Interrupts

</ENT269>

Objectives

- Contra and compare interrupts versus pooling
- Explain the purpose of the ISR
- List all the major interrupts of the PIC18
- Explain the purpose of the IVT
- Enable and disable PIC18 interrupts
- Program the PIC18 interrupt using C

Interrupt

- Whenever any device needs the microcontroller's service the device notifies it by sending an interrupt signal. Upon receiving an interrupt signal, the microcontroller stops whatever it is doing and serve the device. The program associated with the interrupt is called ISR (interrupt service routine) or interrupt handler.
- Each device can get the attention of the microcontroller based on the priority assign to it.
- Can ignore a device request for service

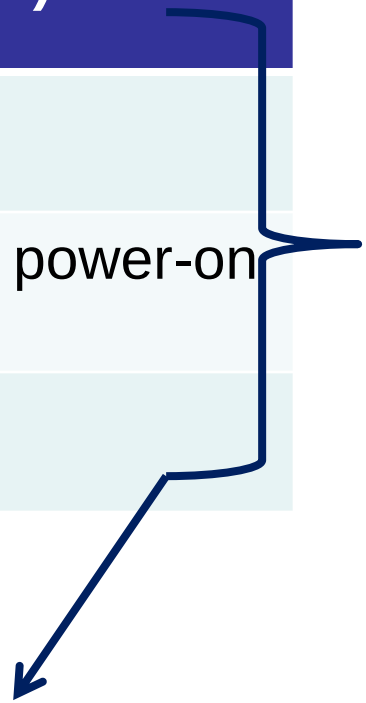
Polling

- The microcontroller continuously monitors the status of a given device; when the status condition met, it performs the service. After that, it moves on to monitor the next device until each one is service.
- Cannot assign priority because it checks all devices in a round-robin fashion.
- Cannot ignore a devices for service

Interrupt Service Routine (ISR)

Interrupt	ROM Location (hex)
Power-on-Reset	0000
High Priority Interrupt	0008 (Default upon power-on reset)
Low Priority Interrupt	0018

Interrupt Vector Table for the PIC18



Fixed Location in Memory

Sources of Interrupt

- Each Timers (Timers 0, 1, 2 and 3)
- 3 interrupts for external hardware. Pin RB0 (INT0), RB1 (INT1) and RB2 (INT2)
- 2 interrupts for serial communication (Receive and Transmit)
- The PORTB-Change interrupt (RB4-RB7)
- The ADC
- The CCP (Compare Capture Pulse-width modulation)

Enable and Disable an Interrupt

REGISTER 9-1: INTCON: INTERRUPT CONTROL REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

GIE (Global Interrupt Enable)
GIE = 0 Disables all interrupts. If GIE = 0, no interrupt is acknowledged, even if they are enabled individually.
If GIE = 1, interrupts are allowed to happen. Each interrupt source is enabled by setting the corresponding interrupt enable bit.

TMR0IE Timer0 interrupt enable
= 0 Disables Timer0 overflow interrupt
= 1 Enables Timer0 overflow interrupt

INT0IE Enables or disables external interrupt 0
= 0 Disables external interrupt 0
= 1 Enables external interrupt 0

These bits, along with the GIE, must be set high for an interrupt to be responded to. Upon activation of the interrupt, the GIE bit is cleared by the PIC18 itself to make sure another interrupt cannot interrupt the microcontroller while it is servicing the current one. At the end of the ISR, the RETFIE instruction will make GIE = 1 to allow another interrupt to come in.

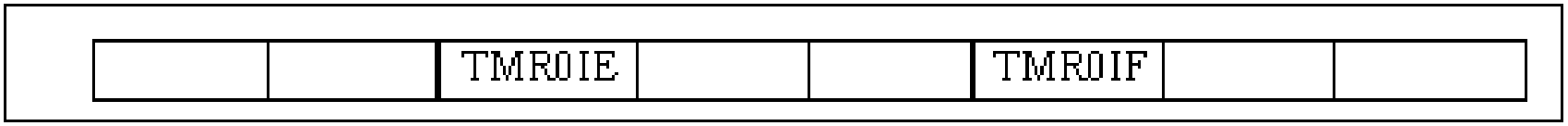
PEIE (PEripheral Interrupt Enable)
For many of the peripherals, such as Timers 1, 2, .. and the serial port, we must enable this bit in addition to the GIE bit. (See Figure 11-2.)

```
INTCONbits.GIE = 1;
INTCONbits.PEIE = 1;
```

Timer Interrupts

Interrupt	Flag Bit	Register	Enable Bit	Register
Timer0	TMR0IF	INTCON	TMR0IE	INTCON
Timer1	TMR1IF	PIR1	TMR1IE	PIE1
Timer2	TMR2IF	PIR1	TMR2IE	PIE1
Timer3	TMR3IF	PIR2	TMR3IE	PIE2

Timer Interrupt Flag Bits and Associated Registers



INTCON Register with Timer0 Interrupt Enable and Interrupt Flag

Example

Uses Timer0 and Timer1 interrupts to generate square waves on pins RB1 and RB7 respectively, while data is being transferred from PORTC to PORTD

1

```
#include <p18f4580.h>
#define myPB1bit PORTBbits.RB1
#define myPB7bit PORTBbits.RB7

void T0_ISR(void);                  // Timer0 Interrupt
void T1_ISR(void);                  // Timer1 Interrupt

#pragma interrupt chk_isr           // Used for high-priority

void chk_isr(void)
{
    if(INTCONbits.TMR0IF == 1)      // Timer0 couses interrupt?
        {
            T0_ISR();               // YES! Execute Timer0 ISR
        }
    if(PIR1bits.TMR1IF == 1)        // OR Timer1 couses interrupt?
        {
            T1_ISR();               // YES! Execute Timer1 ISR
        }
}

#pragma code My_HiPrio_Int = 0x08   // High-Priority Interrupts
void My_HiPrio_Int(void)
{
    _asm
        GOTO chk_isr
    _endasm
}
```

