Metal Detection System

Team 24

Georges Karam, Onkar Singh, Giri Kandasamy, Ian Tiedemann, Jonathan Fuentes Rosales

Binghamton University

Date submitted: 5/9/2025

Abstract

This report documents the development of the final project for the EECE 387 Design Lab. It is a joint effort between the electrical and computer engineering departments, aiming to design a self-contained metal detection system. This design makes use of electromagnetic coupling to detect the strength and position of a metal object, operates using the Basys3 board as the main controller and incorporates custom-designed coils characterized by their resistance and inductance.

The core of our metal detection system lies in the usage of induction coils, which generate electromagnetic fields. When a metal object is brought in proximity to the 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 5V voltage regulator, Colpitt's oscillators, and full-wave rectifiers. The circuit was integrated with a programmed Basys3 board, where it displays which position the washer is being held in and how many times the washer was held in each respective position on the seven-segment display, along with the strength of the signals' output being marked with LEDs.

In order to design our system, we used components used in laboratory assignments, component data sheets, simulation tools such as LTSpice, and measurement tools such as oscilloscopes and multimeters used in our labs. Once the circuitry of our system was working properly, the output signal was connected to the Basys3 board, where a program was then created using the data from the signal. The program uses the DC voltage values from the three full-wave rectifiers to determine the position of the washer and the strength of the washer's effect on the electromagnetic field.

Table of Contents

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, display the lateral position (left, left-center, center, right-center, or right) of where the washer is being held, and display the number of times the washer was placed in each respective position. 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. Voltage regulation, Colpitts oscillators, full-wave rectifiers, and electromagnetics by an induction coil were all included in the design of our circuit.

The hardware/software portion of the metal detector required a BASYS 3 board processing JXADC port readings and driver code. Said code took the values generated by the circuit's output signal in order to display the required positional, strength and count values on the FPGA displays.

A. Requirements

The system was meant to be self-contained, meaning it should not use any external voltage sources or waveform generators. The only voltage source used in this project was a 9V battery pack consisting of 6 1.5V batteries. If any other voltages were needed, they were to be extracted via voltage dividers, regulators, or by using standard electrical components. If any signal waveform 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 inductance coils. Once the coils were created, resistance and inductance values were measured experimentally.

The Basys3 Board was required to be used as the main controller, and the board should display the following on the LEDs and the seven-segment display:

- A specific symbol should be displayed to indicate the position of the washer while it is being detected.
- The total number of objects detected.
- The number of times that the metal object is detected and correctly identified
- A strength meter that indicates the full range of the detection signal.

The design could only use standard electronic components. This means, for example, that only standard values of resistors and capacitors were allowed to be used. Non-standard values can be obtained by parallel/series combination of parts. Potentiometers were also permitted to be used.

II. Hardware Design Description

Our 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 5V voltage regulator to power the Basys3 board and 3 Colpitts oscillators into 3 full-wave rectifiers . The Basys3 board was programmed by the XADC Wizard IP and driver code, which used data values from the output signal of the circuit.

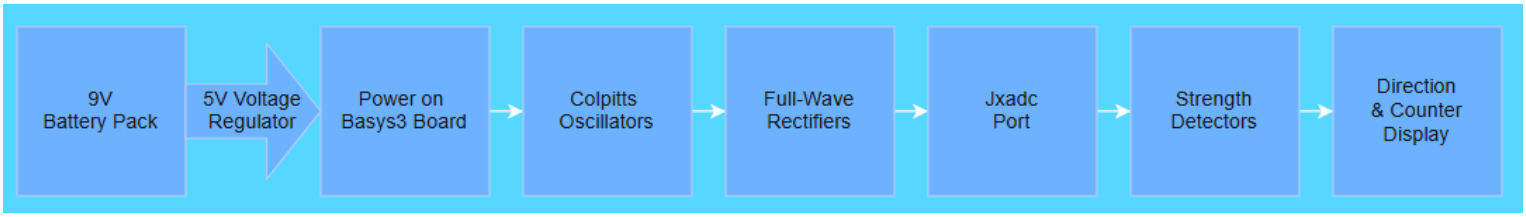


Figure 2.1: Block diagram of entire system

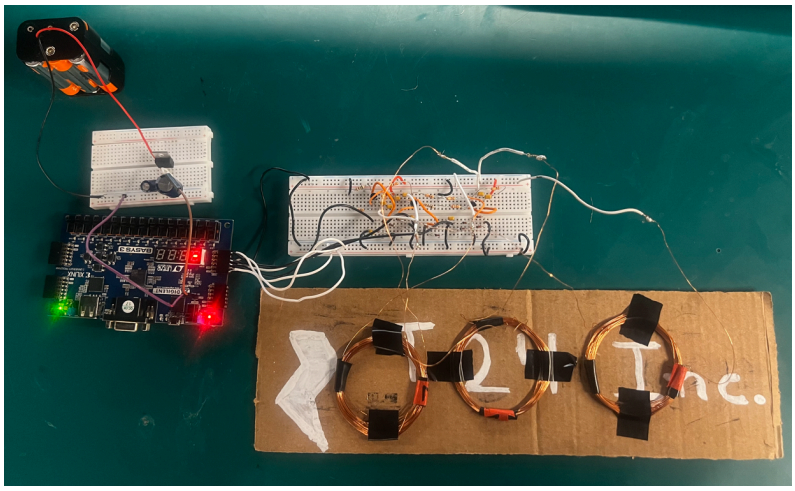


Figure 2.2: Full self contained system

A. Circuit Design

3 Colpitts oscillators, each using an individual induction coil, were designed with a 3.3V input from the Basys3 board to output sinusoidal oscillations. The sinusoidal oscillations were used as the input signals into the 3 full-wave rectifiers. The DC output values of the rectifiers and their changes when a washer is present were used in the program to display the lateral position of the washer.

