit must be cleared to zero by the application program when the interrupt is serviced.

Bit 3~2 Unimplemented, read as “0”

Bit 1 **CTMAE**: CTM Comparator A match interrupt control
 0: Disable
 1: Enable

Bit 0 **CTMPE**: CTM Comparator P match interrupt control
 0: Disable
 1: Enable

• **MF11 Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|-----|-------|-------|---|-----|-------|-------|
| Name | — | DEF | PTMAF | PTMPF | — | DEE | PTMAE | PTMPE |
| R/W | — | R/W | R/W | R/W | — | R/W | R/W | R/W |
| POR | — | 0 | 0 | 0 | — | 0 | 0 | 0 |

Bit 7 Unimplemented, read as “0”

Bit 6 **DEF**: Data EEPROM interrupt request flag
 0: No request
 1: Interrupt request

Note that this bit must be cleared to zero by the application program when the interrupt is serviced.

Bit 5 **PTMAF**: PTM Comparator A match interrupt request flag
 0: No request
 1: Interrupt request

Note that this bit must be cleared to zero by the application program when the interrupt is serviced.

Bit 4 **PTMPF**: PTM Comparator P match interrupt request flag
 0: No request
 1: Interrupt request

Note that this bit must be cleared to zero by the application program when the interrupt is serviced.

Bit 3 Unimplemented, read as “0”

Bit 2 **DEE**: Data EEPROM interrupt control
 0: Disable
 1: Enable

| | |
|-------|---|
| Bit 1 | PTMAE: PTM Comparator A match interrupt control 0: Disable 1: Enable |
| Bit 0 | PTMPE: PTM Comparator P match interrupt control 0: Disable 1: Enable |

Interrupt Operation

When the conditions for an interrupt event occur, such as a TM Comparator P or Comparator A or A/D conversion completion, etc, the relevant interrupt request flag will be set. Whether the request flag actually generates a program jump to the relevant interrupt vector is determined by the condition of the interrupt enable bit. If the enable bit is set high then the program will jump to its relevant vector; if the enable bit is zero then although the interrupt request flag is set an actual interrupt will not be generated and the program will not jump to the relevant interrupt vector. The global interrupt enable bit, if cleared to zero, will disable all interrupts.

When an interrupt is generated, the Program Counter, which stores the address of the next instruction to be executed, will be transferred onto the stack. The Program Counter will then be loaded with a new address which will be the value of the corresponding interrupt vector. The microcontroller will then fetch its next instruction from this interrupt vector. The instruction at this vector will usually be a JMP which will jump to another section of program which is known as the interrupt service routine. Here is located the code to control the appropriate interrupt. The interrupt service routine must be terminated with a RETI, which retrieves the original Program Counter address from the stack and allows the microcontroller to continue with normal execution at the point where the interrupt occurred.

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

If an interrupt requires immediate servicing while the program is already in another interrupt service routine, the EMI bit should be set after entering the routine, to allow interrupt nesting. If the stack is full, the interrupt request will not be acknowledged, even if the related interrupt is enabled, until the Stack Pointer is decremented. If immediate service is desired, the stack must be prevented from becoming full. In case of simultaneous requests, the Interrupt Structure diagram shows the priority that is applied. All of the interrupt request flags when set will wake up the device if it is in SLEEP or IDLE Mode, however to prevent a wake-up from occurring the corresponding flag should be set before the device is in SLEEP or IDLE Mode.

External Interrupts

The external interrupts are controlled by signal transitions on the pins INT0~INT3. An external interrupt request will take place when the external interrupt request flags, INT0F~INT3F, are set, which will occur when a transition, whose type is chosen by the edge select bits, appears on the external interrupt pins. To allow the program to branch to its respective interrupt vector address, the global interrupt enable bit, EMI, and respective external interrupt enable bit, INT0E~INT3E, must first be set. Additionally the correct interrupt edge type must be selected using the INTEG register to enable the external interrupt function and to choose the trigger edge type. As the external interrupt pins are pin-shared with I/O pins, they can only be configured as external interrupt pins if their external interrupt enable bit in the corresponding interrupt register has been set and the external interrupt pin is selected by the corresponding pin-shared function selection bits. The pin must also be setup as an input by setting the corresponding bit in the port control register. When the interrupt

is enabled, the stack is not full and the correct transition type appears on the external interrupt pin, a subroutine call to the external interrupt vector, will take place. When the interrupt is serviced, the external interrupt request flags, INT0F~INT3F, will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts. Note that any pull-high resistor selections on the external interrupt pins will remain valid even if the pin is used as an external interrupt input.

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

A/D Converter Interrupt

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

A/D Converter Auto Mode Interrupt

An A/D Converter Auto Mode Interrupt request will take place when the A/D Converter Auto Mode Interrupt request flag, AUTOADF, is set, which occurs when the number of data to be written into the buffer reaches the value defined by WRNUM[4:0] bits. To allow the program to branch to its interrupt vector address, the global interrupt enable bit, EMI, and A/D Converter Auto Mode Interrupt enable bit, AUTOADE, must first be set. When the interrupt is enabled, the stack is not full and the condition described above occurs, a subroutine call to the A/D Converter Auto Mode Interrupt vector, will take place. When the interrupt is serviced, the A/D Converter Auto Mode Interrupt flag, AUTOADF, will be automatically cleared. The EMI bit will also be automatically cleared to disable other interrupts.

A/D Converter Average Interrupt

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

Multi-function Interrupts

Within the device there are two Multi-function interrupts. Unlike the other independent interrupts, these interrupts have no independent source, but rather are formed from other existing interrupt sources, namely the TM interrupts and EEPROM interrupt.

A Multi-function interrupt request will take place when any of the Multi-function interrupt request flags MFnF is set. The Multi-function interrupt flags will be set when any of their included functions generate an interrupt request flag. To allow the program to branch to its respective interrupt vector address, when the Multi-function interrupt is enabled and the stack is not full, and one of the interrupts contained within each of Multi-function interrupt occurs, a subroutine call to the relevant

Multi-function interrupt vector will take place. When the interrupt is serviced, the related Multi-Function request flag will be automatically reset and the EMI bit will be automatically cleared to disable other interrupts.

However, it must be noted that, although the Multi-function Interrupt request flags will be automatically reset when the interrupt is serviced, the request flags from the original source of the Multi-function interrupts will not be automatically reset and must be manually reset by the application program.

Timer Module Interrupts

The Compact and Periodic type TMs each has two interrupts, one comes from the comparator A match situation and the other comes from the comparator P match situation. All of the TM interrupts are contained within the Multi-function Interrupts. For all of the TM Types there are two interrupt request flags and two interrupt enable bits. A TM interrupt request will take place when any of the TM request flags are set, a situation which occurs when a TM comparator P or A match situation happens.

To allow the program to branch to their respective interrupt vector addresses, the global interrupt enable bit, EMI, respective TM Interrupt enable bit, and relevant Multi-function Interrupt enable bit, MFnE, must first be set. When the interrupt is enabled, the stack is not full and a TM comparator match situation occurs, a subroutine call to the relevant Multi-function Interrupt vector location, will take place. When the TM interrupt is serviced, the EMI bit will be automatically cleared to disable other interrupts. However, only the related MFnF flag will be automatically cleared. As the TM interrupt request flags will not be automatically cleared, they have to be cleared by the application program.

