University of Nottingham
UK | CHINA | MALAYSIA

Spring Semester Technical Report

EEEE2052 Practical Engineering Design Solutions and Project Development

Department of Electrical and Electronic Engineering

Faculty of Science and Engineering

Group 10:

Ng Chin Yuen [20306575]

Arfan Danial Bin Zainal Abidin [20284997]

Ahmad Najjar [20314711]

Sarween Kumar [20313296]

Report Due Date: 8th May 2023

University of Nottingham
UK | CHINA | MALAYSIA

Table of Content

1.0 Introduction..... 3

2.0 Demonstration of Calculations and Design of Circuits & Programming Structure..... 4

2.1 Task 7: Comparator-based Complete Doppler System..... 4

2.2 Task 8: ADC-based Complete Doppler System..... 5

2.3 Task 9.a: RS485 Communications Link (Software)..... 6

2.4 Task 9.b: RS485 Communications Link (Hardware)..... 7

2.5 Task 10: Power Supplies..... 8

2.6 Task 11: External Clock..... 9

2.7 Task 12: LED Interfacing to the CPLD..... 9

2.8 Task 13: Clock Divider..... 10

2.9 Task 14: BCD-7 Segment Decoder..... 11

2.10 Task 15: Shift Register Block..... 12

2.11 Task 16: Control Logic Block..... 13

2.12 Task 17: Demonstrate a Working System..... 19

2.13 Task 18: Completed Radar Speed Detection System..... 24

3.0 Experimental/Simulation Results, Analysis and Discussion.. 27

3.1 Task 7: Comparator-based Complete Doppler System..... 27

3.2 Task 8: ADC-based Complete Doppler System..... 28

3.3 Task 9: RS485 Communications Link (Software & Hardware)..... 29

3.4 Task 10: Power Supplies..... 30

3.5 Task 11: External Clock..... 31

3.6 Task 12: LED Interfacing to the CPLD..... 31

3.7 Task 13: Clock Divider..... 32

3.8 Task 14: BCD-7 Segment Decoder..... 34

3.9 Task 15: Shift Register Block..... 35

3.10 Task 16: Control Logic Block..... 36
3.11 Task 17: Demonstrate a Working System..... 37
3.12 Task 18: Completed Radar Speed Detection System.....38
4.0 Conclusion..... 39
5.0 References.....40
Appendix A: Comparator.c Code Snippet.....41
Appendix B: 7-Segment to Xilinx Connections..... 42
Appendix C: Shift Register.....43
Appendix D: Start Bit Detector..... 44
Appendix E: 8-Bit Counter..... 46
Appendix F: 10x Counter..... 47
Appendix G: RS485 Transmitter Input and Receiver Output..... 48
Appendix H: BCD to 7-Segment Decoder.....54
Appendix I: BCD to 7-Segment Decoder Test Bench Results..... 55
Appendix J: Shift Register Test Bench..... 59
Appendix K: Control Logic Block Test Bench..... 60
Appendix L: Working System.....61

1.0 Introduction

The goal of the project is to design and build a remote display board that can receive serial data from the STM32-NUCLEO-L476RG and display the detected speed on two 7-segment LED displays. To do this, the receiver system was implemented using a Xilinx complex programmable logic device (CPLD), which enables flexibility in design testing and updating without requiring any hardware modifications. Thus, greatly reducing the project development time. The CPLD and microcontroller communicate using an industry standard, RS485, ensuring dependable and efficient communication between the components. The objectives and task distribution of the project is illustrated in Figure 1.

	Name	Duration	Start	Finish
1	Task 7: Comparator-based Complete Doppler System (Technical Assessment)	0.062 days	3/24/23 4:00 PM	3/24/23 4:30 PM
3	Task 8: ADC-based Complete Doppler System (Technical Assessment)	0.062 days	3/24/23 4:30 PM	3/24/23 5:00 PM
5	Task 9.a: RS485 communications link (software)	0.5 days	2/20/23 9:00 AM	2/20/23 2:00 PM
8	Task 9.b: RS485 communications link (hardware)	0.5 days	2/20/23 9:00 AM	2/20/23 2:00 PM
11	Task 10: Power supplies	0.375 days	2/20/23 9:00 AM	2/20/23 1:00 PM
14	Task 11: External clock	0.375 days	2/20/23 9:00 AM	2/20/23 1:00 PM
17	Task 12: LED interfacing to the CPLD	0.375 days	2/20/23 1:00 PM	2/20/23 4:00 PM
20	Task 13: Clock divider	0.5 days	2/20/23 1:00 PM	2/20/23 5:00 PM
25	Task 14: BCD-7 segment decoder	0.5 days	2/20/23 4:00 PM	2/21/23 11:00 AM
30	Task 15: Shift register block	0.5 days	2/21/23 9:00 AM	2/21/23 2:00 PM
33	Task 16: Control logic block	1.875 days	2/21/23 9:00 AM	2/22/23 5:00 PM
36	Task 17: Demonstrate a working system	1 day	3/20/23 9:00 AM	3/21/23 9:00 AM
39	Task 18: Completed radar speed detection system	0.875 days	3/21/23 9:00 AM	3/21/23 5:00 PM

Figure 1. Objectives and task distribution of Semester 2 project.

In literature, vehicle speed measurement systems are important for traffic law enforcement and traffic status assessment in intelligent traffic systems. The Doppler effect is one of the many approaches for detecting speed. The Doppler radar is used to detect the frequency shift in microwaves reflected from a moving vehicle to measure its speed. According to [1], employing the Doppler effect to detect vehicle speed is an effective and precise way for maintaining road safety. Another research in [2] discusses a low-cost Doppler radar-based vehicle speed detection system, which is more precise when there are no other moving objects nearby and operates better at close ranges.

Besides that, this project connects to the SDG targets of "Good Health and Well-Being" and "Quality Education". The project intends to create a vehicle speed detector with a remote display board, which is critical for preserving road safety. Only by reducing road fatalities and injuries can good health and well-being of the community be achieved. Moreover, this project also educates us on the contemporary engineering tools employed in the industry such as the Xilinx CPLD, RS485 communication, and Doppler radar.

2.0 Demonstration of Calculations and Design of Circuits & Programming Structure

2.1 Task 7: Comparator-based Complete Doppler System

In semester 1, the comparator system with the PCB signal conditioning circuit produced very inaccurate frequency calculations. The previous method calculated the comparator output frequency by counting the number of rising edges in 1 second. This method had major drawbacks. The drawbacks were that the method was easily affected by phase delay in the output of the comparator and that the method was very prone to noise. Hence, a new method was proposed to solve this issue.

The new method calculates the frequency of the comparator output signal by finding the time difference between two rising edges. The basic flowchart of this new method is illustrated in Figure 2. As shown in Figure 2, this method has two different states. State 0 is when the first rising edge is encountered and state 1 is the second rising edge. When it is in state 0, it will perform process 1, where the time at which the first rising edge occurs is recorded, T1, and the state is changed to state 1. When the second rising edge is detected, process 2 will be executed where the time at which the second rising edge occurs is recorded, T2. Then, the frequency is found by taking the inverse of the time difference between T2 and T1. Finally, the state is reset to 0 for the next two rising edges.

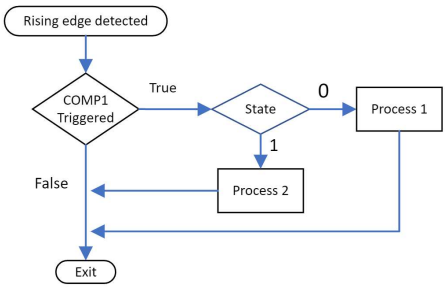


Figure 2. COMP system flowchart.

The internal comparator has a rising edge interrupt mode to pause the current system operation when there is a rising edge and let the interrupt service routine (ISR) call a callback function (see Appendix A). To ensure accurate time allocation when the rising edges occur, a timer with a tick rate of 1 MHz was set up with an overflow counter to consider when the second rising edge time, T2, happens after timer rollover.