3

```
void T0_ISR(void)
{
    myPB1bit = ~myPB1bit;           // Toogle PORTB.1
    TMR0H = 0x35;                   // Load TH0
    TMR0L = 0x00;                   // Load TL0
    INTCONbits.TMR0IF = 0;          // Clear TF0
}

void T1_ISR(void)
{
    myPB7bit = ~myPB7bit;           // Toogle PORTB.7
    TMR1H = 0x35;                   // Load TH1
    TMR1L = 0x00;                   // Load TL1
    PIR1bits.TMR1IF = 0;            // Clear TF1
}
```

2

```
void main(void)
{
    TRISBbits.TRISB1 = 0;           // RB1 = Output
    TRISBbits.TRISB7 = 0;           // RB7 = Output
    TRISC = 255;                    // PORTC = Input
    TRISD = 0;                      // PORTD = Output
    T0CON = 0;                      // Timer0, 16-bit Mode, No Prescaler
    TMR0H = 0x35;                   // Load TH0
    TMR0L = 0x00;                   // Load TL0
    T1CON = 0x88;                   // Timer1, 16-bit Mode, No Prescaler
    TMR1H = 0x35;                   // Load TH1
    TMR1L = 0x00;                   // Load TL1
    INTCONbits.TMR0IF = 0;          // Clear TF0
    PIR1bits.TMR1IF = 0;            // Clear TF1
    INTCONbits.TMR0IE = 1;          // Enable Timer0 Interrupt
    PIE1bits.TMR1IE = 1;            // Enable Timer1 Interrupt
    T0CONbits.TMR0ON = 1;           // Turn ON Timer0
    T1CONbits.TMR1ON = 1;           // Turn ON Timer1
    INTCONbits.GIE = 1;             // Enable All Interrupts Globally
    INTCONbits.PEIE = 1;            // Enable All Peripheral Interupts
    while(1)                        // Keep Looping until Intterupts Comes
        {
            PORTD = PORTC;
        }
}
```

Initialize

C18 does not place an ISR at the interrupt Vector table automatically, we must use Assembly language instruction GOTO At the interrupt vector to transfer control to the ISR

External Hardware Interrupts

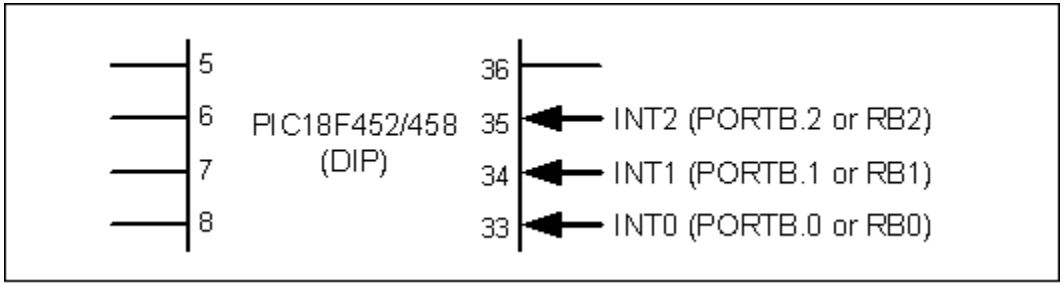
Interrupt	Flag Bit	Register	Enable Bit	Register
INT0(RB0)	INT0IF	INTCON	INT0IE	INTCON
INT1 (RB1)	INT1IF	INTCON3	INT1IE	INTCON3
INT2 (RB2)	INT2IF	INTCON3	INT2IE	INTCON3

Hardware Interrupt Flag Bits and Associated Registers

REGISTER 9-3: INTCON3: INTERRUPT CONTROL REGISTER 3

R/W-1	R/W-1	U-0	R/W-0	R/W-0	U-0	R/W-0	R/W-0
INT2IP	INT1IP	—	INT2IE	INT1IE	—	INT2IF	INT1IF
bit 7							bit 0

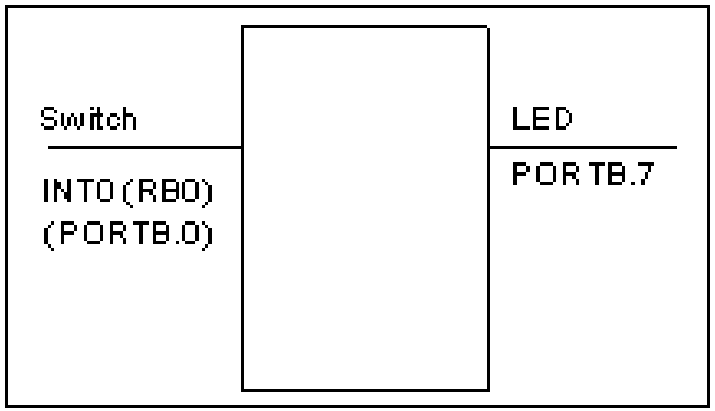
Positive-edge-triggered interrupt



PIC18 External Hardware Interrupt Pins

Example

Connect a switch to INT0 and an LED to PIN RB7. In this program, every time INT0 is activated, it toggles the LED, while at the same time data is being transferred from PORTC to PORTD



1

```c
#include <p18f4580.h>
#define mybit PORTBbits.RB7

void INT0_ISR(void);
#pragma interrupt chk_isr           // Used for high-priority

void chk_isr(void)
{
    if(INTCONbits.INT0IF == 1)      // INT0 Caused Interrupt?
        {
            INT0_ISR();             // YES! Execute INT0 Program
        }
}

#pragma code My_HiPrio_Int = 0x08   // High-Priority Interrupts
void My_HiPrio_Int(void)
{
    _asm
        GOTO chk_isr
    _endasm
}
```

