Module 9

Lab 9 : SysTick Timer

Lab: SysTick Timer

9.0 Objectives

The purpose of this lab is to learn how to use the SysTick timer to manage time.

1. You will first implement an accurate time delay.
2. You will then use the time delay to create a PWM output.
3. With a hardware low pass filter, you will use the PWM to implement a DAC.

Good to Know: Timers, like SysTick, are used in the robot to manage time. SysTick will be used to execute tasks on a periodic basis. The ultrasonic sensors used in the robot calculates distance by measuring the time it takes for sound to travel, reflect off a surface, and return back to the sensor. The concept of PWM will be used to apply a variable power to the robot motors.

9.1 Getting Started

9.1.1 Software Starter Projects

Look at these two projects:
SysTick (example use of the SysTick timer),
Lab09_SysTick (starter project for this lab)

9.1.2 Student Resources (in datasheets directory)

MSP432P4xx Technical Reference Manual (SLAU356)
Meet the MSP432 LaunchPad (SLAU596*)*
MSP432 LaunchPad User's Guide (SLAU597*)*
MSP432P401R Datasheet, msp432p401m.pdf (SLAS826)
CarbonFilmResistor.pdf, resistor datasheet
CeramicCapacitor.pdf, capacitor data sheet

9.1.3 Reading Materials

Volume 1 Sections 4.4 and 8.7
Embedded Systems: Introduction to the MSP432 Microcontroller",
or
Volume 2 Sections 2.6 and 6.3
Embedded Systems: Real-Time Interfacing to the MSP432 Microcontroller",

9.1.4 Components needed for this lab

Quantity	Description	Manufacturer	Mfg P/N
1	MSP-EXP432P401R LaunchPad	TI	MSP-EXP432P401R
1	Ceramic capacitor, 0.47 µF	Kemet	C320C474M5U5TA
1	Carbon 1/6W, 5%, 470 Ω	Yageo	CFR-12JB-470R
1	solderless breadboard	Newark	88W3961

9.1.5 Lab equipment needed

Voltmeter
Oscilloscope (one channel at least 10 kHz sampling)
Logic Analyzer (4 channels at least 10 kHz sampling

9.2 System Design Requirements

In the first part of the lab you will generate a heartbeat wave using the red LED on the TI Launchpad Development kit. You will then use the concept and generate a PWM DAC.

The LED will oscillate from bright to dim to-off to dim almost exhibiting a sine wave, so the LED "looks" like it is breathing. This lab will use the two switches on the LaunchPad to activate and deactivate the heartbeat.

- The heartbeat activates (and continues indefinitely) when the operator pushes SW1
- The heartbeat deactivates when the operator pushes SW2

The operator may push the switches multiple times, and the heartbeat should start and stop as described above. If you can ignore the start button while the heartbeat is active, and ignore the stop button while the heartbeat is inactive.

Lab: SysTick Timer

The basic idea is to use your SysTick wait function to create a digital output signal on P1.0. When active, the period of this signal should be fixed at 10,000 µs. However, software will adjust the duty cycle as a means to control the brightness of the LED. Let *H* be the time the LED is on and *L* be the time the LED is off. The software will guarantee that *H*+*L* is always 10000 µs. However, when active, *H* will vary from 100 to 9900. The duty cycle is defined as

$$Duty = H/(H+L) = H/10000$$

The brightness of the LED is linearly related to the duty cycle. To give your heartbeat flair, you will oscillate the duty cycle sinusoidally, as illustrated in Figure 1. When the duty cycle is large the LED will be bright, when the duty cycle is 50% the LED will be dim, and when the duty cycle is low the LED will be off.

