Self Contained Metal Detection System

Team 19

Katherine McManus, Lauren Stiefel, Ragul Ganesh Anitha Palanivel, Aadhi Maalaven Mahesh

Binghamton University

Binghamton, United States of America

5/8/2024

Abstract

Metal detection is a crucial aspect of security and industrial applications, requiring efficient and accurate methodologies to detect metal objects and discern their sizes. In this project, the task was to design and implement a self contained metal detector. There are three different sized washers that our system is meant to detect. The system combines circuit design and the programming of a Basys3 board in order to detect the strength of the circuit's output signal, the size of the washer, and how many times each washer and the total amount of times the washers are detected.

The core of our metal detection system lies in the utilization of an induction coil, which generates electromagnetic fields. When a metal object is introduced into the vicinity of these coils, it induces eddy currents, altering the properties of the electromagnetic field. By analyzing these changes, the presence of metal can be reliably detected.

Our circuit design consists of a Colpitt Oscillator, an op-amp analog comparator, and voltage dividers. The circuit was integrated with a programmed Basys3 board where it displays the number of total times the metal is introduced along with how many times each size is introduced on the seven segment display and shows the strength of the signals output on the LEDs. Once the circuit portion of the system was completed, the output signal was connected to the Basys3 board where a program was then created using the data from the output signal.

In order to design our system, we used resources found in laboratory exercises, component data sheets, simulation tools such as PSpice, and measurement tools such as oscilloscopes and multimeters found in the labs.

After testing was completed, our system was completely self contained and met all of the requirements specified.

Table of Contents

I. Project Overview......3

A. Requirements......3

II. Technical Design Description......4

A. Circuit Design......5

B. Hardware......7

C. Software......9

III. Design Integration......11

IV. Power Budget Analysis......14

V. Design Verification......16

VI. Demonstration Results......20

VII. Appendices......21

A. Bill of Materials......21

B. Source Code......21

I. Project Overview

The task of the project was to design a circuit metal detector that uses hardware and software to display a strength meter, the size of the metal washer being detected, and count the amount of times each size and the total amount of washers that were detected. The strength meter was implemented to show the strength of detection from the output signal.

The circuit portion of the metal detector was designed by implementing electrical concepts. A Colpitts oscillator, op-amp comparators, voltage dividers, and electromagnetics via an induction coil were all included in the design of our circuit.

The hardware/software portion of the metal detector required a Basys3 board being programmed by the test ports IP and driver code. The code used the data values generated by the output signal of the circuit in order to display the required values on the FPGA displays.

A. Requirements

The design of the metal detector had to meet the following requirements. The system was meant to be self-contained, meaning it should not require any outside source of voltage or signal generation. In order to get voltage to the circuit, rechargeable (7.2V) or non rechargeable (9V) batteries need to be used. If any other voltages were needed, they were to be extracted via voltage dividers, regulators, or by using standard electrical components. That being said, no lithium based battery technologies were allowed to be used. If any signal waveform was needed to be generated it had to be done using electronic circuits, microcontrollers, or the FPGA.

To create an electromagnetic response, magnetic wire was required to be used to create an inductance coil(s). Once the coil was created they were meant to be experimentally characterized in order to find resistance and inductance values.

In regards to hardware, the Basys3 board was required to be used as the main controller. The board should display the following on the LEDs and seven segment display:

- A specific symbol should be shown / lit up to indicate the size of the washer for the duration of time it is detected.
- The total number of objects detected.
- The number of each individual size washer being detected.
- A strength meter that indicates the full range of the detection signal.

For calculations, component datasheets must be used to determine parameters. Components used must also be standard components. Non standard components should be created by placing standard components in parallel or series. Once the design was completed, a power budget analysis of electrical components was required to be calculated.

II. Technical Design Description

The system design consisted of an electric circuit where the information processed from the output signal was used to display the required information on the Basys3 board. The circuit consisted of a Colpitts oscillator, into an analog op-amp comparator, and then scaled using a voltage divider. The Basys3 board was programmed by the test port IP and driver code which used data values from the output signal of the circuit.

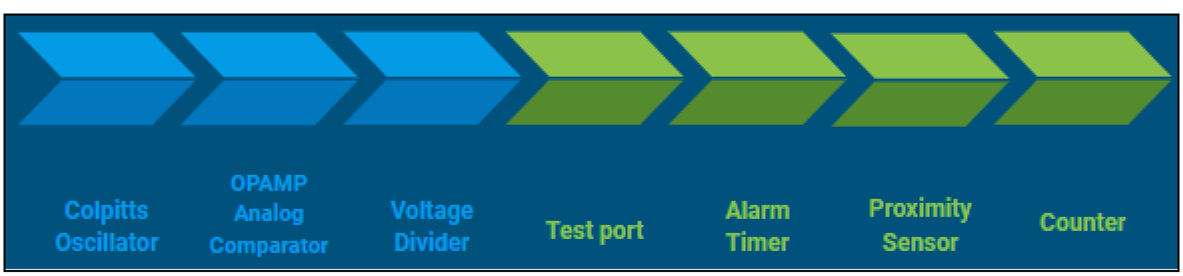

Figure 1: Full system block diagram

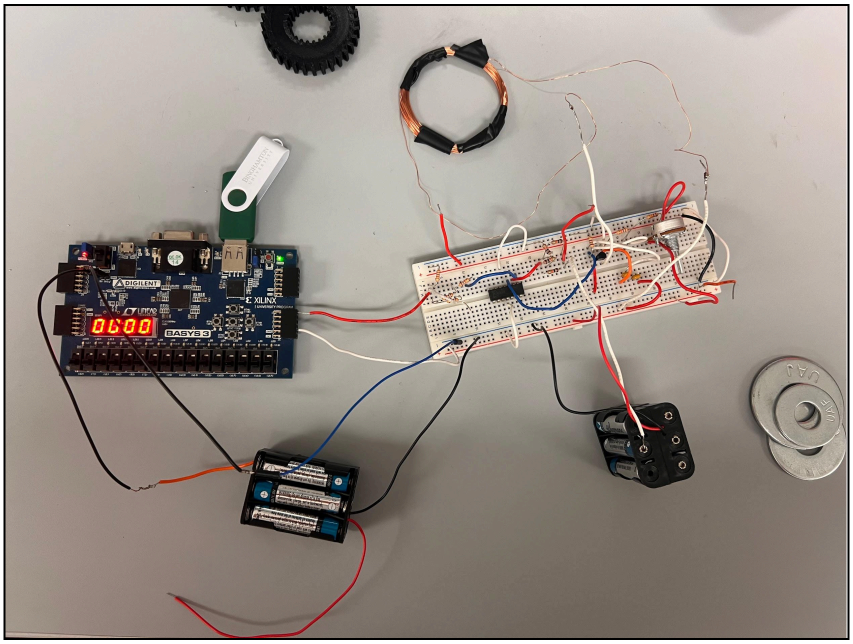

Figure 2: Full physical system

A. Circuit Design

A Colpitts oscillator was created with a 9.8V input to output a sinusoidal oscillation. The oscillation was then used as the input signal into the op-amp. Based on the peak-peak value of the oscillator, the reference voltage was chosen for the comparator at 2.5V. This was achieved by finding a 6.5V node on the battery pack and then voltage dividing it down to 2.5V. The analog comparator was used to produce a square wave which indicated when the sinusoidal wave was above the reference voltage. The peak to peak of the comparator output signal was 8.4V which majorly surpasses what the Basys3 board can take in (3.3V maximum).