2.2 Task 8: ADC-based Complete Doppler System

The ADC-based Doppler system was developed following the "FFT_Student Version" resources in Moodle. The ADC_var.bit was set to 12 as a 12-bit ADC was used and ADC_var.speed was set to 64000000 as the clock of the ADC was 64 MHz. ADC_var.prescaler, ADC_var.sampling_time, and ADC_var.adc_buf_len were set to 64, 247.5, and 4096 respectively to achieve approximately 1 Hz frequency bin as calculated below.

$$Bin = \frac{clock\ speed}{(ADC\ bit\ resolution + sampling\ time) \times prescaler \times buffer\ length}$$

$$= \frac{64 \times 10^6}{(12 + 247.5) \times 64 \times 4096}$$

$$= 0.94\ Hz$$

The frequency bin must be small to increase the accuracy of the ADC system, which is very crucial in a speed detector. The ADC settings of the STM32 are also changed to match the ADC_var settings as shown in Figure 3.

```
hadc1.Instance = ADC1;
hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV64;
hadc1.Init.Resolution = ADC_RESOLUTION_12B;
hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
hadc1.Init.LowPowerAutoWait = DISABLE;
hadc1.Init.ContinuousConvMode = ENABLE;
hadc1.Init.NbrOfConversion = 1;
hadc1.Init.DiscontinuousConvMode = DISABLE;
hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
hadc1.Init.DMAContinuousRequests = ENABLE;
hadc1.Init.Overrun = ADC_OVR_DATA_PRESERVED;
hadc1.Init.OversamplingMode = DISABLE;
```

```
sConfig.Channel = ADC_CHANNEL_6;
sConfig.Rank = ADC_REGULAR_RANK_1;
sConfig.SamplingTime = ADC_SAMPLETIME_247CYCLES_5;
sConfig.SingleDiff = ADC_SINGLE_ENDED;
sConfig.OffsetNumber = ADC_OFFSET_NONE;
sConfig.Offset = 0;
```

Figure 3. The ADC settings of the STM32.

2.3 Task 9.a: RS485 Communications Link (Software)

An RS485 communication link was set up to enable long distance display of the Doppler speed detector. This is important as the speed detector must be placed close to the moving vehicle for accurate measurement, whereas the display of the speed detector must be placed at a distance away for the speeding driver to see it. The software for the RS485 communication link was set up in the STM32 by configuring the UART to transmit 2 BCD values (8-bits) of data. Figure 4 shows the configurations of the UART of the STM32 for the RS485 communication link.

```
huart5.Instance = UART5;
huart5.Init.BaudRate = 57600;
huart5.Init.WordLength = UART_WORDLENGTH_8B;
huart5.Init.StopBits = UART_STOPBITS_1;
huart5.Init.Parity = UART_PARITY_NONE;
huart5.Init.Mode = UART_MODE_TX;
huart5.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart5.Init.OverSampling = UART_OVERSAMPLING_16;
huart5.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
huart5.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
if (HAL_HalfDuplex_Init(&huart5) != HAL_OK)
```

Figure 4. The configurations of the UART for RS485 communication.

As shown in Figure 4, the UART of the STM transmits 8 bits in half-duplex mode at a baud rate of 57600. The transmitted signal has 1 stop bit and no parity bit. The software first converts the 2-digit decimal number into BCD and then transmits it via UART as shown in Figure 5.

```
uint8_t TxData = 55;
```

(a)

```
TxData = Decimal_To_BCD(TxData);
```

(b)

```
while (1)
{
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 */
    HAL_UART_Transmit (&huart5, &TxData, 1, 1);
    HAL_Delay (50);
```

(c)

Figure 5. RS485 communication link software.

Figure 5(a) shows the 2-digit decimal number that will be converted to BCD using the Decimal_To_BCD function shown in Figure 5(b) and transmitted. The program continuously transmits the 8-bit BCD data via UART with some delay between each transmission (Figure 5(c)).

2.4 Task 9.b: RS485 Communications Link (Hardware)

As the STM32 does not have the required hardware for RS485 communication, the SN65HVD1780 IC was used for the RS485 transmitter and receiver. The circuit diagram of the RS485 communication link is shown in Figure 6.

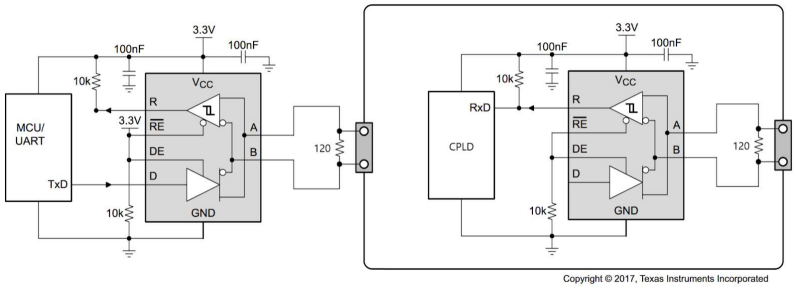


Figure 6. Circuit diagram of the RS485 transmitter and receiver.

As shown in Figure 6, the UART Tx pin of the STM32 is connected to the D pin of the left IC. The DE pin is pulled high to enable transmission. A 120 Ω resistor is placed between pins A and B at both ends as the transient protection, the ESD Suppressors or TVS Diodes are not required in this system. The CPLD is connected to the R pin of the right IC. The DE pin is pulled low to enable the receiver. Figure 7 shows the breadboard implementation of this circuit.

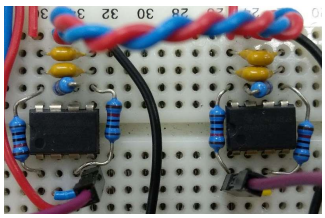


Figure 7. The RS485 communication link on breadboard .

Figure 7 shows the RS485 transmitter on the left and the receiver on the right. The twisted blue and red wire pairs are the communication links between the transmitter and receiver. The wire pairs can extend up to 1200 m for long distance communication between the speed detector and the display board.

2.5 Task 10: Power Supplies

The Low-Dropout (LDO) MAX882 voltage regulator will be used as the power supply to the display board. Voltage regulators are important as they help to filter out any noise or fluctuations in the voltage source to ensure stable voltage supply to the components of the circuit and prevent damage. Two voltage regulator circuits will be designed for the remote display system. One will supply 1.8 V, while the other will supply 3.3 V.

Figure 8 shows the schematic diagram of the voltage regulator circuit. The default output voltage of the MAX882 is 3.3 V but it can be varied by connecting two resistors, R1 and R2 in the configuration as shown in Figure 8.

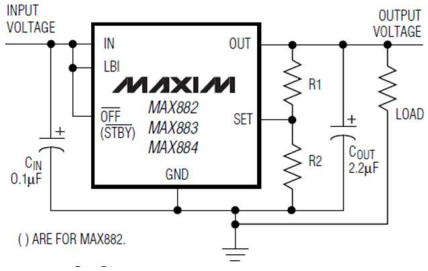


Figure 8. Schematic diagram of the display board power supply.

The values for R1 and R2 are found using (1), where Vset = 1.2 V. The calculation for 1.8V output voltage is as follows, letting R2 = 33 kΩ.

$$V_{OUT} = V_{SET}\frac{R1+R2}{R2} \quad (1)$$

$$R1 = \frac{1.8}{1.2}(33\ k\Omega) - (33\ k\Omega) = 16.5\ k\Omega$$

Therefore, R1 is calculated to be 16.5 kΩ. The calculation for 3.3 V output voltage is shown below, letting R2 = 3 kΩ.

$$R1 = \frac{3.3}{1.2}(3\ k\Omega) - (3\ k\Omega) = 5250\ \Omega$$

Therefore, R1 is calculated to be 5250 Ω. Figure 9 shows the constructed power supply circuits where the 1.8 V output is on the left while the 3.3 V is on the right.

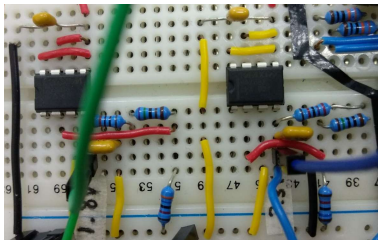


Figure 9. The 1.8 V (left) and 3.3 V (right) power supply.

2.6 Task 11: External Clock

An external clock will be provided to the CPLD of the remote display board as it does not have its own clock. The external clock used is a crystal oscillator, which is optimal for digital systems that require precise timing due to their high stability and accuracy. The external clock is 1.8432MHz. Figure 10 shows the crystal oscillator connections.

Figure 10. Connections of the External Clock.

2.7 Task 12: LED Interfacing to the CPLD

The pins of the Xilinx CPLD connected to the 7-segment display are shown in Appendix B. A series resistor is connected to each segment to limit the forward current across the LED below the maximum 30 mA limit and to ensure uniform brightness of each segment. To illuminate a segment, the

Xilinx board would have to output LOW to ground that segment. Figure 11 illustrates the LED forward current with respect to its forward voltage.

