CALIFORNIA STATE POLYTECHNIC UNIVERSITY, POMONA
COLLEGE OF ENGINEERING

ECE 3301L **Microcontroller Lab** **Felix Pinai**
Session 2

Experiment 9
Timer Interrupt &
Signal Generator Using the SPI and a D2A converter

The purpose of this lab is to get the student to be familiar with the following concepts:

1) System Interrupts
2) Timer Operations
3) SPI (Serial Peripheral Interface) Operations
4) D/A (Digital to Analog) Converter

PART A) Timers and Interrupts

In lab #8, we have generated the routine delay_500ms(). This routine programs the Timer 0 with a fixed constant and then enable that timer to run until the count reaches the value 0xFFFF. When that happens, a flag is set and then the delay operation is completed. A while loop is entered to wait for the flag to be set. Using the same technique but instead of sitting in the while loop, we will use the interrupt technique to do the same delay function.

The first step is to program is to program the Timer 0 to operate with the 1:32 prescaler. Next, load the timer with the same count in order to reach the 500 msec delay. Now we program the registers needed for the interrupt operations:

```
INTCONbits.TMR0IF=0;        // Clear interrupt flag
INTCONbits.TMR0IE=1;        // Set interrupt enable
INTCONbits.GIE=1;           // Set the Global Interrupt Enable
```

The last step is to enable the Timer 0:

```
T0CONbits.TMR0ON=1;         // Turn on Timer0
```

That will kickstart the Timer operation. We will then wait for the interrupt to occur. When that happens, the control of the processor is then transferred to the Interrupt Service Routine (ISR). The T0ISR() function will be then executed.

In that routine, the interrupt flag TMR0IF is cleared to allow further interrupts to happen again. Next, the timer 0 is reloaded with the same count as at the beginning to facilitate another timer operation to interrupt again 500 msec later.

The LED called ‘Second LED’ will be toggled to make its logic state change thus creating a blinking LED.

Here is the sample code of that operation:

```
void Do_Init()                          // Initialize the ports
{
        OSCCON=0x70;                    // Set oscillator to 8 MHz
        ADCON1=0x0F;                    // Configure all pins to digital
        TRISC=0x00;                     // Configure PORT C to be all outputs


        T0CON=0x04;                     // Timer0 off, increment on positive
                                        // edge, 1:32 prescaler
        TMR0H=0x??;                     // Set Timer High
        TMR0L=0x??;                     // Set Timer Low
        INTCONbits.TMR0IE=1;            // Set interrupt enable
        INTCONbits.TMR0IF=0;            // Clear interrupt flag
        INTCONbits.GIE=1;               // Set the Global Interrupt Enable
        T0CONbits.TMR0ON=1;             // Turn on Timer0
}

void interrupt high_priority T0ISR()
{
        INTCONbits.TMR0IF=0;            // Clear the interrupt flag
        TMR0H=0x??;                     // Reload Timer High and
        TMR0L=0x??;                     // Timer Low
        Second_LED = ~Second_LED        // flip logic state of
                                        // Second_LED
}

void main()
{
        Do_Init();                      // Initialization
        while (1)
        {                               // Do nothing,
                                        // wait for interrupt
        }
}
```

Part B) Faster interrupts

Based on the above example, change the programming of the Timer 0 to generate a square wave that has a period of 2 msec (1 msec at logic ‘1’ and 1 msec at logic ‘0’). The new count has to be calculated and the prescaler must be

reduced. The ‘Second LED’ will blink very fast and your eyes would not be able to see all the transitions. Put a scope on that signal and measure its periodl. Make sure to adjust the count that is loaded into the Timer0 registers TMR0L and TMR0H inside the T0ISR() rouine so that the width of the pulse is exactly 1 msec.

PART C) SPI bus operations

We are going to use the SPI bus to send data out. The big advantage of this bus is the use of either 3 or 4 wires to transmit data serially instead of having to connect a lot of wires to transmit data like a minimum of 8 wires for 8-bit data or 16 wires for 16-bit data. To better understand the concept of SPI bus, you can read the following link:

https://en.wikipedia.org/wiki/Serial_Peripheral_Interface

For a typical transaction, run the following program. Place a scope probe on the signal SCLK at pin and another scope probe on the signal SDO at pin 24, both located on the 40-pin header on the development board (Do not use the header JP8 that has the same signals because they have 1K resistors in series making degradations to the quality of the signals).