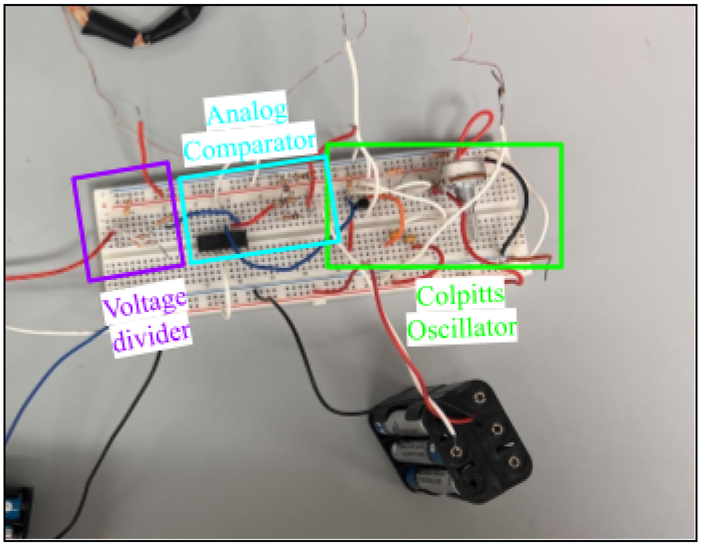

Figure 3: Physical Circuit Labeled

The schematic in Figure 4 shows an accurate representation of the voltages used as well as all of the components used to create the circuit. Important signal outputs are also labeled in order to understand the flow of the circuit easier. The inductance value was calculated by using the equation $L = \sqrt{(\frac{V}{I} - R)^2}/2\pi f$. By measuring the current and frequency across the inductor, the conclusion was made that the inductance of the coil was 6.59 mH.

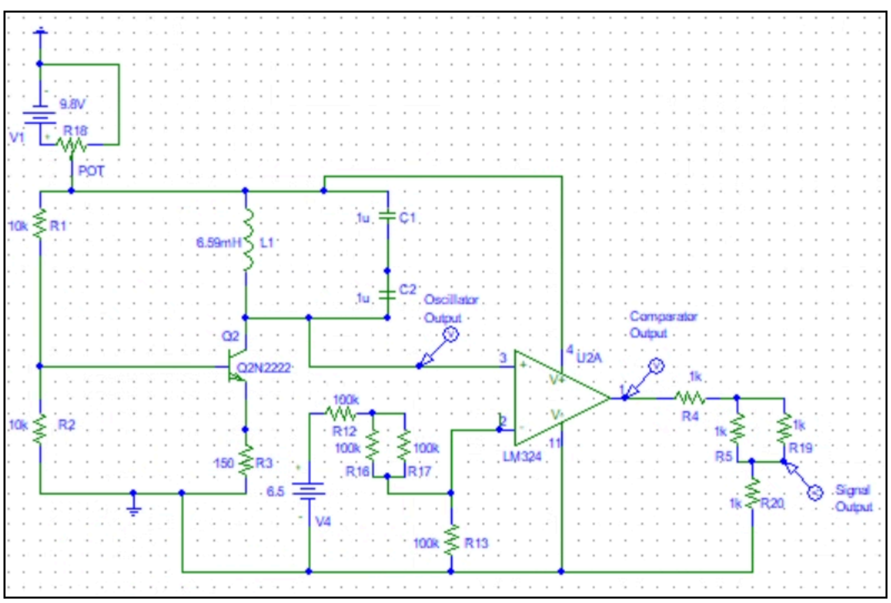

Figure 4: PSpice Schematic of circuit

B. Hardware

The Basys3 board was used as the main controller and display of the metal detector. In order to indicate required values, the seven segment display, LEDs, and switches were utilized.

To display a strength meter, the LEDs were programmed to turn on. When the signal detection is weak the leds on the right will light up. As the strength increases, the LEDs light up from right to left. When no signal is detected the LEDs turn completely off.

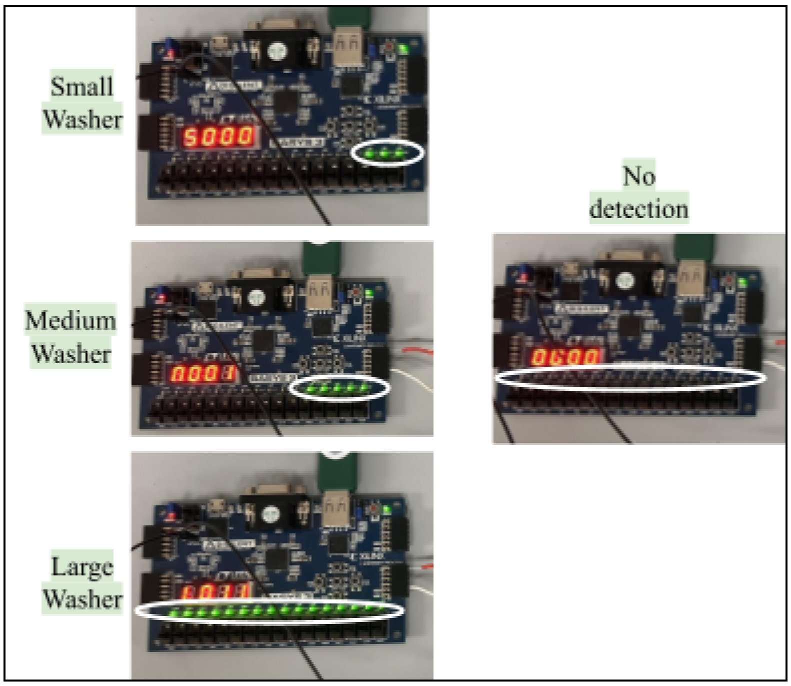

Figure 5: Strength meter depending on what size washer is being detected

When a washer is detected a “S”, “M”, or “L” is displayed on the leftmost sseg (seven segment) display based on the size of the washer being detected. The symbol is only displayed when the washer is detected while 0 is displayed otherwise. After the washer is detected, a designated counter will increment depending on the size. The small washer will increment the rightmost sseg display, the medium washer will increment the right middle sseg display, and the large washer will increment the left middle sseg display.