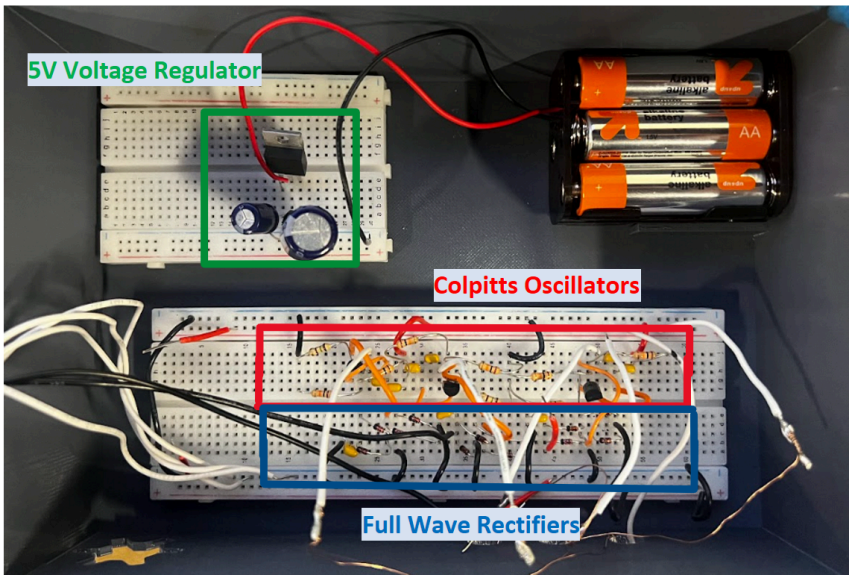


Figure 2.3: Physical Circuit Labeled

The schematic below shows a representation of the voltages and the components used to create the circuit. The inductance value was calculated using the equation

$$L = \sqrt{\left(\frac{V}{I} - R\right)^2}/2\pi f.$$

The inductance of the coils was found to be 0.12 mH.

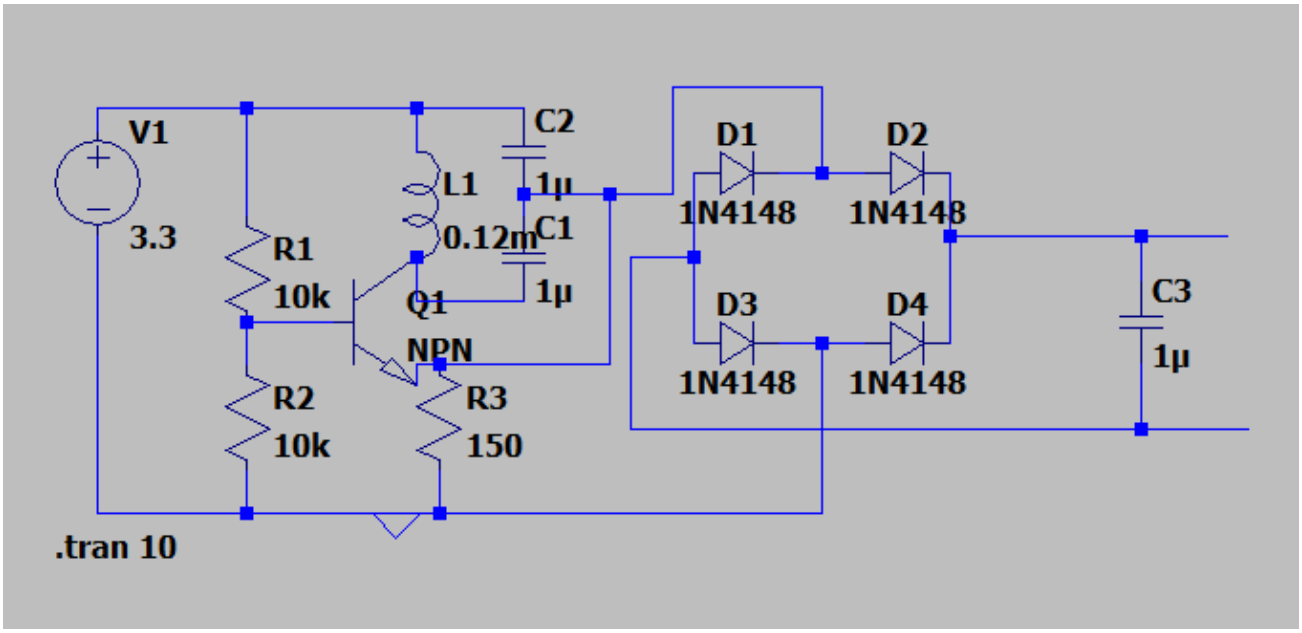


Figure 2.4: LTSpice Schematic of Circuit

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 and LEDs were used.

To display a strength meter, the LEDs were programmed to turn on. When the signal detection is weaker, 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.

