San José State University
Department of Computer Engineering
CMPE 146-03, Real-Time Embedded System Co-Design, Fall 2019

Lab Assignment 4

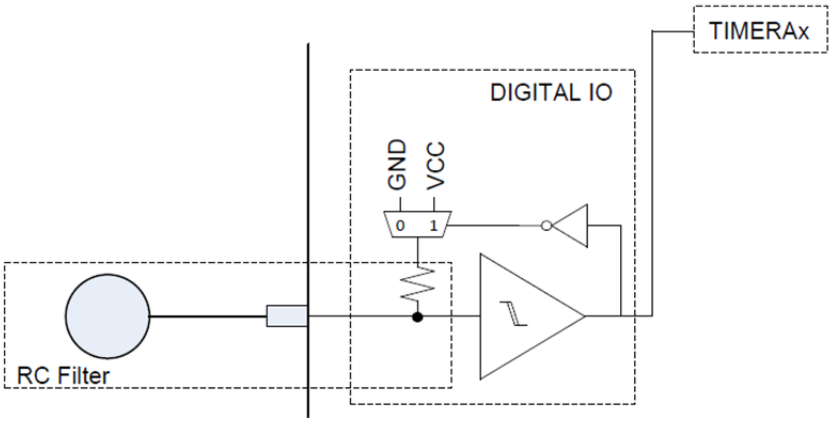
Due date: Friday, 10/18/2019

1. Description

In this assignment, we will be familarized with the GPIO's capacitive-touch capability and the Timer_A module in the MSP432 MCU. We will use the capacitive-touch capability as an input method to control a LED.

2. Exercise 1

We will use one of the 16-bit counters in the Timer_A module to sense the change of the capacitance at the GPIO pin. A GPIO port can be configured to form an oscillator, which is used to drive the counter. We measure how many clock pulses it receives in a specific time window. If the count changes significantly, that means the capacitance at the pin is changed, which means it is touched by a human.



Exercise 1.1

In this exercise, we are going to build a small function to delay a specific number of milliseconds (ms). It will be a useful function for many purposes. Duplicate one of projects that you have built on CCS Cloud. Use only one of the 32-bit timers. Use the following DriverLib functions to set up the timer.

```
Timer32_initModule(TIMER32_0_BASE, TIMER32_PRESCALER_1, TIMER32_32BIT, TIMER32_FREE_RUN_MODE);
Timer32_startTimer(TIMER32_0_BASE, 0);
```

The function prototype is as follows.

```
void delay_ms(uint32_t count);
```

The input argument *count* is the number of ms to delay. Use only the *Timer32_getValue()* DriverLib function in the function. Do not use interrupt. It is okay to use a loop to do the job. By default, the 32-bit timer is running at 3 MHz.

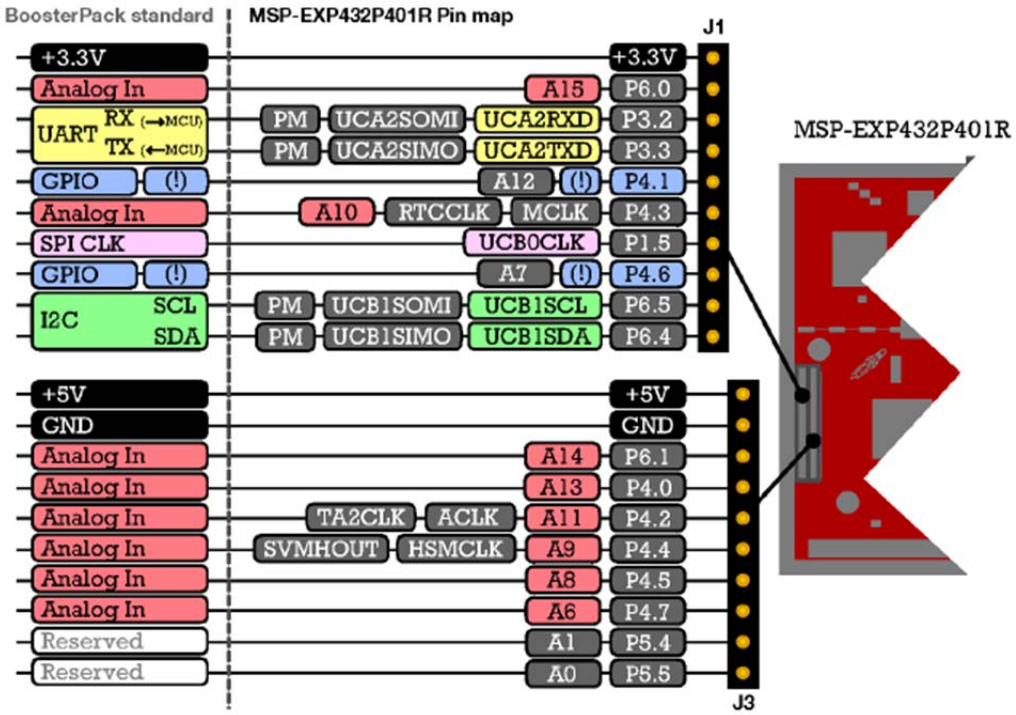
Do some testing to ensure the function is working properly. For example, use a stopwatch to see if it does delay 10 seconds by calling *delay_ms(10000)*.

