Embedded System Design

Chapter 5: C Peripheral Interfaces

(Part 2)
References


- PIC 16F877A datasheet
- PIC C Help
Outline

1. PIC16 C Analog Input
2. PIC16 C Interrupts
3. PIC16 Hardware Timers
Outline

1. PIC16 C Analog Input
2. PIC16 C Interrupts
3. PIC16 Hardware Timers
1. PIC16 C Analog Input

- Allows an external voltage to be converted to digital form, stored and processed
- This type of input occurs in data loggers, control systems, digital audio and signal processors...
- The dsPIC range is designed specifically for high-speed analog signal processing
• The PIC 16F877 has eight analog inputs: RA0, RA1, RA2, RA3, RA5, RE0, RE1, RE2 being renamed AN0 to AN7.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA0/AN0</td>
<td>Analog Input 0</td>
</tr>
<tr>
<td>RA1/AN1</td>
<td>Analog Input 1</td>
</tr>
<tr>
<td>RA2/AN2/VREF-</td>
<td>Analog Input 2</td>
</tr>
<tr>
<td>RA4/T0CKI</td>
<td>Analog Input 4</td>
</tr>
<tr>
<td>RA5/AN4/SS</td>
<td>Analog Input 5</td>
</tr>
<tr>
<td>RA7/AN7/CS</td>
<td>Analog Input 7</td>
</tr>
</tbody>
</table>

**Diagram:**

```
U1
MCLR/Vpp/THV
OSC1/CLKIN
OSC2/CLKOUT
RA0/AN0
RA1/AN1
RA2/AN2/VREF-
RA4/T0CKI
RC0/T1OSO/T1CKI
RC1/T1OSI/CCP2
RC2/CCP1
RE0/AN5/RD
RE1/AN6/WR
RE2/AN7/CS
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RD0/PSP0
RD1/PSP1
RD2/PSP2
RD3/PSP3
RD4/PSP4
RD5/PSP5
RD6/PSP6
RD7/PSP7
```

**Notes:**
- RA0/AN0, RA1/AN1, RA2/AN2, RA4/T0CKI, RA5/AN4, RA7/AN7 are analog inputs.
- RA3/AN3/VREF+ is not shown in the diagram.
- The PIC 16F877 has eight analog inputs: RA0, RA1, RA2, RA3, RA5, RE0, RE1, RE2.
A/D Block diagram
<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC SETUP</td>
<td>Initialize ADC</td>
<td><code>setup_adc(ADC_CLOCK_INTERNAL);</code></td>
</tr>
<tr>
<td>ADC PINS SETUP</td>
<td>Initialize ADC pins</td>
<td><code>setup_adc_ports(RA0_ANALOG);</code></td>
</tr>
<tr>
<td>ADC CHANNEL SELECT</td>
<td>Select ADC input</td>
<td><code>set_adc_channel(0);</code></td>
</tr>
<tr>
<td>ADC READ</td>
<td>Read analog input</td>
<td><code>inval=read_adc();</code></td>
</tr>
</tbody>
</table>
Example

Bùi Minh Thành - Bộ môn Kỹ Thuật Điện Tử - ĐHGBK
Example

```c
#include "16F877A.h"
#device ADC=8   // 8-bit conversion

#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_D0, rcv=PIN_D1)  // LCD output

void main() {
    int vin0;
    setup_adc(ADC_CLOCK_INTERNAL);   // ADC clock
    setup_adc_ports(ALL_ANALOG);     // Input combination
    set_adc_channel(0);              // Select RA0

    for (;;){
        delay_ms(500);
        vin0 = read_adc();          // Get input byte
        vin0 = (vin0/32)+0x30;       // Convert to ASCII

       putc(254); putc(1); delay_ms(10); // Clear screen
        printf("Input = "); putc(vin0); // Display input
    }
}
```
Voltage measurement

[Diagram with voltage measurements and components shown]
Voltage measurement

```c
#include "16F877A.h"
#device ADC=10 // 10-bit operation
#use delay(clock=4000000)
#use rs232(baud=9600, xmit=PIN_D0, rcv=PIN_D1)

void main()
{
    int    chan;
    float  analin[8], disvolts[8]; // Array variables

    setup_adc(ADC_CLOCK_INTERNAL); // ADC clock source
    setup_adc_ports(AN0_AN1_AN2_AN4_AN5_AN6_AN7_VSS_VREF); // ADC inputs

    while(1)
    {
        for(chan=0;chan<8;chan++)
        {
            delay_ms(1000);
            set_adc_channel(chan);
            analin[chan] = read_adc();
            disvolts[chan] = (analin[chan])/400; // Scale input
            putc(254);putc(1);delay_ms(10); // Clear display
            printf(" RA%d = %4.3g",chan,disvolts[chan]); // Display volts
        }
    }
}
```
Outline