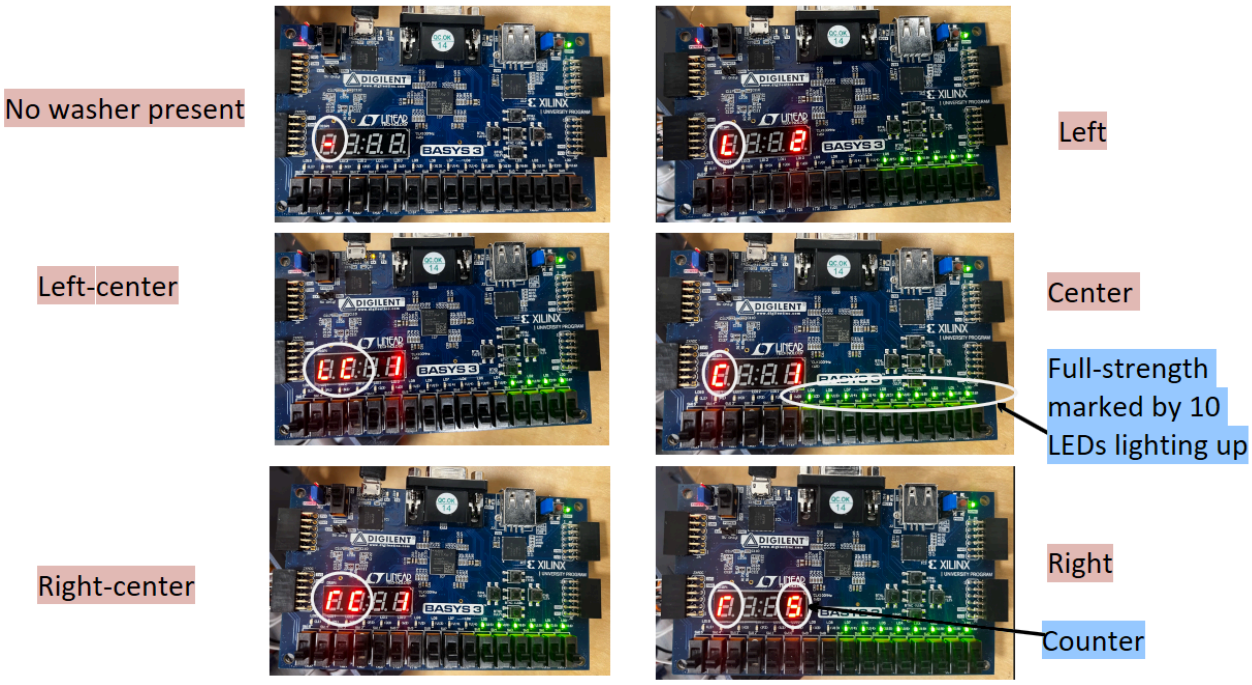


Figure 2.5: SSEG display and strength meter depending on position of washer and proximity

When a washer is detected, an “L,” “LC,” “C,” “RC,” or “R” is displayed on the leftmost SSEG (seven-segment) display based on the lateral position of the washer being detected. The symbol is only displayed when the washer is detected. A “-” will be displayed otherwise. After the washer is detected, a designated counter will increment depending on the position. The counter will increment the rightmost SSEG display.

III. Software Design Description

This section provides a detailed analysis of the Vitis C code implementation for our metal detection system project. The code drives a BASYS FPGA board to detect metal objects and identify their lateral position (left, center-left, center, center-right, or right) using electromagnetic coupling and multiple coils. The system displays the detection position and keeps count of detections in each position.

A. Hardware Interface Overview

The code interfaces with several hardware components through memory-mapped I/O addresses:

```
// Hardware address definitions
#define LEDS    ( *(unsigned volatile *)0x40000000 ) // GPIO-0 16-bit
#define DPSEG   ( *(unsigned volatile *)0x40020000 ) // GPIO-2 {DP, SEG[6:0]}
#define AN      ( *(unsigned volatile *)0x40020008 ) // GPIO-2 4-bit
#define VAUX_6   (*(volatile unsigned *)0x44a10258)   // JXADC0 port
#define VAUX_7   (*(volatile unsigned *)0x44a1025c)   // JXADC1 port

#define VAUX_15   (*(volatile unsigned *)0x44a1027c)   // JXADC2 port
```

Figure 3.1: Memory-mapped I/O addresses

These mappings connect to the physical components:

- **LEDS**: 16-bit LED array used to display signal strength
- **DPSEG**: 7-segment display data pins (segments and decimal point)
- **AN**: Anode control for 4-digit 7-segment display
- **VAUX_6/7/15**: ADC channels connected to the three detection coils (left, center, right)

B. System Design and Flowchart

The overall system architecture follows this basic flow:

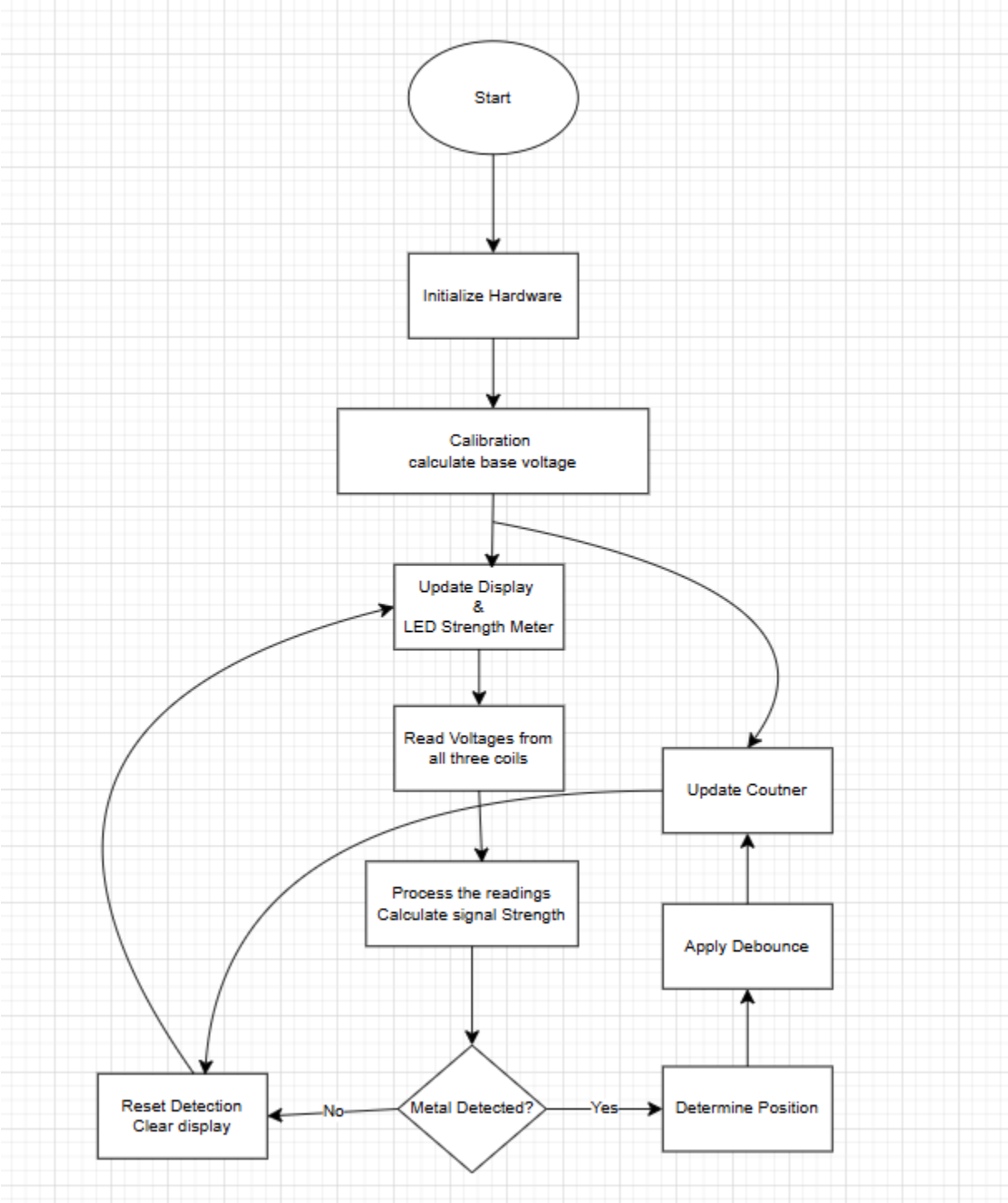


Figure 3.2: System flow chart

C. Main Program Flow

The main program consists of the following stages:

1. **Initialization**: Set up hardware and calculate baseline voltage readings
2. **Main Loop**:
    - Read voltages from all three coils
    - Process readings to detect metal and determine position
    - Apply debouncing to prevent false detections
    - Update counters for each position type
    - Display current detection status and counters
    - Update LED strength meter

IV. Key Components Analysis

A. Display System

The code implements a multiplexed 7-segment display system to show the detection status and counters:

```
// Display patterns for 7-segment (pre-inverted for direct output)
#define SEG_L 0b11000111  // "L" pattern
#define SEG_C 0b11000110  // "C" pattern
#define SEG_R 0b11001110  // "r" pattern
#define SEG_DASH 0b10111111  // "-" pattern
// ... more patterns ...

// Global variables for multiplexed display
uint8_t display_segments[4] = {SEG_OFF, SEG_OFF, SEG_OFF, SEG_OFF}; // All segments off initially

uint8_t current_digit = 0; // Current digit being displayed (0=leftmost, 3=rightmost)
```

Figure 4.1: Multiplexed 7-segment display program

The display multiplexing is handled by two functions:

1. update_display_content() - Sets what should be shown on the display based on detection status
2. refresh_display() - Handles the actual multiplexing, cycling through digits rapidly

B. Metal Detection Algorithm

The code implements a sophisticated algorithm for detecting metal and determining its position:

```
// Process left coil - normalized to 0-10 scale for use with 10 LEDs
if (base_left_mv > left_mv && (base_left_mv - left_mv) >= DETECT_MV_LEFT) {
   left_str = ((base_left_mv - left_mv) * 10) / MAX_CHANGE_LEFT;
   if (left_str > 10) left_str = 10; // Cap at 10
}

// Similar processing for center and right coils...

// Determine direction based on relative strengths
if (left_str > 0 && center_str > 0 && left_str > center_str && left_str > right_str) {
// Center-Left direction (left is stronger than center, but both are active)
   center_left_str = (left_str + center_str) / 2;
   if (center_left_str > strength) {
      strength = center_left_str;
      direction = DIR_CENTER_LEFT;
   }
}
```

Figure 4.2: Metal detection algorithm

Key aspects of the detection algorithm:

1. **Baseline calibration**: The system measures baseline voltages at startup
2. **Normalized strength calculation**: Converts raw ADC readings to a 0-10 scale
3. **Position determination**: Uses relative strengths from different coils to determine position

C. Debouncing Logic

To prevent rapid fluctuations in readings and ensure stable detections, the code implements a sophisticated debouncing system:

```
// Enhanced direction debouncing logic
if (direction == DIR_NONE || strength < STRENGTH_THRESHOLD) {
  // If no significant direction detected, reset the counter
  direction_stable_counter = 0;
  direction_detected = 0;
  // If strength was very low, explicitly set no direction
  if (strength < STRENGTH_THRESHOLD) {
    direction = DIR_NONE;
  }
}
else if (direction != last_direction) {
  // Direction changed - reset counter and update last_direction
  direction_stable_counter = 0;
  last_direction = direction;
  direction_detected = 0;
  xil_printf("Direction changed to %d, resetting stability counter\r\n", direction);
}
else {
  // Direction is stable, increment counter
  if (direction_stable_counter < DIRECTION_STABLE_TIME) {
    direction_stable_counter += 5;  // Assuming ~5ms per loop
  }

  // Check if we've held the direction stable long enough and haven't counted it yet
  if (direction_stable_counter >= DIRECTION_STABLE_TIME && !direction_detected) {
    // Direction is stable for required time - increment appropriate counter
    switch (direction) {
      case DIR_LEFT:
        l_cntr = (l_cntr + 1) % 10;
        break;
      // Additional cases...
    }
    direction_detected = 1;  // Set flag to prevent continuous counting
  }
}
```

Figure 4.3: Debouncing logic

The debouncing logic ensures:

1. A detection must be stable for a minimum time (DIRECTION_STABLE_TIME)
2. Only one detection is counted per object presence
3. The system resets when object is removed or changes position

D. Initialization and Calibration

The system features an intelligent calibration routine that runs at startup:

```
void show_loading_animation() {
  uint16_t left_base_sum = 0, center_base_sum = 0, right_base_sum = 0;
```

```
    uint8_t i = 0;
    uint16_t led_patterns[] = {
      0x0001, 0x0003, 0x0007, 0x000F,  // Fill from right
      // More patterns...
    };

    for (i=0; i < 32; i++) {
      LEDS = led_patterns[i];
      left_base_sum += read_mv_6();
      center_base_sum += read_mv_7();
      right_base_sum += read_mv_15();
      delay_ms(100);
    }
    base_left_mv = left_base_sum / i;
    base_center_mv = center_base_sum / i;
    base_right_mv = right_base_sum / i;
}
```

Figure 4.4: Calibration routine program

This function:

1. Shows a loading animation on the LEDs
2. Takes multiple samples from each coil
3. Averages the readings to establish baseline values
4. These baselines are used to detect changes when metal is present

E. Signal Strength Visualization

The system provides visual feedback on detection strength using the LED array:

```
// Display signal strength on LEDs (0-9)
void show_leds(uint8_t strength) {
    uint16_t led_mask = 0;
    uint8_t i = strength > 10 ? 10 : strength;

    // Create bit mask for LEDs
    while(i--) led_mask = (led_mask << 1) | 1;

    LEDS = led_mask;
}
```