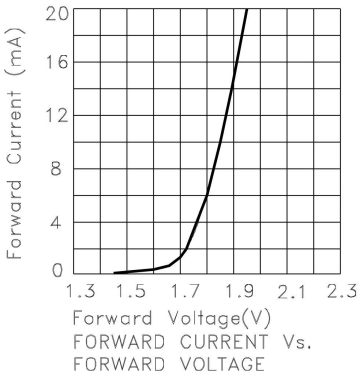


Figure 11. LED forward current and voltage relationship.

The forward current is assumed to be 6 mA to not drive the LED at maximum brightness (20 mA). From Figure 11, the forward voltage of the LED at 6 mA is 1.8 V. The required resistor for each segment is calculated.

$$R = \frac{V_{CC} - V_F}{I} = \frac{3.3 - 1.8}{6 \times 10^{-3}} = 250\ \Omega$$

The closest available resistor in the lab is 240 Ω. The calculation below verifies the 240 Ω resistor assuming the voltage drop across the resistor stays constant.

$$I = \frac{V_{CC} - V_F}{R} = \frac{3.3 - 1.8}{240} = 6.25\ mA$$

According to Figure 11, the forward voltage at 6.25 mA forward current is about 1.8 V.

2.8 Task 13: Clock Divider

The clock divider receives an external 1.8432MHz clock (CLKEXT) and outputs an internal clock (CLKINT) sufficient for detecting a 57600 baud serial signal. To achieve 57600 baud from the 1.8432 MHz signal, five flip-flops were connected in series to divide the clock by 32. The internal clock frequency is obtained as follows:

$$1.8432 \times 10^6 \div 2^5 = 57600\ Hz$$

To identify the start bit and successive bits, the UART must sample the data at a high enough rate to detect the transitions reliably. Transitions may be missed if the sample rate is too low, resulting in inaccurate data being obtained. 16 times the baud rate assures that the UART can detect the start bit and consecutive bits reliably. To achieve the 921600 Hz CLKINT, one D flip flop is used. The internal clock frequency is obtained as follows:

$$1.8432 \times 10^{6} \div 2 = 921600\ Hz$$

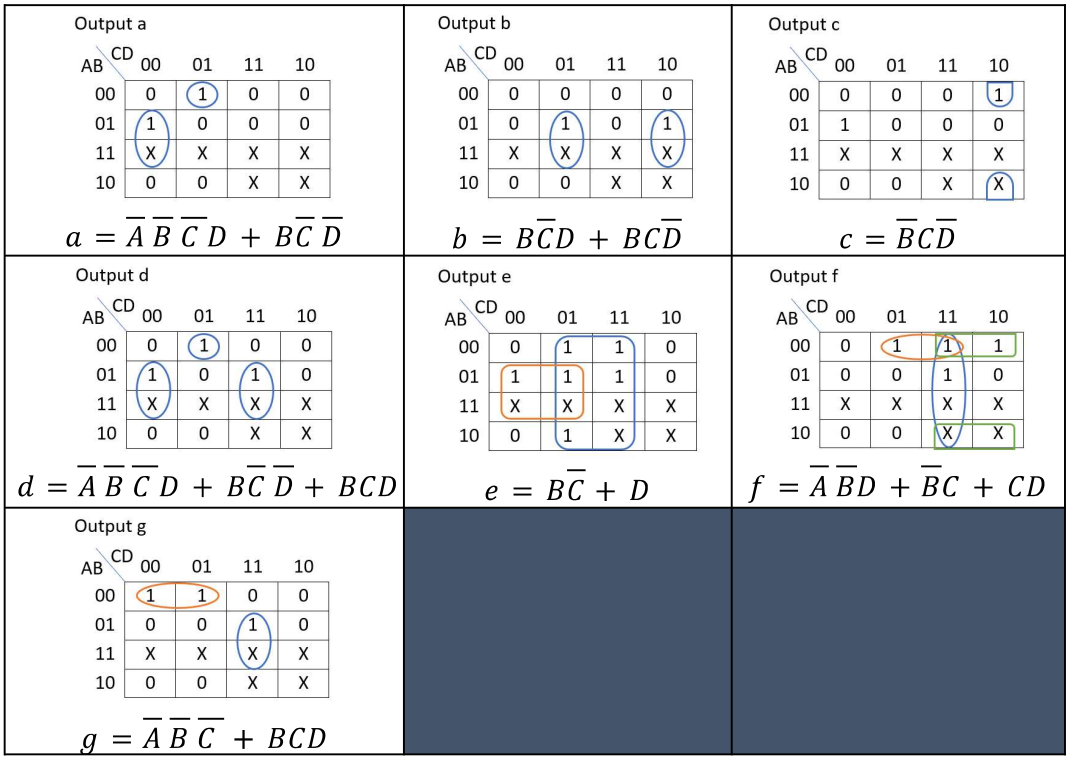
2.9 Task 14: BCD-7 Segment Decoder

The STM32 sends two digit BCD numbers to the CPLD system and the remote display system uses two 7-segment displays. Hence, a BCD to 7-segment decoder is needed to decode the BCD combination to illuminate the correct segments of the 7-segment display. Only one 4-bit BCD decoder will be designed as the two 7-segment displays will have the same schematic. Table 1 depicts the truth table of the BCD decoder, where A is the MSB and D is the LSB. From Table 1, the K-map optimization technique will be used to obtain the optimised logic schematic for each segment as shown in Table 2.

Table 1. Truth table for BCD decoder.

Digit	Input					Output						
	A	B	C	D		a	b	c	d	e	f	g
0	0	0	0	0		0	0	0	0	0	0	1
1	0	0	0	1		1	0	0	1	1	1	1
2	0	0	1	0		0	0	1	0	0	1	0
3	0	0	1	1		0	0	0	0	1	1	0
4	0	1	0	0		1	0	0	1	1	0	0
5	0	1	0	1		0	1	0	0	1	0	0
6	0	1	1	0		0	1	0	0	0	0	0
7	0	1	1	1		0	0	0	1	1	1	1
8	1	0	0	0		0	0	0	0	0	0	0
9	1	0	0	1		0	0	0	0	1	0	0

Table 2. K-map optimization with final output equation.



2.10 Task 15: Shift Register Block

The STM32 board transmits serial data to the remote display board. However, to display the received values on the 7-segment display, parallel data must be used. Hence, a shift register was implemented to convert the serially transmitted data into parallel. The schematic of the 4-bit serial-in parallel-out (SIPO) shift register is shown in Appendix C. The shift register shifts in data bit by bit from Q_A to Q_D at each CLK cycle. The display board receives 8 bits of serial data. Hence, an 8-bit SIPO shift register is needed. This can be done by cascading 8 flip-flops in the same manner as in Appendix C. However, as Xilinx already has a premade 8-bit SIPO shift register (SR8CE) in its library, it is more efficient to use this than to build it from scratch. Table 3 illustrates the truth table for SR8CE.

Table 3. SR8CE truth table.

Inputs				Outputs	
CLR	CE	SLI	C	Q0	Qz: Q1
1	X	X	X	0	0
0	0	X	X	No Change	No Change
0	1	SLI	↑	SLI	Qn-1

When the CLR pin is HIGH, it will ignore its regular operation and reset the outputs to 0. The SR8CE also has a Clock Enable pin (CE) so that the shift register only changes its output when this pin is HIGH, offering more control. As long as there is a rising edge in the CLK and CE is HIGH, the input SLI will be shifted into Q0, and the rest will be shifted to the next output Q bit (see Table C1 in Appendix C).

2.11 Task 16: Control Logic Block

The CPLD receiver of the display system is asynchronous to the transmission clock (Universal Asynchronous Receiver/Transmitter). The receiver is able to approximately sync with the transmitter by detecting the centre of the start bit at the beginning of each transmitted data stream. This requires the receiver to use a high-speed clock to accurately detect the centre of the start bit as shown in Figure 12.

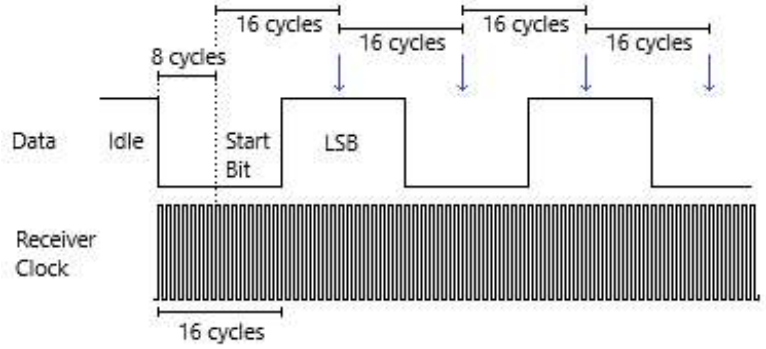


Figure 12. UART receiver.