You should see two sets of 8 eight pulses on the SCLK pin while the waveform of the SDO pin should show two data patterns of 0x55 and 0xAA in serial form.

```
void main()
{
char i;
        OSCCON=0x070;                   // Program oscillator to be at 8Mhz
        TRISC=0x00;                     // Setup port C with output
        SSPSTAT=0x40;                   // SMP:
                                        // Input data sampled at middle of data
                                         // output
                                        // CKE:
                                        // Transmit occurs on transition from active
        SSPCON1=0x20;                   //  SSPEN:
        while (1)
        {
                SSPBUF = 0x55;                  // data 0x55 to be sent out
                while (SSPSTATbits.BF == 0);    // wait for status done

                SSPBUF = 0xaa;                  // data 0xAA to be sent out
                while (SSPSTATbits.BF == 0);    // wait for status done
                for (i=0;i<10;i++);             // small delay
        }
}
```

PART D) Waveform Generations using D/A and SPI

A D/A converter has the function to convert a digital value into an analog voltage. A typical D/A is rated on the number of bits it supports – the higher the number of bits the better the precision. A D/A uses a reference voltage as a base to do the conversion.

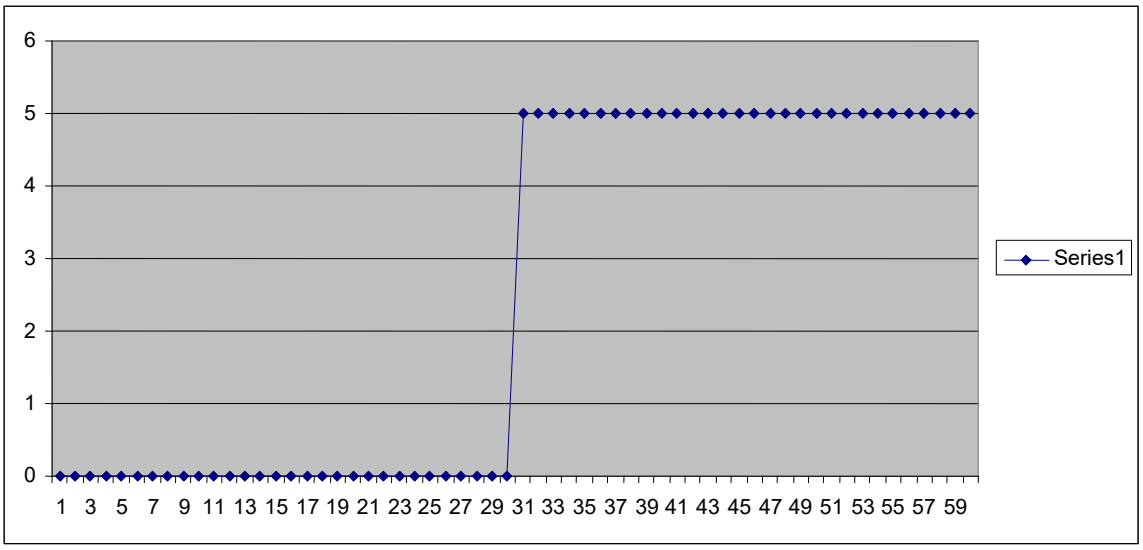
For this lab, we will be using a D/A converter with a SPI bus interface. One typical IC for this use is the MCP4901. Here is the link to its datasheets:

http://ww1.microchip.com/downloads/en/DeviceDoc/22248a.pdf

We are going to combine the use of this MCP4901 and the concept of interrupts to generate a series of pulses that will form a desired type of waveform.

Part 1)

The simplest waveform is a square wave.