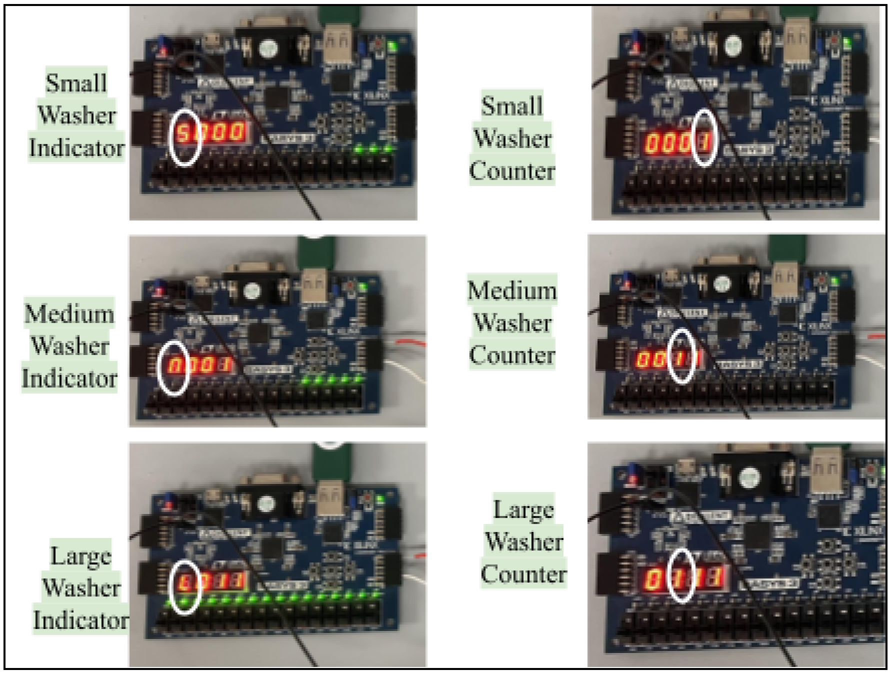

Figure 6: Size indication and size dependent counter implementation

To display the total number of washers detected, the rightmost switch needs to be flipped. This will show the total number of washers that were detected no matter the size. This was implemented because displaying the size indication was done on the sseg display meaning there were no displays left to have the total always being shown.

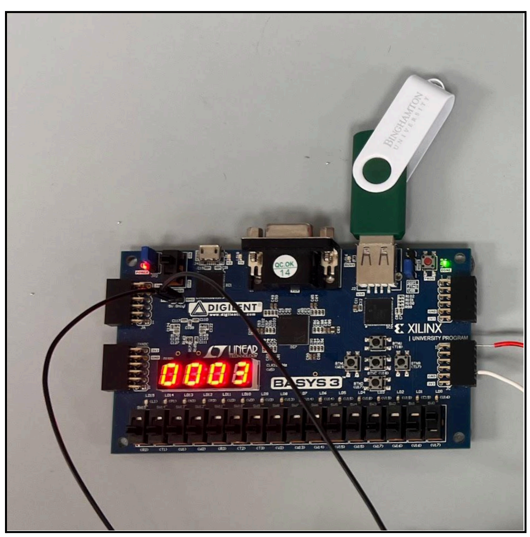

Figure 7: Total number of washers detected display

C. Software

The software was coded in C language. This language was chosen because it is supported by Vitis and is compatible with the Basys3 board. C is also commonly used when integrating electric circuits and FPGAs due to its efficiency and the ability to store values in registers as well as access memory addresses. The software consisted of two main sections, the test port IP and the driver code.

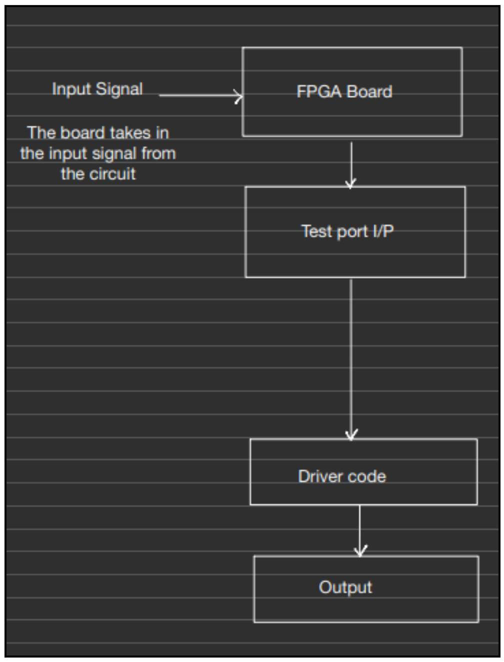

Figure 8: General block diagram

The test port IP was programmed to read the values of the duty cycle. It samples the pwm signal and measures the duty cycle for 100ms. Based on the values of the input signal, a counter is incremented. The range of values specified for each size washer determines which size specific counter is incremented while the total counter is incremented whenever any metal is detected.

