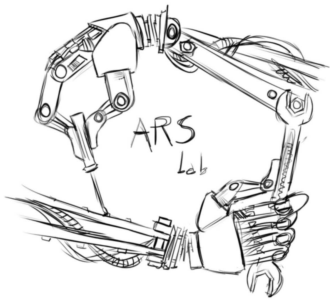
Handling Interrupts in Microchip PIC18F Microcontrollers

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it



L.A.P. 1 Course

Interrupts in MCUs

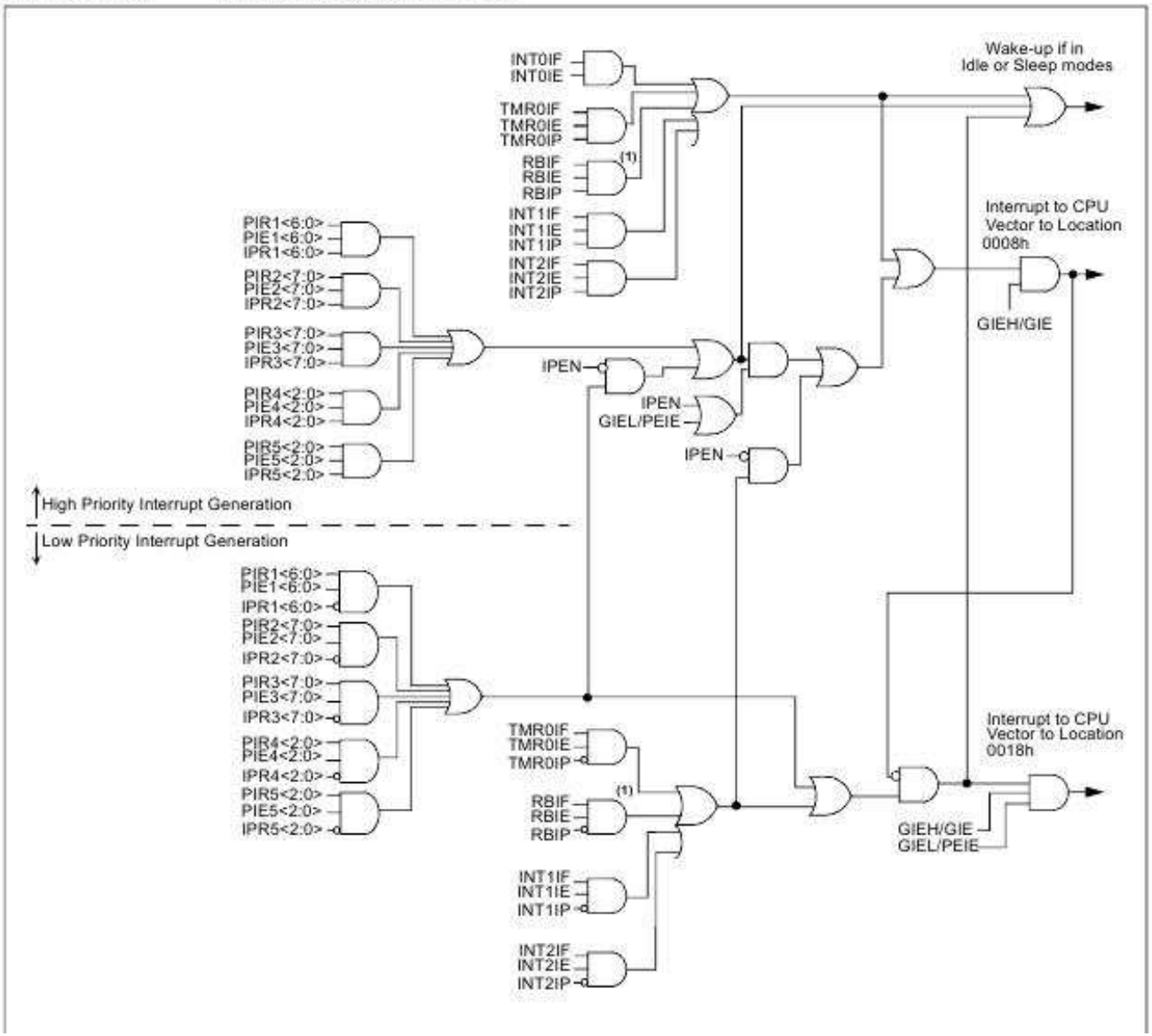
- The core of MCUs manages interrupts, as in normal CPUs
- In a MCU, each peripheral event **can generate** an interrupt, e.g.:
  - The overflow of a timer;
  - The reception/trasmission of a byte through the UART (serial port);
  - The end of conversion of the ADC;
  - ...
- The peripheral is called **interrupt source**
- When an interrupt is generated, the normal program flow is interrupted, a **specific function** is invoked, called **ISR - Interrupt Service Routine**; at the end, the normal program flow is resumed.