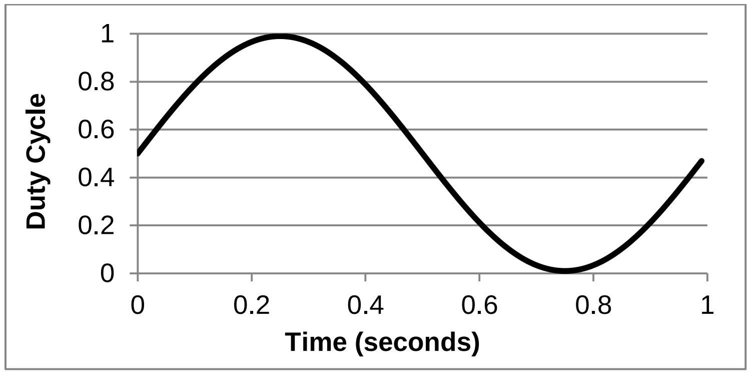


Figure 1. Plot of Duty=H/10000 as a function of time.

Figure 2 shows the MSP432 LaunchPad. You will use the red LED connected to P1.0. You will use switch 1 (SW1) connected to P1.1 and switch 2 (SW2) connected to P1.4.

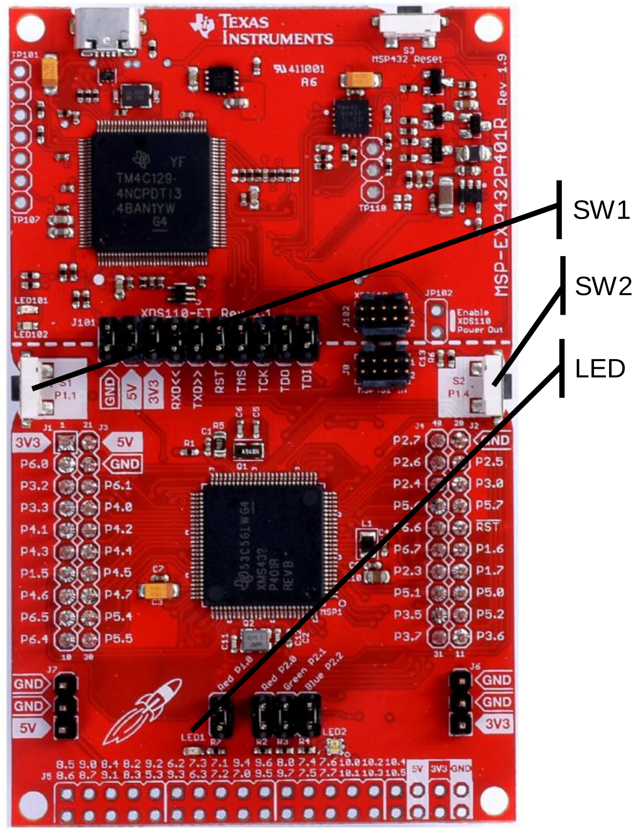


Figure 2. The LaunchPad without external circuits are used for this lab.

Lab: SysTick Timer

9.3 Experiment set-up

9.3.1 Hardware for periodic heartbeat

The LED breathing will be implemented with the MSP432 LaunchPad, without need for additional circuits, see Figure 3.

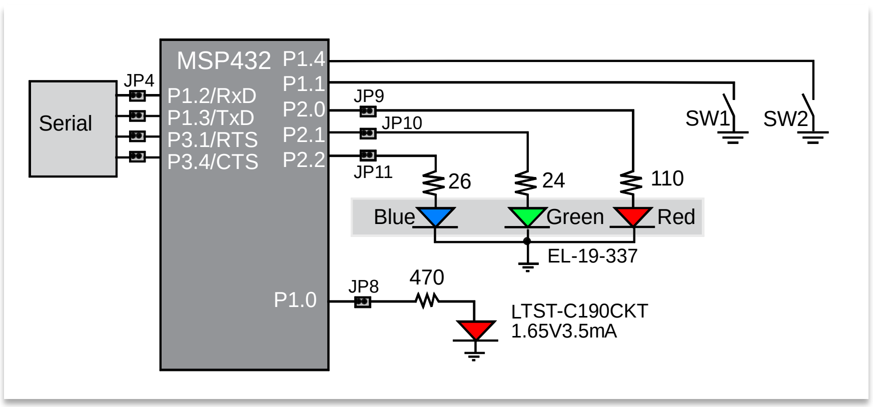


Figure 3. Create the periodic heartbeat on P1.0.