EEPROM Interrupt

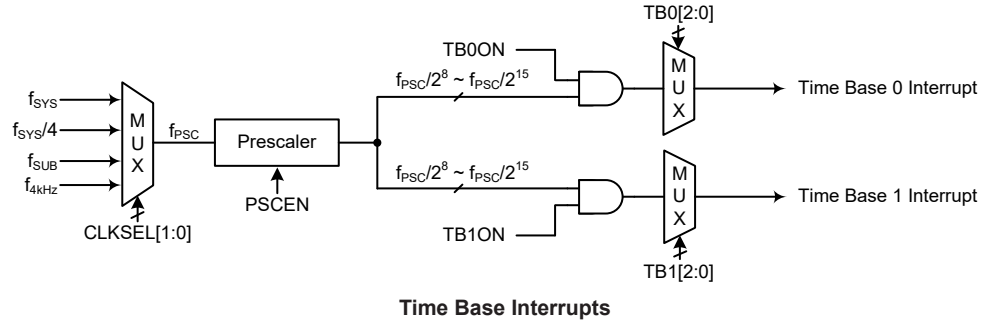
The EEPROM Interrupt is contained within the Multi-function Interrupt. An EEPROM Interrupt request will take place when the EEPROM Interrupt request flag, DEF, is set, which occurs when an EEPROM erase or write cycle ends. To allow the program to branch to its interrupt vector address, the global interrupt enable bit, EMI, EEPROM Interrupt enable bit, DEE, and associated Multi-function interrupt enable bit must first be set. When the interrupt is enabled, the stack is not full and an EEPROM erase or write cycle ends, a subroutine call to the Multi-function Interrupt vector will take place. When the EEPROM Interrupt is serviced, the EMI bit will be automatically cleared to disable other interrupts. However, only the Multi-function interrupt request flag will be automatically cleared. As the DEF flag will not be automatically cleared, it has to be cleared by the application program.

Time Base Interrupts

The function of the Time Base Interrupts is to provide regular time signals in the form of an internal interrupt. They are controlled by the overflow signals from their respective timer functions. When these happens their respective interrupt request flags, TB0F or TB1F will be set. To allow the program to branch to their respective interrupt vector addresses, the global interrupt enable bit, EMI and Time Base enable bits, TB0E or TB1E, must first be set. When the interrupt is enabled, the stack is not full and the Time Base overflows, a subroutine call to their respective vector locations will take place. When the interrupt is serviced, the respective interrupt request flag, TB0F or TB1F, will be automatically cleared, the EMI bit will also be automatically cleared to disable other interrupts.

The purpose of the Time Base Interrupts is to provide an interrupt signal at fixed time periods. Its clock source, f_{PSC} , originates from the internal clock source f_{SYS} , $f_{SYS}/4$, f_{SUB} or f_{4kHz} and then passes through a divider, the division ratio of which is selected by programming the appropriate bits in

the TB0C and TB1C registers to obtain longer interrupt periods whose value ranges. The clock source that generate f_{PSC} , which in turn controls the Time Base interrupt period, is selected using the CLKSEL[1:0] bits in the PSCR register.



• PSCR Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|---|---|---|---|---|-------|---------|---------|
| Name | — | — | — | — | — | PSCEN | CLKSEL1 | CLKSEL0 |
| R/W | — | — | — | — | — | R/W | R/W | R/W |
| POR | — | — | — | — | — | 0 | 0 | 0 |

Bit 7~3 Unimplemented, read as “0”

Bit 2 **PSCEN**: Prescaler clock enable control
0: Disable
1: Enable

The PSCEN bit is the Prescaler clock enable or disable control bit. When the Prescaler clock is disabled, it can reduce extra power consumption.

Bit 1~0 **CLKSEL1~CLKSEL0**: Prescaler clock source f_{PSC} selection
00: f_{SYS}
01: $f_{SYS}/4$
10: f_{SUB}
11: f_{4kHz}

• TB0C Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|---|---|---|---|------|------|------|
| Name | TB0ON | — | — | — | — | TB02 | TB01 | TB00 |
| R/W | R/W | — | — | — | — | R/W | R/W | R/W |
| POR | 0 | — | — | — | — | 0 | 0 | 0 |

Bit 7 **TB0ON**: Time Base 0 Enable Control
0: Disable
1: Enable

Bit 6~3 Unimplemented, read as “0”

Bit 2~0 **TB02~TB00**: Time Base 0 time-out period selection
000: $2^8/f_{PSC}$
001: $2^9/f_{PSC}$
010: $2^{10}/f_{PSC}$
011: $2^{11}/f_{PSC}$
100: $2^{12}/f_{PSC}$
101: $2^{13}/f_{PSC}$
110: $2^{14}/f_{PSC}$
111: $2^{15}/f_{PSC}$

• TB1C Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|-------|---|---|---|---|------|------|------|
| Name | TB1ON | — | — | — | — | TB12 | TB11 | TB10 |
| R/W | R/W | — | — | — | — | R/W | R/W | R/W |
| POR | 0 | — | — | — | — | 0 | 0 | 0 |

Bit 7 **TB1ON**: Time Base 1 Enable Control

0: Disable

1: Enable

Bit 6~3 Unimplemented, read as “0”

Bit 2~0 **TB12~TB10**: Time Base 1 time-out period selection

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

001: $2^9/f_{PSC}$

010: $2^{10}/f_{PSC}$

011: $2^{11}/f_{PSC}$

100: $2^{12}/f_{PSC}$

101: $2^{13}/f_{PSC}$

110: $2^{14}/f_{PSC}$

111: $2^{15}/f_{PSC}$

SPI Interrupt

The serial peripheral Interface interrupt, also known as the SPI interrupt, will take place when the SPI interrupt request flag, SPIF, is set, which occurs when a byte of data has been received or transmitted by the SPI interface. To allow the program to branch to its interrupt vector address, the global interrupt enable bit, EMI, and the serial peripheral interface interrupt enable bit, SPIE, must first be set. When the interrupt is enabled, the stack is not full and a byte of data has been transmitted or received by the SPI interface, a subroutine call to the SPI interrupt vector, will take place. When the interrupt is serviced, the serial peripheral interface interrupt flag, SPIF, will be automatically cleared. The EMI bit will also be automatically cleared to disable other interrupts.

UART Interrupt

The UART Interrupt is controlled by several UART transfer conditions. When one of these conditions occurs, an interrupt pulse will be generated to get the attention of the microcontroller. These conditions are a transmitter data register empty, transmitter idle, receiver reaching FIFO trigger level, receiver overrun, address detect and an RX/TX pin wake-up. To allow the program to branch to its interrupt vector address, the global interrupt enable bit, EMI, and UART Interrupt enable bit, URE, must first be set. When the interrupt is enabled, the stack is not full and any of the conditions described above occurs, a subroutine call to the corresponding UART Interrupt vector, will take place. When the interrupt is serviced, the UART Interrupt flag, URF, will be automatically cleared. The EMI bit will also be automatically cleared to disable other interrupts. However, the USR register flags will only be cleared when certain actions are taken by the UART, the details of which are given in the UART section.

Interrupt Wake-up Function

Each of the interrupt functions has the capability of waking up the microcontroller when in the SLEEP or IDLE Mode. A wake-up is generated when an interrupt request flag changes from low to high and is independent of whether the interrupt is enabled or not. Therefore, even though the device is in the SLEEP or IDLE Mode and its system oscillator stopped, situations such as external edge transitions on the external interrupt pins may cause their respective interrupt flag to be set high and consequently generate an interrupt. Care must therefore be taken if spurious wake-up situations are to be avoided. If an interrupt wake-up function is to be disabled then the corresponding interrupt

request flag should be set high before the device enters the SLEEP or IDLE Mode. The interrupt enable bits have no effect on the interrupt wake-up function.