2

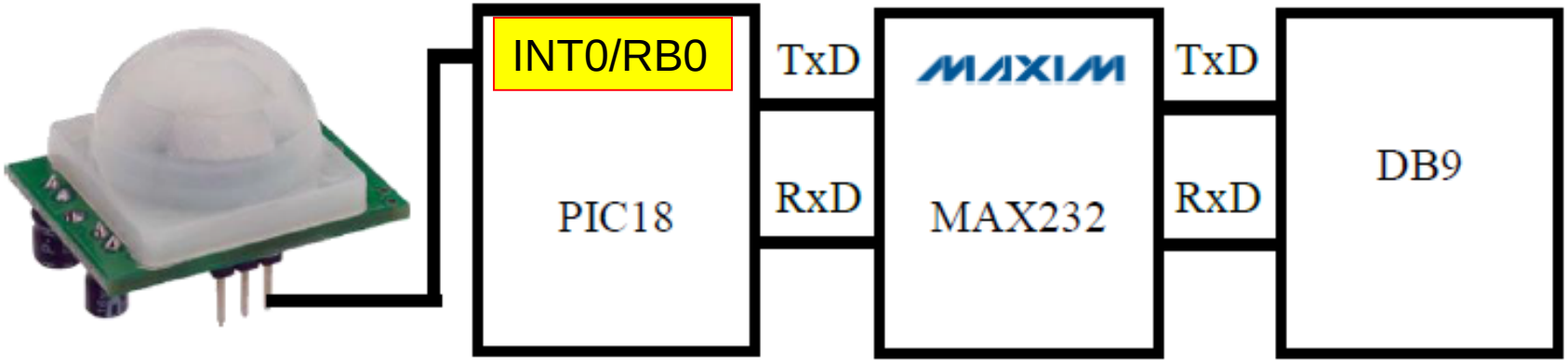
```c
void main(void)
{
    TRISBbits.TRISB7 = 0;           // RB7 = Output
    TRISBbits.TRISB0 = 1;           // INT0 = Input (Switch?)
    TRISC = 0xFF;                   // PORTC = Input
    TRISD = 0;                      // PORTD = Output
    INTCONbits.INT0IF = 0;          // Clear INT0 Flag bit
    INTCONbits.INT0IE = 1;          // Enable INT0 Interrupt
    INTCONbits.GIE = 1;             // Enable All Interrupts
    INTCONbits.PEIE = 1;            // Enable All Peripheral Interupts
    while(1)                        // Keep Looping until Intterupts Comes
        {
            PORTD = PORTC;
        }
}

void INT0_ISR(void)
{
    mybit = ~mybit;
    INTCONbits.INT0IF = 0;          // Clear INT0 Flag Bit
}
```

3

Exercise

Select the suitable SPBRG value and TXSTA configuration for a PIC18 microcontroller to set the baud rate at 9600. Then, sketch a flowchart and write a C18 program to monitor the status of a PIR sensor as shown in Figure 2. If the PIR sensor (INT0) is activated, send a message “ALRT” to the serial port continuously. But if the sensor is not activated, send a message “SAFE” to the serial port. Assume the crystal oscillator frequency is 10 MHz.



1

```
#include <p18f4580.h>

void safe(void);
void alrt(void);
void INT0_ISR(void);

#pragma interrupt chk_isr            // Used for high-priority
void chk_isr(void)
{
    if(INTCONbits.INT0IF == 1)       // INT0 Caused Interrupt?
        {
            INT0_ISR();              // YES! Execute INT0 Program
        }
}

#pragma code My_HiPrio_Int = 0x08    // High-Priority Interrupts
void My_HiPrio_Int(void)
{
    _asm
        GOTO chk_isr
    _endasm
}
```

2

```
void main(void)
{
    TRISBbits.TRISB0 = 1;            // INT0 = Input (Switch?)
    TXSTA = 0x20;                    // Low Baudrate, 8-bit
    SPBRG = 15;                      // 9600 Baudrate / XTAL = 10MHz
    TXSTAbits.TXEN = 1;              // Transmit Enabled
    RCSTAbits.SPEN = 1;              // Serial Port Enable
    INTCONbits.INT0IF = 0;           // Clear INT0 Flag bit
    INTCONbits.INT0IE = 1;           // Enable INT0 Interrupt
    INTCONbits.GIE = 1;              // Enable All Interrupts
    INTCONbits.PEIE = 1;             // Enable All Peripheral Interupts
    while(1)                         // Keep Looping until Intterupts Comes
        {
            safe();
        }
}

void INT0_ISR(void)
{
    alrt();
    INTCONbits.INT0IF = 0;           // Clear INT0 Flag Bit
}
```

3

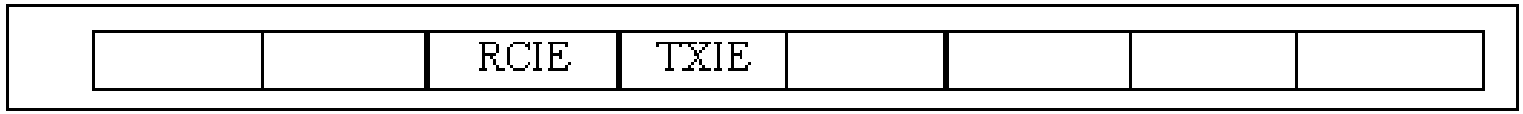
```
void safe(void)
{
    unsigned char z;
    unsigned char Msg2[] = "SAFE";
    for (z = 0; z <4; z++)
            {
                while(PIR1bits.TXIF == 0);  // Wait for Transmit
                TXREG = Msg2[z];
            }
}

void alrt(void)
{
    unsigned char z;
    unsigned char Msg1[] = "ALRT";
    for (z = 0; z <4; z++)
            {
                while(PIR1bits.TXIF == 0);  // Wait for Transmit
                TXREG = Msg1[z];
            }
}
```

Serial Communication Interrupts

Interrupt	Flag Bit	Register	Enable Bit	Register
TXIF (Trasmit)	TXIF	PIR1	TXIE	PIE1
RCIF (Receive)	RCIF	PIR1	RCIE	PIE1

Serial Port Interrupt Flag Bits and Associated Registers



PIE1 Register Bits Holding TXIE and RCIE

Example

The PIC18 gets data from PORTD and send it to TXREG continuously while incoming data from the serial port is sent to PORTB. Assume XTAL = 10MHz and Baud rate = 9600.

```c
#include <p18f4580.h>
void TX_ISR(void);
void RX_ISR(void);
#pragma interrupt chk_isr            // Used for high-priority

void chk_isr(void)
{
    if(PIR1bits.TXIF == 1)
        {
            TX_ISR();
        }
    if(PIR1bits.RCIF == 1)
        {
            RX_ISR();
        }
}

#pragma code My_HiPrio_Int = 0x08    // High-Priority Interrupts
void My_HiPrio_Int(void)
{
    _asm
        GOTO chk_isr
    _endasm
}
```