9.3.2 Hardware for PWM DAC

To implement the PWM DAC, you will need to build an analog low pass filter. The voltmeter and oscilloscope should be connected across the capacitor, see Figure 4.

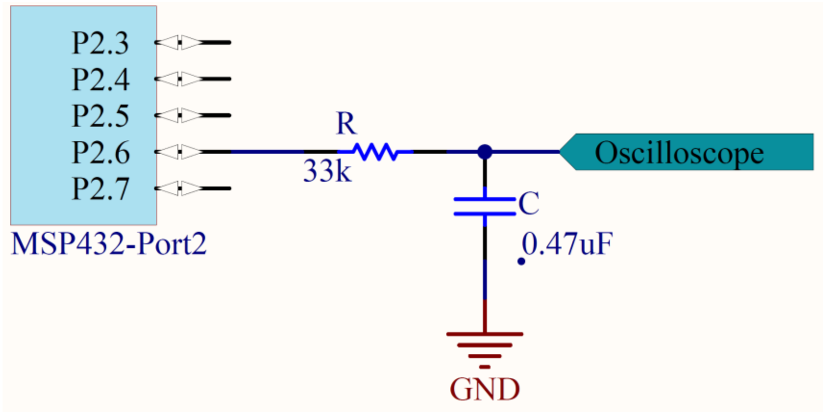


Figure 4. Use an external passive 10 Hz analog low pass filter to convert the PWM signal (P2.6 in this case) into a DAC analog output voltage.

9.4 System Development Plan

9.4.1 SysTick Wait

The first step is to write, develop and test the SysTick wait function.

The following is a software driver function that initializes **SysTick**. In this lab, we will not use interrupts. This initialization function is called once at the beginning of the main program, but before the software uses **SysTick**.

The prototype for this function is:

```
void SysTick_Wait1us(uint32_t delay);
```

where **`delay`** is the prescribed time to wait in µs. You may assume **`delay`** is greater than 2 µs and less than 349,000 µs.

```
// SysTick Initialization
void SysTick_Init(void){
  SysTick->LOAD = 0x00FFFFFF;  // maximum reload value
  SysTick->CTRL = 0x00000005;  // enable, no interrupts
}
```

Lab: SysTick Timer

The sequence of steps for the SysTick wait function are:

1. Write a desired value into the SysTick `LOAD` register
2. Clear the `VAL` counter value, which also clears the `COUNT` bit
3. Wait for the `COUNT` bit in the SysTick `CTRL` register to be set

Program9_1 shows how to test the wait function by creating a 75% duty cycle digital output. Use a logic analyzer or oscilloscope to verify the proper timing of the wait function. The signal should be high for about 7.5 ms, and low for 2.5 ms.

```
int Program9_1(void){
  Clock_Init48MHz();  // makes bus clock 48 MHz
  SysTick_Init();
  LaunchPad_Init();   // buttons and LEDs
  TExaS_Init(LOGICANALYZER_P1);
  while(1){
    P1->OUT |= 0x01;    // red LED on
    SysTick_Wait1us(7500);
    P1->OUT &= ~0x01;  // red LED off
    SysTick_Wait1us(2500);
  }
}
```

9.4.2 Generate a PWM Output

The second step is to extend the operation to implement digital waves with a sinusoidally-varying duty cycle. For example, if ***H*** = 5000, then ***L*** will be 5000, and the LED will have 50% brightness. Alternately, if ***H*** = 100, then ***L*** will be 9900, and the LED will have 1% brightness. To output a wave with fixed frequency and with fixed duty cycle, the main loop will implement these four steps in this order, over and over

1. Set P1.0 high
2. Wait ***H*** µs using your `SysTick_Wait1us` function
3. Clear P1.0 low
4. Wait ***L*** µs using your `SysTick_Wait1us` function

PulseBuf is a ROM-based table consisting of 100 pulse-times, in units of µs, which constitute a sinusoidally-varying duty cycle. Because 100*10 ms is one second, one way to create the sinusoidally-varying heartbeat is execute the following sequence over and over. If you execute steps 1 – 7 over and over again, each time through the loop using a new *H* value, the LED will flash at 1 Hz.