Programming Considerations

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

Where a certain interrupt is contained within a Multi-function interrupt, then when the interrupt service routine is executed, as only the Multi-function interrupt request flags, MFnF, will be automatically cleared, the individual request flag for the function needs to be cleared by the application program.

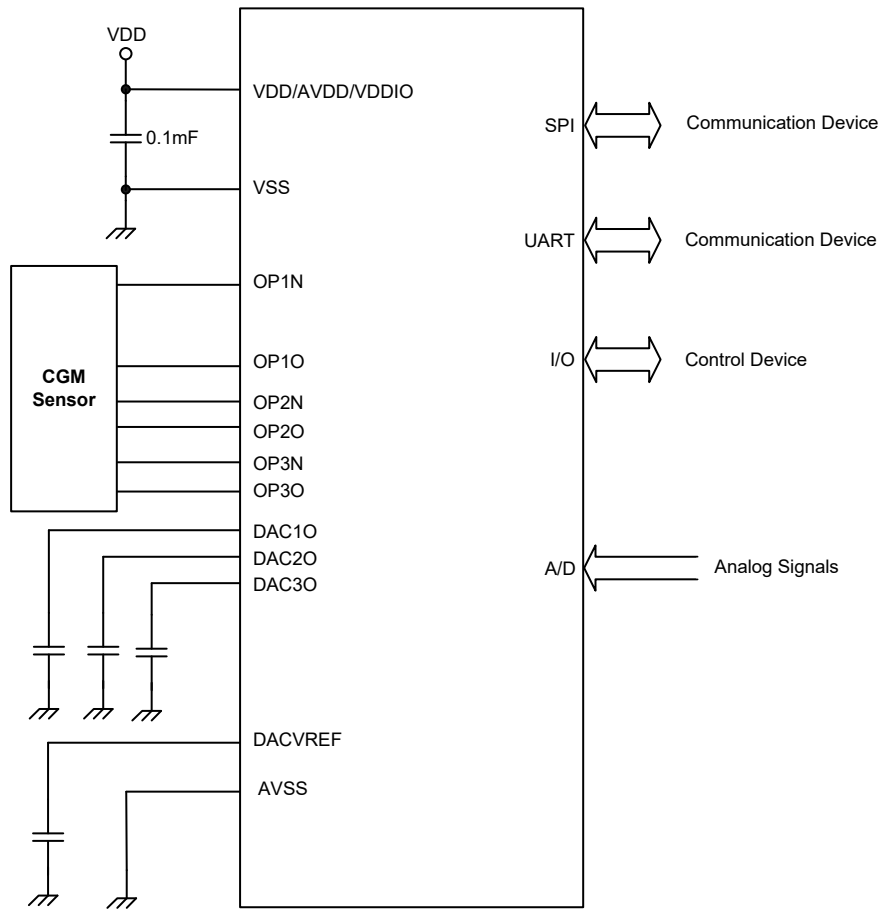
It is recommended that programs do not use the “CALL” instruction within the interrupt service subroutine. Interrupts often occur in an unpredictable manner or need to be serviced immediately. If only one stack is left and the interrupt is not well controlled, the original control sequence will be damaged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

Application Circuits



Instruction Set

Introduction

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

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

Instruction Timing

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

Moving and Transferring Data

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

Arithmetic Operations

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

Logical and Rotate Operation

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

Branches and Control Transfer

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

Bit Operations

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

Table Read Operations

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

Other Operations

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

Instruction Set Summary

The instructions related to the data memory access in the following table can be used when the desired data memory is located in Data Memory sector 0.

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, SC |
| ADDM A,[m] | Add ACC to Data Memory | 1 ^{Note} | Z, C, AC, OV, SC |
| ADD A,x | Add immediate data to ACC | 1 | Z, C, AC, OV, SC |
| ADC A,[m] | Add Data Memory to ACC with Carry | 1 | Z, C, AC, OV, SC |
| ADCM A,[m] | Add ACC to Data memory with Carry | 1 ^{Note} | Z, C, AC, OV, SC |
| SUB A,x | Subtract immediate data from the ACC | 1 | Z, C, AC, OV, SC, CZ |
| SUB A,[m] | Subtract Data Memory from ACC | 1 | Z, C, AC, OV, SC, CZ |
| SUBM A,[m] | Subtract Data Memory from ACC with result in Data Memory | 1 ^{Note} | Z, C, AC, OV, SC, CZ |
| SBC A,x | Subtract immediate data from ACC with Carry | 1 | Z, C, AC, OV, SC, CZ |
| SBC A,[m] | Subtract Data Memory from ACC with Carry | 1 | Z, C, AC, OV, SC, CZ |
| SBCM A,[m] | Subtract Data Memory from ACC with Carry, result in Data Memory | 1 ^{Note} | Z, C, AC, OV, SC, CZ |
| 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] | Skip if Data Memory is not 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) to TBLH and Data Memory | 2 ^{Note} | None |
| TABRDL [m] | Read table (last page) to TBLH and Data Memory | 2 ^{Note} | None |
| ITABRD [m] | Increment table pointer TBLP first and Read table (specific page) to TBLH and Data Memory | 2 ^{Note} | None |
| ITABRDL [m] | Increment table pointer TBLP first and 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.

Extended Instruction Set

The extended instructions are used to support the full range address access for the data memory. When the accessed data memory is located in any data memory sector except sector 0, the extended instruction can be used to directly access the data memory instead of using the indirect addressing access. This can not only reduce the use of Flash memory space but also improve the CPU execution efficiency.

| Mnemonic | Description | Cycles | Flag Affected |
|----------------------------------|---|-------------------|----------------------|
| Arithmetic | | | |
| LADD A,[m] | Add Data Memory to ACC | 2 | Z, C, AC, OV, SC |
| LADDM A,[m] | Add ACC to Data Memory | 2 ^{Note} | Z, C, AC, OV, SC |
| LADC A,[m] | Add Data Memory to ACC with Carry | 2 | Z, C, AC, OV, SC |
| LADCM A,[m] | Add ACC to Data memory with Carry | 2 ^{Note} | Z, C, AC, OV, SC |
| LSUB A,[m] | Subtract Data Memory from ACC | 2 | Z, C, AC, OV, SC, CZ |
| LSUBM A,[m] | Subtract Data Memory from ACC with result in Data Memory | 2 ^{Note} | Z, C, AC, OV, SC, CZ |
| LSBC A,[m] | Subtract Data Memory from ACC with Carry | 2 | Z, C, AC, OV, SC, CZ |
| LSBCM A,[m] | Subtract Data Memory from ACC with Carry, result in Data Memory | 2 ^{Note} | Z, C, AC, OV, SC, CZ |
| LDAA [m] | Decimal adjust ACC for Addition with result in Data Memory | 2 ^{Note} | C |
| Logic Operation | | | |
| LAND A,[m] | Logical AND Data Memory to ACC | 2 | Z |
| LOR A,[m] | Logical OR Data Memory to ACC | 2 | Z |
| LXOR A,[m] | Logical XOR Data Memory to ACC | 2 | Z |
| LANDM A,[m] | Logical AND ACC to Data Memory | 2 ^{Note} | Z |
| LORM A,[m] | Logical OR ACC to Data Memory | 2 ^{Note} | Z |
| LXORM A,[m] | Logical XOR ACC to Data Memory | 2 ^{Note} | Z |
| LCPL [m] | Complement Data Memory | 2 ^{Note} | Z |
| LCPLA [m] | Complement Data Memory with result in ACC | 2 | Z |
| Increment & Decrement | | | |
| LINCA [m] | Increment Data Memory with result in ACC | 2 | Z |
| LINC [m] | Increment Data Memory | 2 ^{Note} | Z |
| LDECA [m] | Decrement Data Memory with result in ACC | 2 | Z |
| LDEC [m] | Decrement Data Memory | 2 ^{Note} | Z |
| Rotate | | | |
| LRRA [m] | Rotate Data Memory right with result in ACC | 2 | None |
| LRR [m] | Rotate Data Memory right | 2 ^{Note} | None |
| LRRCA [m] | Rotate Data Memory right through Carry with result in ACC | 2 | C |
| LRRC [m] | Rotate Data Memory right through Carry | 2 ^{Note} | C |
| LRLA [m] | Rotate Data Memory left with result in ACC | 2 | None |
| LRL [m] | Rotate Data Memory left | 2 ^{Note} | None |
| LRLCA [m] | Rotate Data Memory left through Carry with result in ACC | 2 | C |
| LRLC [m] | Rotate Data Memory left through Carry | 2 ^{Note} | C |
| Data Move | | | |
| LMOV A,[m] | Move Data Memory to ACC | 2 | None |
| LMOV [m],A | Move ACC to Data Memory | 2 ^{Note} | None |
| Bit Operation | | | |
| LCLR [m].i | Clear bit of Data Memory | 2 ^{Note} | None |
| LSET [m].i | Set bit of Data Memory | 2 ^{Note} | None |