1. PIC16 C Analog Input
2. PIC16 C Interrupts
3. PIC16 Hardware Timers
2. PIC16 C Interrupts

- Interrupts allow an external event to initiate a control sequence that takes priority over the current MCU activity.
- The interrupt service routine (ISR) carries out some operation associated with the port or internal device that requested the interrupt.
- Interrupts are frequently used with hardware timers, which provide delays, timed intervals and measurement.
- PIC16F877 has 14 interrupt sources
C Interrupts

- CCS C Interrupt Functions

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERRUPT CLEAR</td>
<td>Clears peripheral interrupt</td>
<td>clear_interrupt(int_timer0);</td>
</tr>
<tr>
<td>INTERRUPT DISABLE</td>
<td>Disables peripheral interrupt</td>
<td>disable_interrupts(int_timer0);</td>
</tr>
<tr>
<td>INTERRUPT ENABLE</td>
<td>Enables peripheral interrupt</td>
<td>enable_interrupts(int_timer0);</td>
</tr>
<tr>
<td>INTERRUPT ACTIVE</td>
<td>Checks if interrupt flag set</td>
<td>interrupt_active(int_timer0);</td>
</tr>
<tr>
<td>INTERRUPT EDGE</td>
<td>Selects interrupt trigger edge</td>
<td>ext_int_edge(H_TO_L);</td>
</tr>
<tr>
<td>INTERRUPT JUMP</td>
<td>Jump to address of ISR</td>
<td>jump_to_isr(isr_loc);</td>
</tr>
</tbody>
</table>
# C Interrupts – Interrupt sources

## PIC16 F877 Primary Interrupt

<table>
<thead>
<tr>
<th>Interrupt Label</th>
<th>Interrupt Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>GLOBAL</td>
<td>Use to enable all interrupt sources</td>
</tr>
<tr>
<td>INT_EXT</td>
<td>External interrupt detect on RB0</td>
</tr>
<tr>
<td>INT_RB</td>
<td>Change on Port B detect</td>
</tr>
<tr>
<td>INT_RTCC</td>
<td>Timer 0 overflow (same as TIMER0)</td>
</tr>
<tr>
<td>INT_TIMER0</td>
<td>Timer 0 overflow (same as RTCC)</td>
</tr>
</tbody>
</table>

## PIC16 F877 Peripheral Interrupts

<table>
<thead>
<tr>
<th>Interrupt Label</th>
<th>Interrupt Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ports</td>
<td></td>
</tr>
<tr>
<td>INT_TBE</td>
<td>USART transmit data done</td>
</tr>
<tr>
<td>INT_RDA</td>
<td>USART receive data ready</td>
</tr>
<tr>
<td>INT_SSP</td>
<td>Serial data received at SPI or I2C</td>
</tr>
<tr>
<td>INT_BUSCOL</td>
<td>I2C collision detected</td>
</tr>
<tr>
<td>INT_PSP</td>
<td>Data ready at parallel serial port</td>
</tr>
<tr>
<td>Timers</td>
<td></td>
</tr>
<tr>
<td>INT_TIMER1</td>
<td>Timer 1 overflow</td>
</tr>
<tr>
<td>INT_CCP1</td>
<td>Timer 1 capture or compare detect</td>
</tr>
<tr>
<td>INT_TIMER2</td>
<td>Timer 2 overflow</td>
</tr>
<tr>
<td>INT_CCP2</td>
<td>Timer 2 capture or compare detect</td>
</tr>
<tr>
<td>Others</td>
<td></td>
</tr>
<tr>
<td>INT_AD</td>
<td>Analog-to-digital converter complete</td>
</tr>
<tr>
<td>INT_COMP</td>
<td>Comparator output change</td>
</tr>
</tbody>
</table>
## Interrupt Sources