1

2

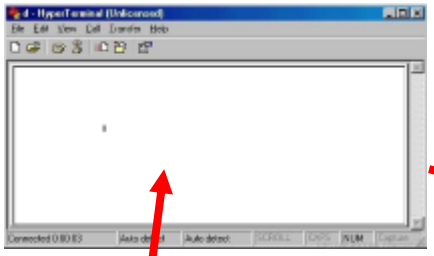
```c
void main(void)
{
    TRISD = 0xFF;                    // PORTD = Input
    TRISB = 0;                       // PORTB = Output
    TRISCbits.TRISC6 = 0;            // TX Pin = Output
    TRISCbits.TRISC7 = 1;            // RX Pin + Input
    TXSTA = 0x20;                    // Choose Low Baudrate, 8-bit
    SPBRG = 15;                      // 9600 Baudrate/XTAL = 10MHz
    RCSTAbits.CREN = 1;
    RCSTAbits.SPEN = 1;
    TXSTAbits.TXEN = 1;
    PIE1bits.RCIE = 1;               // Enable Rx Interrupt
    PIE1bits.TXIE = 1;               // Enable Tx Interrupt
    INTCONbits.GIE = 1;              // Enable All Interrupts
    INTCONbits.PEIE = 1;             // Enable All Peripheral Interupts
    while(1);                        // Keep Looping until Interrupt Occur
}

void TX_ISR(void)
{
    TXREG = PORTD;
}

void RX_ISR(void)
{
    PORTB = RCREG;
}
```