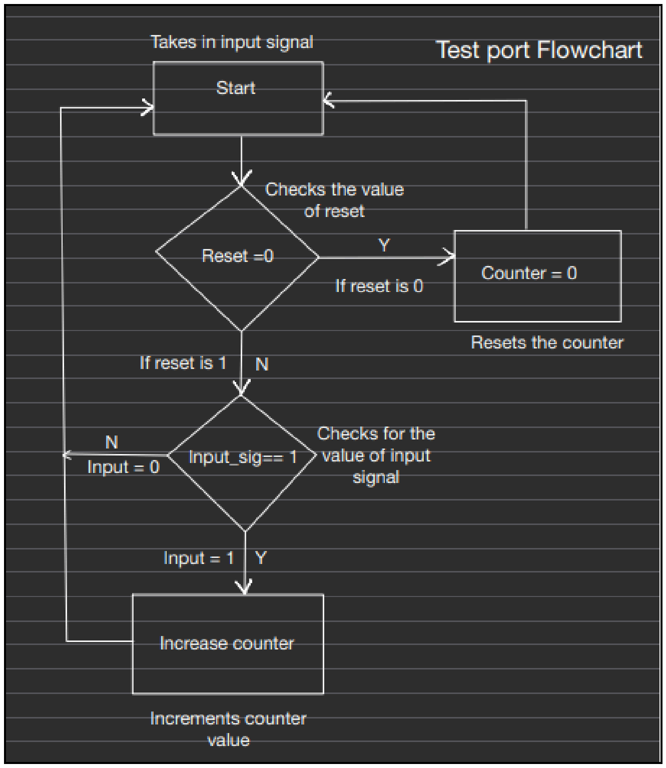

Figure 9: Test Port flow chart

Additionally, there is a process that uses a counter that is synced with the Basys3 boards internal clock. When the reset signal is high, the counter is reset to zero. When the write alarm signal is high, the process writes the counters current value to the alarm0 register which is then accessed by the software.

```
-- Counter process
process( S_AXI_ACLK ) is
begin
  if (rising_edge (S_AXI_ACLK)) then
    -- Reset values when reset is not 0
    if ( S_AXI_ARESETN = '0' ) then
      cntr        <= to_unsigned(0,32);
      alarm0_value<= to_unsigned(0,32);
      alarm0      <= '0';
    else
        -- Increment counter
        cntr <= cntr + to_unsigned(1,32);
        -- Setting alarm0 value to the write data
        if write_alarm0 = '1' then
            alarm0_value <= cntr + unsigned(S_AXI_WDATA);
        end if;
    if (w1c_alarm0 = '1' OR write_alarm0 = '1') then
        alarm0 <= '0';
    -- When counter reaches the value of alarm0, set alarm0
    elsif cntr = alarm0_value then
        alarm0 <= '1';
    end if;
    end if;
  end if;
end process;
```

Figure 10: Counter process

Once the values from the input signal are read, they are stored in a register. The driver code reads the values from the register to produce the proper output. It determines how many LEDs should be lit up by the strength meter, which counter should be incremented, and which size indicator should be displayed.

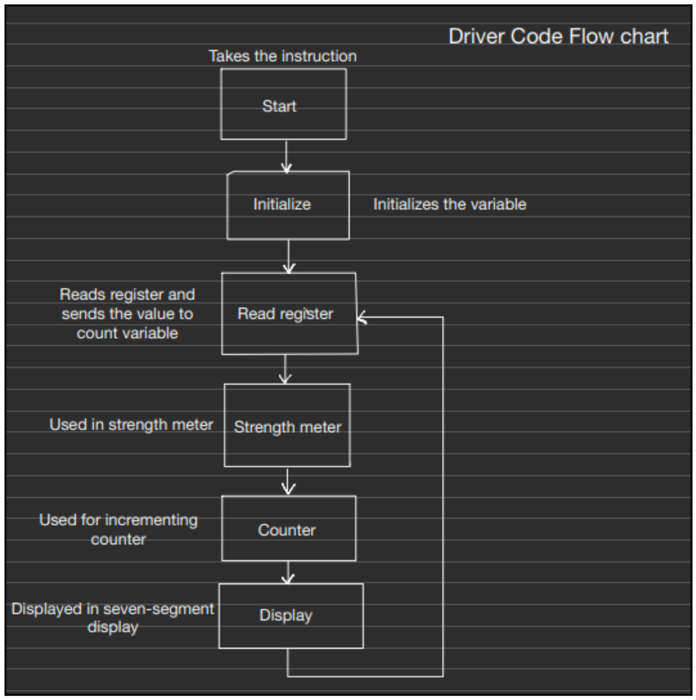

Figure 11: Driver code flow chart

III. Design Integration

In order to integrate the electric circuit system with the hardware/ software system, there needed to be a distinguishable change in the value of the waveform for each size washer. When a washer was introduced to the inductor, the duty cycle increased as the size of the washer increased. The initial duty cycle of the output signal was approximately 58.29%. When the small washer was introduced to the inductor, the duty cycle increased to 60.39%. When the medium washer was introduced the duty cycle increased to 68.52%. When the large washer was introduced, the duty cycle increased to 100%.

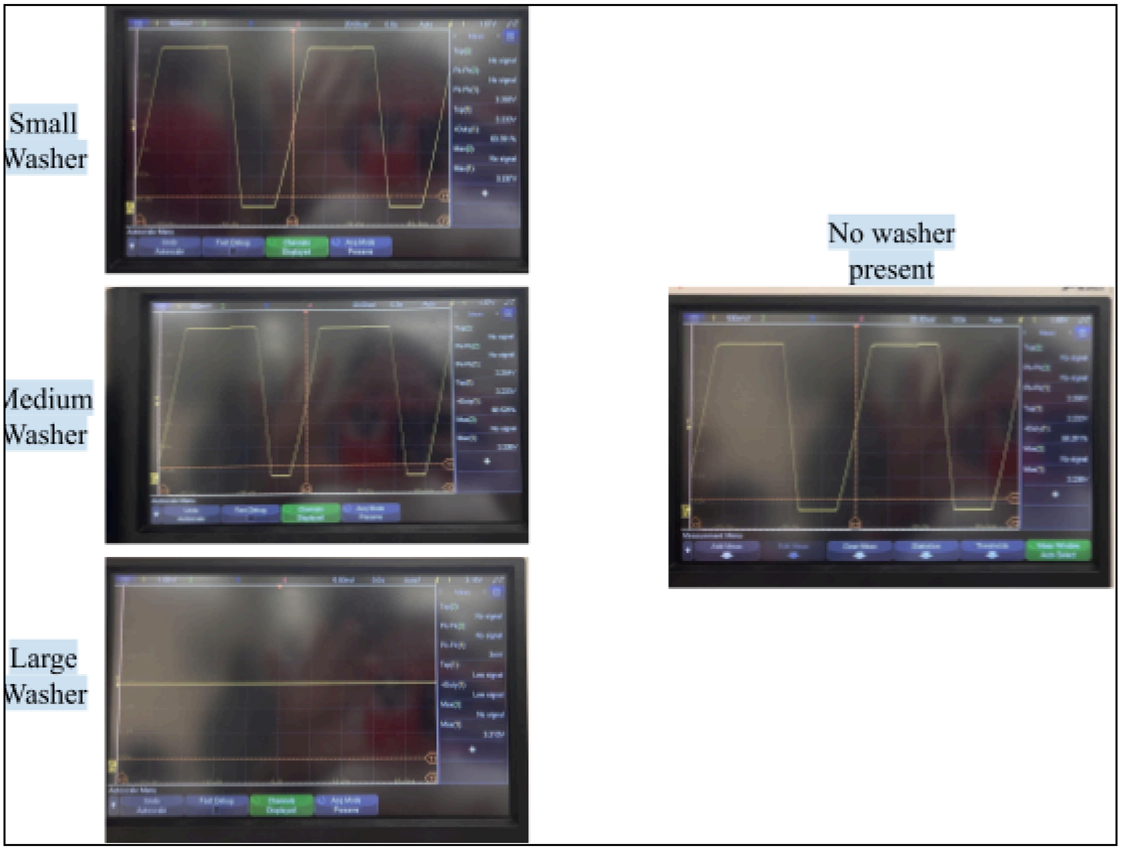

Figure 12: Output signal of circuit on oscilloscope

In order to use the information given by the signal, a custom IP was formulated. The IP takes in the PWM values (duty cycle values produced by the output signal) and stores those values in a register.

```
    -- Add user logic here
--       process( S_AXI_ACLK ) is
--     begin
--         if (rising_edge (S_AXI_ACLK)) then
--             if(sig_echo='1')then
--              cntr
process( S_AXI_ACLK ) is
    begin
      if (rising_edge (S_AXI_ACLK)) then
        if ( S_AXI_ARESETN = '0' ) then
          axi_rdata  <= (others => '0');
          echo_sync<= '0';
          cntr<=(others => '0');
        else
          echo_sync_reg <= echo_sync_reg(0) & echo;
          echo_sync <= echo_sync_reg(1);
          if (echo_sync = '1') then
           cntr <= cntr + 1;
         end if;
        end if;
      end if;
    --countr logic

end process;
    -- User logic ends
```