1. Look up a new ***H= PulseBuf [i]*** value
2. Calculate ***L*** = 10000-***H***
3. Set P1.0 high
4. Wait ***H*** µs using your `SysTick_Wait1us` function
5. Clear P1.0 low
6. Wait ***L*** µs using your `SysTick_Wait1us` function
7. ***i*** = ***i*** +1, if ***i*** ==100, roll back to ***i*** =0
8.

```
// Array used in this lab to create sine wave
const uint32_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};
```

Use an oscilloscope or logic analyzer to test your solution.

Notice, however, that this method of creating a PWM output will require all of the processor's attention. Once we start putting the modules together on the robot, we will create PWM outputs using hardware timers (in the Timers module) so the PWM generation will not require exclusive attention of the software. For now, however, the goal is to simply understand PWM.

9.4.3 Add Switch Functionality

The third step is to add the switch functionality; such that one switch starts the heartbeat and another switch stops the heartbeat. Switch bouncing does not matter in this lab, because you can ignore the start button while the heartbeat is active, and ignore the stop button while the heartbeat is inactive.

9.4.4 Create PWM DAC

The fourth step is to design a digital to analog converter using the PWM output. There are two motivations for this section. First, a DAC is inherently useful device, and using PWM to implement a DAC provides for low-cost, high-resolution implementation for signals less than 1 kHz. Second, the RC circuit in this section mimics the behavior of the motor, so we can consider the voltage output of this circuit to be analogous to the power delivered to the motor.

Recall the frequency of the digital wave is 100 Hz. In the PWM method, the frequency will be fixed. The LED is indeed fast enough to respond on and off to this wave that we have created. Look in the data sheet for the HLMP-4700 LED. You will find it has a response time of 90 ns. So, while running at 100 Hz, the LED will completely turn on and completely turn off.

However, our eyes cannot detect waves at 100 Hz, which is why our eyes perceive the 75% duty wave at 75% brightness. We can use PWM to control other devices that respond slowly as compared to the 100 Hz wave. If the time constant of the device is slow compared to the PWM frequency, the device responds to the average signal (*H*/(*H*+*L*)) and not the instantaneous on and off. To see this powerful method of PWM in another example, we need to move the output to an unused pin, so the pin is not connected to any LED circuits. An example of a slow device is an analog low pass filter implemented with a resistor and capacitor, as shown in Figure 4. The cutoff frequency of the filter will be

$$f_c = 1/(2\pi RC)$$

To make this work, we need 1 Hz < f_c < 100 Hz, so the circuit passes the 1 Hz wave and rejects (or smooths) the 100 Hz wave. In fact, our eyes have a cutoff at about 10 Hz. So, we will choose

$$RC = 1/(2\pi 10\text{Hz}) \approx 0.016 \text{ sec.}$$

One possible combination is *R*=33 kΩ, and *C* = 0.47 µF. It also works at *R* = 3.3 kΩ, and *C* = 4.7 µF.

Warning: Choose a resistor value larger than 3.3 kΩ, in order to restrict the current below 3.3V/3.3 kΩ = 1 mA. Furthermore, we suggest choosing a resistor value much less than the input impedance of your oscilloscope probe.

The static test of your PWM-implemented digital to analog converter uses a voltmeter. Connect the voltmeter across the capacitor in Figure 3. **Program9_2** implements a 100-Hz wave with known duty cycle, (*H*/(*H*+*L*)) on P2.6.

```
int Program9_2(void){
  uint32_t H,L;
  Clock_Init48MHz();  // makes bus clock 48 MHz
  SysTick_Init();
  TExaS_Init(SCOPE);
  P2->SEL0 &= ~0x40;
  P2->SEL1 &= ~0x40; // 1) configure P2.6 as GPIO
  P2->DIR |= 0x40;   // P2.6 output
  H = 7500;
  L = 10000-H;
  while(1){
    P2->OUT |= 0x40;   // on
    SysTick_Wait1us(H);
    P2->OUT &= ~0x40;  // off
    SysTick_Wait1us(L);
  }
}
```

Run this program for five different duty cycles and plot the DC voltage as a function of duty cycle. Your data should look similar to Figure 5.

