CMPE 121L: Microprocessor System Design Lab

Spring 2019

Lab Exercise 1

Check-off Due in lab section, week of April 15, 2019

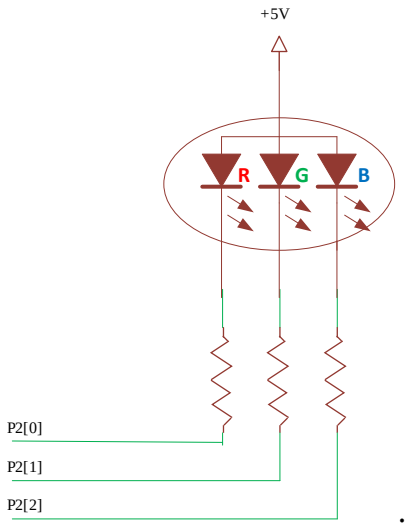
Lab Report due: Monday, April 22, 11:55 PM

In this exercise, you will learn the basics of using the GPIO pins of the PSoC-5 kit and how to monitor and control the inputs/outputs from a program. You will learn the basics of pulse-width modulation. PWM is used for control of motors, servos, etc. It can also be used to control the average DC voltage or current in a circuit (for example, it can be used for controlling the brightness of a light source) digitally. You will also learn how to use timers and counters, and some aspects of power management.

Part 1(a): GPIO Pin Toggling

The purpose of this part is to familiarize with how to configure the GPIO pins as output ports and control them from a C program. The various configuration options associated with the GPIO pins and the means of controlling them from software are described in Application Note AN-72382.

Connect the Common-Anode RGB LED to the GPIO pins as shown below. The datasheet for the LED can be found here.



1. Open the schematic editor of PSoC Creator and add three digital output pins from the component catalog. Right click on the pin symbol and click "open datasheet". Review the data sheet to understand the various configuration options. Make sure to read the Application Programming Interface (API) section.
2. Assign the pin numbers. Make sure to set the *Lock* option to force the assignment of the pins. Note that some pins have dedicated uses and may not be configurable as digital outputs. Refer to the PSoC-5 kit documentation to determine if a specific pin can be used as a general-purpose digital output pin.

3. There are three ways to control the state of an output pin from a program: (i) using Per-Pin APIs, (ii) Using Component APIs, and (iii) using a Control Register. The first two are described in the data sheet. For the third option, add a Control Register from the component catalog to the schematic and connect it to the output pins.
4. Write three programs to toggle the output pins using the three different approaches, to make the three colors go on and off at the same time.
5. Write programs to toggle one of the GPIO output pins between 0 and 1 as fast as possible in a loop, using the three different approaches. This will generate a square wave on the output pin. Using an oscilloscope, compare the frequencies of the wave generated using the three different ways of controlling the output pin. Are the frequencies the same? If not, can you think of the reasons why they are different? Which approach results in the highest frequency?
6. In your report, describe the pros and cons of each approach. When will you use each of them?

CHECKOFFS: Show the square wave generated on the output pin using the three different approaches.

1. Per-Pin method
2. Component API
3. Control Register

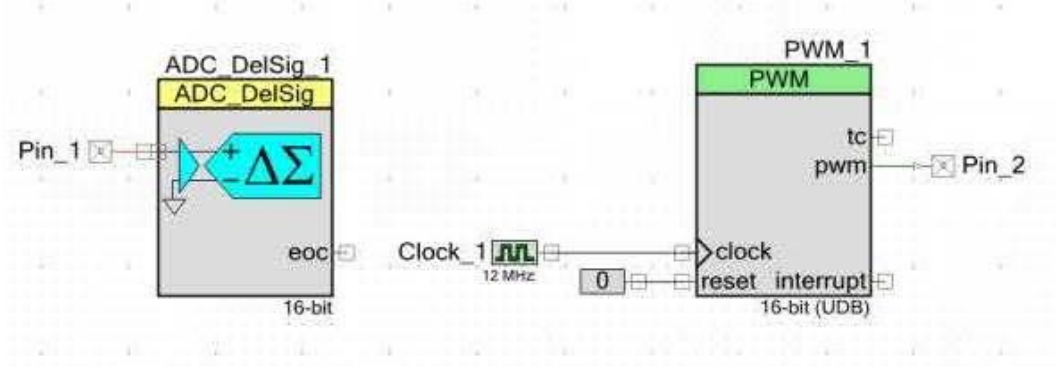
Part 1(b): LED Brightness Control using PWM

In this part, your program will continuously read the setting of a potentiometer and use it to set the brightness of one of the LEDs.