Figure 13: Code responsible for the custom IP

The stored value in the register is then sampled for every 100ms (variable "speed") and then stored in a new variable, "count". The difference of "count" and the originally stored value is taken in order to turn on the LEDs for the strength meter. The proximity meter turns on the LEDs and increments the counter to show the count values on the seven segment display.

```
if(speed == 100){
count = ALT_CNTR;
difference = count - prev_count;
prev_count = count;
xil_printf("%d\n",difference);
speed =0;
}
speed++;
```

Figure 14: Strength meter and counter implementation

IV. Power Budget Analysis

A power budget analysis was completed by first measuring various resistances and currents across the electrical components of the circuit. In this case, only resistors, NPN transistors, and op-amps are relevant components for the analysis. Capacitors and the inductor have an negligible impact on the power budget.

Figure 15 shows the DC voltage and current values that were experimentally found using a digital multimeter. The voltage value reflects the DC voltage across each component.

Component	Value		DC Voltage (V)		DC Current (A)
R1	1.00E+04		5.08		
R2	1.00E+04		3.94		
R3	1.50E+02		3.76		
R4	1.00E+03		2.04		
R5	1.00E+03		1.02		
R12	1.00E+05		2.56		
R13	1.00E+05		2.56		
R16	1.00E+05		1.28		
R17	1.00E+05		1.28		
R18	8.60E+03		9.03		
R19	1.00E+03		1.02		
R20	1.00E+03		2.04		
U2A	LM324		4.81		0.00338
Q2	2N2222	Base Emitter	0.18	Base Current (IB)	0.015
		Collector Emitter	5.96	Collector Current (IC)	0.15

Figure 15: Experimentally found DC voltage and current values (not including IB and IC)

When an NPN transistor is in saturation, the collector current and the base current are constant values. Figure 16 shows where on the data sheet the I_B and I_C values were found. These values are also circled in Figure 15.

Collector – Emitter Saturation Voltage (Note 1) (I_C = 150 mAdc, I_B = 15 mAdc) (I_C = 500 mAdc, I_B = 50 mAdc)	$V_{CE(sat)}$	– –	0.3 1.0	Vdc
Base – Emitter Saturation Voltage (Note 1) (I_C = 150 mAdc, I_B = 15 mAdc) (I_C = 500 mAdc, I_B = 50 mAdc)	$V_{BE(sat)}$	0.6 –	1.2 2.0	Vdc

Figure 16: 2N2222 datasheet values for IB and IC

Using the values above, the power dissipated through each component was found. The following equations were used to calculate these values:

- Resistors: $P = \frac{V^2}{R}$
- Op-amp: $P = I_{out}(V_{cc} - V_{out}) + (+V_{rail} - (-V_{rail}))I_q$
- NPN Transistor: $P = V_{BE}I_b + V_{CE}I_C$

The value of I_q in the op-amp power equation is another constant given in the datasheet. I_q is the value of the supply current drain which doesn't change when the op-amp is integrated into a design. Figure 17 shows where on the datasheet this value was given.

- Very Low Supply Current Drain (700 μA)
—Essentially Independent of Supply Voltage

Figure 17: LM324 datasheet value for Iq

The total power dissipated through the system is calculated by summing the powers of each component. In this case, the total power dissipated through the system is 1.04W. The power for each component as well as the total power can be seen in Figure 18. The equations above were put into Excel in order to calculate these values.

Power Dissipated Through:	Component	Power (W)
P = V^2/R	R1	2.58E-03
	R2	1.55E-03
	R3	9.43E-02
	R4	4.16E-03
	R5	1.04E-03
	R12	6.55E-05
	R13	6.55E-05
	R16	1.64E-05
	R17	1.64E-05
	R18	9.48E-03
	R19	1.04E-03
	R20	4.16E-03
P=Iout(Vcc-Vout) + (+Vrail-(-Vrail))Iq	U2A	2.37E-02
P=Vbe*Ib + Vce*Ic	Q2	8.97E-01
Total Power:		1.04

Constants	
Iq	0.0007
Vcc	9.8
Vee	0

Figure 18: Power budget calculations

V. Design Verification

Our design successfully met all of the requirements. It was powered using two battery packs of 6 AAA batteries. The Basys3 board required a voltage within the range of 4.5V to 5.6V. One battery pack was used to connect the Basys3 board to a 4.6V node of the battery pack so it was independently powered. The second battery pack was used to get 9.8V to the oscillator and the positive rail of the op-amp. There was a 6.2V node on the battery pack used as an input voltage through a voltage divider to get the 2.5V reference voltage for the op-amp comparator. The software was also uploaded to a flash drive in order to program the Basys3 board.

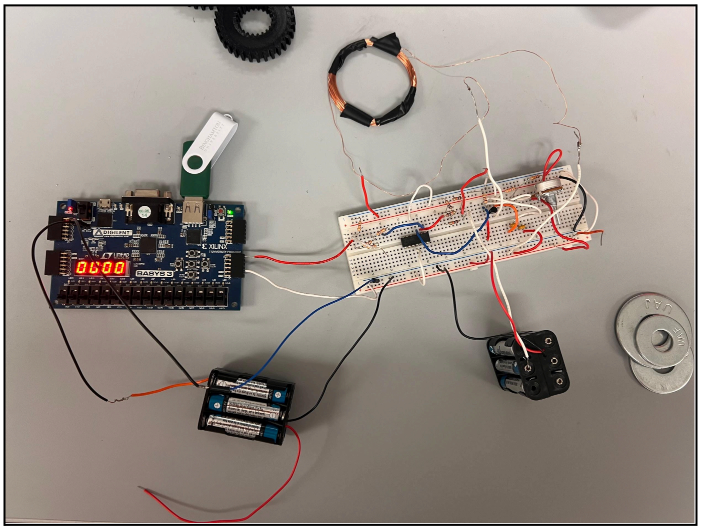

Figure 19: Circuit and Basys3 being powered by batteries and programmed by a flash drive

A Colpitts oscillator was used in order to generate an input signal of the comparator. This meets the design requirement that a signal waveform had to be implemented by circuits, microcontrollers, or the FPGA.