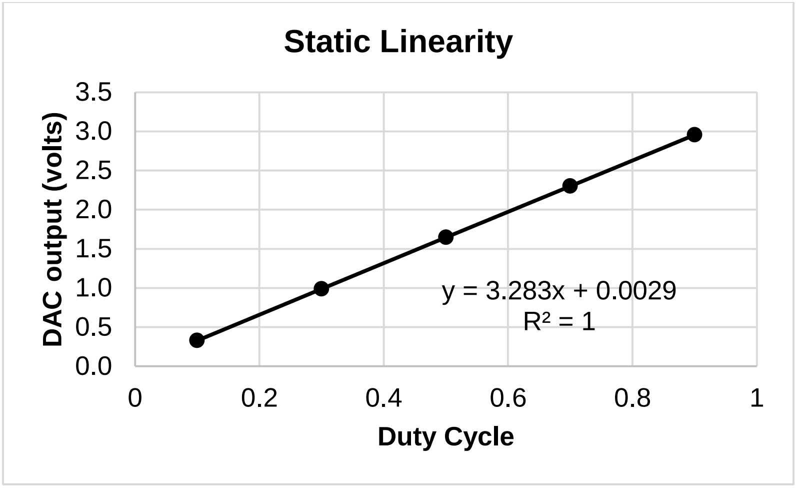


Figure 5. Example measurement data showing DAC linearity.

The above graph shows the relationship between DAC output voltage and duty cycle. Given this implementation of PWM the number of different duty cycles you can create is called the **precision** (with units of alternatives). Typically we define precision in bits.

$$Bits = \log_2(Alternatives)$$

Theoretically, if there are 10,000 alternatives, the equivalent number of bits of this DAC is approximately 13.

Resolution is the smallest change in DAC voltage that can be created. If the H were to increment by 1, In this case, if the H is incremented by 1, then the DAC analog output be (theoretically) increase by 3.3V/10000 = 0.33 mV.

Range is the maximum voltage (3.3V) minus the minimum voltage (0V). Notice that the range (in V), precision (in bits), and resolution (in V) are related.

$$\text{Range} = 2^{\text{Precision}} * \text{Resolution}$$

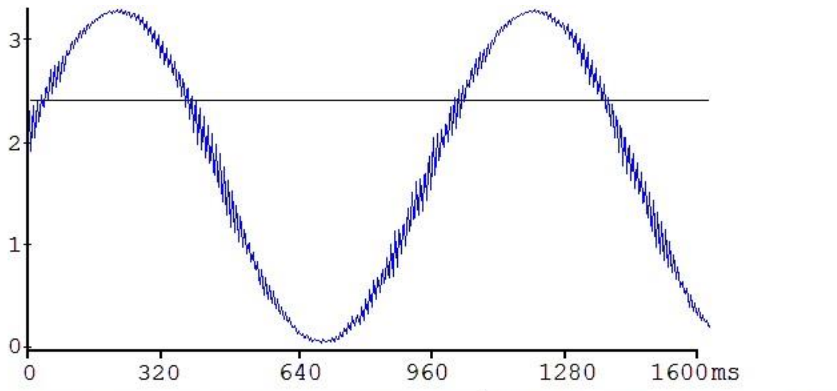
Next, let's measure the actual system performance of the circuit built from Figure 4 and compare actual to theoretical value.

Test (i) Using **Program9_2**, set H=9000 and L=1000. This will set the duty cycle to 90%. Then using a voltmeter measure the DC voltage of the DAC. The voltage on the capacitor should be about 0.9*3.3V. Let *S* (signal) be this DC voltage measurement.