Figure 12 shows a UART receiver using 16 times the transmitter baud rate as its clock. As shown in Figure 12, the transmitter output is HIGH when it is not sending any data (idle). When there is transmitted data, the bits

are preceded by a start bit that is LOW. The receiver is able to detect this start bit by checking the data signal every receiver clock cycle. If 8 consecutive LOW logic levels are detected, the receiver clock is now positioned at the centre of the start bit. The control logic of the receiver then controls a 16-bit counter to count 16 receiver clock cycles so that each data bit is sampled at its centre.

Typically, 8 or 16 times the transmitter baud rate is used. For this project, the receiver clock is 16 times the baud rate of the transmitter. This was chosen as it is less prone to error than the 8 times. This can be proven by assuming the worst-case scenario that the centre of the start bit was detected about 1 whole receiver clock cycle away from the actual centre. If 16 times the baud rate was used, the worst-case scenario is that 9 clock cycles were used to detect the centre of the start bit instead of 8.

$$Errom\ from\ start\ bit\ centre = \frac{9-8}{16} \times 100\%$$
$$= 6.25\%$$

If 8 times the baud rate was used, the worst-case scenario is that 5 clock cycles were used to detect the centre of the start bit instead of 4.

$$Errom\ from\ start\ bit\ centre = \frac{5-4}{8} \times 100\%$$
$$= 12.5\%$$

Hence, the start bit error for 8 times the transmitter baud rate is twice the 16 times. This error calculation did not consider the propagation delay, clock inconsistency, or other errors which would worsen the accuracy of the receiver. Hence, this error should be minimised.

The transmitter baud rate is 57600. Hence, the receiver clock would be 921600 Hz. Besides that, the receiver is to receive 8 bits of data from the RS485 receiver pin. Thus, the control logic block will consist of a start bit detector (StartBD), a 16-bit counter (16Counter), an 8-bit counter (8Counter), and a stop bit detector (StopBD). The control logic controls the enable (SR_CE) and clear (Rx_stop) of all the blocks in the receiver

circuit as well as the signal to trigger the shift register to read a bit (SR_trig). Figure 13 shows the state machine of the control logic block.

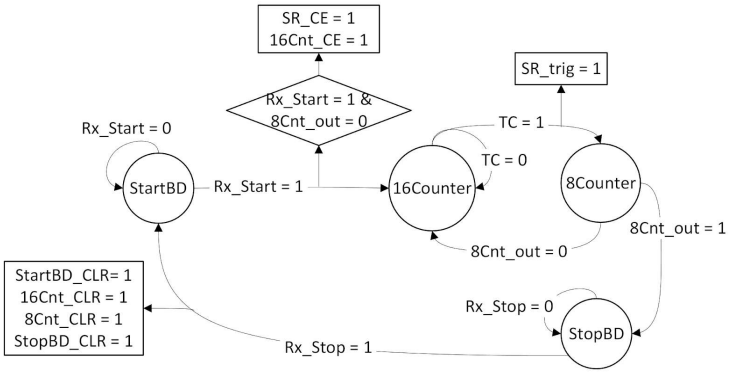


Figure 13. State machine diagram of the control logic block.

From Figure 13, the StartBD will detect 8 consecutive zeros at the start bit of the transmitted byte. This is to prevent glitches in the transmitted signal to falsely trigger the control logic block. When the start bit is detected, Rx_Start = 1. The control logic then enables a 16-bit counter (16Cnt_CE) and an 8-bit shift register (SR_CE). The 16-bit counter triggers the shift register to sample the transmitted data every 16 clock cycles as indicated by TC. Each time the shift register samples a bit, the 8-bit counter increments by 1. When 8 bits of data have been sampled, the 8-bit counter outputs 1 (8Cnt_out = 1). The control logic then disables the shift register and 16-bit counter. The control then moves to the StopBD, which is a 16-bit counter that counts 16 receiver clock cycles for the stop bit. Rx_Stop will then be HIGH and cause all the blocks of the receiver circuit to be reset. The receiver can now receive the next transmitted data. The state machine diagram is implemented in Xilinx as shown in Figure 14.

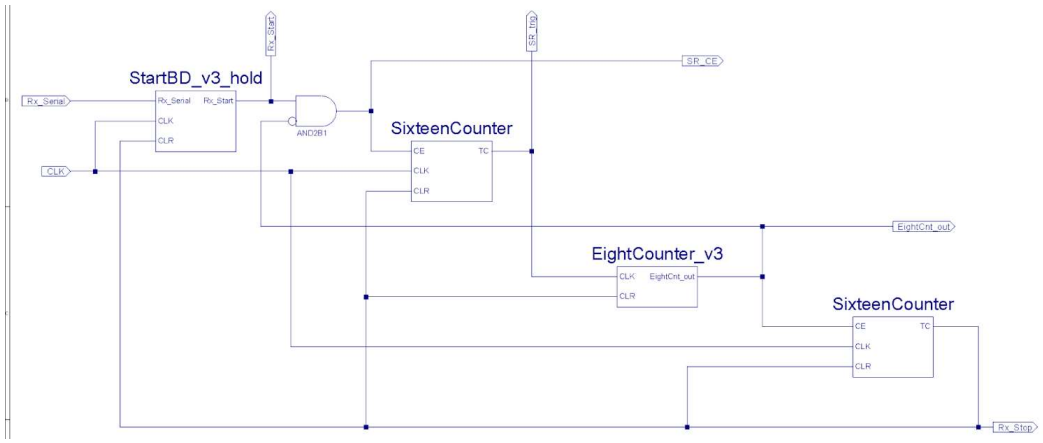


Figure 14. Xilinx implementation of the control logic block.

Start Bit Detector (StartBD_v3_hold)

From Figure 14, the start bit detector (StartBD_v3_hold) was constructed based on the truth table as shown in Table 4, where X is the input (see Figure D1 in Appendix D for the state machine).

Table 4. Truth table of the start bit detector.

Present State (Q)				Next state (D)								Output (Z)
				X=0				X=1				
QA	QB	QC	QD	DA	DB	DC	DD	DA	DB	DC	DD	
0	0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	1	1	0	0	0	0	0
0	0	1	1	0	1	0	0	0	0	0	0	0
0	1	0	0	0	1	0	1	0	0	0	0	0
0	1	0	1	0	1	1	0	0	0	0	0	0
0	1	1	0	0	1	1	1	0	0	0	0	0
0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	0	0	1
1	X	X	X	X	X	X	X	X	X	X	X	X

As shown in Table 4, StartBD_v3_hold is a Moore machine as it is independent of the input X. A Moore machine was chosen for the start bit detector because compared to a Mealy machine, it is more stable, uses less logic gates, and is easier to control as the output is independent of the input data, making it more predictable. The start bit detector reads

the transmitted data signal every receiver clock cycle. If the signal goes to 1 before 8 consecutive zeros are counted, the next state of the flip-flops return to 0000. Moreover, the start bit detector was designed to hold its value at HIGH once 8 consecutive zeros are detected until the flip-flops are reset by Rx_Stop. This is to ensure that the start bit detector will not respond to 8 consecutive zeros in the transmitted data after the start bit. Figure 15 shows the minimised gates of the start bit detector circuit. The circuit was then constructed in Xilinx (see Figure D2 in Appendix D).

```
Minimized:
DA = QA  + X' QB QC QD;
DB = X' QB QC'  + X' QB QD' + X' QB' QC QD;
DC = X' QC' QD + X' QC QD';
DD = X' QA' QD';
Z = QA ;
```

Figure 15. Minimised logic for the start bit detector using Logic Friday.

16-bit Counter (SixteenCounter)

The 16-bit counter (SixteenCounter) used the available 16-bit counter symbol, CB4CE in Xilinx. The schematic of the SixteeenCounter is shown in Figure 16.

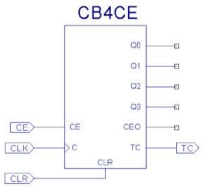


Figure 16. SixteenCounter schematic.

