TB3162

Vectored Interrupt Controller on 8-Bit PIC® Microcontrollers

Author: *June Anthony Asistio*
Microchip Technology Inc.
Mayank Prasad
Microchip Technology Inc.

INTRODUCTION

An interrupt is a request that temporarily stops a microcontroller from running the main routine to execute another task that needs to be handled immediately. The task to be handled is called the Interrupt Service Routine (ISR).

An interrupt can be generated by a multitude of hardware or software sources. Most of the peripherals in the microcontroller can generate an interrupt signal when a certain event happens. For instance, the UART module can generate an interrupt request when the receive buffer is full, or the timer module can generate an interrupt request when the timer counter overflows. Interrupts can also be generated by external signals through a GPIO pin, or generated through the software execution. Refer to the device data sheet to find out the different ways an interrupt can be generated.

Every interrupt source sets an interrupt flag as a signal, which is then sent to the interrupt controller module. The interrupt controller module analyzes the different interrupt requests, resolves the priorities of the interrupt sources, and then sends appropriate signals and addresses to the CPU to suspend the execution of the main routine and jump to the location of the corresponding ISR.

The purpose of this technical brief is to demonstrate the configuration and functionality of the vectored interrupt controller module in implementing and handling interrupt routines.

OVERVIEW

The vectored interrupt controller module reduces the numerous peripheral interrupt request signals to a single interrupt request signal to the CPU. This module includes the following major features:

- Interrupt Vector Table (IVT) with a unique vector for each interrupt source
- Fixed and ensured interrupt latency
- Programmable base address for Interrupt Vector Table (IVT) with lock
- Two user-selectable priority levels - High priority and Low priority
- Two levels of context saving
- Interrupt state Status bits to indicate the current execution status of the CPU

The Vectored Interrupt Controller module assembles all the interrupt request signals and resolves the interrupts based on both a fixed natural order priority (i.e., determined by the Interrupt Vector Table), and a user-assigned priority (i.e., determined by the IPRx registers), thereby eliminating scanning of interrupt sources.

Note: The contents of the Interrupt Vector Table and the locations of individual bits in the PIRx/PIEx/IPRx registers are device dependent. Refer to the device data sheet for the contents of IVT and other interrupt related registers.

MODULE CONFIGURATION

To configure the vectored interrupt controller module, perform the following steps:

1. Enable the MVECEN Configuration bit in the corresponding CONFIG register. This enables the interrupt module to vector directly to ISR by using the IVT. Leaving the MVECEN Configuration bit disabled will make the interrupt controller operate in Legacy mode.
2. Enable the IVT1WAY Configuration bit in the corresponding CONFIG register (if applicable). This is required if the IVTLOCK bit needs to be cleared/set only once. Refer to the device data sheet for more information.
3. Set base address of IVT using the IVTBASE register (or leave it as default to 0x000008). This is required if the user application requires the use of multiple IVTs (bootloader applications for instance).
4. Enable user-assigned priority in interrupts by setting the IPEN bit (if applicable).
5. Enable the desired interrupt sources by setting the appropriate bits in the PIEx registers.
6. Clear the desired interrupt flags in the appropriate PIRx registers. This is to ensure that all the interrupt flags are in the Reset state before the interrupts are enabled globally.
7. Set user-assigned priorities to interrupts by setting/clearing appropriate bits in the IPRx registers (if applicable).
8. Enable interrupts globally by setting the GIEH/GIEL bits. Setting just the GIEH bit and leaving the GIEL bit cleared will enable only user-assigned high priority interrupts.

EXAMPLE 1: INITIALIZING VECTORED INTERRUPT MODULE IN PIC18(L)FxxK42

```
void INTERRUPT_Initialize (void)
{
  //MVECEN/IVT1WAY config bits need to be
  //set separately. Refer to datasheet for
  //more information

  // Enable priority in interrupts- OPTIONAL
   INTCON0bits.IPEN = 1;

  // Set IVTBASE - OPTIONAL
  // Do this only if changing IVTBASE to
  // value other than the default value of
  // 0x000008
  IVTBASEU = 0x00;
  IVTBASEH = 0x40;
  IVTBASEL = 0xF0;

  //Enable interrupts
  PIE3bits.TMR0IE = 1;
  PIE4bits.TMR1IE = 1;

  //Clear interrupt flags
  PIR3bits.TMR0IF = 0;
  PIR4bits.TMR1IF = 0;

  //Make one interrupt low priority -
  //OPTIONAL
  IPR4bits.TMR1IP = 0;

  // Enable interrupts
  INTCON0bits.GIEH = 1;
  INTCON0bits.GIEL = 1;
}
```

INTERRUPT SERVICE ROUTINE (ISR) SYNTAX

The following is the syntax to write an ISR.