Figure 4.5: Signal strength marked by LEDs

V. Design Integration

Our design integrates the electromagnetic coil system with the FPGA through analog-to-digital converters. Our implementation directly measures voltage changes when metal objects are introduced to the coils:

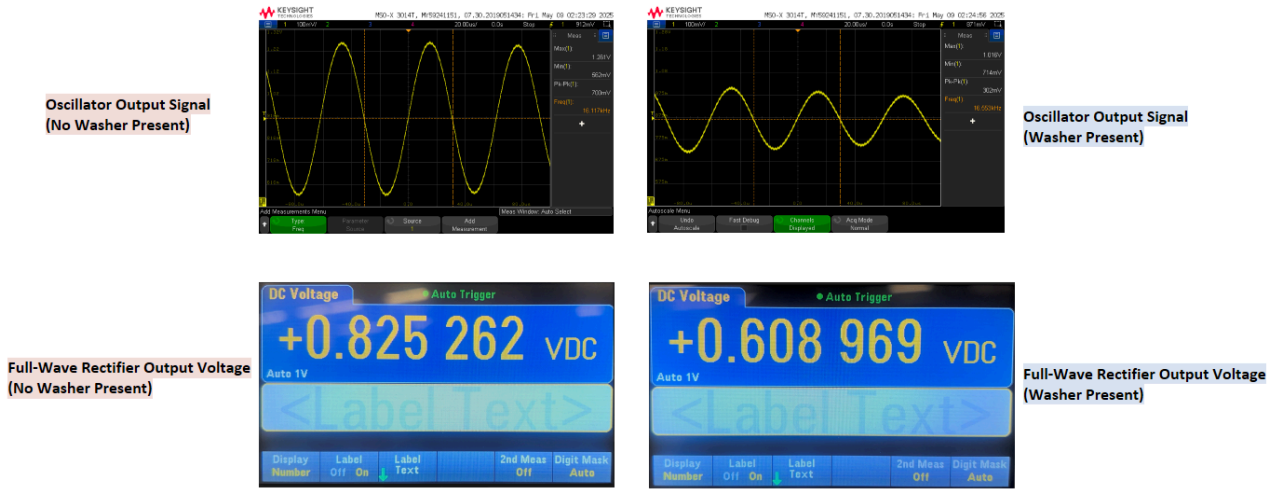
```
// Read voltage and convert directly to millivolts

uint16_t read_mv_6() {
   return ((VAUX_6 >> 4) & 0xFFF) * 1000 / 4095;
}
uint16_t read_mv_7() {
   return ((VAUX_7 >> 4) & 0xFFF) * 1000 / 4095;
}
uint16_t read_mv_15() {
   return ((VAUX_15 >> 4) & 0xFFF) * 1000 / 4095;

}
```

Figure 5.1: Voltage change detection

When a metal object is introduced to one of the coils, we observe a measurable drop in voltage. The magnitude of this drop correlates with both the position of the metal object and its proximity to the coil. By sampling these voltage changes across three strategically placed coils, we can determine both the presence and lateral position of metal objects.



Full-Wave Rectifier Output Voltage
(Washer Present)

Figure 5.2: AC and DC voltage changes when a washer is present

VI. Design Verification

Our design successfully met all of the requirements. It was powered using one battery pack of 6 AA batteries.