| Mnemonic | Description | Cycles | Flag Affected |
|----------------------|---|-------------------|---------------|
| Branch | | | |
| LSZ [m] | Skip if Data Memory is zero | 2 ^{Note} | None |
| LSZA [m] | Skip if Data Memory is zero with data movement to ACC | 2 ^{Note} | None |
| LSNZ [m] | Skip if Data Memory is not zero | 2 ^{Note} | None |
| LSZ [m].i | Skip if bit i of Data Memory is zero | 2 ^{Note} | None |
| LSNZ [m].i | Skip if bit i of Data Memory is not zero | 2 ^{Note} | None |
| LSIZ [m] | Skip if increment Data Memory is zero | 2 ^{Note} | None |
| LSIZ [m] | Skip if decrement Data Memory is zero | 2 ^{Note} | None |
| LSIZA [m] | Skip if increment Data Memory is zero with result in ACC | 2 ^{Note} | None |
| LSIZA [m] | Skip if decrement Data Memory is zero with result in ACC | 2 ^{Note} | None |
| Table Read | | | |
| LTABRD [m] | Read table (specific page) to TBLH and Data Memory | 3 ^{Note} | None |
| LTABRDL [m] | Read table (last page) to TBLH and Data Memory | 3 ^{Note} | None |
| LITABRD [m] | Increment table pointer TBLP first and Read table (specific page) to TBLH and Data Memory | 3 ^{Note} | None |
| LITABRDL [m] | Increment table pointer TBLP first and Read table (last page) to TBLH and Data Memory | 3 ^{Note} | None |
| Miscellaneous | | | |
| LCLR [m] | Clear Data Memory | 2 ^{Note} | None |
| LSET [m] | Set Data Memory | 2 ^{Note} | None |
| LSWAP [m] | Swap nibbles of Data Memory | 2 ^{Note} | None |
| LSWAPA [m] | Swap nibbles of Data Memory with result in ACC | 2 | None |

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

2. Any extended instruction which changes the contents of the PCL register will also require three 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 | $ACC \leftarrow ACC + [m] + C$ |
| Affected flag(s) | OV, Z, AC, C, SC |
| ADCM A,[m] | Add ACC to Data Memory with Carry |
| Description | The contents of the specified Data Memory, Accumulator and the carry flag are added. The result is stored in the specified Data Memory. |
| Operation | $[m] \leftarrow ACC + [m] + C$ |
| Affected flag(s) | OV, Z, AC, C, SC |
| ADD A,[m] | Add Data Memory to ACC |
| Description | The contents of the specified Data Memory and the Accumulator are added. The result is stored in the Accumulator. |
| Operation | $ACC \leftarrow ACC + [m]$ |
| Affected flag(s) | OV, Z, AC, C, SC |
| ADD A,x | Add immediate data to ACC |
| Description | The contents of the Accumulator and the specified immediate data are added. The result is stored in the Accumulator. |
| Operation | $ACC \leftarrow ACC + x$ |
| Affected flag(s) | OV, Z, AC, C, SC |
| ADDM A,[m] | Add ACC to Data Memory |
| Description | The contents of the specified Data Memory and the Accumulator are added. The result is stored in the specified Data Memory. |
| Operation | $[m] \leftarrow ACC + [m]$ |
| Affected flag(s) | OV, Z, AC, C, SC |
| AND A,[m] | Logical AND Data Memory to ACC |
| Description | Data in the Accumulator and the specified Data Memory perform a bitwise logical AND operation. The result is stored in the Accumulator. |
| Operation | $ACC \leftarrow ACC \text{ "AND" } [m]$ |
| Affected flag(s) | Z |
| AND A,x | Logical AND immediate data to ACC |
| Description | Data in the Accumulator and the specified immediate data perform a bit wise logical AND operation. The result is stored in the Accumulator. |
| Operation | $ACC \leftarrow ACC \text{ "AND" } x$ |
| Affected flag(s) | Z |

| | |
|-------------------|---|
| ANDM A,[m] | Logical AND ACC to Data Memory |
| Description | Data in the specified Data Memory and the Accumulator perform a bitwise logical AND operation. The result is stored in the Data Memory. |
| Operation | $[m] \leftarrow \text{ACC} \text{ "AND" } [m]$ |
| Affected flag(s) | Z |
| CALL addr | Subroutine call |
| Description | Unconditionally calls a subroutine at the specified address. The Program Counter then increments by 1 to obtain the address of the next instruction which is then pushed onto the stack. The specified address is then loaded and the program continues execution from this new address. As this instruction requires an additional operation, it is a two cycle instruction. |
| Operation | Stack \leftarrow Program Counter + 1 Program Counter \leftarrow addr |
| Affected flag(s) | None |
| CLR [m] | Clear Data Memory |
| Description | Each bit of the specified Data Memory is cleared to 0. |
| Operation | $[m] \leftarrow 00H$ |
| Affected flag(s) | None |
| CLR [m].i | Clear bit of Data Memory |
| Description | Bit i of the specified Data Memory is cleared to 0. |
| Operation | $[m].i \leftarrow 0$ |
| Affected flag(s) | None |
| CLR WDT | Clear Watchdog Timer |
| Description | The TO, PDF flags and the WDT are all cleared. |
| Operation | WDT cleared $TO \leftarrow 0$ $PDF \leftarrow 0$ |
| Affected flag(s) | TO, PDF |
| CPL [m] | Complement Data Memory |
| Description | Each bit of the specified Data Memory is logically complemented (1's complement). Bits which previously contained a 1 are changed to 0 and vice versa. |
| Operation | $[m] \leftarrow [m]$ |
| 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 | $\text{ACC} \leftarrow [m]$ |
| Affected flag(s) | Z |