1. Using PSoC Creator, generate the datasheets of the following components and study them:
   a. PWM
   b. Delta Sigma ADC (this will be covered in more detail in class later).
   c. Clock

   The datasheets also provide the API functions associated with each of the components. You need to be familiar with them to develop the program.

2. Create a schematic with the components as shown below:



Right click to configure the Analog-to-Digital Converter (ADC):

a. Set the ADC as single ended
b. Set the output resolution to 16 bits

Set the input range between Vssa(analog ground) and Vdda(analog supply voltage). The output of the ADC should range from 0 to 0xffff (16 bits). Define the clock frequency as 1 MHz and the period of the PWM signal as 999, which results in a 1 kHz signal on the PWM output.

Connect the middle pin of the potentiometer to any unused pin of the microcontroller and the outer pins to power and ground. Connect the input terminal of the ADC (Pin_1) to the pin connected to the potentiometer, and the PWM output to the pin connected to the red LED.

a. Under the project workspace open the file [project name].cydrw
b. Assign the pins.
c. Make sure to set the *Lock* option to force the assignment of the pins.

3. Write a program to read the ADC output and control the duty cycle of the PWM output in a continuous loop. The duty cycle can be controlled by keeping the period of the PWM signal constant and changing the ON interval between 1 cycle and 999 cycles, based on the value read from the potentiometer.
4. Observe the PWM output on the oscilloscope and verify that the duty cycle changes as expected.
5. You might notice that, when the Sigma-Delta ADC is used to convert the potentiometer setting and the resolution is set to 8 bits, the reading from the ADC goes out of range at the two ends (that is, the reading may reach 0 followed by 255, or 255 followed by 0). Similarly, when the resolution is set to 16 bits, the reading may change from 0 to 65,535 at the low end, and from 65,535 to 0 at the high end. The reason for this is explained on page 6 of the ADC datasheet. The solution is to read the ADC output with higher resolution (that is, read the ADC output as a 16-bit value even though its resolution is set to 8 bits, and read it as a 32-bit value when the resolution is set to 16 bits). You can then limit the values on both ends to fit the value within the range as an unsigned number.

   For example, when the ADC resolution is set to 16 bits and you read the ADC output as a 16-bit value using ADC_Pot_Read32(), you will get a signed 32-bit number. If the ADC output jumps over from 0 to 0xffffffff, you will be reading a negative value, so all you need to do is to replace negative values with 0 to avoid a discontinuity on the low end.

Similarly, any positive value larger than 65535 can be replaced with 65535 to avoid a discontinuity at the high end.

6. Read the potentiometer in as a 16-bit and as a 32-bit value. What differences do you see? Which option will you use and why?

CHECKOFFS:

1. LED is controlled by the potentiometer.
2. LED can go from completely off to completely on.

Part 2(a): Proximity Detector

In this part, you will implement a proximity detector using the HC-SR04 ultrasonic module in the parts kit. The ultrasonic module consists of a transmitter that transmits ultrasonic pulses (at 40 kHz) and a receiver that detects echoes. By measuring the round-trip time of the pulses, it can be used to measure the distance to a nearby object.

The datasheet of the HC-SR04 ultrasonic module is posted in the Lab Resources area. Apart from power and ground, it has two signal pins. To measure the distance to an object, the microcontroller applies a 10 μs pulse on the *Trigger* pin, and monitors the state of the *Echo* pin. The ultrasonic module sets the Echo pin high on start of the measurement cycle and sets it back to low when the echo pulse returns to it. By measuring the duration of the high state of the Echo pin, the controller can measure the distance to the object that was on the path of the pulse. Based on a velocity of 340 m/sec for sound waves in the air, the distance to the object can be calculated as $d = 340 \times t/2$, where t is width of the pulse on the Echo pin.

Steps:

1. Using PSoC Creator, generate the datasheets of the following additional components and study them:
    a. Counter
    b. Timer
    c. Sleep Timer
    d. Control Register

2. Add a 1-bit control register to the design. Bring out its output on one of the pins and connect it to the Trigger pin of the ultrasonic module. This enables the software to control the state of the pin. Write some test code to toggle this pin and check if the Echo pin pulses.
3. Add an 8-bit counter to the design and configure it for count-up with one-shot operation. In this mode, the counter needs a hardware reset to start counting. This can be done by connecting its reset input to a second control register and toggling this control register from software. The counter generates a pulse on its TC output on reaching its terminal count and stops. It must be started again by applying a pulse on the reset input to initiate another count cycle. Set the terminal count so that it corresponds to an interval of 10 microseconds. Connect the TC output of the counter to the reset input of the first control

register driving the *Trigger* pin. This will synchronize the start of counting with the trigger.
Write a test loop to verify the operation of the counter, by resetting the counter repeatedly through the first control register and observing its TC output using a pin. Note that the counter has an asynchronous reset, so the reset pulse must be wider than the clock cycle.