Interrupts in Microchip MCUs

- Each peripheral which can generate interrupt has two control bits:
  - **`xxxIF`**, the *interrupt flag*, it is set to **`''1''`**, by the hardware, when the "event" occurs; it must be reset by the software;
  - **`xxxIE`**, the *interrupt enable bit*; when set to **`''1''`** (by the software) the "event", when occurs, **generates a CPU interrupt**
- Moreover, there are other bits which control the interrupt circuit:
  - **`GIE`**, the *global interrupt enable flag*; when set to **`''1''`**, interrupt sources are routed to the CPU;
  - **`IPEN`**, the *interrupt priorities enable flag*; when set to **`''1''`**, two priorities are handled **low** and **high/urgent**;
  - **`PEIE`**, the *peripheral interrupt enable flag*; when set to **`''1''`**, interrupt sources from peripherals are enabled.

The Interrupt Circuit

FIGURE 9-1: PIC18 INTERRUPT LOGIC



Enabling Interrupts in Microchip MCUs (No priorities)

1. Program the peripheral according to its working mode;
2. Reset the peripheral interrupt flag `xxxIF = 0;`;
3. Set the peripheral interrupt enable flag `xxxIE = 1;`;
4. Disable priorities handling `RCONbits.IPEN = 0;`;
5. Enable global interupt flag `INTCONbits.GIE = 1;`;
6. Enable peripheral interupt flag `INTCONbits.PEIE = 1;`;

Enabling Interrupts in Microchip MCUs (With priorities)

1. Program the peripheral according to its working mode;
2. Reset the peripheral interrupt flag `xxxIF = 0;`;
3. Set the peripheral interrupt enable flag `xxxIE = 1;`;
4. Set the peripheral interrupt priority flag (0 = low, 1 = high) `xxxIP = yy`;
5. Enable priorities handling `RCONbits.IPEN = 1;`;
6. Enable/disable high interupts `INTCONbits.GIEH = yy;`;
7. Enable/disable low priority interupts `INTCONbits.GIEL = yy;`;

Handling Interrupts in Microchip MCUs

- Define a C function marked as **interrupt**;
- Check if the peripheral interrupt flag is on;
- Serve the peripheral interrupt;
- Reset the peripheral interrupt flag.

Handling TMR0 interrupt

```
...
void interrupt isr()
{
  if (INTCONbits.T0IF == 1) {
    // ...  handle the TMR interrupt
    INTCONbits.T0IF = 0;
  }
}
...
```

Handling Several Interrupts in Microchip MCUs

Handling TMR0 & TMR1 interrupts

```
...
void interrupt isr()
{
  if (INTCONbits.TMR0IF == 1) {
    // ...  handle the TMR0 interrupt
    INTCONbits.TMR0IF = 0;
  }
  if (PIR1bits.TMR1IF == 1) {
    // ...  handle the TMR1 interrupt
    PIR1bits.TMR1IF = 0;
  }
}
...
```

Example

- A LED **RB0**
- A pushbuttons **RA3**
- Pressing pushbutton starts/stops flashing at a period of 200*ms*

Let's determine timer setup

- We want to use the system clock, `T0CS = 0;`;
- We have $FOSC = 64MHz$, therefore the basic frequency is $FOSC/4 = 16MHz$, the $P = 62.5ns$;
- Let's use the prescaler and divide the frequency by 256, so `PSA = 0;` `T0PS = 0b111;`
- The timer increments using a period $P = 62.5ns * 256 = 16\mu s$.
- So $200ms/16\mu s = 12500$ counts, therefore the TMR0 setup value is -12500.

Let's determine timer setup

Timer Setup

```
...
T0CONbits.TMR0ON = 0; // stop the timer
T0CONbits.T08BIT = 0; // timer configured as 16-bit
T0CONbits.T0CS = 0; // use system clock
T0CONbits.PSA = 0; // use prescaler
T0CONbits.T0PS = 0b111;
// prescaler 1:256 ('0b' is a prefix for binary)
TMR0 = -12500; // setup initial timer value

INTCONbits.T0IF = 0; // reset timer interrupt flag
INTCONbits.T0IE = 1; // enable timer interrupts

RCONbits.IPEN = 0; // do not use priorities
INTCONbits.PEIE = 1; // enable peripheral interrupts
INTCONbits.GIE = 1; // enable interrupts globally
...
```

Let's handle interrupts

Timer Interrupt Handling

```
...
void interrupt isr()
{
  if (INTCONbits.T0IF == 1) {
    TMR0 = -12500; // reload timer value

     // invert the LED
    LATBbits.LATB0 = !LATBbits.LATB0;

    INTCONbits.T0IF = 0;
  }
}
...
```

Let's handle timer on/off

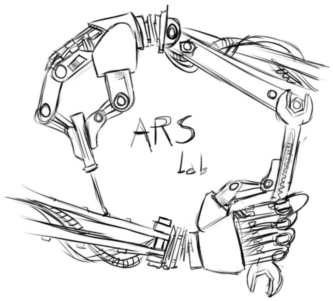
Timer on/off

```
  ...
  for (;;) { // loop forever
    while (PORTAbits.RA3 == 0) {};
    // if the push button is DOWN, wait

    while (PORTAbits.RA3 == 1) {};
    // if the push button is UP, wait

    // transition got, let's invert the TMRON flag
    T0CONbits.TMR0ON = !T0CONbits.TMR0ON;
  }
}
```

Handling Interrupts in Microchip PIC18F Microcontrollers

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it



L.A.P. 1 Course