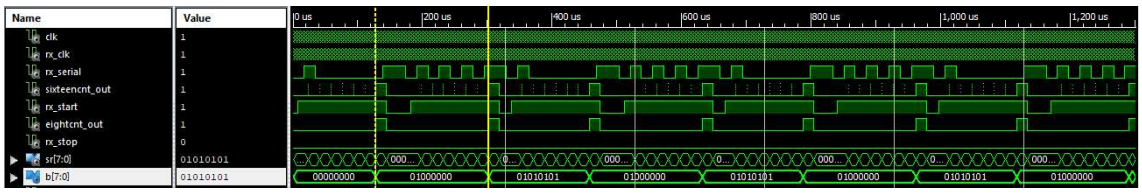
As shown in Figure 16, the Q0, Q1, Q2, Q3, and CEO of the CB4CE symbol are not used. Xilinx will automatically trim these ports during implementation. The counter is enabled when CE is HIGH and the counter increments by 1 for each rising edge at CLK. TC is the output of the 16-bit counter and is HIGH when Q0, Q1, Q2, and Q3 are all HIGH, which means that 16-bits have been counted. CLR clears the counter when it is HIGH.

8-bit Counter (EightCounter_v3)

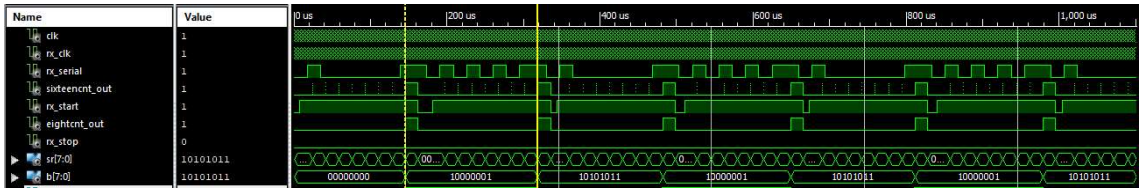
As there are no premade 8-bit counter symbols in Xilinx, an 8-bit counter was made from 4 flip-flops. Table 5 shows the truth table of the counter. As shown in Table 5, the 8-bit counter is a Moore machine and counts from 0000 to 1000. The counter does not count from 000 to 111 as it would not read the last bit of the transmitted data as shown in Figure 17.

Table 5. Truth table of EightCounter_v3.

Present State (Q)				Next state (D)				Output (Z)
QA	QB	QC	QD	DA	DB	DC	DD	
0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	1	0	0
0	0	1	0	0	0	1	1	0
0	0	1	1	0	1	0	0	0
0	1	0	0	0	1	0	1	0
0	1	0	1	0	1	1	0	0
0	1	1	0	0	1	1	1	0
0	1	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0	1
1	X	X	X	X	X	X	X	X



(a)

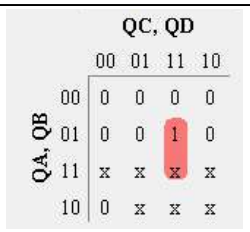
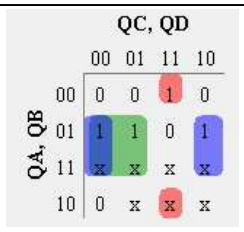
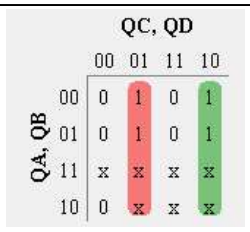
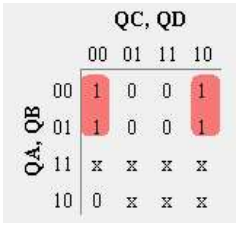
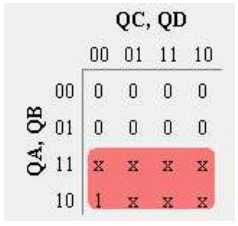


(b)

Figure 17. The test bench output when using (a) EightCounter (000 to 111) and (b) EightCounter_v3 (0000 to 1000).

As shown in Figure 17a, the 000 to 111 counter always fails to read the last bit of the transmitted data. Whereas in Figure 17b, the 0000 to 1000 counter is able to read the full transmitted byte. Hence, it was decided to use the EightCounter_v3 design. The K-map simplification for QA, QB, QC, QD, and Z of the EightCounter_v3 design are shown in Table 6. The circuit is constructed in Xilinx as shown in Appendix E.

Table 6. K-map simplification of the EightCounter_v3.

 $DA = QB\ QC\ QD$	 $DB = \overline{QB}\ QC\ QD + QB\ \overline{QC} + QB\ \overline{QD}$	 $DC = \overline{QC}\ QD + QC\ \overline{QD}$
 $DD = \overline{QA}\ \overline{QD}$	 $Z = QA$	

2.12 Task 17: Demonstrate a Working System

The logic circuits developed from Task 13 to 16 were combined as shown in Figure 18. The state machine diagram for the whole system is shown in Figure 19. The working system uses a register block to store the shift register values once the transmitted data has been fully read. This is crucial as the display board should display the speed accurately and stably. If the BCD to 7-segment decoder was connected directly to the shift register, the display would show unstable values as the transmitted bit is shifted into the shift register.

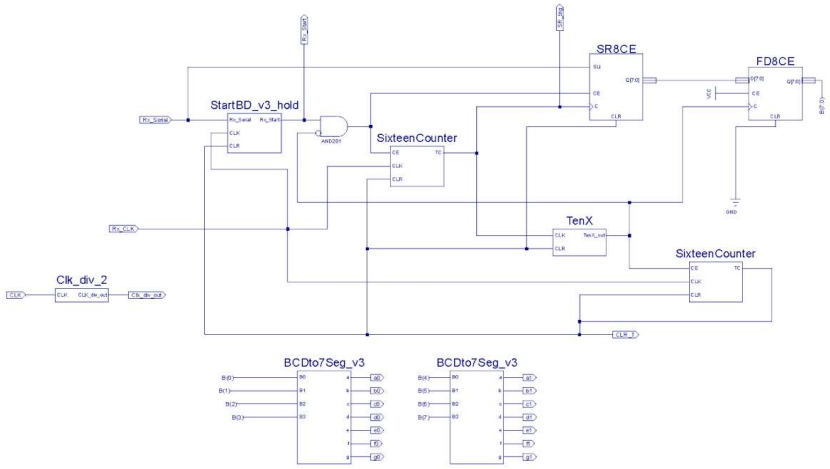


Figure 18. Schematic of the CPLD UART receiver.

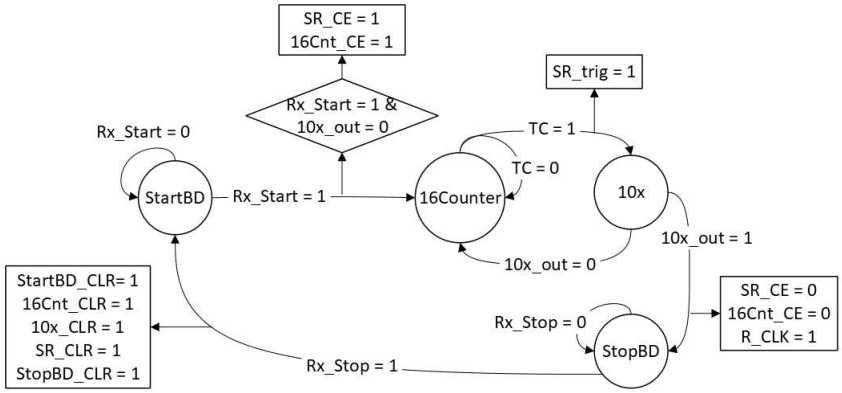


Figure 19. State machine diagram for the CPLD UART receiver.

From Figure 18 and 19, the start bit detector (StartBD) detects the start bit of the transmitted data (Rx_Serial). When a start bit is detected Rx_Start goes HIGH. If the Rx_Start is HIGH and the 10x counter output (10x_out) is LOW, indicating that the system has not finish reading 8 bits from Rx_Serial, the enable pins of the shift register (SR_CE) and 16-bit counter (16Cnt_CE) are set to HIGH (enabled). The control is now passed to the 16-bit counter (16Counter), which outputs HIGH (TC = 1) each 16 counts of the receiver clock cycle. TC is connected to the shift register clock (SR_trig). Hence, the 16Counter will control when the shift register shifts in a bit. TC is also connected to the 10x clock to count the number

of bits shifted into the shift register. Once 8 bits have been shifted into the register, the 10x_out goes HIGH. This disables the 16Counter and the shift register. 10x_out also triggers the clock of a register to parallelly shift in the bits stored in the shift register. This ensures that the display is always stable. The 10x_out passes the control to the stop bit detector (StopBD). Once the stop bit has been detected, the Rx_Stop becomes HIGH. This resets all the blocks in the circuit. Figure 20 shows the demonstration of the working system.