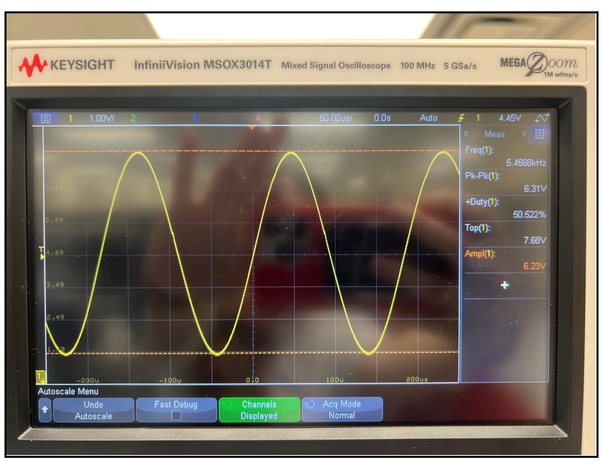

Figure 20: Colpitts Oscillator shown on oscilloscope

Magnetic wire was used to create an inductor. The inductor was made with approximately 100 turns. The inductance value was calculated using the equation: $L = \sqrt{(\frac{V}{I} - R)^2}/2\pi f$. The unknown current and frequency values were found using measurement tools in the oscilloscope and a digital multimeter. The inductance value was found to be 6.59mH [$L = \sqrt{(\frac{7.2}{16.21m} - 10k)^2}/2\pi(10.721k)$]. The resistance value was found by using a multimeter and measuring across the leads of the inductor. The resistance of the inductor used is 2.996 Ω.

The circuit only contained standard parts. Any nonstandard parts that were needed were created by placing resistors in parallel. For a voltage divider, a 2.5k resistor was needed so one 1k was placed in series with two 1k resistors in parallel to make a 2.5k resistor.

In this system the Basys3 board is the main controller. A strength meter is implemented on the LEDs to show a weaker signal (rightmost LEDs) up to a strong signal (leftmost LEDs). The program displays an “S”, “M”, or “L” depending on what size washer it is detecting. These requirements being met can all be seen in Figure 16.

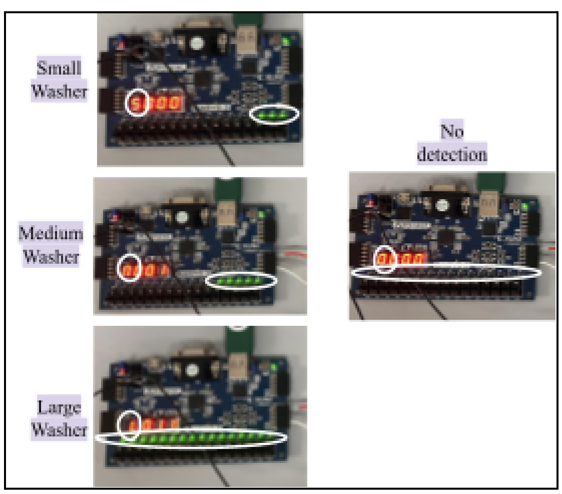

Figure 21: Size indicators and the strength meter depending on which size washer is being detected

The total number of objects detected is visible once the far right switch is flipped while the number of times each sized washer is counted is always displayed. Otherwise a seven segment display is designated for counting each size washer. The rightmost display shows the small counter, the right middle display shows the medium counter, and the left middle display shows the counter for the number of times the large washer has been detected. This can be verified in Figure 17.

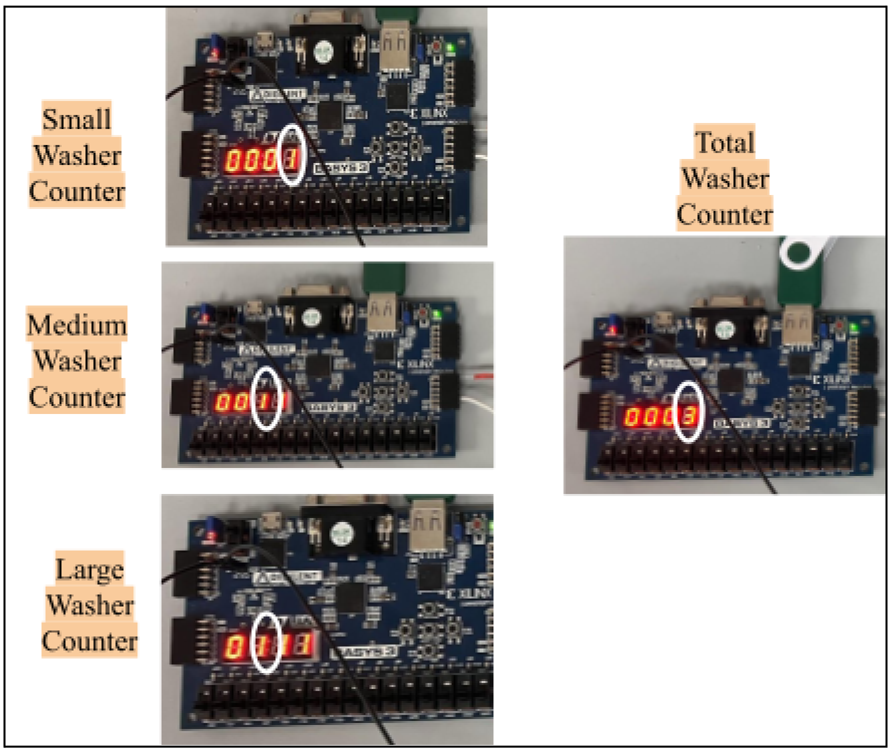

Figure 22: Size dependent counters for each washer and the total counter being displayed

After the project was completed, a power budget analysis of components was made. An oscilloscope was used in order to find the voltages across components as well as currents in and out of components. In order to complete the power budget, values were taken from the datasheet in order to perform proper calculations. Figures 23 and 24 show what content from the datasheets were used in order to perform said calculations. With this information a power budget was created using Excel in order to find the total dissipation of power.

