Department of Electrical and Computer Engineering
University of Victoria
CENG 355 - Microprocessor-Based Systems

PROJECT REPORT

Submitted on November 28th, 2017
by:
Patrick More
Rickus Senekal

Problem description / Specifications	_____/5
Design / Solution	_____/15
Testing / Results	_____/10
Discussion	_____/15
Code design and documentation	_____/15
Total	_____ /60

1.0 Problem Description

The focus of this project was on the use of a STM32F0 Discovery microcontroller (the microcontroller) and a PBMCUSLK board (the main board) to generate, monitor, and display a pulse width modulated (PWM) signal. A potentiometer on the main board manages a signal voltage that is sent to the analog-to-digital (ADC) converter on the microcontroller. Using the generated ADC voltage, the microcontroller calculates the potentiometer resistance and outputs a voltage through the digital-to-analog (DAC) converter, also located on the microcontroller. The output signal is relayed into the main board and isolated using a 4N35 optocoupler and then used to control the frequency of the NE555 Timer. The timer output (square wave) is passed back into the microcontroller and the respective signal frequency is measured. A serial peripheral interface (SPI) is used by the microcontroller to pass data to the LCD, which continually displays the timer frequency and potentiometer resistance. Figure 1 presents a view of the major system component interactions.

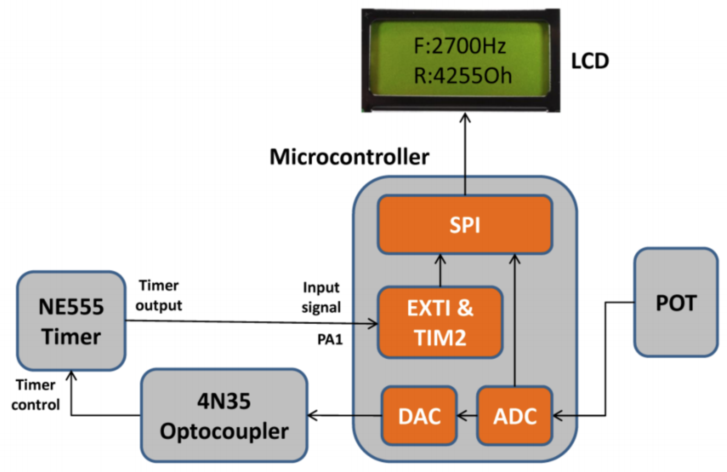

Figure 1: Block diagram of microcontroller resistance and frequency measurement system [1]

2.0 Design Solution

By the end of the second lab session, we had gained the ability to use a Function Generator to generate a square-wave signal and to display its period and frequency on the Eclipse console. The next objectives were to use the 555 Timer instead of the Function Generator, as well as, to display the signal frequency and potentiometer resistance using the LCD.

Our first step was to initialize the SPI to interface with the LCD. Once this was done, we were ready to configure the ACD to measure an input voltage from the potentiometer. Using a polling approach allowed for continual readings of the frequency, period, potentiometer voltage, and equivalence resistance. The timer circuit was then constructed and tested using external voltages. Last, but not least, the DAC was configured to act as a control signal for the newly constructed timer circuit.

2.1 LCD

The LCD portion of our project was controlled using the SPI of the microcontroller and an 8-bit serial to parallel register. The LCD we used was a P/N CM0826, which has a 4-bit, 2-by-8 character display. The LCD was mounted on a Hitachi HD44780 LCD controller. The controller can be configured to accept input in either 8 or 4-bit mode. We configured the LCD controller to use 4-bit mode by sending the word “0x02”, which was split up and sent as two different commands, 0x00 and 0x02. In 8 bit mode, which is the LCD controller initialization mode, 0x00 has no effect and 0x02 sets the LCD controller to 4-bit mode.The controller has bits zero to three hardwired low. If the controller is in 4-bit mode, the 0x02 instruction sends the cursor home. The SPI to LCD connection can be seen in Figure 2.

In order to send an 8-bit word to our 4-bit configured LCD, the sent word must be split into upper and lower parts. As shown in Figure 2, the data lines LCD_D4 to LCD_D7 correspond to the lower half of the shift register and the RS - EN lines correspond to the upper half. The process for sending data to the shift register is defined as follows:

1. Disable LCD, send high data
2. Enable LCD, send high data
3. Disable LCD, send high data

4. Disable LCD, send low data
5. Enable LCD, send low data
6. Disable LCD, send low data

The RS bit indicates whether the data being sent is of type command or data, and is always constant. The upper half of the word is sent first, with the toggling of the LCD from "Disable" to "Enable" causing a final pulse to be sent, resulting in a loaded word. The same happens with the lower half of the word.