EXAMPLE 2: INTERRUPT SERVICE ROUTINE (ISR) SYNTAX

```
// ISR Syntax

void __interrupt(irq(...), base(...), priority) ISR_NAME(void)
{
    // Clear appropriate interrupt flag(s)
    // Interrupt handler code follows
}
```

Inside the `__interrupt(...)` handler, the following arguments need to be provided:

- `irq(...)` argument lists the vector number of all the interrupt requests handled by the ISR. Refer to the device data sheet for a list of available interrupt requests and their vector numbers.
- `base(...)` argument specifies the base address of IVT. This is optional and needs to be included if the IVTBASE is changed from its default value or if there are multiple IVTs in the program memory.
- `priority` argument specifies the priority of the ISR and takes the values as either `high_priority` or `low_priority`. If nothing is specified, `high_priority` is assigned by default. This argument is relevant only when MVECEN = OFF and IPEN = `1`.

EXAMPLE 3: ISR Examples with MVECEN = ON for PIC18(L)F24/25K42

```
// ISRs with MVECEN=ON
// base(...) argument must be used when IVTBASE is changed from default

// ISR for TMR0 interrupt with IVTBASE=default (0x0008)
void __interrupt(irq(IRQ_TMR0)) TMR0_ISR(void)
{
    PIR3bits.TMR0IF = 0;    // Clear TMR0 interrupt flag
    // Interrupt handler code goes here
}

// Common ISR for TMR1 and CCP1 interrupts with IVTBASE=0x40F0
void __interrupt(irq(IRQ_TMR1, IRQ_CCP1), base(0x40F0)) TMR1_ISR(void)
{
    PIR4bits.TMR1IF = 0;    // Clear TMR1 interrupt flag
    PIR4bits.CCP1IF = 0;    // Clear CCP1 interrupt flag
    // Interrupt handler code goes here
}

// Default ISR for all unhandled interrupts with IVTBASE=default (0x0008)
void __interrupt(irq(default)) DEFAULT_ISR(void)
{
    // Unhandled interrupt code
}
```

Refer to the examples section of MPLAB® Xpress IDE for a working demonstration and application of vectored interrupts.

BACKWARD COMPATIBILITY

The vectored interrupt controller module is fully backward compatible with the legacy interrupt controllers available in earlier PIC16 and PIC18 devices. The backward compatibility can be established in multiple ways.

Method 1: Using Legacy ISR

Legacy ISRs using `interrupt` handler are still functional with the vectored interrupt controller module. The following code example is a demonstration.

EXAMPLE 4: ISR Examples with MVECEN = OFF for PIC18(L)F24/25K42

```
// Legacy ISR with MVECEN=OFF and IPEN=1

void interrupt INTERRUPT_InterruptManagerHigh (void)
{
    // Check for the appropriate interrupt flag
    if(INTCON0bits.GIE == 1 && PIE3bits.TMR0IE == 1 && PIR3bits.TMR0IF == 1)
    {
        PIR3bits.TMR0IF = 0;    // Clear TMR0 interrupt flag
        TMR0_ISR();             // TMR0 interrupt handler
    }
    else
    {
        //Unhandled Interrupts
    }
}

void interrupt low_priority INTERRUPT_InterruptManagerLow (void)
{
    // Check for the appropriate interrupt flag
    if(INTCON0bits.GIE == 1 && PIE4bits.TMR1IE == 1 && PIR4bits.TMR1IF == 1)
    {
        PIR4bits.TMR1IF = 0;    // Clear TMR1 interrupt flag
        TMR1_ISR();             // TMR1 interrupt handler
    }
    else
    {
        //Unhandled Interrupts
    }
}
```

Method 2: Using __interrupt Handler

Another way to implement the ISR and use the vectored interrupt controller in Legacy mode is to use the `__interrupt(...)` handler. The following example is an illustration.

