**Lawrence Technological University**

MRE 6183 – Mechatronic Systems II

Spring 2023



**Lab 3: Tradeoffs between Hardware and Software**

Ahmed Khan Mohammed Zia

Shanawaz Hussain Mohammed

*"I pledge that on all academic work that I submit, I will neither give nor receive unauthorized*

*aid, nor will I present another person's work as my own."*

##### Lab Summary

Lab 3 was a comparative study on differences in implementing Boolean logic on hardware and through software. Relative merits for both cases were noted.

##### Prelab

We went over the Arduino code provided in Appendix A at the end. The code implements Inverted OR logic (NOT of OR) between two digital inputs at pins 2 and 3. The symbol ‘**!’** represents a NOT logic and the symbol ‘**||**’ represents OR logic. The result is then written on pin 4. The code can be executed at a higher rate by decreasing the delay in the loop function.

The second code in Appendix A reads analog values from pins A0 and A1, compares the first analog value (A0) with the second analog value (A1) using Boolean operators and outputs a logic high or logic low at digital pin 12. Reading analog values is done using the analogRead() function [1] and the digital output is generated using the digitalWrite() function [2]. The analogRead() function returns a value between 0 to 1023 for voltage between 0v to 5v, i.e. 1 unit per 4.9v at the analog pin [1]. Since there is no need to use the read values from analog pins elsewhere in the code, the analogRead() functions were used directly inside the logic instead of saving the values in variable and then use those variables in the logic.

##### Boolean operators in hardware and software

In this section of the lab, we implemented Boolean operations in hardware and software. The NAND Resistor-Transistor Logic circuit shown in Figure 1 was built on a breadboard (Figure 2).

|  |  |
| --- | --- |
| A diagram of a circuit  Description automatically generated | A circuit board with wires connected to it  Description automatically generated |
| (a) | (b) |

Figure 1: NAND RTL Circuit (a) wiring diagram and (b) built circuit

The inputs to the circuit were given as connections to either GND (0V) or VCC (5V) following the pattern listed in the truth table in Table 1 where 0 represents logic low and 1 represents logic high. The output was measured on the oscilloscope as shown in Figure 2 (a) high logic output and (b) low logic output. The behavior of the RTL circuit is the same as that of a NAND gate.

Table 1: Truth table for the given RTL circuit

|  |  |  |
| --- | --- | --- |
| **INPUT 1** | **INPUT 2** | **OUTPUT** |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

|  |  |
| --- | --- |
| A screen shot of a graph  Description automatically generated | A screen shot of a graph  Description automatically generated |
| (a) | (b) |

Figure 2: RTL Circuit output (a) High and (b) Low

Another circuit using the 7400 IC was constructed on the breadboard. As seen from the circuit diagram in Figure 3, this IC has 4 NAND gates, each with its own input pins and output pins.

|  |  |
| --- | --- |
| A diagram of a circuit  Description automatically generated |  |
| (a) | (b) |

Figure 3: NAND gate implementation using the 74HC00 IC (a) wiring diagram and (b) built circuit

The NAND IC circuit Shown in Figure 4 (b) is very convenient, neat and compact compared to the RTL circuit in Figure 1 (b). The setup effort, time and troubleshooting were greatly reduced.

The NAND IC circuit was analyzed by applying a 0-5 VDC square wave as a simultaneous signal source for inputs A and B. Figure 4 shows the delay in polarity change for the input and output signals at 100 Hz for (a) 74HC00 IC and (b) Arduino UNO R4. Delay over other frequencies on a logarithmic scale (from 102 Hz to 106 Hz) was observed for both cases and plotted in Figure 5. Software-based NAND logic implemented on Arduino UNO R4 using the code in Appendix B.

|  |  |
| --- | --- |
| A screen shot of a graph  Description automatically generated |  |
| (a) | (b) |

Figure 4: Propagation delay at 100Hz for (a) 74HC00 IC and (b) Arduino UNO R4

Figure 5: Plot of propagation delay between output and input signals versus the input signal frequency.

From Figures 5 and 6, striking differences can be noted between the two approaches (Hardware and Software), and as such, there advantages and disadvantages for each.