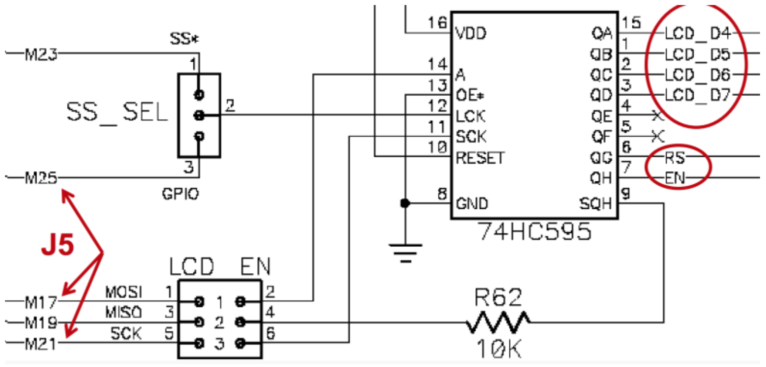

Figure 2: LCD Wiring and SPI Communication lines [2]

2.2 ADC

The ADC was used to sample the voltage across the potentiometer and calculate the resistance from the 12-bit value obtained from it. Configuring the ADC is a more involved process than the GPIO or DAC due to the particular order of register manipulation that is needed. To initialize a disabled ADC, which is the state our software assumed it was in, the ADCAL bit in the ADC_CR register must be written to a 1. This begins the calibration of the ADC and will store a calibration factor in the ADC_DR register, which is internally applied to the ADC. The status of the calibration can be read by examining the ADCAL bit which is cleared by hardware as soon as the calibration is complete. The calibration must be complete before the ADC is enabled or the ADC starts conversions. Once the calibration is complete, the ADC is enabled by setting ADEN in ADC_CR to 1 and waiting until the ADRDY flag is set by hardware in the ADC_ISR register. [3] The ADSTART bit is then set and conversions will begin based on the rest of the settings in the ADC_CR register. For this project the ADC was set to continuous and overrun management mode, which was done by setting the CONT and OVERMOD in ADC_CFGR1. This made the ADC continuously sample the input and replace the old data with the newest measurement. Once the ADC is calibrated, ready, and all the configurations are set, ADSTART is set and conversions begin. The ADC can be configured to a number of channels, however we only used channel zero which corresponds to PA0. PA0 was initialized as an analog pin in myGPIOA_Init().

2.3 NE555 Timer

The digital signal obtained from the ADC was used to adjust the frequency of the PWM signal generated by the 555 Timer. The hardware configurations of the 555 Timer is as per Figure 3. We used the DAC to convert the mentioned digital value to an analog voltage signal feeding into the 4N35 optocoupler. The optocoupler isolates the output signal and relays it into the 555 Timer. [4] Frequencies generated by the Timer are fed back into the microcontroller, where they are configured with the SPI to transmit to the LCD display. The connection configuration from the optocoupler to the timer can be seen in Figure 4.

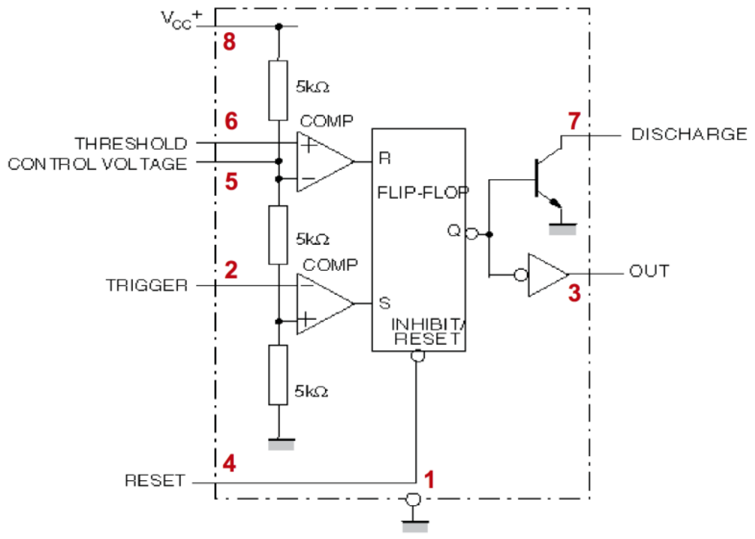

Figure 3: Block diagram of the NE555 Timer [2]

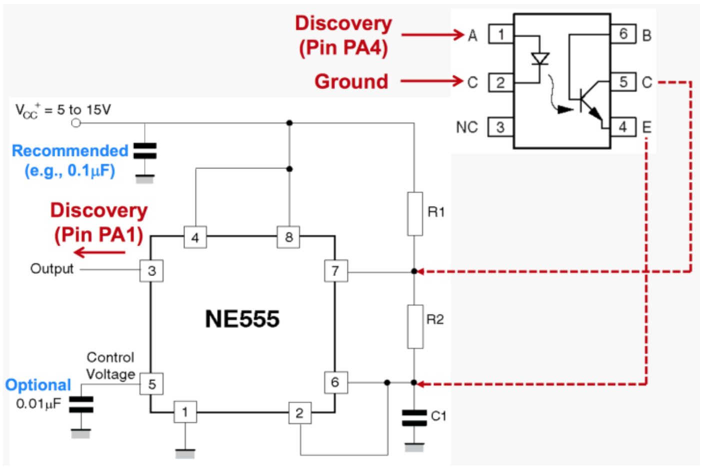

Figure 4: Summary of the 4N35 Optocoupler connection to N555 Timer [2]