Collector – Emitter Saturation Voltage (Note 1) (I_C = 150 mAdc, I_B = 15 mAdc) (I_C = 500 mAdc, I_B = 50 mAdc)	$V_{CE(sat)}$	– –	0.3 1.0	Vdc
Base – Emitter Saturation Voltage (Note 1) (I_C = 150 mAdc, I_B = 15 mAdc) (I_C = 500 mAdc, I_B = 50 mAdc)	$V_{BE(sat)}$	0.6 –	1.2 2.0	Vdc

Figure 23: 2N2222 datasheet values for IB and IC - first row of currents were used

- Very Low Supply Current Drain (700 μA)
  —Essentially Independent of Supply Voltage

Figure 24: LM324 datasheet value for Iq

VI. Demonstration Results

This system was successfully demonstrated for our proctor. We were required to detect each sized washer in a random order five times. We detected each size twice first and then detected a random order until each size washer was detected five times. All fifteen detections were successfully indicated on the seven segment display counters and size indicator. When the rightmost switch was flipped the total amount of washers detected was displayed on the seven segment display. During the demonstration, when the rightmost switch was flipped 15 appeared on the right two seven segment displays.

We also implemented the bonus requirement of being able to detect the washer from about 2 inches away. The display was programmed to show the duty cycle when the leftmost switch on the Basys3 board was flipped. When the large washer was placed approximately 2 inches away from the inductor, the duty cycle increased indicating the washer was being detected. Overall, the system demonstration was a success.

VII. Appendices

A. Bill of Materials

Part Number	Part Name	Value	Quantity
K104K10X7RF53L2	Capacitor	1u	2
S0603CPX150J	Resistor	150	1
S0603CPX102J	Resistor	1k	5
S0603CPX103J	Resistor	10k	2
S0603CPX104j	Resistor	100k	4
P160KN21K	Potentiometer	21k - 8.6k used	1
2N2222	NPN Transistor		1
LM324	Opamp		1
-	Inductor	6.59 mH	1
BH26AAAW	6AAA Battery Pack		2

Figure 25: Bill of Materials

B. Source Code

```
#include "xil_printf.h"

#define ONE_US 100 // 10ns * 100
#define ONE_MS 100*1000 // 1us * 100
#define INCH_CONST 1

#define ALARM_CNTR          (* (volatile unsigned *)0x44a00000)
#define ALARM0              (* (volatile unsigned *)0x44a00004)
#define ALARM1        (* (volatile unsigned *)0x44a00005)
#define ALARM0_VALUE        (* (volatile unsigned *)0x44a00008)
#define ALARM1_VALUE    (* (volatile unsigned *)0x44a0000C)
#define ALT_CNTR    (* (volatile unsigned *)0x44a10000)

#define DELAY_UNIT 81

#define LEDS   (*(unsigned volatile *)0x40000000 ) //GPIO-0 16-bit
#define SW     (*(unsigned volatile *)0x40000008 ) //GPIO-0 16-bit
#define JB     (*(unsigned volatile *)0x40010000 ) //GPIO-1 8-bit
#define DPSEG  (*(unsigned volatile *)0x40020000 ) //GPIO-2 {DP, SEG[6:0]}
#define AN     (*(unsigned volatile *)0x40020008 ) //GPIO-2 4-bit
#define BTN    (*(unsigned volatile *)0x40030000 ) //GPIO-3 4-bit, {btnR, btnL, btnD, btnU};

void delay_ms(unsigned t){
unsigned cntr1, cntr2;
while(t--)
for (cntr1 = 0; cntr1<100; cntr1++){
for (cntr2 = 0; cntr2<DELAY_UNIT; cntr2++){}
}
}
```

Figure 26: Variable Initiation for creating delay and delay _ms code

```
void seg_disp(uint8_t data[4], uint8_t cursor){
const uint8_t disp_lut[16] = {
0b00111111, //0
0b00000110, //1
0b01011011, //2
0b01001111, //3
0b01100110, //4
0b01101101, //5
0b01111101, //6
0b00000111, //7
0b01111111, //8
0b01101111, //9
0b01110111, //10
0b01101101,//0b01111100, //11 small metal
0b00110111,//0b00111001,//12 med metal
0b00111000,//0b01011110,//13 large  metal
0b01111001,//14
0b01110001,//15
};

static uint8_t digit = 0;

AN = ~(1<<(3-digit)); // enable the LEDS
DPSEG = ~disp_lut[data[digit]]; //Write data??

if(digit == 3){
digit = 0;
}
else{
digit++;
}
```

Figure 27: LUT and loop for activation of Anode and DPSEG

```c

}

int main (){
uint8_t data[4];
uint8_t cursor = 1;
uint8_t left = 0;
uint8_t right = 0;
uint8_t leftmiddle = 0;
uint8_t rightmiddle = 0;
int32_t metal_ctr = 0;
int32_t count = 0;
int32_t prev_count = 0;
int32_t difference = 0;
uint8_t speed = 0;
_Bool display;
int32_t metal;
int32_t metal_p=0;
int32_t small = 0;
int32_t med = 0;
int32_t large = 0;
int32_t small_p = 0;
int32_t med_p = 0;
int32_t large_p = 0;
int32_t small_count = 0;
int32_t med_count = 0;
int32_t large_count = 0;
print("final Launched!\n\r");
```

Figure 28: Main loop variable initialization

```
while(1){
delay_ms(1);
LEDS &= ~(1<<0);
LEDS &= ~(1<<1);
LEDS &= ~(1<<2);
LEDS &= ~(1<<3);
LEDS &= ~(1<<4);
LEDS &= ~(1<<5);
LEDS &= ~(1<<6);
LEDS &= ~(1<<7);
LEDS &= ~(1<<8);
LEDS &= ~(1<<9);
LEDS &= ~(1<<10);
LEDS &= ~(1<<11);
LEDS &= ~(1<<12);
LEDS &= ~(1<<13);
LEDS &= ~(1<<14);
LEDS &= ~(1<<15);
if(speed == 100){
count = ALT_CNTR;
difference = count - prev_count;
prev_count = count;
xil_printf("%d\n",difference);
speed =0;
}
speed++;
//led driver
if(difference>=9935596){
LEDS |= (1<<0);
LEDS |= (1<<1);
```

Figure 29: Start of code for proximity sensor

```
LEDS |= (1<<2);
LEDS |= (1<<3);
LEDS |= (1<<4);
LEDS |= (1<<5);
LEDS |= (1<<6);
LEDS |= (1<<7);
LEDS |= (1<<8);
LEDS |= (1<<9);
LEDS |= (1<<10);
LEDS |= (1<<11);
LEDS |= (1<<12);
LEDS |= (1<<13);
LEDS |= (1<<14);
LEDS |= (1<<15);
metal= 1;
large = 1;
    }
else if(difference>=9763527){
LEDS |= (1<<0);
LEDS |= (1<<1);
LEDS |= (1<<2);
LEDS |= (1<<3);
LEDS |= (1<<4);
LEDS |= (1<<5);
LEDS |= (1<<6);
LEDS |= (1<<7);
LEDS |= (1<<8);
LEDS |= (1<<9);
LEDS |= (1<<10);
LEDS |= (1<<11);
LEDS |= (1<<12);
LEDS |= (1<<13);
LEDS |= (1<<14);
metal= 1;
large = 1;
```