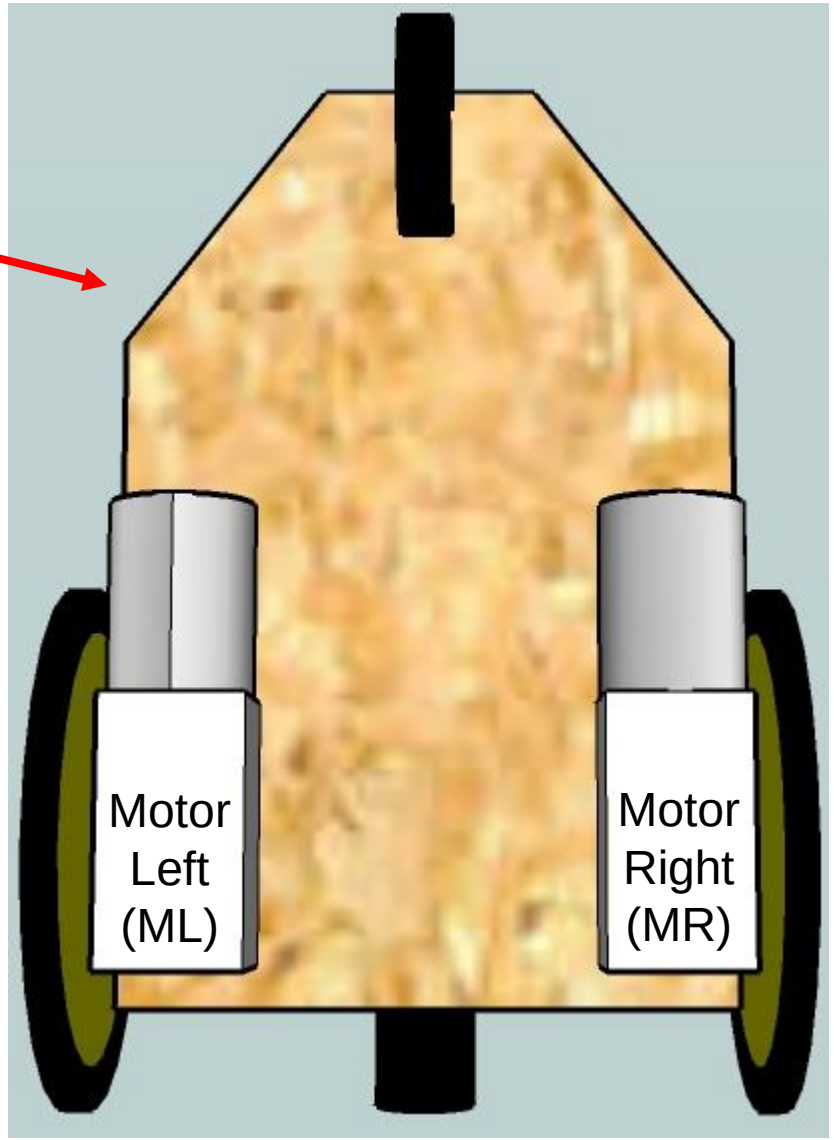
3

Exercise

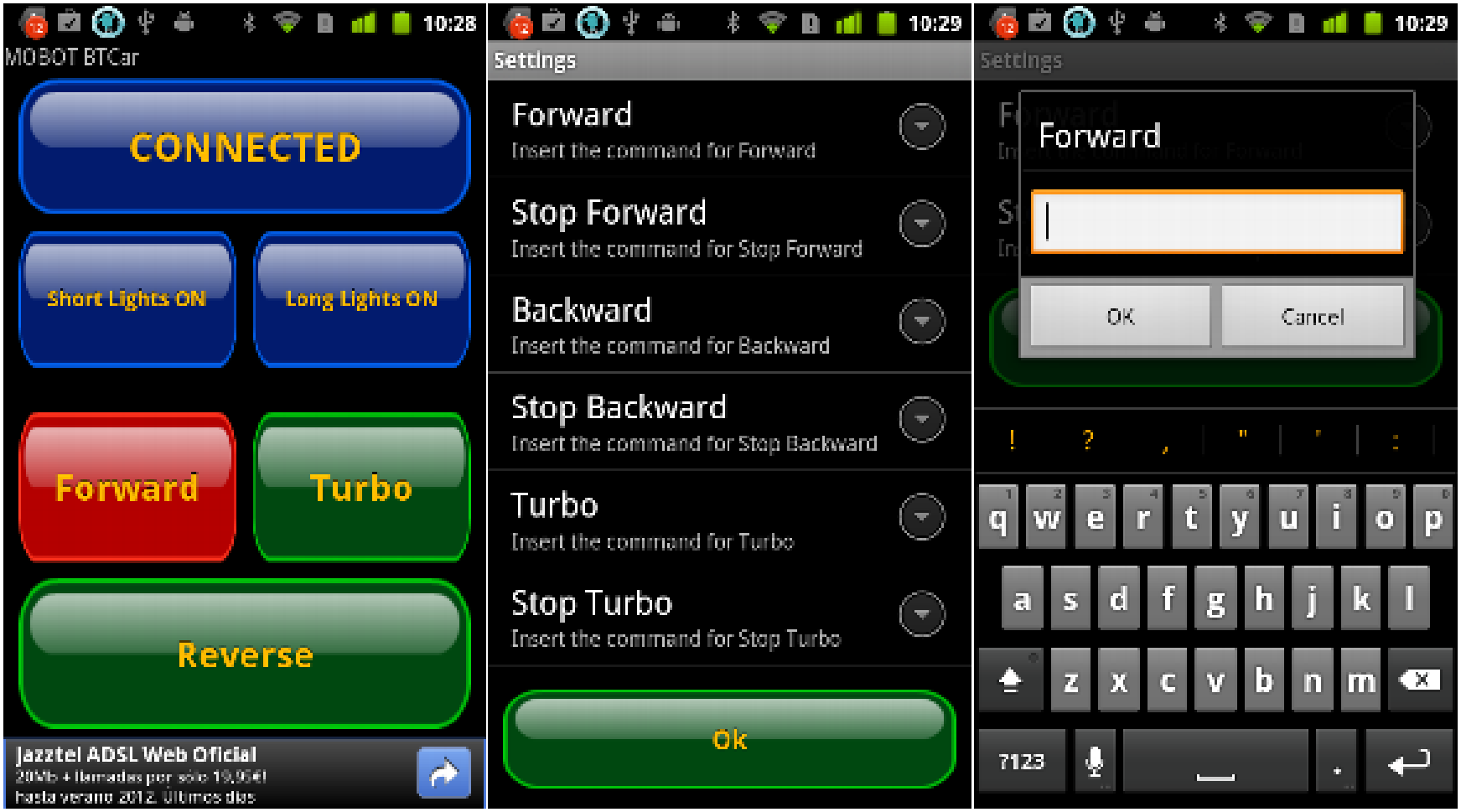


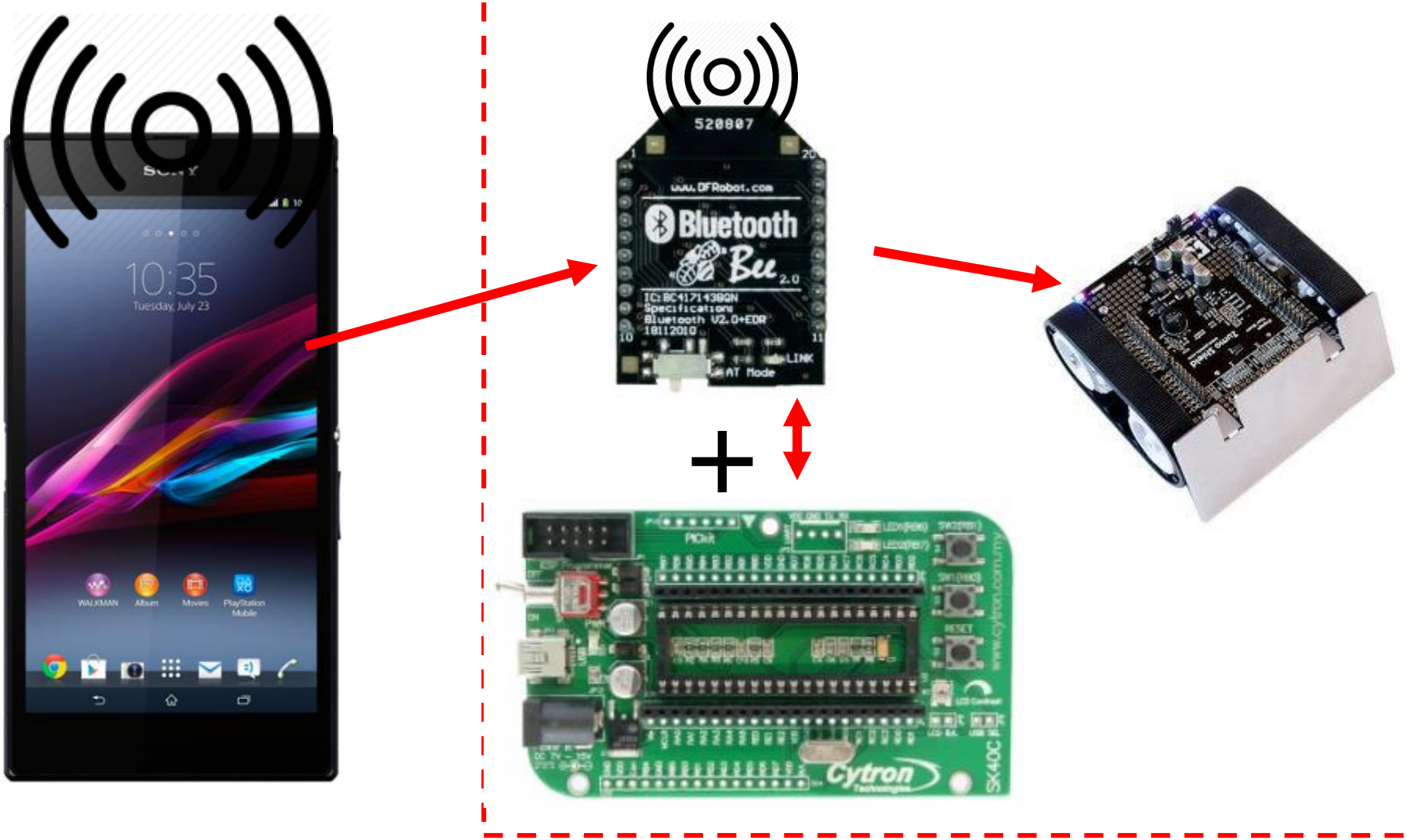
Letter	Action	ML	MR
L	Left	0	1
R	Right	1	0
S	Stop	0	0

