Lab7: SysTick Timer

Preparation:

Before you start, ensure that you have completed Lab by interfacing switches and LED to MSP 432 LaunchPad and tested it. You shall require only the LaunchPad to complete this lab.

Reference Materials:

- MSP432P4xx Technical Reference Manual.
- MSP 4322 Launchpad User's Guide.
- MSP432P401R Datasheet.

Purpose:

The purpose of this lab is to learn to use the SysTick timer to manage time. You'll first implement an accurate time delay; this is followed by using the time delay to create PWM output. Finally, using low-pass filter, you'll implement a DAC.

Procedure:

To generate periodic heartbeat:

The LED periodic heartbeat will be implemented using the MSP 432 LaunchPad and you should not require any additional circuits. The hardware connection is as shown in figure 1.

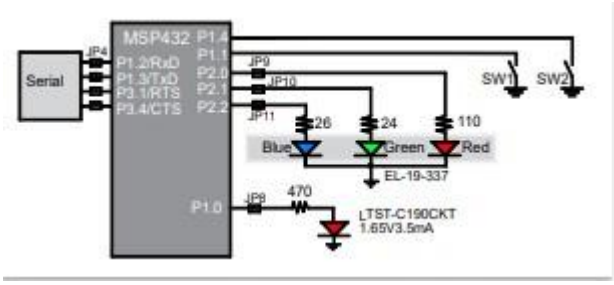

Figure 1: Hardware connection for switches and LEDs on LaunchPad

The first step now is to develop and validate the SysTick wait function. Initialize SysTick by implementing the driver using the pseudo code below. Name the function as 'Systick_Init'.

- Initialize the LOAD register with 0x00FFFFFF (Maximum 24-bit value)
- Set SysTick control register to 5.

In this lab, we'll not be using interrupts. The initialization function is called once at the beginning of the main program, however, before the use of SysTick.

The prototype for SysTick wait would be **Void SysTick_Wait1us(uint32_t delay);** where delay is time of delay in us.

The sequence of steps for the SysTick wait function are:

- Write a desired value into the SysTick LOAD register.
- Clear the VAL Counter value, which also clears the COUNT bit.
- Wait for the COUNT bit in the SysTick CTRL register to be set.

Now, Test the wait function using the function program9_1 below by creating a 50% duty cycle digital output. Use a logic analyzer or oscilloscope to verify the proper timing of the wait function.

```
int Program9_1(void)
{
    Clock_Init48MHz();
    SysTick_Init();
    LaunchPad_Init(); // LEDs and Switches
    TExaS_Init(LOGICANALYZER_P1);
    while(1)
    {
        p1->OUT |= 0x01;
        SysTick_Wait1us(5000);
        p1->OUT &= ~0x01;
        SysTick_Wait1us(5000);
    }
}
```

Test function for SysTick_wait1us

In the second step, use the two switches on the LaunchPad to implement the heartbeat in such a way that one Switch (Switch 1) starts the heartbeat while the other (Switch 2) stops it. Further, ignore Switch 1 while heartbeat is active and Switch 2 otherwise.

Generate PWM Output:

The next step is to extend SysTick_wait1us function operation to implement digital waves with a sinusoidally-varying duty cycle. To implement an output wave with fixed frequency and duty cycle, the main loop will implement the following four steps in order indefinitely.

- Set P1.0 high
- Wait H us using SysTick_Wait1us function.
- Clear P1.0 low
- Wait L us using SysTick_wait1us function.

PulseBuf is a ROM-based table consisting of 100 pulse-times, in units of us, which constitute a sinusoidally-varying duty cycle. Because 100*10ms is one second, execute the following steps indefinitely. Each time through the loop using a new H value, the LED with flash at 1Hz.

- Look up new H= PulseBuf[i] value.
- Calculate L = 10000 – H

- Set P1.0 to high
- Wait H us using your SysTick_Wait1us function.
- Clear P1.0 to low.
- Wait L us using your SysTick_Wait1us function
- I = I+1, when I is 100, roll back to 0.