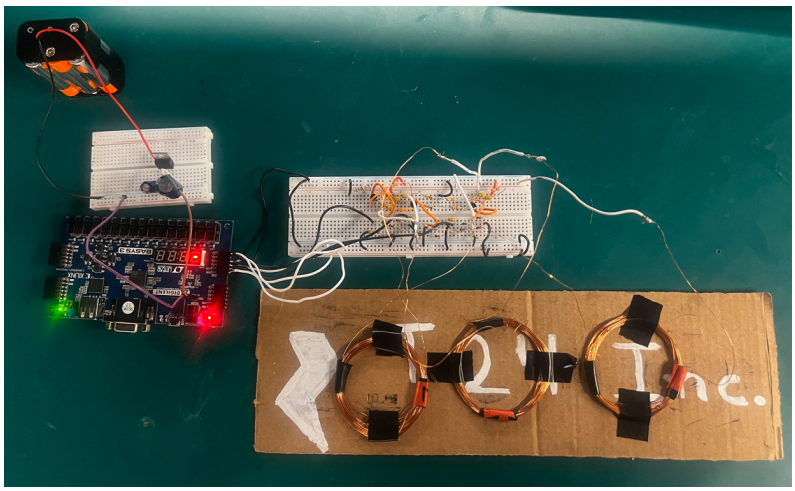


Figure 6.1: Self-contained metal detector

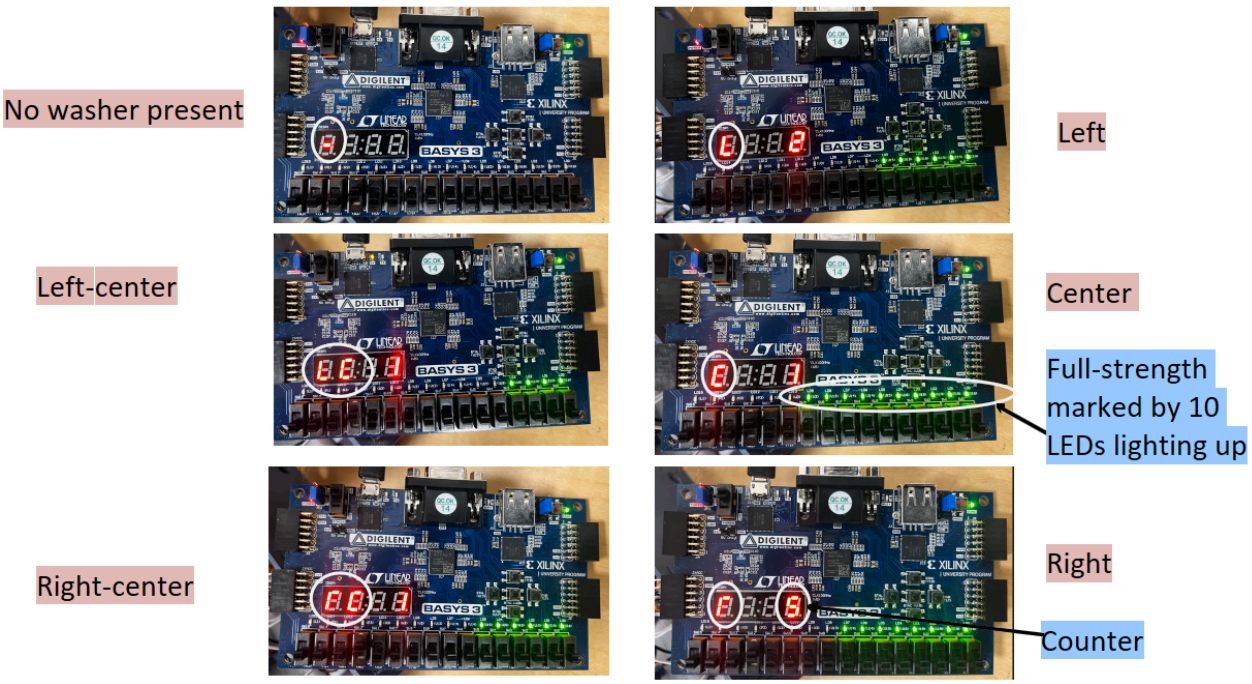


Figure 6.2: Washer positions, signal strength, and counter successfully displayed on Basys3 board

VII. Power Budget Analysis

Since the circuit is composed of 3 identical subcircuits, analysis was done for one, and then the power consumption was multiplied by 3 to receive total power consumption. Power consumption was calculated by obtaining voltage drop and current across components and multiplying both factors. The BJT's base-emitter current was ignored due to its negligible impact on the BJT's power draw.

Component	Value	Quantity	DC Voltage (V)	DC Current (mA)	Power (mW)	
R1	10000	1	1.79	Not used	0.32041	P = V^2/R
R2	10000	1	1.5	Not used	0.225	
R3	150	1	0.89	Not used	5.280666667	
Diodes		4	0.7	0.93	2.604	
Inductors		1	0.0004	0.01	0.000004	
C1	1uC	1	1.68	0.7	1.176	
C2	1uC	1	1.74	0.7	1.218	
C3	1uC	1	0.9	2	1.8	
BJT (Collector-Emitter)		1	2.4	2.2	5.28	P = Vce*Ic
Total power	For 3 circuits				53.712242	

Figure 7.1: Experimentally found DC voltage and current values (not including IB and IC)

VIII. Appendices

A. Bill of Materials

Part Number	Part Name	Value	Quantity
ESK108M6R3AH9	Capacitor	1 mF	2
K104K10X7RF53L2	Capacitor	1 uF	9
S0603CPX150J	Resistor	150 Ω	3
S0603CPX103J	Resistor	10 kΩ	6
1N4148	Diode	-	12
2N4401	NPN Transistor	-	3
LM2940T-5.0	5V Regulator	-	1
-	Inductor	0.12 mH	3
BH26AAW	6AA Battery Pack	9 V	1