4. You can now generate a 10 μs pulse on the Trigger pin as follows. Set the control register to 1 by software and start the counter immediately. The counter will now reset the control register to 0 after 10 μs, returning the Trigger pin to 0.
5. To determine the distance to the object, you need to measure the width of the pulse on the Echo pin. This can be done by a timer.

   Add a 16-bit timer to the design and clock it with a 1 MHz clock derived from the 24 MHz system clock. This can be done by connecting the clock input to a Clock component and setting its frequency to 1 MHz, leaving the clock source as Auto. Bring out the enable input of the timer to a pin and connect it to the Echo pin of the ultrasonic module (Note: It is always a good practice to use the Sync component to synchronize inputs to the internal clock domain of the PSoc. Insert a Sync block between the input pin and the enable input of the timer, and connect its clock input to the system clock.). You can now measure the width of the pulse on the Echo pin by enabling the timer before issuing the trigger and reading the timer value after the pulse has ended. You can avoid the software reading the timer too early by generating an interrupt on the falling edge of the pulse and reading the timer value in the ISR. Do not forget to clear the timer interrupt by reading the timer status using the API function *Timer_ReadStatusRegister()* before returning from the ISR.

   Note that timers count down, starting from an initial setting towards zero.

6. Write a main loop to continuously perform proximity sensing with the ultrasonic module. The main loop should wait for 500 milliseconds using the CyDelay() function between measurements. When it detects an object within a distance of 100 centimeters, it should immediately turn on an LED and keep it on for 5 seconds. The measurement sequence must be performed every 500 milliseconds while the LED remains on, but the LED should not be turned off before 5 seconds even if the object moves away (in the case when the object stays within range, the LED should remain ON continuously).
7. Measure the average power consumed while running the program. This requires disconnecting the USB ports and connecting the VDD pin to a 5V bench power supply. Note that the power will change while performing measurements as compared to when running the CyDelay loop. To compute the average power accurately, measure the power separately for the two cases: (i) running the CyDelay loop with no other activity, and (ii) continuously performing measurements in the main loop with no CyDelay. The average power is then the weighted sum of the two measurements. To avoid the LED power from affecting the results, turn off all LEDs while performing power measurements.
8. Assume you were to power this design from two CR2032 coin batteries. The CR2032 has a voltage of 3V, so can use two of them in series and a voltage regulator to bring the voltage down to 5V. Assume that no energy is lost in the conversion. Each CR2032 has

an energy capacity of 225 mAh. In your report, estimate the lifetime of the battery when running your design.

9. This is an application that does not require very precise measurement. Errors within a few percent are quite acceptable, so you should use integer arithmetic for all operations.

CHECKOFFS:

1. The design is able to measure the distance to nearby objects correctly and is able to distinguish between objects closer than 100 cm and those farther.
2. The LED remains ON continuously if an object stays within 100 cm and turns off 5 seconds after the object is moved out of range.
3. Show the power measurement results and explain how you calculated the total power (you don't need to repeat the measurements during checkoff).

Part 2(b): Proximity Detector with Lower Power

You can reduce the power in your design of Part 2(a) by making the system go into a low-power state between measurement cycles. Application Note AN 77900 explains the low-power states of the CPU. In this part, we will make use of the *sleep* state of the CPU to save power.