Next, without changing the duty cycle, change the voltmeter setting and measure the AC voltage of the DAC. Let *N* (noise) be this measurement in volts. Calculate signal to noise ratio as *SNR = S/N.* In this measurement, we define the RMS AC voltage as the resolution of the DAC. Similarly, we approximated DAC range as the value at 90%.Therefore, the equivalent number of bits considering noise is

$$\text{Precision (bits)} = \log_2 S/N$$

Test (ii) For the same circuit as shown in Figure 4, we will use an oscilloscope. Connect the scope probe across the capacitor. Now run your sinusoidally-varying duty system and observe the output on the scope. Figure 6 shows a typical analog output, measured with the TExaS oscilloscope. Figure 6 shows the filter does not remove all the 100 Hz components; it does pass the 1 Hz, but also passes some of the 100 Hz. There is a large 100-Hz component in the signal arising from the PWM signal. If your scope has a spectrum analyzer function, you can use it to see the amplitude at 100 Hz, caused by the PWM frequency.



```
Ave=1.85V, Peak-peak=3.25V, Period=1009.8ms, Freq=  1Hz
high-pulse=413.4ms, low-pulse=596.4ms
```

Figure 6. Example analog output of the PWM DAC.

9.5 Troubleshooting

Can't program LaunchPad:

- Check the cables, jumpers on the LaunchPad development board.
- Check the Windows driver to see if the board is recognized by the operating system.
- Try another LaunchPad on this computer. Try this LaunchPad on another computer

SysTick delay is not correct:

- Make sure the MSP432 clock is running at 48 MHz.
- Make sure all the integers used in the SysTick functions all fit into 24 bits (less than 16,777,216).

LED doesn't DIM:

- Measure the port output on the scope or logic analyzer. Make sure the frequency is fixed, but the duty cycle varies.
- Echo the output onto two pins of the microcontroller (output same value to two pins). Your port pin may be damaged.

DAC isn't analog:

- Verify the resistor and capacitor values. Calculate f=1/(2πRC), f should be between 1 and 100 Hz.

9.6 Things to think about

In this section, we list thought questions to consider after completing this lab. These questions are meant to test your understanding of the concepts in this lab.

- Precision is the number of different PWM outputs that can be generated. This lab describes a system capable of creating about 10,000 different PWMs, which is equivalent to about 13 bits. What could you do to increase the precision?
- What is the relationship between the PWM period (10ms), the resolution of your SysTick timer wait (1us) and the PWM precision? Give an equation for this relationship.
- What would happen in your implementation if you tried to set the PWM period larger than 350ms?
- In what way does the RC circuit model (represent) the behavior of the visual processing of our eyes and brain?
- Why is the RC circuit classified as an analog low pass filter?

  How would you experimentally determine the frequency response of your visual system? One of the early Pokémon anime shows had a 5-sec 12 Hz scene that caused neurological responses in children (search "Pokémon induced seizures").

9.7 Additional challenges

In this section, we list additional activities you could do to further explore the concepts of this module. You could extend the system or propose something completely different. For example,

- Improve the precision by reducing the units of the timer wait function (e.g., go from 13 bits to 15 bits by reducing timer wait from 1us to 250ns)
- Remove the noise in the PWM DAC (100 Hz ripple in Figure 5), by switching from 100 Hz PWM to 1000 Hz PWM (precision will drop from 13 bits to 10 bits)
- Implement this lab using the hardware timer (we will eventually switch PWM to use the timer)

Lab: SysTick Timer

9.8 Which modules are next?

It turns out the motors using the robot also have a time constant of about 10 Hz. We will use PWM outputs to allow the software to set the power delivered to the motors. However, we cannot use 100% of the processor time to implement the PWMs for our robot. Therefore, we will use hardware timers built into the MSP432 microcontroller, so the software will be free to perform other tasks, while the hardware generates the PWMs automatically. The software will however set the period once, and adjust the duty cycle dynamically to control the behavior of the robot. These are future modules that build on the concepts learned in this module.

Module 10) Use software arrays to verify proper functionality of the system
Module 12) Use this PWM output to adjust power to the DC motor on the robot
Module 13) Use periodic interrupts to create PWM output in hardware

9.9 Things you should have learned

In this section, we review the important concepts you should have learned in this module how to:

- Measure resistance and voltage
- Measure time with a logic analyzer and an oscilloscope
- Create accurate time delays
- Implement PWM output
- Use PWM output to create time-varying behavior
- Create a simple analog low pass filter
- Balance the tradeoffs between range, resolution, and precision