<table>
<thead>
<tr>
<th>Interrupt Source</th>
<th>Interrupt Trigger Event</th>
<th>Interrupt Label</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Timers</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timer0</td>
<td>Timer0 register overflow</td>
<td>INT_TIMER0</td>
</tr>
<tr>
<td>Timer1</td>
<td>Timer1 register overflow</td>
<td>INT_TIMER1</td>
</tr>
<tr>
<td>CCP1</td>
<td>Timer1 capture or compare detected</td>
<td>INT_CCP1</td>
</tr>
<tr>
<td>Timer2</td>
<td>Timer2 register overflow</td>
<td>INT_TIMER2</td>
</tr>
<tr>
<td>CCP2</td>
<td>Timer2 capture or compare detected</td>
<td>INT_CCP2</td>
</tr>
<tr>
<td><strong>Ports</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RB0/INT pin</td>
<td>Change on single pin RB0</td>
<td>INT_EXT</td>
</tr>
<tr>
<td>Port B pins</td>
<td>Change on any of four pins, RB4-RB7</td>
<td>INT_RB</td>
</tr>
<tr>
<td>Parallel Slave Port</td>
<td>Data received at PSP (write input active)</td>
<td>INT_PSP</td>
</tr>
<tr>
<td>Analog Converter</td>
<td>A/D conversion completed</td>
<td>INT_AD</td>
</tr>
<tr>
<td>Analog Comparator</td>
<td>Voltage compare true</td>
<td>INT_COMP</td>
</tr>
<tr>
<td><strong>Serial</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UART Serial Port</td>
<td>Received data available</td>
<td>INT_RDA</td>
</tr>
<tr>
<td>UART Serial Port</td>
<td>Transmit data buffer empty</td>
<td>INT_TBE</td>
</tr>
<tr>
<td>SPI Serial Port</td>
<td>Data transfer completed (read or write)</td>
<td>INT_SSP</td>
</tr>
<tr>
<td>I²C Serial Port</td>
<td>Interface activity detected</td>
<td>INT_SSP</td>
</tr>
<tr>
<td>I²C Serial Port</td>
<td>Bus collision detected</td>
<td>INT_BUSCOL</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EEPROM</td>
<td>Nonvolatile data memory write complete</td>
<td>INT_EEPROM</td>
</tr>
</tbody>
</table>
Interrupt example
Interrupt example

// Demo external interrupt RBO low interrupts foregroundoutput count
#include "16F877A.h"
#include "delay.h"

#include "16F877A.h"
#include "delay.h"

#define clock 4000000

// Interrupt name

#define ext_int edge H_TO_L

// Interrupt service routine

void isrext() {
    output_D(255); // ISR action
    delay_ms(1000);
}

// Enable named interrupt
enable_interrupts(int_ext);

// Enable all interrupts
enable_interrupts(global);

// Interrupt signal polarity

while(1) {
    output_D(x);
    x++;
    delay_ms(100);
}

// Foreground loop

Bùi Minh Thành - Bộ môn Kỹ Thuật Điện Tử - ĐHBK
Interrupt statements

• #int Xxx
  – Tells the compiler that the code immediately following is the service routine for this particular interrupt
  – The interrupt name is preceded by #(hash) to mark the start of the ISR definition and to differentiate it from a standard function block.
  – An interrupt name is defined for each interrupt source.

• enable_interrupts(int_ext);
  – Enables the named interrupt by loading the necessary codes into the interrupt control registers
Interrupt statements

- **enable_interrupts(level);**
  - Enables the interrupt at the given level.
  - Examples:
    
    ```
    enable_interrupts(GLOBAL);
    enable_interrupts(INT_TIMER0);
    enable_interrupts(INT_TIMER1);
    ```

- **Disable_interrupts(level)**
  - Disable interrupt at the given level

- **ext_int_edge(H_TO_L);**
  - Enables the edge on which the edge interrupt should trigger. This can be either rising or falling edge.
Timer interrupt process

1. Start Counter Statement
2. Run Counter until Overflow
3. Time-out Interrupt
4. Jump to ISR
5. Time-out Process (Interrupt Service Routine)
6. Return from Interrupt
7. Continue
Outline

1. PIC16 C Analog Input
2. PIC16 C Interrupts
3. PIC16 Hardware Timers
3. PIC16 Hardware Timers

• The PIC 16F877 has three hardware timers built in: Timer0 (8-bit) (originally called RTCC, the real-time counter clock), Timer1 (16-bit), and Timer2 (8-bit).

• The principal mode of operation of these registers are as counters for external events or timers using the internal clock.

• Additional registers are used to provide Capture, Compare, and Pulse Width Modulation (PWM) modes.
Counter/Timer Operation

• A counter/timer register consists of a set of bistable stages (flip-flops) connected in cascade (8, 16, or 32 bits).