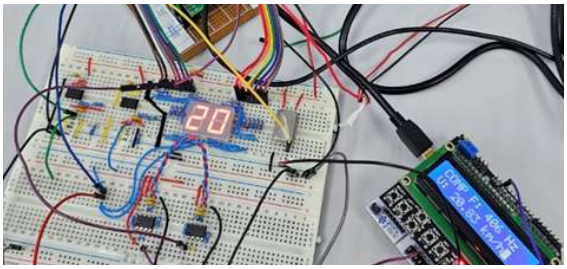


Figure 20. Demonstration of a working system.

As shown in Figure 20, the remote display board is able to correctly read and display the transmitted speed value from the STM32 via UART and the RS485 communication link.

10x Counter

As shown in Figure 18 and 19, the working system uses a 10x counter in place of the EightCounter_v3 from Task 16. This is because the experimental results show that the EightCounter_v3 would miss the last bit even though the test bench simulation results show otherwise as illustrated in Figure 21.

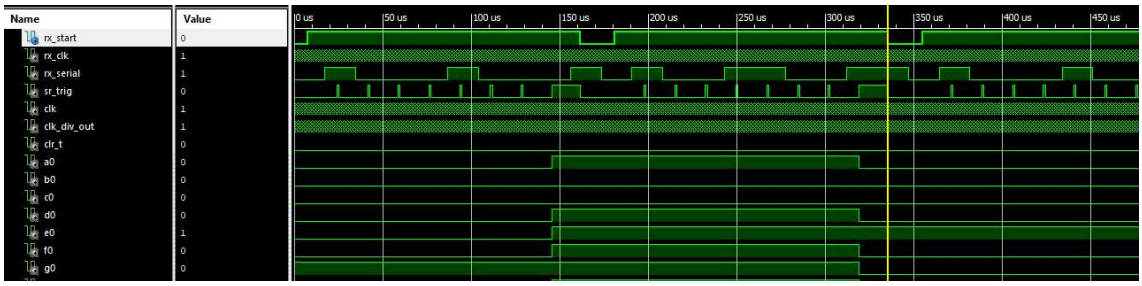


Figure 21. The UART receiver system with EightCounter_v3.

As shown in Figure 21, the inputs of the UART receiver system are 11 (0001 0001) and 99 (1001 1001). The EightCounter_v3 correctly allows 8 data bits to be shifted into the shift register as shown in the output of the BCD decoder (a, b, c, d, e, f, and g). However, when this design was uploaded and tested in the lab, the results were incorrect as shown in Table 7.

Table 7. Experiment results of the UART receiver system using EightCounter_v3.

Sent Value		Displayed Value		
Decimal	Binary	Decimal	Binary	Binary (Corrected)
00	0000 0000	00	0000 0000	0000 0000
11	0001 0001	44	0100 0100	0010 0010
22	0010 0010	22	0010 0010	0100 0100
33	0011 0011	64	0110 0100	0010 0110
44	0100 0100	11	0001 0001	1000 1000
55	0101 0101	55	0101 0101	1010 1010
66	0110 0110	33	0010 0010	0100 0100
77	0111 0111	77	0111 0111	1110 1110
88	1000 1000	00	0000 0000	0000 0000
99	1001 1001	44	0100 0100	0010 0010

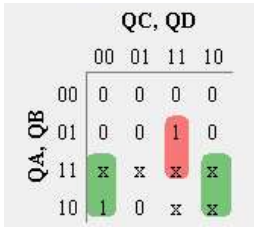
At the time of testing, the LSB and MSB of the data were flipped. This error was corrected in the working system. Aside from that, if the displayed value in binary was corrected (flipped), it shows that the last bit (MSB) was not shifted into the shift register. Hence, the 10x counter, which counts from 0000 to 1001, is used to ensure the last bit will be

shifted into the shift register. The truth table, K-map minimization, and schematic of the 10x counter is shown in Table 8, Table 9 and in Appendix F respectively.

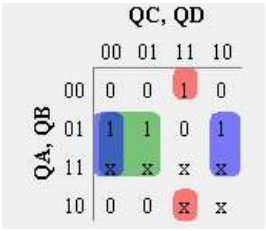
Table 8. Truth table of the 10x counter.

Present State (Q)				Next state (D)				Output (Z)
QA	QB	QC	QD	DA	DB	DC	DD	
0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	1	0	0
0	0	1	0	0	0	1	1	0
0	0	1	1	0	1	0	0	0
0	1	0	0	0	1	0	1	0
0	1	0	1	0	1	1	0	0
0	1	1	0	0	1	1	1	0
0	1	1	1	1	0	0	0	0
1	0	0	0	1	0	0	1	0
1	0	0	1	0	0	0	0	1
1	X	X	X	X	X	X	X	X

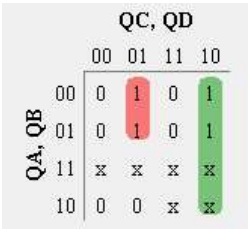
Table 9. K-map minimization of the 10x counter.



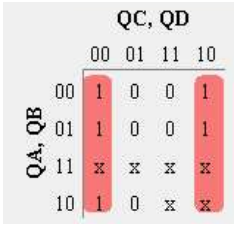
$$DA = QB\,QC\,QD + QA\,\overline{QD}$$



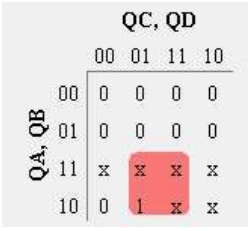
$$DB = \overline{QB}\,QC\,QD + QB\,\overline{QC} + QB\,\overline{QD}$$



$$DC = \overline{QA}\,\overline{QC}\,QD + QC\,\overline{QD}$$



$$DD = \overline{QD}$$



$$Z = QA\,QD$$

UCF File

Figure 22 shows the UCF file of the working system. As shown in Figure 22 line 2 and 3, the receiver clock (Rx_CLK) is buffered to the global clock (GCK) of the CPLD to provide low skew and reduce loading delays of the clock signal to the other blocks.

```
1   NET CLK LOC = "P42";
2   NET RX_CLK LOC = "P43";
3   NET "Rx_CLK" BUFG=CLK;
4   NET Rx_Serial LOC = "P2";
5   NET SR_trig LOC = "P3";
6   NET Clk_div_out LOC = "P39";
7
8   //NET Rx_Stop LOC = "P2";
9   NET CLR_T LOC = "P30";
10  //NET "CLR_T" BUFG=SR;
11
12
```

```
13  //MSB
14  NET a0 LOC = "P31";
15  NET b0 LOC = "P32";
16  NET c0 LOC = "P33";
17  NET d0 LOC = "P34";
18  NET e0 LOC = "P36";
19  NET f0 LOC = "P37";
20  NET g0 LOC = "P38";
21
22  //LSB
23  NET a1 LOC = "P5";
24  NET b1 LOC = "P6";
25  NET c1 LOC = "P8";
26  NET d1 LOC = "P12";
27  NET e1 LOC = "P13";
28  NET f1 LOC = "P14";
29  NET g1 LOC = "P16";
```

Figure 22. UCF file of the working system.

2.13 Task 18: Completed Radar Speed Detection System

This task combines all the tasks to create a complete speed detector system. Figure 23 depicts the flowchart of the complete speed detector system. The signal conditioning circuit is the PCB version. In the complete system, only the STM32 codes had to be altered to use all the functions, such as the remote display board, LCD display, COMP, and ADC system.

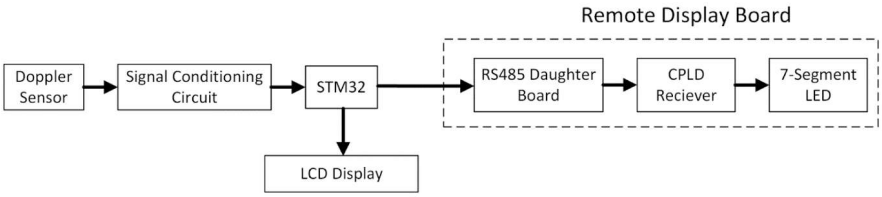


Figure 23. Complete speed detector system flowchart.

The display.c and display.h codes were adjusted to have functions to return the frequency detected from either the COMP or ADC system. These two files allow the correct mode, frequency, and speed to be displayed on the LCD. In addition, these files also ensure that correct BCD data is transmitted from the STM32 to the remote display board. There

are only four push buttons on the LCD keypad shield with distinct analogue values that can be used. The flowchart of the display.c file is depicted in Figure 24. Press the left button to switch to the ADC system or right for the COMP systems. The LCD displays both frequency and speed, the user can easily change the unit of speed between m/s, km/h and mph by pressing the down button.