| | |
|------------------|--|
| DAA [m] | Decimal-Adjust ACC for addition with result in Data Memory |
| Description | Convert the contents of the Accumulator value to a BCD (Binary Coded Decimal) value resulting from the previous addition of two BCD variables. If the low nibble is greater than 9 or if AC flag is set, then a value of 6 will be added to the low nibble. Otherwise the low nibble remains unchanged. If the high nibble is greater than 9 or if the C flag is set, then a value of 6 will be added to the high nibble. Essentially, the decimal conversion is performed by adding 00H, 06H, 60H or 66H depending on the Accumulator and flag conditions. Only the C flag may be affected by this instruction which indicates that if the original BCD sum is greater than 100, it allows multiple precision decimal addition. |
| Operation | [m] ← ACC + 00H or [m] ← ACC + 06H or [m] ← ACC + 60H or [m] ← ACC + 66H |
| Affected flag(s) | C |
| DEC [m] | Decrement Data Memory |
| Description | Data in the specified Data Memory is decremented by 1. |
| Operation | [m] ← [m] – 1 |
| Affected flag(s) | Z |
| DECA [m] | Decrement Data Memory with result in ACC |
| Description | Data in the specified Data Memory is decremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged. |
| Operation | ACC ← [m] – 1 |
| Affected flag(s) | Z |
| HALT | Enter power down mode |
| Description | This instruction stops the program execution and turns off the system clock. The contents of the Data Memory and registers are retained. The WDT and prescaler are cleared. The power down flag PDF is set and the WDT time-out flag TO is cleared. |
| Operation | TO ← 0 PDF ← 1 |
| Affected flag(s) | TO, PDF |
| INC [m] | Increment Data Memory |
| Description | Data in the specified Data Memory is incremented by 1. |
| Operation | [m] ← [m] + 1 |
| Affected flag(s) | Z |
| INCA [m] | Increment Data Memory with result in ACC |
| Description | Data in the specified Data Memory is incremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged. |
| Operation | ACC ← [m] + 1 |
| Affected flag(s) | Z |

| | |
|------------------|--|
| JMP addr | Jump unconditionally |
| Description | The contents of the Program Counter are replaced with the specified address. Program execution then continues from this new address. As this requires the insertion of a dummy instruction while the new address is loaded, it is a two cycle instruction. |
| Operation | Program Counter \leftarrow addr |
| Affected flag(s) | None |
| MOV A,[m] | Move Data Memory to ACC |
| Description | The contents of the specified Data Memory are copied to the Accumulator. |
| Operation | ACC \leftarrow [m] |
| Affected flag(s) | None |
| MOV A,x | Move immediate data to ACC |
| Description | The immediate data specified is loaded into the Accumulator. |
| Operation | ACC \leftarrow x |
| Affected flag(s) | None |
| MOV [m],A | Move ACC to Data Memory |
| Description | The contents of the Accumulator are copied to the specified Data Memory. |
| Operation | [m] \leftarrow ACC |
| Affected flag(s) | None |
| NOP | No operation |
| Description | No operation is performed. Execution continues with the next instruction. |
| Operation | No operation |
| Affected flag(s) | None |
| OR A,[m] | Logical OR Data Memory to ACC |
| Description | Data in the Accumulator and the specified Data Memory perform a bitwise logical OR operation. The result is stored in the Accumulator. |
| Operation | ACC \leftarrow ACC “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 \leftarrow 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] \leftarrow 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 \leftarrow 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 \leftarrow Stack ACC \leftarrow 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 \leftarrow Stack EMI \leftarrow 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) \leftarrow [m].i; (i=0~6) [m].0 \leftarrow [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) \leftarrow [m].i; (i=0~6) ACC.0 \leftarrow [m].7 |
| Affected flag(s) | None |
| RLC [m] | Rotate Data Memory left through Carry |
| Description | The contents of the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into bit 0. |
| Operation | [m].(i+1) \leftarrow [m].i; (i=0~6) [m].0 \leftarrow C C \leftarrow [m].7 |
| Affected flag(s) | C |

| | |
|------------------|---|
| RLCA [m] | Rotate Data Memory left through Carry with result in ACC |
| Description | Data in the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into the bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | $ACC.(i+1) \leftarrow [m].i; (i=0\sim6)$ $ACC.0 \leftarrow C$ $C \leftarrow [m].7$ |
| Affected flag(s) | C |
| RR [m] | Rotate Data Memory right |
| Description | The contents of the specified Data Memory are rotated right by 1 bit with bit 0 rotated into bit 7. |
| Operation | $[m].i \leftarrow [m].(i+1); (i=0\sim6)$ $[m].7 \leftarrow [m].0$ |
| Affected flag(s) | None |
| RRA [m] | Rotate Data Memory right with result in ACC |
| Description | Data in the specified Data Memory is rotated right by 1 bit with bit 0 rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | $ACC.i \leftarrow [m].(i+1); (i=0\sim6)$ $ACC.7 \leftarrow [m].0$ |
| Affected flag(s) | None |
| RRC [m] | Rotate Data Memory right through Carry |
| Description | The contents of the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. |
| Operation | $[m].i \leftarrow [m].(i+1); (i=0\sim6)$ $[m].7 \leftarrow C$ $C \leftarrow [m].0$ |
| Affected flag(s) | C |
| RRCA [m] | Rotate Data Memory right through Carry with result in ACC |
| Description | Data in the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | $ACC.i \leftarrow [m].(i+1); (i=0\sim6)$ $ACC.7 \leftarrow C$ $C \leftarrow [m].0$ |
| Affected flag(s) | C |
| SBC A,[m] | Subtract Data Memory from ACC with Carry |
| Description | The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | $ACC \leftarrow ACC - [m] - C$ |
| Affected flag(s) | OV, Z, AC, C, SC, CZ |

| | |
|-------------------|---|
| SBC A, x | Subtract immediate data from ACC with Carry |
| Description | The immediate data and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | $ACC \leftarrow ACC - [m] - C$ |
| Affected flag(s) | OV, Z, AC, C, SC, CZ |
| SBCM A,[m] | Subtract Data Memory from ACC with Carry and result in Data Memory |
| Description | The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | $[m] \leftarrow ACC - [m] - C$ |
| Affected flag(s) | OV, Z, AC, C, SC, CZ |
| SDZ [m] | Skip if decrement Data Memory is 0 |
| Description | The contents of the specified Data Memory are first decremented by 1. If the result is 0 the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | $[m] \leftarrow [m] - 1$ Skip if $[m]=0$ |
| Affected flag(s) | None |
| SDZA [m] | Skip if decrement Data Memory is zero with result in ACC |
| Description | The contents of the specified Data Memory are first decremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction. |
| Operation | $ACC \leftarrow [m] - 1$ Skip if $ACC=0$ |
| Affected flag(s) | None |
| SET [m] | Set Data Memory |
| Description | Each bit of the specified Data Memory is set to 1. |
| Operation | $[m] \leftarrow FFH$ |
| Affected flag(s) | None |
| SET [m].i | Set bit of Data Memory |
| Description | Bit i of the specified Data Memory is set to 1. |
| Operation | $[m].i \leftarrow 1$ |
| Affected flag(s) | None |

| | |
|-------------------|--|
| SIZ [m] | Skip if increment Data Memory is 0 |
| Description | The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | $[m] \leftarrow [m] + 1$ Skip if $[m]=0$ |
| Affected flag(s) | None |
| SIZA [m] | Skip if increment Data Memory is zero with result in ACC |
| Description | The contents of the specified Data Memory are first incremented by 1. If the result is 0, the following instruction is skipped. The result is stored in the Accumulator but the specified Data Memory contents remain unchanged. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | $ACC \leftarrow [m] + 1$ Skip if $ACC=0$ |
| Affected flag(s) | None |
| SNZ [m].i | Skip if bit i of Data Memory is not 0 |
| Description | If bit i of the specified Data Memory is not 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is 0 the program proceeds with the following instruction. |
| Operation | Skip if $[m].i \neq 0$ |
| Affected flag(s) | None |
| SNZ [m] | Skip if Data Memory is not 0 |
| Description | The contents of the specified Data Memory are read out and then written back to the specified Data Memory again. If 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 $[m] \neq 0$ |
| Affected flag(s) | None |
| SUB A,[m] | Subtract Data Memory from ACC |
| Description | The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | $ACC \leftarrow ACC - [m]$ |
| Affected flag(s) | OV, Z, AC, C, SC, CZ |
| SUBM A,[m] | Subtract Data Memory from ACC with result in Data Memory |
| Description | The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | $[m] \leftarrow ACC - [m]$ |
| Affected flag(s) | OV, Z, AC, C, SC, CZ |