Default = Move FORWARD (ML = 1, MR = 1)

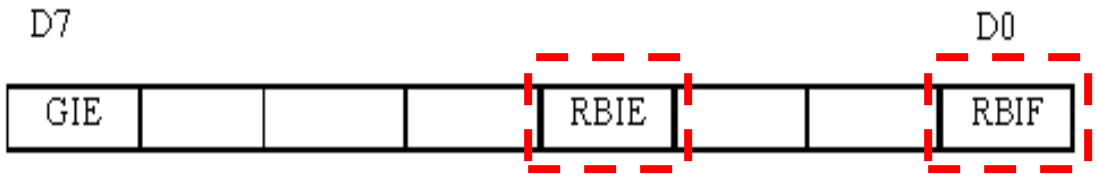


Mobot Bluetooth




10:35
Tuesday, July 23
WALKMAN
Album
Movies
PlayStation Mobile
520807
www.DFRobot.com
Bluetooth
Bee 2.0
IC:BC417143BQN
Specifications
Bluetooth V2.0+EDR
18112010
LINK
AT Mode
Cytron
SK40C

PORTB-Change Interrupt



GIE (Global Interrupt Enable)

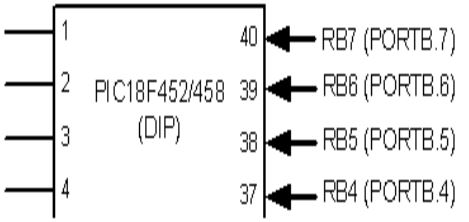
GIE = 0 Disables all interrupts. If GIE = 0, no interrupt is acknowledged, even if they are enabled individually.

If GIE = 1, interrupts are allowed to happen. Each interrupt source is enabled by setting the corresponding interrupt enable bit.

RBIE PORTB-Change Interrupt Enable
= 0 Disables PORTB-Change interrupt
= 1 Enables PORTB-Change interrupt

RBIF PORTB-Change Interrupt Flag.
= 0 None of the RB4–RB7 pins have changed state
= 1 At least one of the RB4–RB7 pins have changed state

The RBIE bit, along with the GIE, must be set high for any changes on the pins RB4–RB7 to cause an interrupt. The RB4–RB7 pins must also have been configured as input pins for this interrupt to work. In order to clear the RBIF flag we must read the pins of RB4–RB7 **and** use the instruction "BCF INTCON,RBIF".



INTCON Register

*** Use in Keypad Interfacing**

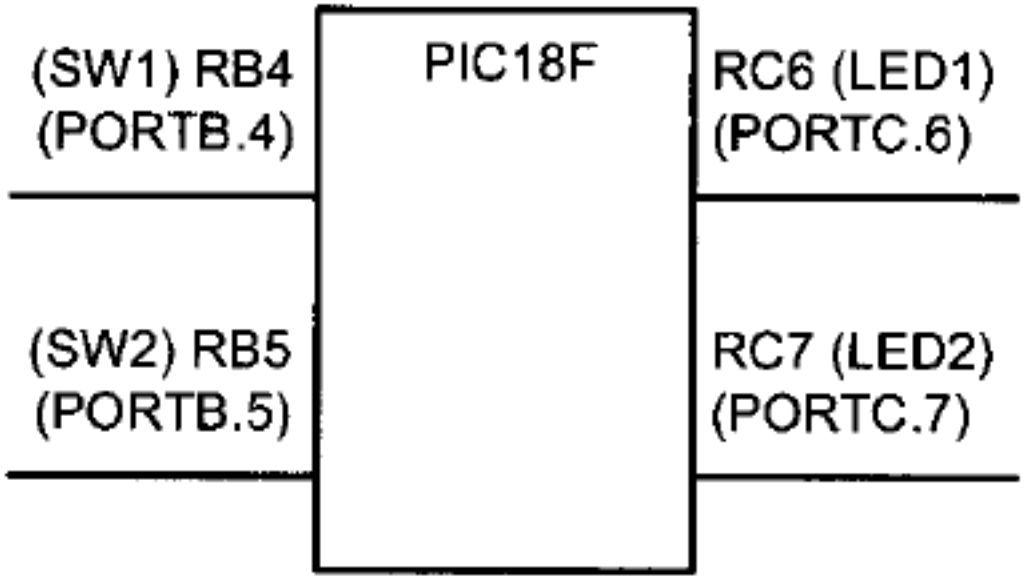
*** Status: H-L or L-H**

Differences Between External Hardware and PORTB-Change Interrupt

External Hardware	PORTB-Change
This (RB0-RB2) interrupts has its own pin and is independent each other	Use s all four (RB4-RB7) and considered to be a single interrupt even though it can use up to four pins
Has it own flag (INTxIF) and independent each other	Single flag only (RBIF)
Can be programmed to trigger on the negative or positive edge	Cause an interrupt either pin changes status from HIGH-LOW or LOW-HIGH

Example

SW1 and SW2 are connected to pins RB4 and RB5 respectively. The activation of SW1 and SW2 will result in changing the state of LED1 and LED2 respectively.



1

```c
#include <p18f4580.h>
#define LED1 PORTCbits.RC6
#define LED2 PORTCbits.RC7

void RBINT_ISR(void);
#pragma interrupt chk_isr           // Used for high-priority

void chk_isr(void)
{
    if(INTCONbits.RBIF == 1)        // RBIF Caused Interrupt?
        {
            RBINT_ISR();            // YES! Execute RBIF Program
        }
}

#pragma code My_HiPrio_Int = 0x08   // High-Priority Interrupts
void My_HiPrio_Int(void)
{
    _asm
        GOTO chk_isr
    _endasm
}
```