In the case of Hardware approach using the 74HC00 IC, the circuit is simple to set up and the propagation delay is very low and steady, offering very precise, predictable and fast operation. But this analog approach is subjected to physical limitations such as the size of the circuit and semiconductor physics such as drift in performance over time. They also lack flexibility in adapting to other applications.

On the other hand, the Software approach offers robust performance and repeatability. The code can be written to be highly scalable and covers a wide variety of applications. But the circuitry needed to run and process the code is complicated and cannot be made readily without sophisticated machinery. Moreover, due to relatively large processing overhead, software implementations are bound to have longer propagation delays than hardware implementations.

##### FSR Alarm in Hardware and Software

In this section of the lab, we built the FSR Circuit with adjustable threshold for force. Figure 7 shows the schematic and the built circuit.

|  |  |
| --- | --- |
|  |  |
| (a) | (b) |

Figure 6: FSR circuit with adjustable force threshold.

We were able to change the force required to turn on the LED by adjusting the blue potentiometer. This can be implemented in software by reading the FSR values at one of the analogue pins of Arduino UNO R4. The FSR voltage value will be mapped from 0 to 1023 units for 0V to 5V by the DAC in Arduino UNO. This then can be compared with a threshold value between 0 (GND) and 1023 (5V). Essentially, the UA741 chip and the potentiometer will be replaced with an Arduino UNO running the logic program that compares the analogue input to a threshold value. Since this is meant to be an alarm, the output can be binary (HIGH or LOW) which then can be used to light up an LED or sound a buzzer.

The output function could be as simple as:  
digitalWrite(ALARM\_PIN, (analogRead(FSR\_PIN) > FSR\_THRESHOLD\_CONSTANT)

Hardware Implementation Advantages:

* Can be optimized for lower power consumption by adjusting resistor values.
* Threshold can be adjusted on demand

Hardware Implementation Disadvantages:

* Manual assembly is taxing
* Circuit requires more space

Software Implementation Advantages:

* Quicker and convenient to write and execute the logic code.
* Circuit is compact and space saving

Software Implementation Disadvantages:

* Need to reprogram the Arduino to adjust the threshold.
* Sophisticated IC and programming software necessary.

**Citations**

|  |  |
| --- | --- |
| [1] | Arduino LLC, "analogRead() - Arduino Reference," Arduino Reference, [Online]. Available: https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/. |
| [2] | Arduino LLC, "digitaWrite() - Arduino Reference," Arduino Reference, [Online]. Available: https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/. |

##### Appendix A: Prelab Arduino codes

1. Given Arduino code

/\*

Implements a two variable truth table

\*/

// initialize pin 4 as an output

// (it’s not necessary to initialize input pins...)

void setup() {

pinMode(4, OUTPUT);

}

// the loop routine runs over and over

void loop() {

// read the input pins

int A = digitalRead(2);

int B = digitalRead(3);

// determine output value

int Z = !(A || B)

// write to output pin

digitalWrite(4, Z)

delay(500);

}

1. Arduino code to logically compare two analog pin values

// Define the analog input pins and the digital output pin

#define A\_IN1 A0 // First analog input pin

#define A\_IN2 A1 // Second analog input pin

#define D\_OUT 4 // Digital output pin

void setup() {

// Set pin modes

pinMode(D\_OUT, OUTPUT);

pinMode(A\_IN1, INPUT);

pinMode(A\_IN2, INPUT);

}

void loop() {

// Read the analog values and use boolean operators to generate digital output

digitalWrite(D\_OUT, !(analogRead(D\_IN1)||analogRead(D\_IN2)));

delay(10); // to keep sampling rate at 100Hz

}

##### Appendix B: Arduino code to logically compare two digital pin values

// Define the analog input pins and the digital output pin

#define D\_IN1 2 // First digital input pin

#define D\_IN2 3 // Second digital input pin

#define D\_OUT 4 // Digital output pin

void setup() {

// Set output mode

pinMode(D\_OUT, OUTPUT);

pinMode(D\_IN1, INPUT);

pinMode(D\_IN2, INPUT);

}

void loop() {

// Read the digital input and use boolean operators to generate digital output

digitalWrite(D\_OUT, !(digitalRead(D\_IN1)||digitalRead(D\_IN2)));

}