Lab Report Submission

List the function. Describe how the testing is done.

Exercise 1.2

We configure the GPIO port to create an oscillator. There are several GPIO pins we can get access to on the LaunchPad's connectors. We will choose Port 4, Pin 1 (P4.1) on J1, as our input pin.



To form an oscillator, we enable the capacitive-touch feature on P4.1 by executing the following statements:

```
CAPTIO0CTL |= (1 << 8);    // Enable CAPTIO
CAPTIO0CTL |= 0b0100 << 4; // Choose Port 4
CAPTIO0CTL |= 0b0001 << 1; // Choose Pin 1
```

Incorporate these statements in your program. To read the state the of the oscillator's output, you can do something like this:

```
bool state = CAPTIO0CTL & 0x200
```

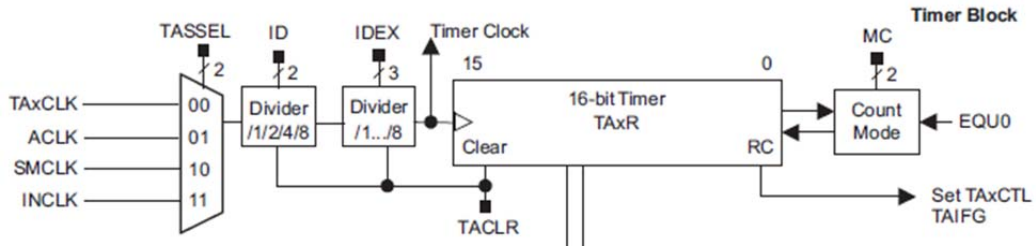
Write a loop to continually to print out the state. To see a stream of outputs, you can use *printf()* without a newline. Make sure to add a *fflush(stdout)* call after *printf()* so that the output can be seen immediately. Add a small delay using the delay function you wrote earlier so that the printing is not too fast. Use a delay from 10 to 100 ms. You should see a stream of random 1's and 0's on the screen.

Lab Report Submission

List the program in your report. Show the output stream.

Exercise 1.3

Each counter in the Timer_A module accepts one of the four input clock signals. The oscillator output from GPIO is routed to the module as the INCLK signal.



The P4.1 oscillator output is routed to the A2 couter/timer in the Timer_A module. Use the Timer_A DriverLib functions to set up the counter to count the incoming pulses. To set up the clock source for the library function, use the TIMER_A_CLOCKSOURCE_INVERTED_EXTERNAL_TXCLK type, which is essentially INCLK. Configure the counter to run in continuous mode. Set the clock divider to divide by 1, essentially not dividing because we want to be able to measure with more precision. You can consult some Timer_A example projects to see how the module can be configured.

Write a small program to see how many counter ticks will be recorded in 2 ms. Basically, before the measurement, you clear the timer so that it counts up from 0 again. Delay for 2 ms. Then read the timer register to get the count by calling *Timer_A_getCounterValue()*. Translate that to frequency and print it out to the debug console. The value tells us the approximate frequency of the oscillator.

Lab Report Submission

List the program in your report. Report the frequency computed.

Exercise 1.4

Similar to Exercise 1.2, write a loop to repeatedly do the above measurement and print out the counts. While you use your finger to touch the connector pin on P4.1, you should see the count decrease. Touching the pin adds capacitance to the pin and thus lowers the oscillator's frequency, effecting lower counts.

In the loop, add control to a LED. If the finger touches the pin, the program turns on the LED; otherwise, it turns it off.

Lab Report Submission

List the program in your report. Report the changes of the counts between touch and no tounch.

3. Grading

	Points
Exercise 1.1	2
Exercise 1.2	2

Exercise 1.3	2
Exercise 1.4	3
Report Contents	1
Total	10

Grading Policy:

The lab's grade is primarily determined by the report. We will grade the report first. Based on what we see in the demo, we may adjust the grade by either deducting or adding points.

Demo is mandatory. If you don't show up for demo, half of the points will be deducted.

If you miss the submission deadline for the report, you will get zero point. However, you can still come to the demo to show what you have done and hopefully learn something as well. And you may get a maximum of 30% of the full grade for the assignment. I will probably drop the lowest score of the lab assignments when I determine the final grade.

As for the report contents, do not use screenshots to show your codes. In the report body, list the relevant codes only for the exercise you are discussing. Put the entire program listing in appendix. Put the contents of one exercise in one section. Do not separate them to two different sections. This will make the grading easier.