• An 8-bit counter counts up from 0x00 to 0xFF
Counter/Timer Operation

- Timer0 is an 8-bit register that can count pulses at RA4; for this purpose, the input is called T0CKI (Timer0 clock input).
- Timer1 is a 16-bit register that can count up to 0xFFFF (65,535) connected to RC0 (T1CKI).
- The count can be recorded at any chosen point in time; alternatively, an interrupt can be generated on overflow to notify the processor that the maximum count has been exceeded.
- If the register is preloaded with a suitable value, the interrupt occurs after a known count.
- Timer0 has a prescaler that divides by up to 128;
- Timer1 has one that divides by 2, 4, or 8;
- Timer2 has a prescaler and postscaler that divide by up to 16.
## Timer Functions

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMERX SETUP</td>
<td>Set up the timer mode</td>
<td>`setup_timer0(RTCC_INTERNAL</td>
</tr>
<tr>
<td>TIMERX READ</td>
<td>Read a timer register (8 or 16 bits)</td>
<td><code>count0 = get_timer0();</code></td>
</tr>
<tr>
<td>TIMERX WRITE</td>
<td>Preload a timer register (8 or 16 bits)</td>
<td><code>set_timer0(126);</code></td>
</tr>
<tr>
<td>CCPX SETUP</td>
<td>Select PWM, capture, or compare mode</td>
<td><code>setup_ccp1(ccp_pwm);</code></td>
</tr>
<tr>
<td>PWMX DUTY</td>
<td>Set PWM duty cycle</td>
<td><code>set_pwm1_duty(512);</code></td>
</tr>
</tbody>
</table>
Exercise
The program that carries out the function of a counting circuit counts from 00 to 19 and displays on two 7-segment LEDs connected to port C.
C source code with Assembly block

/*
Source code file: FAST.C
Author, date, version: MPB 19-10-07 V1.0
Program function: Demo of assembler block
Simulation circuit: ASSEM.DSN
*/

#include "16F877A.h"
#define delay(clock=4000000)

// ISR switches off output and waits for button ***********
#define int_ext

void isrext()
{
    output_low(PIN_D0);
    delay_ms(100);
    while(input(PIN_B0));
}

// Main block initializes interrupt and waits for button ***
void main()
{
    enable_interrups(int_ext);
    enable_interrups(global);
    ext_int_edge(L_TO_H);
    // Assembler block outputs high speed pulse wave ********
    #asm
    Start:
    BSF 8, 0
    BCF 8, 0
    GOTO Start
    #endasm
    // End of source code
*/

If the MCU clock is 20MHz
⇒ the output frequency is 1.25MHz.
PWM mode

• In Pulse Width Modulation mode, a CCP module can be used to generate a timed output signal.
• This provides an output pulse waveform with an adjustable high (mark) period.
PWM mode - Example

```c
// Demo PWM output, MCU clock = 4MHz

#include "16F877A.h"

void main()
{
    setup_ccp1(ccp_pwm); // Select timer and mode
    setup_timer_2(T2_DIV_BY_16, 250, 1); // Set on time
    set_pwm1_duty(500); // Clock rate & output period

    while(1){} // Waiting until reset
}
```

Produce an output at CCP1 of 250Hz (4ms) and a mark-space ratio of 50% with a 4-MHz MCU clock. Explain?
Compare Mode

- Generate a timed output in conjunction with Timer1.
- The 16-bit CCPR register is preloaded with a set value, which is continuously compared with the Timer1 count. When the count matches the CCPR value, the output pin toggles and a CCP interrupt is generated. If this operation is repeated, an interrupt and output change with a known period can be obtained.
Capture mode

- The CCP pin is set to input and monitored for a change of state.
- When a rising or falling edge (selectable) is detected, the timer register is cleared to 0 and starts counting at the internal clock rate.
- When the next active edge is detected at the input, the timer register value is copied to the CCP register. The count therefore corresponds to the period of the input signal. With a 1MHz instruction clock, the count is in microseconds.
Exercise – Timer Interrupt

1) Calculate the frequency of the pulse on PIN B0 created by the program on the right figure, given that $F_{\text{OSC}} = 4\text{MHz}$

2) Write the program that create a 2Hz pulse on PIN_B1, given that $F_{\text{OSC}} = 4\text{MHz}$ and dutycycle $= 20\%$

```c
// Timer interrupt
#include "16F877A.h"
#include <16F877A.h>

// Use delay (clock=4000000)
#include "delay.h"

int timer1; // Interrupt name

void isrimer1() // Interrupt service routine
{
    output_toggle(PIN_B0);
    set_timer1(-50000);
}

void main() {
    int x;

    enable_interrupts(int_timer1); // Enable named interrupt
    enable_interrupts(global); // Enable all interrupts

    // ext_int_edge(H_TO_L); // Interrupt signal polarity
    setup_timer_1(T1_INTERNAL | T1_DIV_BY_8);
    set_timer1(-50000);
    while(1);
}
```