ECE 362 Experiment 5: Timers

Evaluation:

IMPORTANT! You must complete this experiment during your scheduled lab period. All work for this experiment must be demonstrated and verified by your lab instructor *before the end* of your scheduled lab period.

Step		Points
4.1	Prelab	10
6.1	Configure Timer	10
6.2	Configure GPIO	5
6.3	Blink LED using timer interrupt	10
6.4	Wiring	10
6.5	Debounce keypad	20
7.0	Post lab submission	*
Total		65

***All lab points are contingent on submission of code completed in lab.**

1.0 Introduction

Each timer subsystem allows the microcontroller to produce periodic interrupts while the CPU is busy with other operations. In this experiment, we will use timer interrupts to periodically scan a matrix keypad and reliably report button presses. By recording the history of button presses with an interrupt handler, the main program is free to do other things, or it can wait for a button press to take place.

2.0 Objectives

1. To configure the timer subsystem
2. Use timer interrupts
3. To reliably debounce a keypad

3.0 Equipment and Software

1. Standard ECE362 Software Toolchain (Installation instructions available in Experiment 0: Getting Started)
2. STM32F0-DISCOVERY development board (For procurement instructions, see Experiment 0: Getting Started)

4.0 Prelab

Read through this entire lab document.

4.1 Complete the prelab on the course website.

4.2 Wire your keypad as described in section 6.4.

5.0 Background

5.1 Debouncing a keypad

Because switches are mechanical devices, they bounce. This means we do not get a stable electrical signal on a press or a release. Because of switch bounce, the voltage fluctuates between Vdd and ground before settling down to a stable value. There are several ways to fix this. Debouncing can be performed in hardware or software.

1) Hardware debouncing: You have already witnessed this in previous labs (lab 3). Hardware debouncing is done via a simple RC network to filter high frequency glitches that occur from mechanical bounce.

2) Software debouncing: In certain applications it might be beneficial to debounce switches in software rather than use an RC filter on each switch.

Software debouncing can be achieved in a variety of methods. The simplest debouncing method is to read the voltage, wait for the switch to settle and read it again. However there are more robust approaches, which are more suited for a keypad matrix. Here we describe one such method that we dub the “history method”.

In this “history” method we store N previous button states. Each button's history is represented by a single variable. This variable is referred from here on, as the 'history variable'. The least significant bit of the *history variable* represents the most recent button state and the most significant bit represents the oldest button state in history.

When a new button state is read, the *history variable* is shifted left by one, making space for the new state in the least significant bit. The new state value (i.e. 0 or 1) is ORed into the *history variable*.

Now that we have a history of button states we can look for two specific patterns to identify a button press and release. A 0b0000 0001 represents a button press and 0b1111 1110 represents a button release. Note these patterns are applicable when the button's logic is active high, else the patterns are switched. Therefore, by checking if the history variable equals 1 (i.e. 0b0000 0001) we can check for a button press. Similarly by checking if history variable is -2 (i.e. 0b1111 1110) we can check for a button release.

For this experiment we will implement the 'history' method of debouncing the keypad. For each key on the keypad, an 8-bit unsigned variable is used. The 16-key history is stored as an array in RAM. The array's first 4 elements represent the history for the first column of keys, the elements from 4-7 represent the history values of the second column of keys and so on.

6.0 Experiment

6.1 Configure timer

For this section of the experiment you will need to configure TIM6. You should type your code under “init_TIM6” in lab5.s. The code should perform the following:

- Enable clock to Timer 6.
- Set PSC and ARR values so that the timer update event occurs exactly once every 1ms.
- Enable UIE in the TIMER6's DIER register.
- Enable TIM6 interrupt in NVIC's ISER register.

Upon completion uncomment *testInitTIM6()*. If your program works correctly, it will flash the green LED. Show this to your TA.

6.2 Configure GPIO

For this section of the experiment, you will need to configure GPIO pins. You should type your code under “init_GPIO” in lab5.s. The code should perform the following

- Enable clock to Port C and Port A.
- Set PC0, PC1, PC2, PC3 and PC8 as outputs.
- Set PA0, PA1, PA2 and PA3 as outputs.
- Set up a pull down resistance on pins PA4, PA5, PA6 and PA7.

Upon completion recomment *testInitTIM6()* and uncomment *testInitGPIO()*. If your program works correctly, it will flash the green LED.

Show this to your TA.

6.3 Blink blue LED using Timer 6 interrupt

For this section of the experiment you will write the TIM6 interrupt handler. Remember to give the subroutine the correct (exact) name and define it to be thumb code. The code should perform the following things:

- Acknowledge the interrupt (refer to the class notes to see how to perform this).
- Increment a global variable 'tick'.
- Check if 'tick' is 1000. If so, toggle (hint: XOR) bit 8 of GPIOC's ODR and set 'tick' to 0.

Upon completion, uncomment *testProb3()*. If your program works correctly, it will flash the blue LED once every second. Show this to your TA

6.4 Wiring

For the next part of this experiment, we will be using the keypad, reading a key press and displaying the identified key press (i.e. key value) using the AD2's LEDs. You will need to make the following connections to AD2, connect PC0, PC1, PC2 and PC3 of the micro-controller to D0, D1, D2 and D3 of the AD2, respectively. Configure them as inputs (LED) in AD2 (refer to lab 0 for a more detailed guide on configuring them as inputs). Connect the ground of the AD2 to the ground of the micro-controller. Wire the keypad to the micro-controller as shown below.