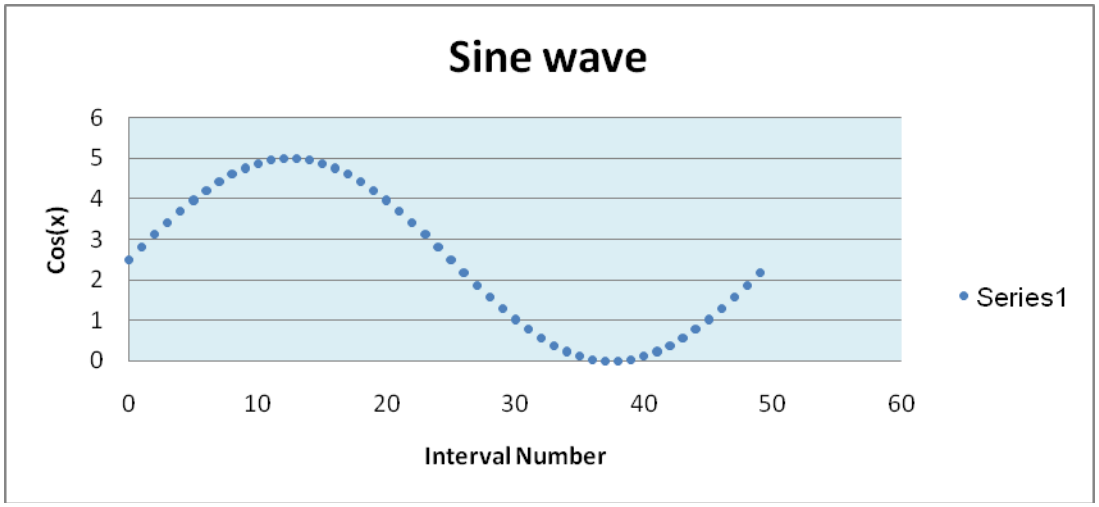


You need to generate the above waveform to be a square wave with a frequency of 16.66Hz or a period of 60 msec. Its amplitude must be from 0 to 5V For the purpose of this lab, 60 periodical outputs should be generated with a spacing of 60/60=1.00 msec between outputs. Therefore, program the timer to interrupt every 1 msec period.

Use the schematics on Figure 1 to connect the PIC microcontroller to a D/A converter (MCP4901).
This D/A will output a voltage up to VDD which is in this case +5V. Since the digital input for a D/A is from 0 to 255 and the requirements for the voltage output is from 0 to +5V, the proper values for the D/A have to be converted.

Part 2)

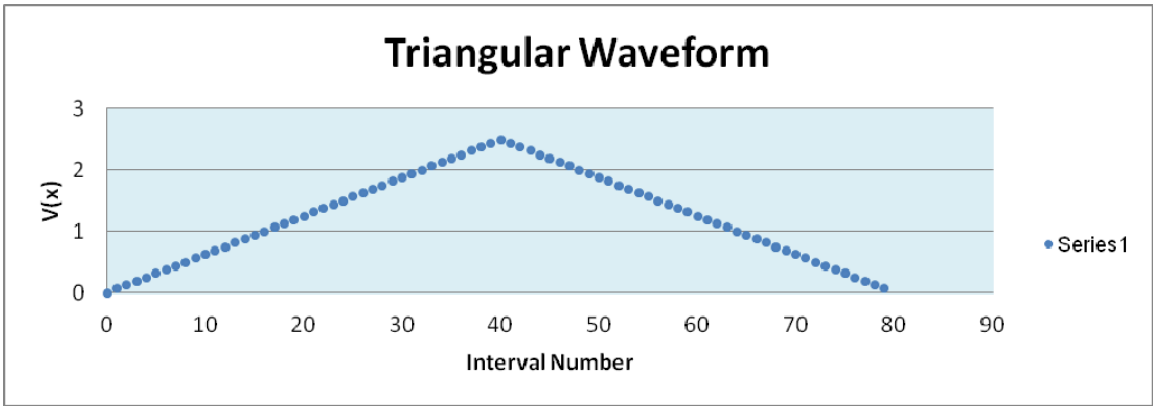
Modify Part 1) to generate a 33Hz sine wave as shown below:



The period of the waveform must be then 30 msec. The amplitude of the above waveform must be from 0 to 5V or its mid-range is at 2.5V with a +/- 2.5V swing. For the purpose of this lab, 50 periodical outputs should be generated with a spacing of 30/50=0.600 msec between outputs.