Figure 8.1: Circuit Bill of Materials

B. Source Code

```
#include "xil_printf.h"
#include <stdint.h>

// Hardware address definitions
#define LEDS    ( *(unsigned volatile *)0x40000000 ) // GPIO-0 16-bit
#define DPSEG   ( *(unsigned volatile *)0x40020000 ) // GPIO-2 {DP, SEG[6:0]}
#define AN      ( *(unsigned volatile *)0x40020008 ) // GPIO-2 4-bit
#define VAUX_6   (*(volatile unsigned *)0x44a10258)   // JXADC0 port
#define VAUX_7   (*(volatile unsigned *)0x44a1025c)   // JXADC1 port
#define VAUX_15   (*(volatile unsigned *)0x44a1027c)   // JXADC2 port

// Display patterns for 7-segment (pre-inverted for direct output)
#define SEG_L 0b11000111  // "L" pattern
#define SEG_C 0b11000110  // "C" pattern
#define SEG_R 0b11001110  // "r" pattern
#define SEG_DASH 0b10111111  // "-" pattern
#define SEG_1 0b11111001  // "1" pattern
#define SEG_2 0b10100100  // "2" pattern
#define SEG_3 0b10110000  // "3" pattern
#define SEG_4 0b10011001  // "4" pattern
#define SEG_5 0b10010010  // "5" pattern
#define SEG_6 0b10000010  // "6" pattern
#define SEG_7 0b11111000  // "7" pattern
#define SEG_8 0b10000000  // "8" pattern
#define SEG_9 0b10010000  // "9" pattern
#define SEG_0 0b11000000  // "0" pattern
```

```
#define SEG_OFF 0b11111111  // All segments off

// Direction codes
#define DIR_NONE 0
#define DIR_LEFT 1
#define DIR_CENTER_LEFT 2
#define DIR_CENTER 3
#define DIR_CENTER_RIGHT 4
#define DIR_RIGHT 5

// Array of digit patterns for easy indexing
const uint8_t DIGIT_PATTERNS[10] = {
   SEG_0, SEG_1, SEG_2, SEG_3, SEG_4, SEG_5, SEG_6, SEG_7, SEG_8, SEG_9
};

// Simplified threshold values as integer millivolts to avoid floats
#define DETECT_MV_LEFT    40    // Minimum change for left coil to be detected
#define DETECT_MV_CENTER  40    // Minimum change for center coil to be detected
#define DETECT_MV_RIGHT   20    // Minimum change for right coil to be detected

// Maximum expected change in voltage for normalization (0-10 scale)
#define MAX_CHANGE_LEFT    240  // Maximum drop from base for left coil (~840 to ~600)
#define MAX_CHANGE_CENTER  150  // Maximum drop from base for center coil (~960 to ~800)
#define MAX_CHANGE_RIGHT   120  // Maximum drop from base for right coil (~970 to ~850)

// Debounce timing constants (in milliseconds)
#define DIRECTION_STABLE_TIME  100  // Time a direction must be stable before counting (ms)
#define STRENGTH_THRESHOLD    1     // Minimum strength to consider detection valid

// Global variables for multiplexed display
uint8_t display_segments[4] = {SEG_OFF, SEG_OFF, SEG_OFF, SEG_OFF}; // All segments off initially
uint8_t current_digit = 0; // Current digit being displayed (0=leftmost, 3=rightmost)

// Debounce variables
uint8_t last_direction = DIR_NONE;
uint16_t direction_stable_counter = 0;
uint8_t direction_detected = 0;  // Flag to indicate a direction has been detected and counted

// Simple delay function
void delay_ms(unsigned t) {
   volatile unsigned i;
   for(i = 0; i < t * 5000; i++); // Reduced iterations
}

uint16_t base_left_mv = 0, base_center_mv = 0, base_right_mv = 0;
```

```
// Read voltage and convert directly to millivolts
uint16_t read_mv_6() {
   return ((VAUX_6 >> 4) & 0xFFF) * 1000 / 4095;
}
uint16_t read_mv_7() {
   return ((VAUX_7 >> 4) & 0xFFF) * 1000 / 4095;
}
uint16_t read_mv_15() {
   return ((VAUX_15 >> 4) & 0xFFF) * 1000 / 4095;
}

void show_loading_animation() {
   uint16_t left_base_sum = 0, center_base_sum = 0, right_base_sum = 0;
   uint8_t i = 0;
   uint16_t led_patterns[] = {
      0x0001, 0x0003, 0x0007, 0x000F,  // Fill from right
      0x001F, 0x003F, 0x007F, 0x00FF,
      0x01FF, 0x03FF, 0x07FF, 0x0FFF,
      0x1FFF, 0x3FFF, 0x7FFF, 0xFFFF,  // All on
      0x7FFF, 0x3FFF, 0x1FFF, 0x0FFF,  // Empty from left
      0x07FF, 0x03FF, 0x01FF, 0x00FF,
      0x007F, 0x003F, 0x001F, 0x000F,
      0x0007, 0x0003, 0x0001, 0x0000   // All off
   };

   for (i=0; i < 32; i++) {
      LEDS = led_patterns[i];
      left_base_sum += read_mv_6();
      center_base_sum += read_mv_7();
      right_base_sum += read_mv_15();
      delay_ms(100);
   }
   base_left_mv = left_base_sum / i;
   base_center_mv = center_base_sum / i;
   base_right_mv = right_base_sum / i;
}

// Display signal strength on LEDs (0-9)
void show_leds(uint8_t strength) {
   uint16_t led_mask = 0;
   uint8_t i = strength > 10 ? 10 : strength;

   // Create bit mask for LEDs
   while(i--) led_mask = (led_mask << 1) | 1;

   LEDS = led_mask;
```

```
}

// Function to update the display content (set what should be shown)
void update_display_content(uint8_t direction, uint8_t l_count, uint8_t cl_count, uint8_t c_count, uint8_t
cr_count, uint8_t r_count) {
  // Display direction and counter based on current direction
  switch (direction) {
    case DIR_LEFT:
      // Left direction - show "L" and counter
      display_segments[0] = DIGIT_PATTERNS[l_count % 10]; // Counter
      display_segments[1] = SEG_OFF;
      display_segments[2] = SEG_OFF;
      display_segments[3] = SEG_L;    // Direction indicator
      break;

    case DIR_CENTER_LEFT:
      // Center-Left direction - show "CL" and counter
      display_segments[0] = DIGIT_PATTERNS[cl_count % 10]; // Counter
      display_segments[1] = SEG_OFF;
      display_segments[2] = SEG_C;    // "C" of "CL"
      display_segments[3] = SEG_L;    // "L" of "CL"
      break;

    case DIR_CENTER:
      // Center direction - show "C" and counter
      display_segments[0] = DIGIT_PATTERNS[c_count % 10]; // Counter
      display_segments[1] = SEG_OFF;
      display_segments[2] = SEG_OFF;
      display_segments[3] = SEG_C;    // Direction indicator
      break;

    case DIR_CENTER_RIGHT:
      // Center-Right direction - show "CR" and counter
      display_segments[0] = DIGIT_PATTERNS[cr_count % 10]; // Counter
      display_segments[1] = SEG_OFF;
      display_segments[2] = SEG_C;    // "C" of "CR"
      display_segments[3] = SEG_R;    // "R" of "CR"
      break;

    case DIR_RIGHT:
      // Right direction - show "R" and counter
      display_segments[0] = DIGIT_PATTERNS[r_count % 10]; // Counter
      display_segments[1] = SEG_OFF;
      display_segments[2] = SEG_OFF;
      display_segments[3] = SEG_R;    // Direction indicator
      break;
```

```
        default:
            // No direction - show dash
            display_segments[0] = SEG_OFF;
            display_segments[1] = SEG_OFF;
            display_segments[2] = SEG_OFF;
            display_segments[3] = SEG_DASH;
            break;
    }
}

// Function to perform actual multiplexing (call this frequently)
void refresh_display() {
    // Disable all digits first (set all bits to 1)
    AN = 0x0F;

    // Set segment pattern for current digit
    DPSEG = display_segments[current_digit];

    // Enable only the current digit (active low - set its bit to 0)