EXAMPLE 5: ISR Examples with MVECEN = OFF for PIC18(L)F24/25K42

```
// Legacy ISR with MVECEN=OFF and IPEN=1

void __interrupt INTERRUPT_InterruptManagerHigh (void)
{
    // Check for the appropriate interrupt flag
    if(INTCON0bits.GIE == 1 && PIE3bits.TMR0IE == 1 && PIR3bits.TMR0IF == 1)
    {
        PIR3bits.TMR0IF = 0;    // Clear TMR0 interrupt flag
        TMR0_ISR();             // TMR0 interrupt handler
    }
    else
    {
        //Unhandled Interrupts
    }
}

void __interrupt(low_priority) INTERRUPT_InterruptManagerLow (void)
{
    // Check for the appropriate interrupt flag
    if(INTCON0bits.GIE == 1 && PIE4bits.TMR1IE == 1 && PIR4bits.TMR1IF == 1)
    {
        PIR4bits.TMR1IF = 0;    // Clear TMR1 interrupt flag
        TMR1_ISR();             // TMR1 interrupt handler
    }
    else
    {
        //Unhandled Interrupts
    }
}
```

Method 3: Using Vector Number

The most efficient way to implement the ISR in Legacy mode is to use the `__interrupt(...)` handler and use the vector number stored in the WREG to switch to the appropriate interrupt handler. The following example is an illustration.

EXAMPLE 6: ISR Example using Vector Number with MVECEN = OFF for PIC18(L)F24/25K42

```
// ISR with MVECEN=OFF and IPEN=1

void __interrupt() INTERRUPT_InterruptManagerHigh (void)
{
    uint8_t vectorID_High = WREG;

    // Switch using the appropriate vector number
    switch(vectorID_High)
    {
        case IRQ_TMR0:
            PIR3bits.TMR0IF = 0;    // Clear TMR0 interrupt flag
            TMR0_ISR();             // TMR0 interrupt handler
            break;
        case IRQ_CCP1:
            PIR4bits.CCP1IF = 0;    // Clear CCP1 interrupt flag
            CCP1_ISR();             // CCP1 interrupt handler
            break;
        default:
            //Unhandled Interrupts
            break;
    }
}

void __interrupt(low_priority) INTERRUPT_InterruptManagerLow (void)
{
    uint8_t vectorID_Low = WREG;

    // Switch using the appropriate vector number
    switch(vectorID_Low)
    {
        case IRQ_TMR1:
            PIR4bits.TMR1IF = 0;    // Clear TMR1 interrupt flag
            TMR1_ISR();             // TMR1 interrupt handler
            break;
        case IRQ_TMR3:
            PIR6bits.TMR3IF = 0;    // Clear TMR3 interrupt flag
            TMR3_ISR();             // TMR3 interrupt handler
            break;
        default:
            //Unhandled Interrupts
            break;
    }
}
```

CONCLUSION

The vectored interrupt controller module uses the Interrupt Vector Table (IVT) to uniquely determine the interrupt source and execute the appropriate ISR directly, thereby eliminating scanning of interrupt sources in the software. It assembles all the interrupt request signals and resolves the interrupts based on both a fixed natural order priority and a user-assigned priority. The operation of the vectored interrupt controller is fully backward compatible with the legacy interrupt controller module available in earlier PIC® Microcontrollers.

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

ISBN: 978-1-5224-4844-0

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Australia - Sydney
Tel: 61-2-9868-6733

China - Beijing
Tel: 86-10-8569-7000

China - Chengdu
Tel: 86-28-8665-5511

China - Chongqing
Tel: 86-23-8980-9588

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115

China - Hong Kong SAR
Tel: 852-2943-5100

China - Nanjing
Tel: 86-25-8473-2460

China - Qingdao
Tel: 86-532-8502-7355

China - Shanghai
Tel: 86-21-3326-8000

China - Shenyang
Tel: 86-24-2334-2829

China - Shenzhen
Tel: 86-755-8864-2200

China - Suzhou
Tel: 86-186-6233-1526

China - Wuhan
Tel: 86-27-5980-5300

China - Xian
Tel: 86-29-8833-7252

China - Xiamen
Tel: 86-592-2388138

China - Zhuhai
Tel: 86-756-3210040

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444

India - New Delhi
Tel: 91-11-4160-8631

India - Pune
Tel: 91-20-4121-0141

Japan - Osaka
Tel: 81-6-6152-7160

Japan - Tokyo
Tel: 81-3-6880- 3770

Korea - Daegu
Tel: 82-53-744-4301

Korea - Seoul
Tel: 82-2-554-7200

Malaysia - Kuala Lumpur
Tel: 60-3-7651-7906

Malaysia - Penang
Tel: 60-4-227-8870

Philippines - Manila
Tel: 63-2-634-9065

Singapore
Tel: 65-6334-8870

Taiwan - Hsin Chu
Tel: 886-3-577-8366

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600

Thailand - Bangkok
Tel: 66-2-694-1351

Vietnam - Ho Chi Minh
Tel: 84-28-5448-2100

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-72400

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-7288-4388

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820