The array used to generate sine wave is as shown below.

```
const unint32_t PulseBuf[100] = {
    5000, 5308, 5614, 5918, 6219, 6514, 6804, 7086,
    7361, 7626, 7880, 8123, 8354, 8572, 8776, 8964,
    9137, 9294, 9434, 9556, 9660, 9746, 9813, 9861,
    9890, 9900, 9890, 9861, 9813, 9746, 9660, 9556,
    9434, 9294, 9137, 8964, 8776, 8572, 8354, 8123,
    7880, 7626, 7361, 7086, 6804, 6514, 6219, 5918,
    5614, 5308, 5000, 4692, 4386, 4082, 3781, 3486,
    3196, 2914, 2639, 2374, 2120, 1877, 1646, 1428,
    1224, 1036,  863,  706,  566,  444,  340,  254,
     187,  139,  110,  100,  110,  139,  187,  254,
     340,  444,  566,  706,  863, 1036, 1224, 1428,
    1646, 1877, 2120, 2374, 2639, 2914, 3196, 3486,
     3781, 4082, 4386, 4692
};
```

PulseBuf Array

Use Oscilloscope or Logic Analyzer to verify your output.

Create PWM DAC:

The final step is to design DAC (Digital to Analog Converter) using the PWM output as shown in figure 2.

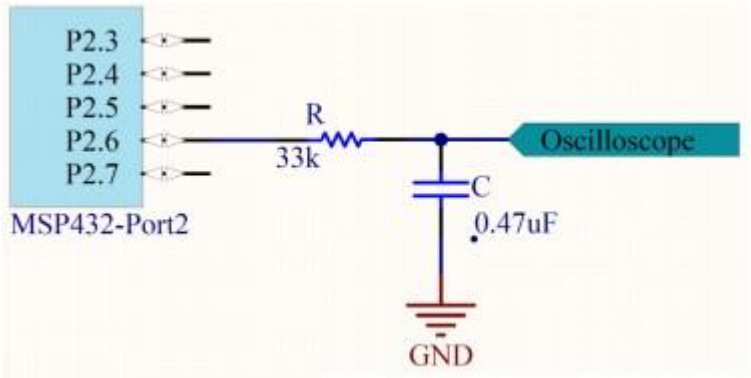

Figure 2: Design for PWM to DAC

PWM can be used to control other devices which respond slowly as comparted to the 100hz wave. An example of a slow device is an analog low pass filter implemented with a resistor and a capacitor. The cutoff frequency of the filter will be given by $f_c = (1/2\prod RC)$

To ensure that the circuit is working, we need 1Hz < fc < 100Hz, so that the circuit passes 1Hz wave and rejects the 100Hz. As a matter of fact, our eyes have a cutoff frequency at about 10Hz, so we will choose RC = 1/(2∏10Hz) ≈ 0.016 sec.

The design is possible with R=33kΩ and C =0.47µF. Connect the voltmeter across the capacitor in figure 3. The program below implements 100Hz wave with known duty cycle on pin P2.6.

```
int program9_2(void){
    uint32_t H,L;
    clock_Init48MHz();
    SysTick_Init();
    TExaS_Init(SCOPE);
    p2->SEL0 &= ~0x40;
    p2->SEL1 &= ~0x40; // P2.6 as GPIO
    p2->DIR |= 0x40; // P2.6 as output
    H = 7500;
    L = 10000-H;
    while(1){
        p2->OUT |= 0x40;
        SysTick_waitlus(H);
        p2->OUT &= ~0x40;
        SysTick_waitlus(L);
    }
}
```

Run the program for five different duty cycles and plot the DC voltage as a function of duty cycle.

Test the above program , with H=9000 and L=1000. This will set the duty cycle to 90%. Now, measure the voltage across the capacity. Change the voltmeter setting to AC and measure the voltage. With ‘S’ being the DC signal voltage and ‘N’ be the noise measurement in volts. The signal to noise ratio (SNR) is S/N. For the same circuit, connect an oscilloscope, run sinusoidally- varying input signal and measure the output on oscilloscope.

Questions

1. What would happen to your implementation if you have PWM period more than 350ms? why or why not?
2. In what way does RC circuit model represent the behavior of a Motor and why?
3. A PWM system uses 48MHz clock and a 16-bit timer to generate a wave. What is the longest period that can be generated?
4. Write two functions to implement a stopwatch. Start() will start the measurement and stop() will return the elapsed time in bus cycles. Assume that elapsed time is less than 349ms.