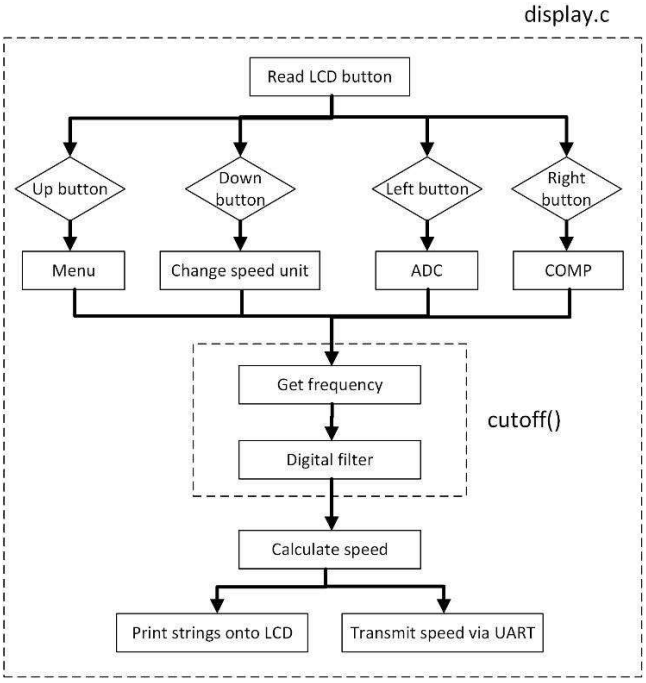


Figure 24. Flowchart of the display.c file

Figure 25 shows a code snippet of a function in the display.c file that processes the analog reading obtained after a push button is pressed. This function will change the variables according to the button pressed. A variable named mode is used to keep track of the current mode of the LCD, which are the menu, ADC system, and COMP system. While another variable named speed_unit is used to track the speed unit of displayed speed. There are three possible values for speed_unit, 1, 2, and 3, which correspond to (m/s), (km/h), and (mph), respectively. This variable will be reset to 1 if the value stored is more than 3.

```
void keypad_read_key(uint32_t adc_readout, UART_HandleTypeDef *huart){
    //for when up button is pressed
    if (adc_readout > 700 && adc_readout < 800){
        //reset button to show menu
        mode=0;
    }

    //for when down button is pressed
    else if (adc_readout > 1700 && adc_readout < 1900 ){
        //change the speed unit each time the down button is pressed
        if(speed_unit==3){
            //reset back to M/S
            speed_unit=1;
        }
        else{
            speed_unit++;
        }
    }

    //for when left button is pressed
    else if (adc_readout > 2000 && adc_readout < 3000){
        //change to ADC mode
        mode = 1;
    }

    //for when right button is pressed
    else if (adc_readout == 0){
        //change to COMP mode
        mode = 2;
    }

    checking();
    Output(huart);
}
```

Figure 25. Read button function in the display.c file.

From Figure 25, the next stage is cut-off, which limits the frequency returned by the ADC or COMP system before display. This is because the large gain of the amplifier in the COMP signal conditioning circuit results in inaccurate frequency detection of input signals outside the passband (100 Hz to 600 Hz). Therefore, a function is used to set input signal frequencies outside the passband to 0. Otherwise, the frequency detected will remain unchanged. The subsequent stage calculates the speed, which can be found using (2), where the f_t is the transmitted frequency and c is the speed of light. To change the speed unit obtained from (2), c can be converted from the standard 3×10^8 m/s into km/h or mph.

$$v = c\frac{f}{2f_t} \quad (2)$$

The two last stages from Figure 25 display the speed and frequency on the LCD as well as transmitting the speed value through UART to the remote display system. Since the remote display can only display values ranging from 0 to 99, a function is used to set the speed value to 99 if it exceeds 99. Otherwise, the speed value remains unchanged and is converted to BCD before transmission to the remote display system.

3.0 Experimental/Simulation Results, Analysis and Discussion

3.1 Task 7: Comparator-based Complete Doppler System

The new and old methods are experimented and compared to validate the new method. The old method is called COMP1, while the new method is called COMP2. The experiment setup is shown in Figure 26. The signal generator was set to output a sine wave with a ripple of 400 mV. This signal will go through the signal conditioning PCB before inputted to the STM32 internal comparator. Both COMP1 and COMP2 systems have the same comparator configuration such as a reference voltage of 1.22 V.

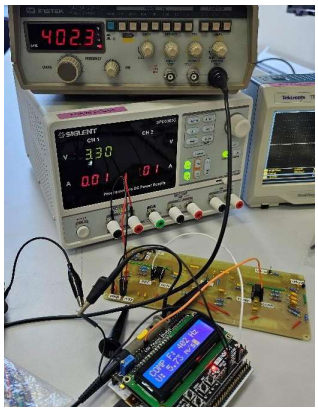


Figure 26. COMP system experiment setup.

Both systems were tested with varying signal input frequencies ranging from 100 Hz to 600 Hz. Table 10 illustrates the results obtained from the experiment. The results show that COMP1 is more unstable than COMP2. In particular, it was found that the frequency readings from COMP1 frequently deviated from the expected values and fluctuated erratically. Table 10 shows that for all test cases, COMP1 always had a significantly higher frequency detected than COMP2. The maximum error percentage was 8% for COMP1, while COMP2 was 0.2%. Thus, COMP1 had significantly higher error than COMP2. Across all 6 test cases, COMP1 had a mean error in frequency measurement of around 26.7 Hz, while COMP2 only had a mean error of 0.67 Hz, a reduction of 26.03 Hz error. These results are enough to prove that COMP2 has significantly improved the

comparator system accuracy. This could be due to COMP2's significantly higher sampling rate of around 1 MHz while COMP1 is only 1 Hz.

Table 10. COMP1 and COMP2 frequency measurement results.

Actual signal frequency (Hz)	COMP1 detected frequency (Hz)	COMP2 detected frequency (Hz)
100	120	100
200	220	201
300	330	301
400	430	400
500	540	501
600	620	601

3.2 Task 8: ADC-based Complete Doppler System

The ADC-based Doppler system was tested using a DAC sine wave generator developed on the STM32 board as shown in Figure 27. The DAC sine wave generator is an efficient way to test the Doppler system at home when the lab is inaccessible.

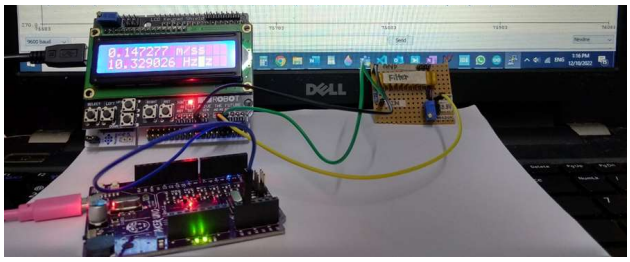


Figure 27. Set up for the ADC-based Doppler system experiment.

As shown in Figure 27, the STM32 generates a sine wave using a DAC. The sine wave is passed to the filter then to the ADC input of the ADC-based Doppler system. The filter circuit was replaced by the signal conditioning circuit later for more accurate experimentation. The frequency of the DAC waveform is calculated using (3) where 100 is the size of the array storing the sine wave magnitude. The result of the experiment is tabulated in Table 11.

$$Sine\ wave\ generator\ frequency\ = \frac{timer\ clock\ frequency}{prescaler \times counter\ period \times 100} \quad (3)$$

Table 11. Experiment results of the ADC-based Doppler system.

Timer Counter Period	Sine wave generator frequency (Hz)	ADC reading (Hz)	Error (%)
100-1	100	99.53	0.470
50-1	200	200.00	0.000
33-1	303	303.30	0.100
25-1	400	400.02	0.005
20-1	500	500.49	0.098
17-1	588	587.82	0.031

As shown in Table 11, the ADC-based Doppler system can measure the input signal frequency with good accuracy as the maximum error is only 0.47%. It is also noticed that the ADC reading is very stable and does not glitch. Hence, the ADC-based Doppler system is very robust to noise and very accurate in measuring frequency and velocity. The ADC system was briefly tested again in the lab and the results were the same as Table 11.

3.3 Task 9: RS485 Communications Link (Software & Hardware)

The software and hardware of the RS485 communication link was tested by sending a 2-digit decimal number using the RS485 software and probing the STM32 UART output as well as the RS485 receiver. Table 12 shows the results of the experiment (see Appendix G for the figures).

Table 12. Experiment results for the RS485 communication link.