2.4 DAC

The DAC was fairly trivial to setup and use compared to the SPI or DAC. There is only one pin on the STM32F0 that is capable of connecting to the DAC, which is PA4. We configured PA4 to analog mode in myGPIOA_Init() which is called before myDAC_Init(). The only thing that remains to do is enable the clock and then the DAC by setting their respective enable bits. To use the DAC, we simply write a value into one of three registers. There are two 12-bit data registers, a left aligned and a right aligned, and one 8-bit register which is right aligned. We used the 12-bit right aligned register in the main loop of the application. Additionally since changing the DAC values in the low ranges did not cause any frequency changes in the circuit for the bottom third of the values, we created a offset which normalized the range of the DAC output to the upper two thirds which was done in Set_DAC().

3.0 Test procedure and results

3.1 Frequency measurement

3.1.1 Upper limit

We measured the upper limit of our frequency measurement to be 648 KHz. This was measured by a variable that tracked the largest calculated frequency and only printed it to the console when the ADC value was at it's maximum. This suggest an interrupt overhead of roughly 74 machine instructions. This was calculated by:

$$No. instructions \approx \frac{Clock\ frequency}{Max. measurable\ frequency} = \frac{48{,}000{,}000}{648{,}000} = 74.07$$

Given the number and kind of instructions before the timer reaches a stopped state on the second detected edge this seems like a reasonable estimate of interrupt overhead.

3.1.1 Lower limit

We found the lower limit of our frequency measurement to be 0.01117 Hz which corresponds to a period of about 89 seconds. The lower limit was calculated from:

$$P = \frac{Counter\ Max.}{Clock\ frequency} = \frac{2^{32} - 1}{48 \times 10^{6}} = 89.478\ s$$

$$f = \frac{1}{P} = \frac{1}{89.478} = 0.00117\ s^{-1}$$

We measured the lower limit by setting up a timer that increments a variable every second and stopped when a “Timer overflowed” message appeared on the console.

3.2 Resistance measurement

We selected $R_1 = 330\ \Omega$, $R_2 = 5\ K\Omega$, and $C_1 = 0.1\ \mu F$. This gave us a frequency of 1396.9Hz as calculated by:

$$f = \frac{1.443}{(R_1 + 2R_2)C_1} = 1396.9\ Hz$$ [5]

The tested frequency range of the circuit was 1200 Hz - 2400 Hz. This was caused by the optocoupler changing the resistance of the R_2section of the circuit. By rearranging the above equation we can find the range of resistances of the R_2 portion of the the circuit as given by:

$$R_2 = \frac{1}{2}(\frac{1.443}{fC_1} - R_1)$$

This gives us a resistance range of 5847 to $2841\ \Omega$.

4.0 Discussion

Our design specifications follows that of the CENG 355 lab manual [1]. This section will discuss design decisions included in our project to make the process of running our application easier and more efficient.

LCD Display Placeholders

In order to maximize efficiency of writing to the LCD display, two variable lists are defined as follows:

```
char upper[9] = {'F', ':', 'h', 'e', 'l', 'p', 'H', 'z', '\0'};
char lower[9] = {'R', ':', 'm', 'e', ' ', ' ', 'O', 'h', '\0'};
```

Two separate functions, write_upper() and write_lower(), handle the variable assignments, respectively. Each of these functions use the predefined snprintf() function, which rounds our frequency and resistance values to our selected format of four significant figures and stores the formatted values in a newly created char array. This new char array is written to the LCD overwriting characters two to five , resulting in "F:____Hz" and "R:____Oh" remaining constant with only the 4 center characters continually overwritten.

LCD Re-Initialization Configuration

As referred to in Section 2.1, we discuss the transmission of the word 0x20 of type Command. There exists a problem upon the main board's start-up in which the LCD is set to its default configuration. To avoid any discrepancies with the initialization, we re-initialize the LCD into 4-bit mode upon debugging with the 0x20 command. The command is split into its high and low bits, 0x00 and 0x20. The high half 0x00 will have no effect on the controller, whereas, the low half 0x20 sets the controller into 4-bit mode. If the controller was for some reason in 4-bit mode upon startup of the main board, sending of the 0x20 command will send the cursor home. Thus, we have written our LCD initialization in such a way as to where we can freely continue further operations in 4-bit mode after the debugging phase.

References

[1] A. Jooya, K. Jones, D. Rakhmatov, and B. Sirna, CENG 355 Laboratory manual, University of Victoria, 2016.

[2] D. Rakhmatov, CENG 355 Interface Examples Lecture Slides, University of Victoria, 2017.

[3] ST Microelectronics, Reference manual, STM32F0x1, STM32F0x2, STM32F0x8 advanced ARM-based 32-bit MCUs, version 5, Jan. 2014.

[4] Vishay Semiconductors, Optocoupler, phototransistor output, with base connection, 4N35, version 1.2, Jan. 7, 2010.

[5] ST Microelectronics, General-purpose single bipolar timers, NE555, version 6, Jan. 2012.