1. Start with your design of Part 2(a). Add a Sleep Timer component and set its interval to 512 ms. The Sleep Timer is clocked by the 1 kHz low-power Internal Low-Speed Oscillator (ILO) that remains active when the CPU is in a low-power state.
2. Start the Sleep Timer and enable its interrupt using its API function calls, before entering the main loop. An example for use of the sleep state can be found in the PSoC Director code example *SleepTimer_Interrupt_Sleep*.
3. In the main loop, enter the sleep state by calling *CyPmSaveClock()* to save the configuration of various clocks, followed by *cyPmSleep()* to enter the sleep state.
4. The CPU will now enter the sleep state. It will wake up after 512 ms and enter the ISR for the Sleep Timer.
5. In the Sleep Timer ISR, read the status using *SleepTimer_GetStatus()* to clear the interrupt. No other action is required. On returning from the ISR, control will return to the main loop.
6. Restore the clock configuration using *CyPmRestoreClocks().*
7. Perform the sequence for proximity sensing and control the LED, as in Part 2(a). Repeat the sequence in the main loop, thus performing a measurement every 512 ms, and staying in the sleep state between measurements.
8. Note that most hardware components in your design will lose their state when the CPU enters the sleep state. Their state must be saved before entering sleep, and must be restored when the CPU is woken up, before they can be used again. Refer to the data sheets of the components (Timer, Counter, etc.) for the API functions to save and restore their state (for example, *Timer_Sleep()* and *Timer_Wakeup()* for the Timer component).
9. All control registers hold their outputs in the sleep state, but they are reset to 0 on wake-up. This means any pin connected to a control register will be reset on wake-up. This is not a problem for the Trigger pin, but could be a problem if you have the LED connected to a pin driven by a control register. You can avoid this problem by using Per-Pin API

functions to set the LED state, rather than using a control register. See Section 3.11 of Cypress Application Note AN 77900 for a more detailed description of this issue.

10. Measure the average power consumed while running the program. This requires disconnecting the USB ports and connecting the VDD pin to a 5V bench power supply. Note that the power will change while performing measurements as compared to when sleeping. To compute the average power accurately, measure the power separately for the two cases: (i) CPU always in the sleep state with no other activity, and (ii) continuously performing measurements in the main loop with the CPU always remaining active. The average power is then the weighted sum of the two measurements. To avoid the LED power from affecting the results, turn off all LEDs while performing power measurements.
11. To further reduce the power, use the MOSFET in the parts kit to turn off power to the sensor when not performing measurements. Connect a GPIO output of the PSoC to the gate of the MOSFET to control it.
12. Proceeding as in Part 2(a), estimate the lifetime of the battery when running your design.

CHECKOFFS:

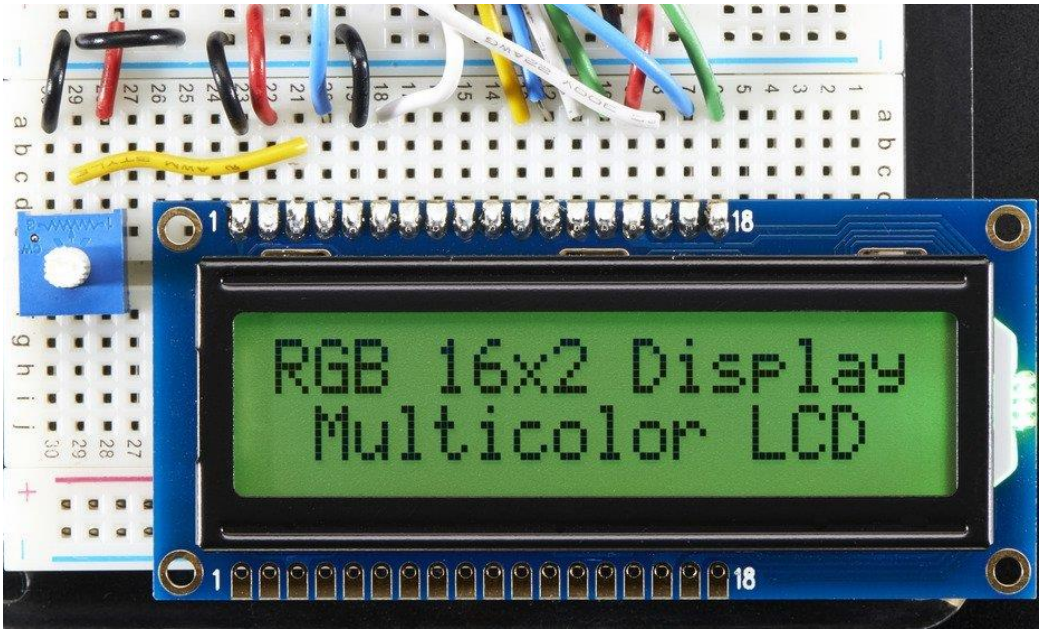
1. The design is able to measure the distance to nearby objects correctly and is able to distinguish between objects closer than 100 cm and those farther.
2. The LED remains ON continuously if an object stays within 100 cm and turns off 5 seconds after the object is moved out of range.
3. The sensor is turned off using the MOSFET when not performing measurements.
4. Show the power measurement results and explain how you calculated the total power (you don't need to repeat the measurements during checkoff).