Part 3)

Modify part 2) or part 3) to generate a 25 Hz triangular wave as shown below:



The period of the waveform must be then 40 msec. The amplitude of the above waveform must be from 0 to 2.55V. For the purpose of this lab, 80 periodical outputs should be generated with a spacing of 40/80=0.500 msec between outputs.

Hint:

For the parts 1 through 3, as you can see on the picture above with the waveforms, one period can be subdivided into equal segments. Hence, if you can start a timer that will interrupt the processor at a fixed interval, then at each interrupt you can program the D/A chip with the proper value of the voltage to be outputted. That value is obtained from an array with pre-stored values and you would use an interval counter as an index to that array.

1) In the initialization, you would need to perform the following steps:

1) Setup up the processor to run at 8Mhz (don't use 4Mhz because it is best to use the fastest speed of the processor) – Use OSCCON register to set the proper speed.
2) Setup the ADCON1 to make sure that PORTA and PORTB are all digital
3) Fill an array that should contain all the values for the waveform to be generated
4) Setup the TRISC register to be output
5) Setup the processor's SPI port.
6) Setup the TIMER0 (T0CON) with the mode 16-bit and no prescale
7) Enable the timer's interrupt (TMR0IE in INTCON register)
8) Clear the timer's interrupt flag (TMR0IF in INTCON register)
9) Enable the Global Register
10) Program the timer with the proper interval (TMR0H and TMR0L).
11) Turn on the timer

Here are the lines that you should fill:

```
OSCCON=0x70;          // Program oscillator to be at 8Mhz
TRISC=0x??;           // Setup port C with output
SSPSTAT=0x40;         // SMP:
                      // Input data sampled at middle of data output
                      // CKE:
                      // Transmit occurs on transition from active
SSPCON1=0x20;         //  SSPEN:
                      // Enables serial port and configures SCK, …
T0CON=0x??;           // Set Timer 0 in Timer mode
```

```
TMR0H=0x??;                 // Program Timer High byte
TMR0L=0x??;                 // Program Timer Low byte
 INTCONbits.TMR0IE=1;       // Enable Timer 0 interrupt
INTCONbits.TMR0IF=0;        // Clear Timer 0 Interrupt Flag
INTCONbits.GIE=1;           // Enable Global Interrupt

T0CONbits.TMR0ON=1;         // Turn on Timer 0
```

2) The main loop will be a do nothing while loop waiting for the timer interrupt to arrive.

3) T0 Interrupt Service Routine T0ISR()

```
void interrupt high_priority T0ISR()
{

// Place your code here
    1) Clear the timer interrupt flag
    2) Reload the timer register
    3) Update the interval counter
    4) Read the array based on counter
    5) Use the SPI_out() module to output the value to the D/A chip
}
```

Here is the code for the SPI_out(char) module:

```
void  SPI_out(unsigned char SPI_data)
{
char First_byte, Second_byte;
        First_byte = (SPI_data & 0xf0) >> 4;    // take the upper nibble of data and >> 4
                                                //times
        First_byte = 0x30 | First_byte;         // set the upper nibble with 0x30
        Second_byte = (SPI_data & 0x0f) <<4;// take the lower nibble of data and << 4 times

        Chip_Select = 0;                        // set the chip select of the D/A chip to be low

        SSPBUF = First_byte;                    // output the first byte to the SPI bus
        while (SSPSTATbits.BF == 0);            // wait for status done
        for (i=0;i<1;i++);                      // small delay

        SSPBUF = Second_byte;                   // output the second byte to the SPI bus
        while (SSPSTATbits.BF == 0);            // wait for status done
        for (i=0;i<1;i++);                      // small delay


        Chip_Select = 1;                        // raise chip select high
}
```

You do need to have the definition for the variable “Chip_Select” done through the use of “# define” and look on the schematics for the hardware assignment of that signal.