Pre-lab Assignment 6

Due: Fri, 16 Oct 2020 23:59:59 (approximately 55 days ago)
[Score: 17 / 20 points possible]

For this lab experiment, you will use C to configure the digital-to-analog converter (DAC), the analog-to-digital converter (ADC), set up timers, and invoke interrupt service routines (ISRs).

Academic Integrity Statement [0 ... -100 points]

By typing my name, below, I hereby certify that the work on this prelab is my own and that I have not copied the work of any other student (past or present) while completing it. I understand that if I fail to honor this agreement, I will receive a score of zero for the lab, a one letter drop in my final course grade, and be subject to possible disciplinary action.

Raghuram Selvaraj 

---

(1) GPIO Programming in C [5 points]

Write a C subroutine named **`setup_portc()`** that does the following operations:

- Enable the RCC clock to GPIOC. (Remember that you can use <ctrl>-<space> or <command>-<space> in System Workbench to autocomplete predefined constants such as RCC_APB1ENR_TIM6EN.)
- Set PC0 – PC10 as outputs.
- Write the value 0x03f to the Port C ODR.

(Yes, this has nothing to do with the DAC or ADC. It's just a warm-up for doing peripheral programming in C instead of assembly language.) Use the example on pages 29 – 31 of Lecture 09 on Embedded C for reference. The end result of this should be the digit '0' visible on the leftmost 7-segment display of the lab 5 hardware setup.

```
void setup_portc() {
    RCC->AHBENR |= RCC_AHBENR_GPIOCEN;
    GPIOC->MODER &= ~(GPIO_MODER_MODER0_1 | GPIO_MODER_MODER1_1 |
                      GPIO_MODER_MODER2_1 | GPIO_MODER_MODER3_1 |
                      GPIO_MODER_MODER4_1 | GPIO_MODER_MODER5_1 |
                      GPIO_MODER_MODER6_1 | GPIO_MODER_MODER7_1 |
                      GPIO_MODER_MODER8_1 | GPIO_MODER_MODER9_1 |
                      GPIO_MODER_MODER10_1);
    GPIOC->MODER |= (GPIO_MODER_MODER0_0 | GPIO_MODER_MODER1_0 |
                     GPIO_MODER_MODER2_0 | GPIO_MODER_MODER3_0 |
                     GPIO_MODER_MODER4_0 | GPIO_MODER_MODER5_0 |
                     GPIO_MODER_MODER6_0 | GPIO_MODER_MODER7_0 |
                     GPIO_MODER_MODER8_0 | GPIO_MODER_MODER9_0 |
                     GPIO_MODER_MODER10_0);
    GPIOC->ODR &= ~(0xffff);
    GPIOC->ODR |= 0x03f;
    return;
}
```

(2) GPIO Programming in C [5 points]

Write a C subroutine named **`copy_pa0_pc6()`** that does the following operations:

- Enable the RCC clock to GPIOA.
- Set PA0 as an input.
- Set the PUPDR to pull down PA0.
- Enable the RCC clock to GPIOC.
- Set PC6 as an output.
- Continually copy the value read from PA0 to PC6.

The end result of this circuit is that every time you press the SW2 button, the RED LED will illuminate.

Be careful here. If you change any part of the configuration for pins PA13 or PA14, you will lose the ability to debug or reprogram the STM32. When you make this mistake, consult the lab document to learn how to restore operation of the debugger/programmer.

```
void copy_pa0_pc6() {
    RCC->AHBENR |= RCC_AHBENR_GPIOAEN;
    GPIOA->MODER &= ~(GPIO_MODER_MODER0);
    GPIOA->PUPDR &= ~(GPIO_PUPDR_PUPDR0_0);
    GPIOA->PUPDR |= GPIO_PUPDR_PUPDR0_1;
    RCC->AHBENR |= RCC_AHBENR_GPIOCEN;
    GPIOC->MODER &= ~(GPIO_MODER_MODER6_1);
    GPIOC->MODER |= GPIO_MODER_MODER6_0;
    for(;;) {
        int pa0 = ((GPIOA->IDR) & 0x1);
        GPIOC->BSRR = ((1<<6)<<16) | (pa0 << 6);
    }
    return;
}
```

(3) [1 point]

When the STM32's analog-to-digital converter is operating in **10-bit** mode, how many distinct quantization levels can it resolve?

1024 

(4) [1 point]

How many **cycles** of the high-speed internal clock does the STM32's analog-to-digital converter take to make a sample-and-hold and conversion to a 8-bit result?

11 ✗

---

(5) [1 point]

How **long** does the STM32's analog-to-digital converter take to sample, hold, and convert to a 12-bit result when using the high-speed internal clock?

1893 ns ✗

---

(6) [1 point]

What external pin(s) can ADC_IN8 be connected to?

PB0 ✓

---

(7) [1 point]

Suppose the ADC is supplied with reference voltages V_{SSA}=0V and V_{DDA}=2.95V. When operating in 12-bit mode, what is the smallest change in the input voltage that the STM32's ADC can resolve? (In other words, what change in the input voltage will produce a 1-bit change to the least significant bit of the converted value?)

0.0007202148438 ✓

---

(8) [5 points]

Given an STM32 ADC operating in 12-bit right-aligned mode with reference voltages V_{SSA}=0V and V_{DDA}=3.6V, determine the converted hexadecimal values for the following input voltages: (If you're off by one binary bit too high or too low, it is acceptable. We will ascribe it to noise in the ADC.)

0.50 V: 0x239 ✓

1.10 V: 0x4E3 ✓

1.80 V: 0x800 ✓

2.20 V: 0x9C7 ✓

3.80 V: 0xE3 ✗