Part 3: Frequency Meter

In this third part, you will generate a variable frequency signal and measure its frequency using a timer and counter. For this part, you will need to connect the 16x2 LCD display to the PSoC.

Interfacing the LCD Display

The 16x2 LCD display in the parts kits includes a header strip and a 10K potentiometer for controlling its contrast. Solder the 18-pin header as shown. The potentiometer is used to control the contrast for the display. There is no datasheet available for this specific display module, but it is compatible with the Hitachi HD44780 controller (datasheet posted in the Lab Resources area).



The connections between the LCD module and the PSoC board are specified in the following table. There are 7 signals between the two, which include a 4-bit data bus and three control signals. The LCD macro in the PSoC library allocates pins P2[6:0] for these signals. If you want to control the contrast of the display, you can do so by varying the voltage on pin 3 using a potentiometer (connect the middle terminal of the potentiometer to this pin and the outer pins to GND and 5V). Similarly, if you want to add backlight to the display, there are three LEDs that can be controlled through pins 15–18.

To use the LCD module in your design, add the Character LCD component in the PSoC Creator library. The API functions for using the display can be found in the datasheet for the LCD component.

LCD Pin Number	LCD Pin Name	PSoC Port/Pin	Description
1	VSS		GND
2	VDD		5V
3	V0		LCD contrast control (middle terminal of potentiometer)
4	RS	P2[5]	Register Select (selects between control and data)
5	RW	P2[6]	Selects operation (read or write)
6	E	P2[4]	LCD Enable
7	DB0	Unconnected	
8	DB1	Unconnected	
9	DB2	Unconnected	
10	DB3	Unconnected	
11	DB4	P2[0]	Data Bit 0

12	DB5	P2[1]	Data Bit 1
13	DB6	P2[2]	Data Bit 2
14	DB7	P2[3]	Data Bit 3
15	LED+		5V (backlight LED power). There is an internal current-limiting resistor within the module, so this pin can be connected to 5V directly.
16	RED		Backlight Red. Connect to GND for red backlight at full brightness. Can be connected to PWM signal to control brightness.
17	GREEN		Backlight Green. Connect to GND for green backlight at full brightness. Can be connected to PWM signal to control brightness.
18	BLUE		Backlight Blue. Connect to GND for blue backlight at full brightness. Can be connected to PWM signal to control brightness.

Draw a clean schematic of the connections between the PSoC and the LCD module before connecting them together and include it in your report.

Measuring Frequency

1. Using PSoC Creator, generate the datasheets of the following additional components and study them:
    a. Counter
    b. Timer
    c. Control and Status Registers
2. Modify the design in part 1 to control the frequency of the PWM output using the external potentiometer. The PWM must generate a square wave with frequency in the range 1 kHz – 1 MHz, based on the setting of the potentiometer (this requires changing both the period and the ON interval of the PWM block based on the value read from the ADC).
3. Add a counter block to the design.
    a. Use the UDB option to implement the counter. This will make the count input appear.
    b. Take the PWM output (not TC) and set that as the count input.
4. Add a timer block to the design. Set it up to generate an interrupt every 2 milliseconds.

5. Write the interrupt service routine (ISR). This code should read the counter value, calculate the frequency (by dividing the counter value by the period of the timer), and display the value on the LCD display.
6. Note that, to measure the frequency continuously, you should either clear the counter or remember its current value before returning from the ISR. This gives you two options to determine the number of pulses during the measurement interval. Which of these do you think will be more accurate? Try out the two options in your design and report the percentage difference in the measured values, at several different points. Include this information in your lab report.
7. Did you observe that the measurements at the low end of the range are not very accurate? How can you improve the accuracy when the frequency is low?

CHECKOFFS:

1. LCD displays potentiometer value, expected frequency, calculated frequency for the entire range 1 kHZ – 1 MHz.
2. Clear counter method is working
3. Track counter method is working
4. ISR routine is short

What to submit?

- Schematics and code for all parts (Please review guidelines in lecture slides to decide which files to submit).
- Report containing descriptions of your designs and answers to the questions above, as a PDF file
- Your project folder. Make sure to delete all generated files.
- You should also demonstrate the working design for all parts to your TA.

MOSFET references: If you are not familiar with MOSFETs, the following references will be helpful. They essentially act as switches, controlled by the gate input.

https://en.wikipedia.org/wiki/Field-effect_transistor

https://www.eeweb.com/profile/elizabeth-simon/articles/how-to-read-data-sheets-field-effect-transistors-fets