2

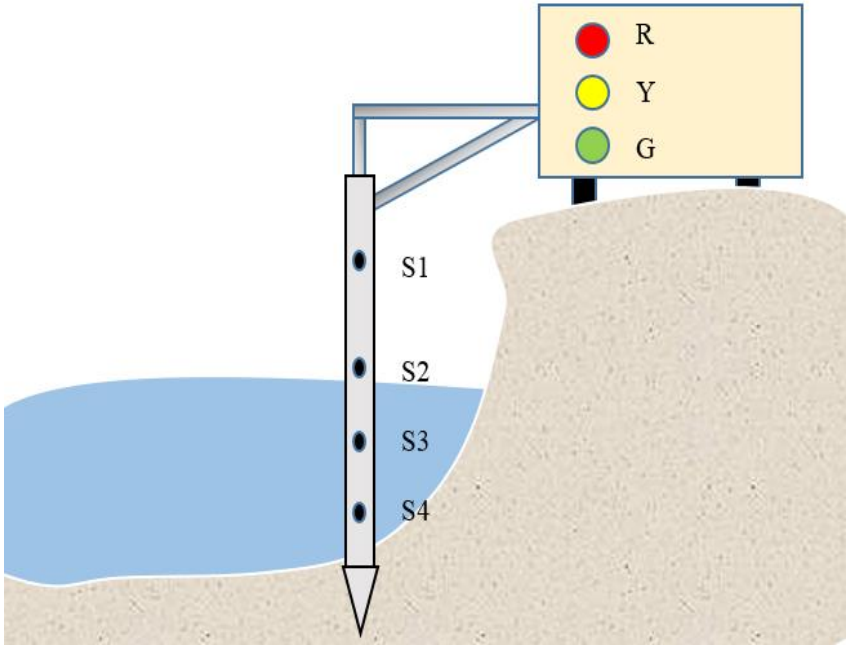
```c
void main(void)
{
    TRISCbits.TRISC6 = 0;           // RC6 = Output
    TRISCbits.TRISC7 = 0;           // RC7 = Output
    TRISBbits.TRISB4 = 1;           // RB4 = Input for Interrups
    TRISBbits.TRISB5 = 1;           // RB5 = Input for Interrups
    INTCONbits.RBIF = 0;            // Clear RBIF Flag bit
    INTCONbits.RBIE = 1;            // Enable RB Interrupt
    INTCONbits.GIE = 1;             // Enable All Interrupts
    // Notice for PORTB-Change Interrupt, there is no need to enable
    // the PEIE, however we still need to enable the GIE bit.
    while(1);
}
```

3

```c
void RBINT_ISR(void)
{
    LED1 = PORTBbits.RB4;
    LED2 = PORTBbits.RB5;
    INTCONbits.RBIF = 0;            // Clear RBIF Flag Bit
}
```

Exercise

Figure below shows the water level monitoring system for Timah Tasoh Lake. Four sensors are connected to a microcontroller (Active-HIGH) at RB4 (S4), RB5 (S3), RB6 (S2) and RB7 (S1) and perform as PORTB-Change Interrupt. Three indicator lamp is connected to the microcontroller (Active-HIGH) at RC0 (R), RC1 (Y) and RC2 (G). From the Figure and Table write a C program to perform the operation continuously. Assume that the XTAL frequency is 20MHz and use Timer1 for delay



Sensor Activation	Indicator Lamp	Siren Sounded
S4	G	-
S4 and S3	Y	500 Hz (Square wave)
S4, S3 and S2	Y	50 Hz (Square wave)
All Sensors	R	5Hz (Square wave)

Interrupt Priority in the PIC18

- What happen if two interrupts are activated at the same time? Which of these two interrupts is responded to first?
- Only two levels of interrupts priority: i) Low Level ii) High Level
- How?... Programming the IPEN (Interrupt Priority Enable) and Programming the Interrupt Priority (IP) register.

Interrupt Vector Table

Interrupt	ROM Location (hex)
Power-on-Reset	0000
High Priority Interrupt	0008 (Default upon power-on reset)
Low Priority Interrupt	0018

INTCONbits.GIEL = 1;
INTCONbits.GIEH = 1;

Interrupt Priority

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF(1)
bit 7							bit 0

IPEN							

IPEN Interrupt Priority Enable bit

0 = All the interrupts are directed to the vector location 0008 (default).

1 = Interrupts can be assigned a low or high priority.

Upon Power-on Reset

The importance of IPEN: Upon power-on reset, all the interrupts of PIC18 are directed to location 0008, making it a single-priority system, just like PIC16xxx. To prioritize the PIC18 interrupts into low- and high-level priorities, we must make IPEN = 1. When IPEN = 1, we can assign either a low or a high priority to any of the interrupts by manipulating the corresponding bit in the IPR (interrupt priority register) for that interrupt. When interrupt priority is enabled (IPEN = 1), we must set both the GIEH and GIEL bits to high in order to enable the interrupts globally. Notice in Figure 11-21 that GIE is the same as GIEH.

Interrupt Flag Bits for PIC18

Interrupt Priority (IP) register

Table 11-6: Interrupt Flag Bits for PIC18

Interrupt	Flag bit (Register)	Enable bit (Register)	Priority (Register)
Timer0	TMR0IF (INTCON)	TMR0IE (INTCON)	TMR0IP (INTCON2)
Timer1	TMR1IF (PIR1)	TMR1IE (PIE1)	TMR1IP (IPR1)
Timer2	TMR2IF (PIR1)	TMR2IE (PIE1)	TMR2IP (IPR1)
Timer3	TMR3IF (PIR3)	TMR3IE (PIE2)	TMR3IP (IPR2)
INT1	INT1IF (PIR1)	INT1IE (PIE1)	INT1IP (INTCON3)
INT2	INT2IF (PIR1)	INT2IE (PIE1)	INT2IP (INTCON)
TXIF	TXIF (PIR1)	TXIE (PIE1)	TXIP (IPR1)
RCIF	RCIF (PIR1)	RCIE (PIE1)	RCIP (IPR1)
RB_INT	RBIF (INTCON)	RBIE (INTCON)	RBIP (INTCON2)

Note: INT0 has only the high-level priority.