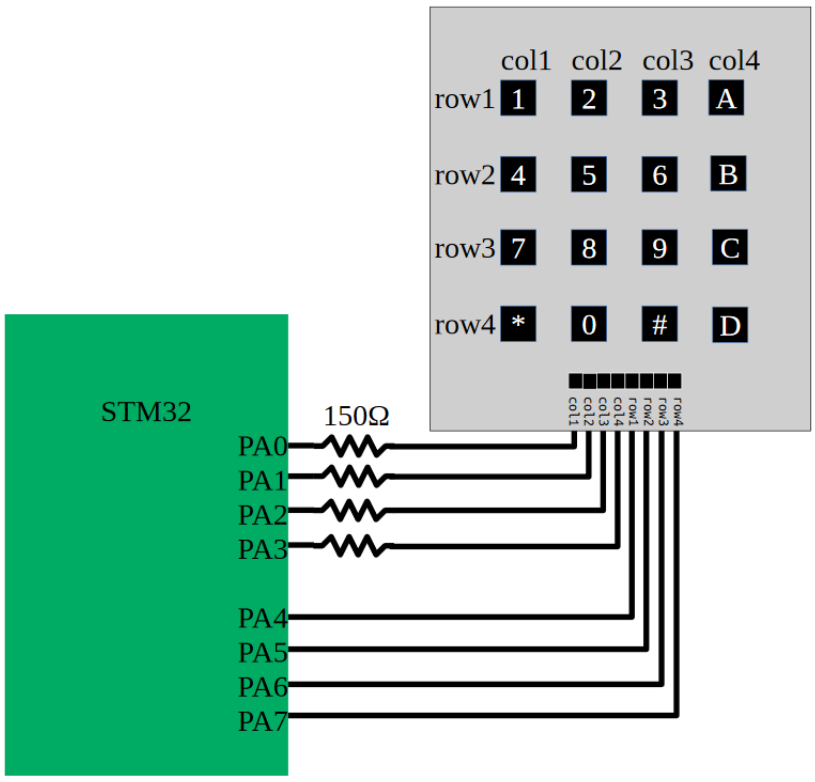

Figure 1. Wiring for lab experiment 5.

6.5 Debouncing the keypad

For this section of the experiment, we shall identify the key pressed on the key pad an report a corresponding key value (in binary) to the AD2 to visualize. The keypad needs to be debounced using the method described in the prelab. To complete this experiment, you will have to complete the following subroutines.

1) getKeyPressed(): The functionality of this subroutine is described in C below. Convert it to assembly and type your code under 'getKeyPressed' in lab5.s.

```
int getKeyPress() {
  while(1) {
    for(int i = 0; i < 16; i++) {
      if(history[i] == 1) {
        tick = 0;
        return i;
      }
    }
  }
}
```

2) getKeyReleased(): The functionality of this subroutine is described in C below. Convert it to assembly and type your code under 'getKeyReleased' in lab5.s.

```
int getKeyReleased() {
  while(1) {
    for(int i = 0; i < 16; i++) {
      if(history[i] == -2) {
        tick = 0;
        return i;
      }
    }
  }
}
```

3) Modify the ISR you wrote for section 6.3. **Add the functionality for the code described below.** Convert the following C description and add it to the Timer 6 ISR. The variable 'col' is a global variable defined in main.c.

```
// Update the selected column.
//
col = (col + 1) & 3; // col counts 0,1,2,3,0,1,2,3,...
GPIOA->ODR = (1 << col);

// index is the starting index of the four history variables
// that represent the buttons in the selected column.
//
int index  = col << 2;

// Left shift all of the history variables.
//
history[index] <<= 1;  // read history[index], shift left, store it back
history[index+1] <<= 1;
history[index+2] <<= 1;
history[index+3] <<= 1;

// Read the row indicators for the selected column.
// We put this off after asserting the column line so that
// the capacitance in the button matrix has been overcome
// and the row lines will be stable.
// This is probably not necessary for your current push button
// matrix with 150 Ohm resistors, but if you had a much higher
// resistance and higher capacitance, it would be.
//
int row = (GPIOA->IDR >> 4) & 0xf;

// OR the new key sample into the each history variable.
//
history[index] |= row & 0x1;
history[index+1] |= (row >> 1) & 0x1;
history[index+2] |= (row >> 2) & 0x1;
history[index+3] |= (row >> 3) & 0x1;

return;
```

Upon completion, uncomment *testDebounce()*. Open the 'waveforms' tool for AD2 and set DIO 0, DIO 1, DIO 2 and DIO 3 as LEDs. Press a key on the key pad, and observe the corresponding key code in binary. The list below described the key and binary value combination.

Show this to your TA

Key on keypad	Binary value
0	0b0111
1	0b0000
2	0b0100
3	0b1000
4	0b0001
5	0b0101
6	0b1001
7	0b0010
8	0b0110
9	0b1010
A	0b1100
B	0b1101
C	0b1110
D	0b1111
*	0b0011
#	0b1011

7.0 Complete post lab and submit code

Submit your code and complete the post lab on the website. The portal for post-lab will close 10 minutes after your scheduled lab.

8.0 Clean up your lab station and log out

Demonstrate to your TA that your lab station is clean and that you have logged out of your workstation