Figure 30: Proximity sensor (large metal)

```
}
else if(difference>=9262728){
LEDS |= (1<<0);
LEDS |= (1<<1);
LEDS |= (1<<2);
LEDS |= (1<<3);
    LEDS |= (1<<4);
LEDS |= (1<<5);
LEDS |= (1<<6);
LEDS |= (1<<7);
LEDS |= (1<<8);
LEDS |= (1<<9);
LEDS |= (1<<10);
LEDS |= (1<<11);
LEDS |= (1<<12);
LEDS |= (1<<13);
metal= 1;
large = 1;
}
else if(difference>=8963728){
LEDS |= (1<<0);
LEDS |= (1<<1);
LEDS |= (1<<2);
LEDS |= (1<<3);
LEDS |= (1<<4);
LEDS |= (1<<5);
LEDS |= (1<<6);
LEDS |= (1<<7);
LEDS |= (1<<8);
LEDS |= (1<<9);
LEDS |= (1<<10);
LEDS |= (1<<11);
LEDS |= (1<<12);
metal= 1;
large=1;
```

Figure 31: Proximity sensor (large metal)

```
}
else if(difference>=8792837){
LEDS |= (1<<0);
LEDS |= (1<<1);
LEDS |= (1<<2);
LEDS |= (1<<3);
LEDS |= (1<<4);
LEDS |= (1<<5);
LEDS |= (1<<6);
LEDS |= (1<<7);
LEDS |= (1<<8);
LEDS |= (1<<9);
LEDS |= (1<<10);
LEDS |= (1<<11);
metal= 1;
large=1;
}
else if(difference>=8478275){
LEDS |= (1<<0);
LEDS |= (1<<1);
LEDS |= (1<<2);
LEDS |= (1<<3);
LEDS |= (1<<4);
LEDS |= (1<<5);
LEDS |= (1<<6);
LEDS |= (1<<7);
LEDS |= (1<<8);
LEDS |= (1<<9);
LEDS |= (1<<10);
metal= 1;
med = 1;
}
else if(difference>=8187363){
LEDS |= (1<<0);
LEDS |= (1<<1);
```

Figure 32: Proximity sensor code(start of medium metal)

```
LEDS |= (1<<2);
LEDS |= (1<<3);
LEDS |= (1<<4);
LEDS |= (1<<5);
LEDS |= (1<<6);
LEDS |= (1<<7);
LEDS |= (1<<8);
LEDS |= (1<<9);
metal= 1;
med = 1;
}
else if(difference>=7968840){
LEDS |= (1<<0);
LEDS |= (1<<1);
LEDS |= (1<<2);
LEDS |= (1<<3);
LEDS |= (1<<4);
LEDS |= (1<<5);
LEDS |= (1<<6);
LEDS |= (1<<7);
LEDS |= (1<<8);
metal= 1;
med = 1;
}
else if(difference>=7738390){
LEDS |= (1<<0);
LEDS |= (1<<1);
LEDS |= (1<<2);
LEDS |= (1<<3);
LEDS |= (1<<4);
LEDS |= (1<<5);
LEDS |= (1<<6);
LEDS |= (1<<7);
metal= 1;
med = 1;
```

Figure 33: Proximity sensor(medium metal)

```
    }
    else if(difference>=7570000){
        LEDS |= (1<<0);
        LEDS |= (1<<1);
        LEDS |= (1<<2);
        LEDS |= (1<<3);
        LEDS |= (1<<4);
        LEDS |= (1<<5);
        LEDS |= (1<<6);
        metal= 1;
        med = 1;
    }
    else if(difference>=7275000){
        LEDS |= (1<<0);
        LEDS |= (1<<1);
        LEDS |= (1<<2);
        LEDS |= (1<<3);
        LEDS |= (1<<4);
        LEDS |= (1<<5);
        metal= 1;
        med = 1;
        }
    else if(difference>=7150000){
        LEDS |= (1<<0);
        LEDS |= (1<<1);
        LEDS |= (1<<2);
        LEDS |= (1<<3);
        LEDS |= (1<<4);
        metal= 1;
        med = 1;
            }
    else if(difference>=7000000){
        LEDS |= (1<<0);
        LEDS |= (1<<1);
        LEDS |= (1<<2);
```

Figure 34: Proximity sensor(medium metal)

```
        LEDS |= (1<<3);
        metal= 1;
        med = 1;
            }
    else if(difference>6700000){
        LEDS |= (1<<0);
        LEDS |= (1<<1);
        LEDS |= (1<<2);
        metal= 1;
        small = 1;
            }
    else if(difference>=6665000){
        LEDS |= (1<<0);
        LEDS |= (1<<1);
        metal= 1;
        small = 1;
            }
    else if(difference>=6540000){
        LEDS |= (1<<0);
        metal= 0;
        small =0;
        med = 0;
        large =0;

            }
    else {
    metal =0;
    small =0;
    med = 0;
    large =0;
    }
```

Figure 35: Proximity sensor (small and no metal)

```
//total metal counter
if(metal_p == 1){
if(metal == 0){
metal_ctr++;

}

}
metal_p= metal;
//small metal counter
    if(small_p == 1){
    if(small == 0){
    small_count++;

    }

    }
    small_p= small;
    //med metal counter
        if(med_p == 1){
        if(med == 0){
        med_count++;
        small_count--;

        }

        }
        med_p= med;
        //large metal counter
            if(large_p == 1){
            if(large == 0){
            large_count++;
            med_count--;


            }
```

Figure 36: Metal counter code for the total, small, medium, and large metal

```

            }
            large_p= large;
    //total display
    if(SW & (1<<0)){

    left = (metal_ctr / 1000) % 10;
    leftmiddle = (metal_ctr / 100) % 10;
    rightmiddle = (metal_ctr / 10) % 10;
    right = metal_ctr % 10;
    }
   else if(SW & (1<<15)){

        left = (difference / 1000000) % 10;
        leftmiddle = (difference / 100000) % 10;
        rightmiddle = (difference  / 10000) % 10;
        right = (difference  / 1000) % 10;
    }
    else{
    if(large==1){
    left=13;
    }
    else if(med==1){
    left = 12;
    }
    else if(small==1){
    left=11;
    }
    else{
    left=0;
    }
    leftmiddle = large_count % 10;
    rightmiddle = med_count % 10;
```

Figure 37: Metal counter display logic using switches

```
    right = small_count % 10;
    }
    //display data
    data[3] = right;
    data[2] = rightmiddle;
    data[1] = leftmiddle;
    data[0] = left;
    seg_disp(data, cursor);

    }
}

```

Figure 38: Code for displaying specific numbers on specific anode displays