IPR1 Peripheral Interrupt Priority Register

		RCIP	TXIP			TMR2IP	TMR1IP

RCIP USART (Serial COM) Receive Interrupt Priority bit
0 = Low priority
1 = High priority

TXIP USART (Serial COM) Transmit Interrupt Priority bit
0 = Low priority
1 = High priority

TMR2IP Timer2 Interrupt Priority bit
0 = Low priority
1 = High priority

TMR1IP Timer1 Interrupt Priority bit
0 = Low priority
1 = High priority

REGISTER 9-10: IPR1: PERIPHERAL INTERRUPT PRIORITY REGISTER 1

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP
bit 7							bit 0

bit 7 **PSPIP:** Parallel Slave Port Read/Write Interrupt Priority bit(1)
1 = High priority
0 = Low priority

bit 6 **ADIP:** A/D Converter Interrupt Priority bit
1 = High priority
0 = Low priority

bit 5 **RCIP:** EUSART Receive Interrupt Priority bit
1 = High priority
0 = Low priority

bit 4 **TXIP:** EUSART Transmit Interrupt Priority bit
1 = High priority
0 = Low priority

bit 3 **SSPIP:** Master Synchronous Serial Port Interrupt Priority bit
1 = High priority
0 = Low priority

bit 2 **CCP1IP:** CCP1 Interrupt Priority bit
1 = High priority
0 = Low priority

bit 1 **TMR2IP:** TMR2 to PR2 Match Interrupt Priority bit
1 = High priority
0 = Low priority

bit 0 **TMR1IP:** TMR1 Overflow Interrupt Priority bit
1 = High priority
0 = Low priority

Details of IPR1

Example

Program Timer0 and Timer1 interrupts to generate square waves on pins RC0 and RC1, respectively. It also uses the transmit and receive interrupts to send and receive data serially. Data from PORTB is transmitted and the received byte is placed on PORTD. Timer0 and Receive ISRs have the high priority while Timer1 and Transmit ISRs are assigned low priority level.

```c
#include<p18f4580.h>
#define myPC0bit PORTCbits.RC0
#define myPC1bit PORTCbits.RC1

void chk_isr(void);
void chk_low_isr(void);
void T0_ISR(void);
void T1_ISR(void);
void TX_ISR(void);
void RX_ISR(void);

#pragma code My_HiPrio_Int = 0x08        // High-Priority Interrupts
void My_HiPrio_Int(void)
{
    _asm
        GOTO chk_isr
    _endasm
}

#pragma code My_LoPrio_Int = 0x18        // Low-Priority Interrupts
void My_LoPrio_Int(void)
{
    _asm
        GOTO chk_low_isr
    _endasm
}
```

1

2

```c
#pragma interrupt chk_low_isr            // Used for low-priority
void chk_low_isr(void)
{
    if(PIR1bits.TMR1IF == 1)             // Timer1 Causes Interrupt?
        {
            T1_ISR();                    // YES! Execute Timer1 ISR
        }
    if(PIR1bits.TXIF == 1)               // Transmit Causes Interrupt?
        {
            TX_ISR();                    // YES! Execute Transmit (Tx) ISR
        }
}

#pragma interrupt chk_isr                // Used for high-priority
void chk_isr(void)
{
    if(INTCONbits.TMR0IF == 1)           // Timer0 Causes Interrupt?
        {
            T0_ISR();                    // YES! Execute Timer0 ISR
        }
    if(PIR1bits.RCIF == 1)               // Receiver Causes Interrupt?
        {
            RX_ISR();                    // YES! Execute Receiver (Rx) ISR
        }
}
```

```
void main(void)
{
    TRISCbits.TRISC0 = 0;               // RC0 = Output
    TRISCbits.TRISC1 = 0;               // RC1 = Output
    TRISD = 255;                        // PORTD = Input
    TRISB = 0;                          // PORTB = Output
    T0CON = 0x08;                       // Timer0, 16-bit Mode
                                        // No Prescaler
    TMR0H = 0xFF;                       // Load TH0
    TMR0L = 0x00;                       // Load TL0
    INTCONbits.TMR0IF = 0;              // Clear TF0
    T1CON = 0x0;                        // Timer1, 16-bit Mode
                                        // No Prescaler
    TMR1H = 0xFF;                       // Load TH1
    TMR1L = 0x00;                       // Load TL1
    PIR1bits.TMR1IF = 0;                // Clear TF1
    TXSTA = 0x20;                       // Choose Low Baudrate, 8-bit
    SPBRG = 15;                         // 9600 Baudrate/XTAL = 10MHz
    RCSTAbits.CREN = 1;                 // Set CREN (Continous Reception)
    RCSTAbits.SPEN = 1;
    TRISCbits.TRISC6 = 0;               // Tx Pin = Output
    TRISCbits.TRISC7 = 1;               // Rx Pin = Input
    RCONbits.IPEN = 1;
    PIE1bits.RCIE = 1;
    PIE1bits.TXIE = 1;
    INTCONbits.TMR0IE = 1;
    PIE1bits.TMR1IE = 1;
    IPR1bits.TMR1IP = 0;                // Make Timer1 a Low Priority
    IPR1bits.TXIP = 0;                  // Make Tx a Low Priority
    T0CONbits.TMR0ON = 1;               // Turn ON Timer0
    T1CONbits.TMR1ON = 1;               // Turn ON Timer1
    INTCONbits.GIEL = 1;                // Enable Low Priority Interrupts
    INTCONbits.GIEH = 1;                // Enable High Priority Interrupts
    while(1);                           // Keep Looping Until Interrups Comes
}
```

3

```
void T0_ISR(void)
{
    TMR0H = 0xFF;                   // Load TH0
    TMR0L = 0x00;                   // Load TL0
    myPC0bit = ~myPC0bit;           // Toogle PORTC.0
    INTCONbits.TMR0IF = 0;          // Clear TF0
}

void T1_ISR(void)
{
    TMR1H = 0xFF;                   // Load TH1
    TMR1L = 0x00;                   // Load TL1
    myPC1bit = ~myPC1bit;           // Toogle PORTC.1
    PIR1bits.TMR1IF = 0;            // Clear TF1
}

void TX_ISR(void)
{
    TXREG = PORTD;
}

void RX_ISR(void)
{
    PORTB = RCREG;
    RCSTAbits.CREN = 0;             // Clear CREN (To Clear Any Error)
    RCSTAbits.CREN = 1;             // Set CREN (Continous Reception)
}
```

Thank You

“Never interrupt your enemy when he/she is making a mistake“