| | |
|------------------|---|
| SUB A,x | Subtract immediate data from ACC |
| Description | The immediate data specified by the code is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | $ACC \leftarrow ACC - x$ |
| Affected flag(s) | OV, Z, AC, C, SC, CZ |
| SWAP [m] | Swap nibbles of Data Memory |
| Description | The low-order and high-order nibbles of the specified Data Memory are interchanged. |
| Operation | $[m].3 \sim [m].0 \leftrightarrow [m].7 \sim [m].4$ |
| Affected flag(s) | None |
| SWAPA [m] | Swap nibbles of Data Memory with result in ACC |
| Description | The low-order and high-order nibbles of the specified Data Memory are interchanged. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged. |
| Operation | $ACC.3 \sim ACC.0 \leftarrow [m].7 \sim [m].4$ $ACC.7 \sim ACC.4 \leftarrow [m].3 \sim [m].0$ |
| Affected flag(s) | None |
| SZ [m] | Skip if Data Memory is 0 |
| Description | The contents of the specified Data Memory are read out and then written back 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 | $ACC \leftarrow [m]$ Skip if $[m]=0$ |
| Affected flag(s) | None |
| SZ [m].i | Skip if bit i of Data Memory is 0 |
| Description | If bit i of the specified Data Memory is 0, the following instruction is skipped. As this requires the insertion of a dummy instruction while the next instruction is fetched, it is a two cycle instruction. If the result is not 0, the program proceeds with the following instruction. |
| Operation | Skip if $[m].i=0$ |
| Affected flag(s) | None |

| | |
|--------------------|--|
| TABRD [m] | Read table (specific page) to TBLH and Data Memory |
| Description | The low byte of the program code (specific page) addressed by the table pointer (TBLP and TBHP) is moved to the specified Data Memory and the high byte moved to TBLH. |
| Operation | [m] ← program code (low byte) TBLH ← program code (high byte) |
| Affected flag(s) | None |
| TABRDL [m] | Read table (last page) to TBLH and Data Memory |
| Description | The low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH. |
| Operation | [m] ← program code (low byte) TBLH ← program code (high byte) |
| Affected flag(s) | None |
| ITABRD [m] | Increment table pointer low byte first and read table (specific page) to TBLH and Data Memory |
| Description | Increment table pointer low byte, TBLP, first and then the program code (specific page) addressed by the table pointer (TBHP and TBLP) is moved to the specified Data Memory and the high byte moved to TBLH. |
| Operation | [m] ← program code (low byte) TBLH ← program code (high byte) |
| Affected flag(s) | None |
| ITABRDL [m] | Increment table pointer low byte first and read table (last page) to TBLH and Data Memory |
| Description | Increment table pointer low byte, TBLP, first and then the low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH. |
| Operation | [m] ← program code (low byte) TBLH ← program code (high byte) |
| Affected flag(s) | None |
| XOR A,[m] | Logical XOR Data Memory to ACC |
| Description | Data in the Accumulator and the specified Data Memory perform a bitwise logical XOR operation. The result is stored in the Accumulator. |
| Operation | ACC ← ACC “XOR” [m] |
| Affected flag(s) | Z |
| XORM A,[m] | Logical XOR ACC to Data Memory |
| Description | Data in the specified Data Memory and the Accumulator perform a bitwise logical XOR operation. The result is stored in the Data Memory. |
| Operation | [m] ← ACC “XOR” [m] |
| Affected flag(s) | Z |
| XOR A,x | Logical XOR immediate data to ACC |
| Description | Data in the Accumulator and the specified immediate data perform a bitwise logical XOR operation. The result is stored in the Accumulator. |
| Operation | ACC ← ACC “XOR” x |
| Affected flag(s) | Z |

Extended Instruction Definition

The extended instructions are used to directly access the data stored in any data memory sections.

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

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

Operation $ACC \leftarrow ACC + [m] + C$

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

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

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

Operation $[m] \leftarrow ACC + [m] + C$

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

LADD A,[m] Add Data Memory to ACC

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

Operation $ACC \leftarrow ACC + [m]$

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

LADDM A,[m] Add ACC to Data Memory

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

Operation $[m] \leftarrow ACC + [m]$

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

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

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

Operation $ACC \leftarrow ACC \text{ "AND" } [m]$

Affected flag(s) Z

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

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

Operation $[m] \leftarrow ACC \text{ "AND" } [m]$

Affected flag(s) Z

| | |
|-------------------|--|
| LCLR [m] | Clear Data Memory |
| Description | Each bit of the specified Data Memory is cleared to 0. |
| Operation | $[m] \leftarrow 00H$ |
| Affected flag(s) | None |
| LCLR [m].i | Clear bit of Data Memory |
| Description | Bit i of the specified Data Memory is cleared to 0. |
| Operation | $[m].i \leftarrow 0$ |
| Affected flag(s) | None |
| LCPL [m] | Complement Data Memory |
| Description | Each bit of the specified Data Memory is logically complemented (1's complement). Bits which previously contained a 1 are changed to 0 and vice versa. |
| Operation | $[m] \leftarrow [m]$ |
| Affected flag(s) | Z |
| LCPLA [m] | Complement Data Memory with result in ACC |
| Description | Each bit of the specified Data Memory is logically complemented (1's complement). Bits which previously contained a 1 are changed to 0 and vice versa. The complemented result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | $ACC \leftarrow [m]$ |
| Affected flag(s) | Z |
| LDAA [m] | Decimal-Adjust ACC for addition with result in Data Memory |
| Description | Convert the contents of the Accumulator value to a BCD (Binary Coded Decimal) value resulting from the previous addition of two BCD variables. If the low nibble is greater than 9 or if AC flag is set, then a value of 6 will be added to the low nibble. Otherwise the low nibble remains unchanged. If the high nibble is greater than 9 or if the C flag is set, then a value of 6 will be added to the high nibble. Essentially, the decimal conversion is performed by adding 00H, 06H, 60H or 66H depending on the Accumulator and flag conditions. Only the C flag may be affected by this instruction which indicates that if the original BCD sum is greater than 100, it allows multiple precision decimal addition. |
| Operation | $[m] \leftarrow ACC + 00H$ or $[m] \leftarrow ACC + 06H$ or $[m] \leftarrow ACC + 60H$ or $[m] \leftarrow ACC + 66H$ |
| Affected flag(s) | C |
| LDEC [m] | Decrement Data Memory |
| Description | Data in the specified Data Memory is decremented by 1. |
| Operation | $[m] \leftarrow [m] - 1$ |
| Affected flag(s) | Z |