    AN = ~(1 << current_digit);

    // Advance to next digit for next time
    current_digit = (current_digit + 1) % 4;
}

int main() {
    uint16_t left_mv, center_mv, right_mv;
    uint8_t strength = 0, direction = DIR_NONE;
    uint8_t l_cntr = 0, cl_cntr = 0, c_cntr = 0, cr_cntr = 0, r_cntr = 0;
    uint8_t right_str, left_str, center_str;
    uint8_t center_left_str, center_right_str;

    // Initialize hardware
    LEDS = 0;

    // Initialize display with dash
    display_segments[0] = SEG_OFF;
    display_segments[1] = SEG_OFF;
    display_segments[2] = SEG_OFF;
    display_segments[3] = SEG_DASH;

    xil_printf("Calculating Base Voltages.");
    show_loading_animation();
    xil_printf("Base Calculation Complete.L=%d C=%d R=%d\r\n",
               base_left_mv, base_center_mv, base_right_mv);
```

```
  while(1) {
    // Read voltages (in millivolts)
    left_mv = read_mv_6();
    center_mv = read_mv_7();
    right_mv = read_mv_15();
    right_str = 0;
    left_str = 0;
    center_str = 0;
    center_left_str = 0;
    center_right_str = 0;

    // Reset signal strengths
    right_str = 0;
    left_str = 0;
    center_str = 0;
    center_left_str = 0;
    center_right_str = 0;

    // Process left coil - normalized to 0-10 scale
    if (base_left_mv > left_mv && (base_left_mv - left_mv) >= DETECT_MV_LEFT) {
      left_str = ((base_left_mv - left_mv) * 10) / MAX_CHANGE_LEFT;
      if (left_str > 10) left_str = 10; // Cap at 10
    }

    // Process center coil - normalized to 0-10 scale
    if (base_center_mv > center_mv && (base_center_mv - center_mv) >= DETECT_MV_CENTER) {
      center_str = ((base_center_mv - center_mv) * 10) / MAX_CHANGE_CENTER;
      if (center_str > 10) center_str = 10; // Cap at 10
    }

    // Process right coil - normalized to 0-10 scale
    if (base_right_mv > right_mv && (base_right_mv - right_mv) >= DETECT_MV_RIGHT) {
      right_str = ((base_right_mv - right_mv) * 10) / MAX_CHANGE_RIGHT;
      if (right_str > 10) right_str = 10; // Cap at 10
    }

    // Check if no significant signal on any coil
    if (left_str < STRENGTH_THRESHOLD && center_str < STRENGTH_THRESHOLD && right_str <
STRENGTH_THRESHOLD) {
      // No signal - reset counters and update display
      direction = DIR_NONE;
      last_direction = DIR_NONE;
      direction_stable_counter = 0;
      direction_detected = 0;  // Reset detection flag
      update_display_content(DIR_NONE, l_cntr, cl_cntr, c_cntr, cr_cntr, r_cntr);
```

```
    LEDS = 0;


    // Make sure to refresh display during wait
    for (int i = 0; i < 25; i++) {
        refresh_display();
        delay_ms(1);
    }
    continue;
}

// Calculate overall strength and determine direction
strength = 0;
direction = DIR_NONE;

// Determine direction based on relative strengths
if (left_str > 0 && center_str > 0 && left_str > center_str && left_str > right_str) {
    // Center-Left direction (left is stronger than center, but both are active)
    center_left_str = (left_str + center_str) / 2;
    if (center_left_str > strength) {
        strength = center_left_str;
        direction = DIR_CENTER_LEFT;
    }
}
else if (center_str > 0 && right_str > 0 && right_str > center_str && right_str > left_str) {
    // Center-Right direction (right is stronger than center, but both are active)
    center_right_str = (center_str + right_str) / 2;
    if (center_right_str > strength) {
        strength = center_right_str;
        direction = DIR_CENTER_RIGHT;
    }
}
else if (left_str > 0 && left_str > center_str && left_str > right_str) {
    // Left direction (only left coil is significantly active)
    if (left_str > strength) {
        strength = left_str;
        direction = DIR_LEFT;
    }
}
else if (center_str > 0 && center_str > left_str && center_str > right_str) {
    // Center direction (only center coil is significantly active)
    if (center_str > strength) {
        strength = center_str;
        direction = DIR_CENTER;
    }
}
```

```
else if (right_str > 0 && right_str > left_str && right_str > center_str) {
   // Right direction (only right coil is significantly active)
   if (right_str > strength) {
      strength = right_str;
      direction = DIR_RIGHT;
   }
}

// Enhanced direction debouncing logic
if (direction == DIR_NONE || strength < STRENGTH_THRESHOLD) {
   // If no significant direction detected, reset the counter
   direction_stable_counter = 0;
   direction_detected = 0;
   // If strength was very low, explicitly set no direction
   if (strength < STRENGTH_THRESHOLD) {
      direction = DIR_NONE;
   }
}
else if (direction != last_direction) {
   // Direction changed - reset counter and update last_direction
   direction_stable_counter = 0;
   last_direction = direction;
   direction_detected = 0;
   xil_printf("Direction changed to %d, resetting stability counter\r\n", direction);
}
else {
   // Direction is stable, increment counter
   if (direction_stable_counter < DIRECTION_STABLE_TIME) {
      direction_stable_counter += 5;  // Assuming ~5ms per loop
   }

   // Check if we've held the direction stable long enough and haven't counted it yet
   if (direction_stable_counter >= DIRECTION_STABLE_TIME && !direction_detected) {
      // Direction is stable for required time - increment appropriate counter
      switch (direction) {
         case DIR_LEFT:
            l_cntr = (l_cntr + 1) % 10;
            break;
         case DIR_CENTER_LEFT:
            cl_cntr = (cl_cntr + 1) % 10;
            break;
         case DIR_CENTER:
            c_cntr = (c_cntr + 1) % 10;
            break;
         case DIR_CENTER_RIGHT:
            cr_cntr = (cr_cntr + 1) % 10;
```

```
                break;
            case DIR_RIGHT:
                r_cntr = (r_cntr + 1) % 10;
                break;
        }

        direction_detected = 1;  // Set flag to prevent continuous counting
        xil_printf("Direction %d stable for %dms - counter incremented\r\n",
                direction, direction_stable_counter);
      }
    }

    // Update LEDs with signal strength
    show_leds(strength);

    // Update the display content based on current state
    update_display_content(direction, l_cntr, cl_cntr, c_cntr, cr_cntr, r_cntr);

    // Debug output with normalized values
    xil_printf("L=%d(%d) C=%d(%d) R=%d(%d) L_str=%d CL_str=%d C_str=%d CR_str=%d R_str=%d dir=%d
time=%d\r\n",
            left_mv, base_left_mv - left_mv, center_mv, base_center_mv - center_mv,
            right_mv, base_right_mv - right_mv, left_str, center_left_str, center_str,
            center_right_str, right_str, direction, direction_stable_counter);

    // Refresh display multiple times during the wait period
    for (int i = 0; i < 50; i++) {
      refresh_display();
      delay_ms(1);
    }
```