| | |
|-------------------|---|
| LDECA [m] | Decrement Data Memory with result in ACC |
| Description | Data in the specified Data Memory is decremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged. |
| Operation | $ACC \leftarrow [m] - 1$ |
| Affected flag(s) | Z |
| LINC [m] | Increment Data Memory |
| Description | Data in the specified Data Memory is incremented by 1. |
| Operation | $[m] \leftarrow [m] + 1$ |
| Affected flag(s) | Z |
| LINCA [m] | Increment Data Memory with result in ACC |
| Description | Data in the specified Data Memory is incremented by 1. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged. |
| Operation | $ACC \leftarrow [m] + 1$ |
| Affected flag(s) | Z |
| LMOV A,[m] | Move Data Memory to ACC |
| Description | The contents of the specified Data Memory are copied to the Accumulator. |
| Operation | $ACC \leftarrow [m]$ |
| Affected flag(s) | None |
| LMOV [m],A | Move ACC to Data Memory |
| Description | The contents of the Accumulator are copied to the specified Data Memory. |
| Operation | $[m] \leftarrow ACC$ |
| Affected flag(s) | None |
| LOR A,[m] | Logical OR Data Memory to ACC |
| Description | Data in the Accumulator and the specified Data Memory perform a bitwise logical OR operation. The result is stored in the Accumulator. |
| Operation | $ACC \leftarrow ACC \text{ "OR" } [m]$ |
| Affected flag(s) | Z |
| LORM 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 | $[m] \leftarrow ACC \text{ "OR" } [m]$ |
| Affected flag(s) | Z |

| | |
|------------------|---|
| LRL [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 | $[m].(i+1) \leftarrow [m].i; (i=0\sim6)$ $[m].0 \leftarrow [m].7$ |
| Affected flag(s) | None |
| LRLA [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 | $ACC.(i+1) \leftarrow [m].i; (i=0\sim6)$ $ACC.0 \leftarrow [m].7$ |
| Affected flag(s) | None |
| LRLC [m] | Rotate Data Memory left through Carry |
| Description | The contents of the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into bit 0. |
| Operation | $[m].(i+1) \leftarrow [m].i; (i=0\sim6)$ $[m].0 \leftarrow C$ $C \leftarrow [m].7$ |
| Affected flag(s) | C |
| LRLCA [m] | Rotate Data Memory left through Carry with result in ACC |
| Description | Data in the specified Data Memory and the carry flag are rotated left by 1 bit. Bit 7 replaces the Carry bit and the original carry flag is rotated into the bit 0. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | $ACC.(i+1) \leftarrow [m].i; (i=0\sim6)$ $ACC.0 \leftarrow C$ $C \leftarrow [m].7$ |
| Affected flag(s) | C |
| LRR [m] | Rotate Data Memory right |
| Description | The contents of the specified Data Memory are rotated right by 1 bit with bit 0 rotated into bit 7. |
| Operation | $[m].i \leftarrow [m].(i+1); (i=0\sim6)$ $[m].7 \leftarrow [m].0$ |
| Affected flag(s) | None |

| | |
|--------------------|---|
| LRRA [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. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | $ACC.i \leftarrow [m].(i+1); (i=0\sim6)$ $ACC.7 \leftarrow [m].0$ |
| Affected flag(s) | None |
| LRRC [m] | Rotate Data Memory right through Carry |
| Description | The contents of the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. |
| Operation | $[m].i \leftarrow [m].(i+1); (i=0\sim6)$ $[m].7 \leftarrow C$ $C \leftarrow [m].0$ |
| Affected flag(s) | C |
| LRRCA [m] | Rotate Data Memory right through Carry with result in ACC |
| Description | Data in the specified Data Memory and the carry flag are rotated right by 1 bit. Bit 0 replaces the Carry bit and the original carry flag is rotated into bit 7. The rotated result is stored in the Accumulator and the contents of the Data Memory remain unchanged. |
| Operation | $ACC.i \leftarrow [m].(i+1); (i=0\sim6)$ $ACC.7 \leftarrow C$ $C \leftarrow [m].0$ |
| Affected flag(s) | C |
| LSBC A,[m] | Subtract Data Memory from ACC with Carry |
| Description | The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | $ACC \leftarrow ACC - [m] - C$ |
| Affected flag(s) | OV, Z, AC, C, SC, CZ |
| LSBCM A,[m] | Subtract Data Memory from ACC with Carry and result in Data Memory |
| Description | The contents of the specified Data Memory and the complement of the carry flag are subtracted from the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | $[m] \leftarrow ACC - [m] - C$ |
| Affected flag(s) | OV, Z, AC, C, SC, CZ |

| | |
|-------------------|---|
| LSDZ [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 three cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | $[m] \leftarrow [m] - 1$ Skip if $[m]=0$ |
| Affected flag(s) | None |
| LSDZA [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 three cycle instruction. If the result is not 0, the program proceeds with the following instruction. |
| Operation | $ACC \leftarrow [m] - 1$ Skip if $ACC=0$ |
| Affected flag(s) | None |
| LSET [m] | Set Data Memory |
| Description | Each bit of the specified Data Memory is set to 1. |
| Operation | $[m] \leftarrow FFH$ |
| Affected flag(s) | None |
| LSET [m].i | Set bit of Data Memory |
| Description | Bit i of the specified Data Memory is set to 1. |
| Operation | $[m].i \leftarrow 1$ |
| Affected flag(s) | None |
| LSIZ [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 three cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | $[m] \leftarrow [m] + 1$ Skip if $[m]=0$ |
| Affected flag(s) | None |

| | |
|--------------------|--|
| LSIZA [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 three cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | $ACC \leftarrow [m] + 1$ Skip if ACC=0 |
| Affected flag(s) | None |
| LSNZ [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 three cycle instruction. If the result is 0 the program proceeds with the following instruction. |
| Operation | Skip if $[m].i \neq 0$ |
| Affected flag(s) | None |
| LSNZ [m] | Skip if Data Memory is not 0 |
| Description | The contents of the specified Data Memory are read out and then written to the specified Data Memory again. If the content 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 three cycle instruction. If the result is 0 the program proceeds with the following instruction. |
| Operation | Skip if $[m] \neq 0$ |
| Affected flag(s) | None |
| LSUB A,[m] | Subtract Data Memory from ACC |
| Description | The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Accumulator. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | $ACC \leftarrow ACC - [m]$ |
| Affected flag(s) | OV, Z, AC, C, SC, CZ |
| LSUBM A,[m] | Subtract Data Memory from ACC with result in Data Memory |
| Description | The specified Data Memory is subtracted from the contents of the Accumulator. The result is stored in the Data Memory. Note that if the result of subtraction is negative, the C flag will be cleared to 0, otherwise if the result is positive or zero, the C flag will be set to 1. |
| Operation | $[m] \leftarrow ACC - [m]$ |
| Affected flag(s) | OV, Z, AC, C, SC, CZ |

| | |
|-------------------|--|
| LSWAP [m] | Swap nibbles of Data Memory |
| Description | The low-order and high-order nibbles of the specified Data Memory are interchanged. |
| Operation | $[m].3 \sim [m].0 \leftrightarrow [m].7 \sim [m].4$ |
| Affected flag(s) | None |
| LSWAPA [m] | Swap nibbles of Data Memory with result in ACC |
| Description | The low-order and high-order nibbles of the specified Data Memory are interchanged. The result is stored in the Accumulator. The contents of the Data Memory remain unchanged. |
| Operation | $ACC.3 \sim ACC.0 \leftarrow [m].7 \sim [m].4$ $ACC.7 \sim ACC.4 \leftarrow [m].3 \sim [m].0$ |
| Affected flag(s) | None |
| LSZ [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 three cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | Skip if $[m]=0$ |
| Affected flag(s) | None |
| LSZA [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 three cycle instruction. If the result is not 0 the program proceeds with the following instruction. |
| Operation | $ACC \leftarrow [m]$ Skip if $[m]=0$ |
| Affected flag(s) | None |
| LSZ [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 three 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 |
| LTABRD [m] | Read table (specific page) to TBLH and Data Memory |
| Description | The low byte of the program code (specific page) addressed by the table pointer (TBHP and TBLP) is moved to the specified Data Memory and the high byte moved to TBLH. |
| Operation | $[m] \leftarrow \text{program code (low byte)}$ $TBLH \leftarrow \text{program code (high byte)}$ |
| Affected flag(s) | None |

| | |
|---------------------|--|
| LTABRDL [m] | Read table (last page) to TBLH and Data Memory |
| Description | The low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH. |
| Operation | [m] ← program code (low byte) TBLH ← program code (high byte) |
| Affected flag(s) | None |
| LITABRD [m] | Increment table pointer low byte first and read table (specific page) to TBLH and Data Memory |
| Description | Increment table pointer low byte, TBLP, first and then the program code (specific page) addressed by the table pointer (TBHP and TBLP) is moved to the specified Data Memory and the high byte moved to TBLH. |
| Operation | [m] ← program code (low byte) TBLH ← program code (high byte) |
| Affected flag(s) | None |
| LITABRDL [m] | Increment table pointer low byte first and read table (last page) to TBLH and Data Memory |
| Description | Increment table pointer low byte, TBLP, first and then the low byte of the program code (last page) addressed by the table pointer (TBLP) is moved to the specified Data Memory and the high byte moved to TBLH. |
| Operation | [m] ← program code (low byte) TBLH ← program code (high byte) |
| Affected flag(s) | None |
| LXOR 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 |
| LXORM 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 |

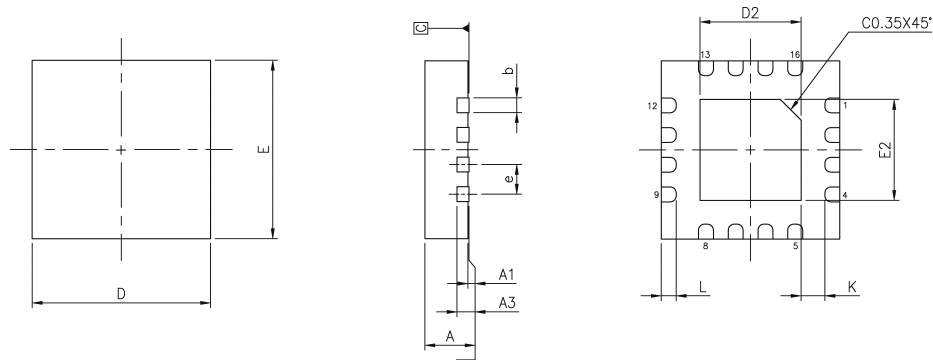
Package Information

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

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

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

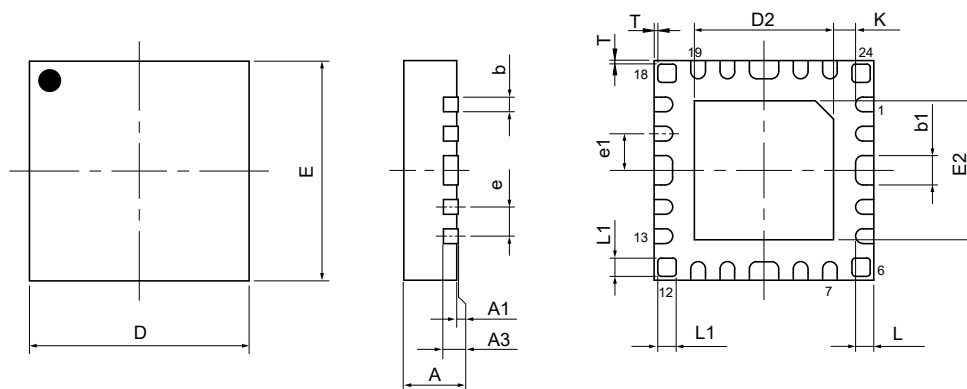
SAW Type 16-pin QFN (3mm×3mm×0.75mm, FP0.25mm) Outline Dimensions



| Symbol | Dimensions in inch | | |
|--------|--------------------|-------|-------|
| | Min. | Nom. | Max. |
| A | 0.028 | 0.030 | 0.031 |
| A1 | 0.000 | 0.001 | 0.002 |
| A3 | 0.008 REF | | |
| b | 0.007 | 0.010 | 0.012 |
| D | 0.118 BSC | | |
| E | 0.118 BSC | | |
| e | 0.020 BSC | | |
| D2 | 0.063 | — | 0.071 |
| E2 | 0.063 | — | 0.071 |
| L | 0.008 | — | 0.016 |
| K | 0.008 | — | — |

| Symbol | Dimensions in mm | | |
|--------|------------------|------|------|
| | Min. | Nom. | Max. |
| A | 0.70 | 0.75 | 0.80 |
| A1 | 0.00 | 0.02 | 0.05 |
| A3 | 0.203 REF | | |
| b | 0.18 | 0.25 | 0.30 |
| D | 3.00 BSC | | |
| E | 3.00 BSC | | |
| e | 0.50 BSC | | |
| D2 | 1.60 | — | 1.80 |
| E2 | 1.60 | — | 1.80 |
| L | 0.20 | — | 0.40 |
| K | 0.20 | — | — |

SAW Type 24-pin QFN (3mm×3mm×0.55mm) Outline Dimensions



| Symbol | Dimensions in inch | | |
|--------|--------------------|-------|-------|
| | Min. | Nom. | Max. |
| A | 0.020 | 0.022 | 0.024 |
| A1 | 0.000 | 0.001 | 0.002 |
| A3 | 0.006 REF | | |
| b | 0.006 | 0.008 | 0.010 |
| b1 | 0.014 | 0.016 | 0.018 |
| D | 0.118 BSC | | |
| E | 0.118 BSC | | |
| e | 0.016 BSC | | |
| e1 | 0.020 BSC | | |
| D2 | 0.073 | — | 0.077 |
| E2 | 0.073 | — | 0.077 |
| L | 0.006 | 0.010 | 0.014 |
| L1 | 0.008 | 0.010 | 0.012 |
| K | 0.008 | — | — |
| T | 0.000 | 0.002 | 0.004 |

| Symbol | Dimensions in mm | | |
|--------|------------------|------|------|
| | Min. | Nom. | Max. |
| A | 0.50 | 0.55 | 0.60 |
| A1 | 0.00 | 0.02 | 0.05 |
| A3 | 0.15 REF | | |
| b | 0.15 | 0.20 | 0.25 |
| b1 | 0.35 | 0.40 | 0.45 |
| D | 3.00 BSC | | |
| E | 3.00 BSC | | |
| e | 0.40 BSC | | |
| e1 | 0.50 BSC | | |
| D2 | 1.85 | — | 1.95 |
| E2 | 1.85 | — | 1.95 |
| L | 0.15 | 0.25 | 0.35 |
| L1 | 0.20 | 0.25 | 0.30 |
| K | 0.20 | — | — |
| T | 0.00 | 0.05 | 0.10 |

Copyright© 2024 by HOLTEK SEMICONDUCTOR INC. All Rights Reserved.

The information provided in this document has been produced with reasonable care and attention before publication, however, HOLTEK 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. HOLTEK 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. HOLTEK disclaims all liability arising from the information and its application. In addition, HOLTEK does not recommend the use of HOLTEK's products where there is a risk of personal hazard due to malfunction or other reasons. HOLTEK hereby declares that it does not authorise the use of these products in life-saving, life-sustaining or safety critical components. Any use of HOLTEK'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 HOLTEK 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 HOLTEK (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 HOLTEK herein. HOLTEK reserves the right to revise the information described in the document at any time without prior notice. For the latest information, please contact us.