Transmitted decimal	Expected Logic Levels (LSB to MSB)	STM Output (LSB to MSB)	RS485 Receiver Output (LSB to MSB)
00	0 0000 0000 1	0 0000 0000 1	0 0000 0000 1
01	0 1000 0000 1	0 1000 0000 1	0 1000 0000 1
02	0 0100 0000 1	0 0100 0000 1	0 0100 0000 1
03	0 1100 0000 1	0 1100 0000 1	0 1100 0000 1
04	0 0010 0000 1	0 0010 0000 1	0 0010 0000 1
05	0 1010 0000 1	0 1010 0000 1	0 1010 0000 1
06	0 0110 0000 1	0 0110 0000 1	0 0110 0000 1
07	0 1110 0000 1	0 1110 0000 1	0 1110 0000 1
08	0 0001 0000 1	0 0001 0000 1	0 0001 0000 1
09	0 1001 0000 1	0 1001 0000 1	0 1001 0000 1

As shown in table 12, the RS485 communication link is functioning as expected. Therefore, the RS485 software and hardware is working.

3.4 Task 10: Power Supplies

Before the voltage regulators can be used to power the JTAG and the Xilinx board, it must be thoroughly tested. For the 1.8 V voltage regulator, when the supply voltage to the IC is larger than 1.8 V, the output maintains at 1.74 V as depicted in Figure 28. This is a reduction of 0.06 V from the expected 1.8 V output. When the supply voltage is less than 1.8 V, the voltage regulator's output significantly drops from 1.8 V to be the same as the supply voltage.

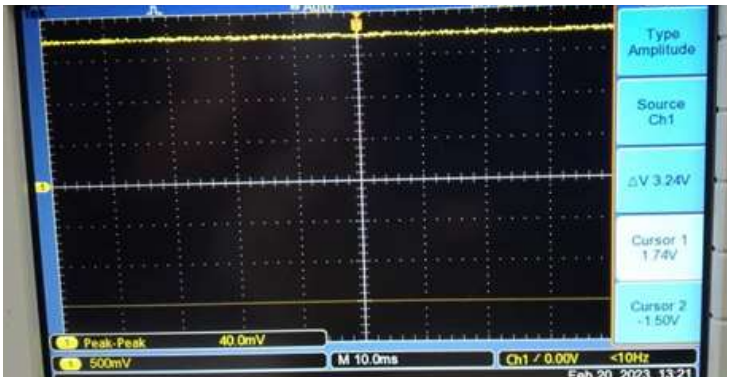


Figure 28. Voltage regulator with 1.8 V output probed using oscilloscope.

For the 3.3 V voltage regulator, when the supply voltage was higher than 3.3 V, the output of the voltage regulator maintained at 3.28 V as depicted in Figure 29. The 3.3 V voltage regulator behaves similar to the 1.8 V when its supply voltage is below 3.3 V. The slight difference between the actual and expected output voltage could be due component tolerance.

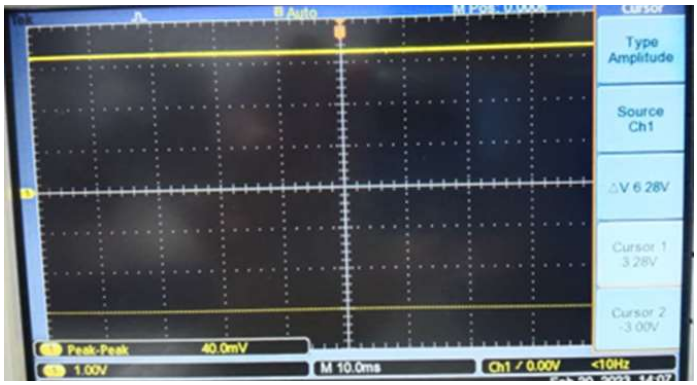


Figure 29. Voltage regulator with 3.3 V output probed using oscilloscope.

3.5 Task 11: External Clock

The output of the external clock shows a square wave with a fast rise and fall time. The peak-to-peak voltage is 3.3V as illustrated in Figure 30. The external clock frequency was measured to be 1.8431MHz, which is close to the expected frequency of 1.8432MHz.

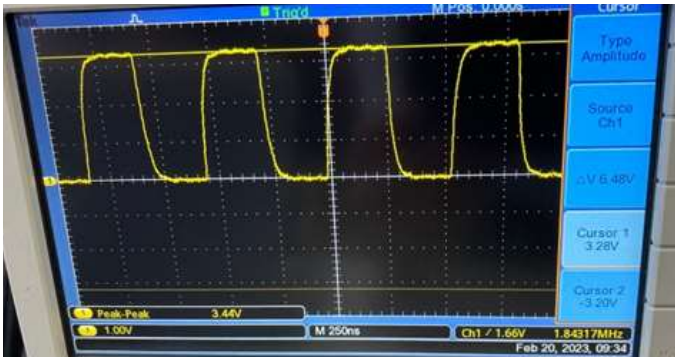


Figure 30. Output of the external clock.

3.6 Task 12: LED Interfacing to the CPLD

Figure 31 shows the physical connections made for each 7-segment display. The subsequent sections will cover the 7-segment display working with these connections.

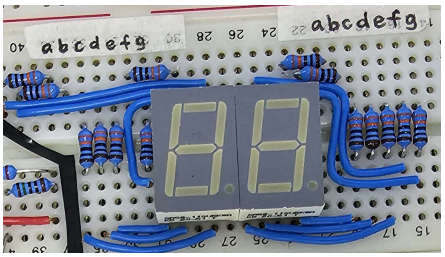


Figure 31. Physical connections of the 7-segment display.

3.7 Task 13: Clock Divider

The clock divider schematic used to achieve 57600 baud is illustrated in Figure 32. The Xilinx code was merged into a test bench file. A constant time of 0.542 us was used as the clock period. Figure 33 shows the simulation results.

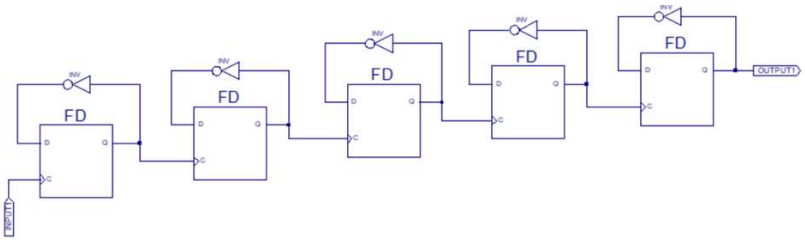


Figure 32. Clock divider for 57600 baud.

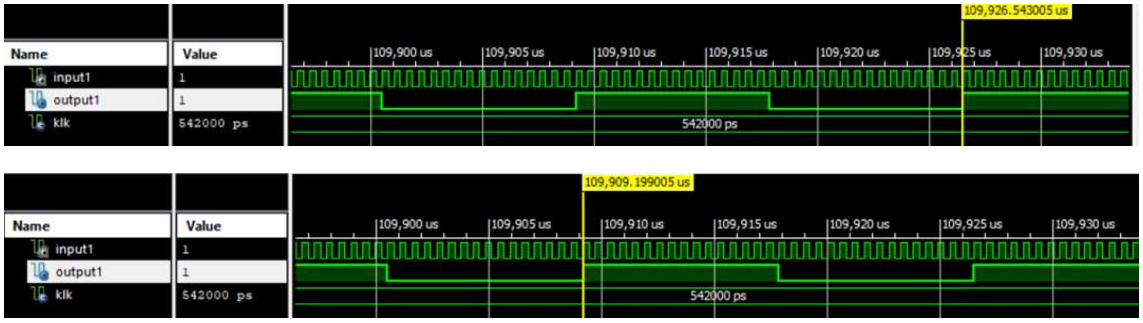


Figure 33. Output of the clock divider circuit.

$$F = \frac{1}{(109926-109909)\times 10^{-6}} = 57636\ Hz$$

The calculated frequency was determined to be 57.636 kHz, the value corresponds to the expected frequency of 57600 baud. As a result, the clock divider circuit can successfully generate an internal clock signal of

57600 baud in the simulation. The circuit for 921600 Hz baud serial signal was then created in a Xilinx schematic file, as shown in figure 34. The circuit was then put into a test bench file and tested similar to the 57600 baud. The simulation results are shown in Figure 35.

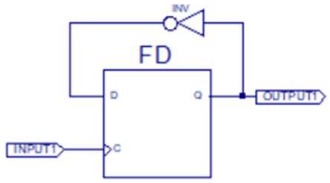


Figure 34. Clock divided by 2.

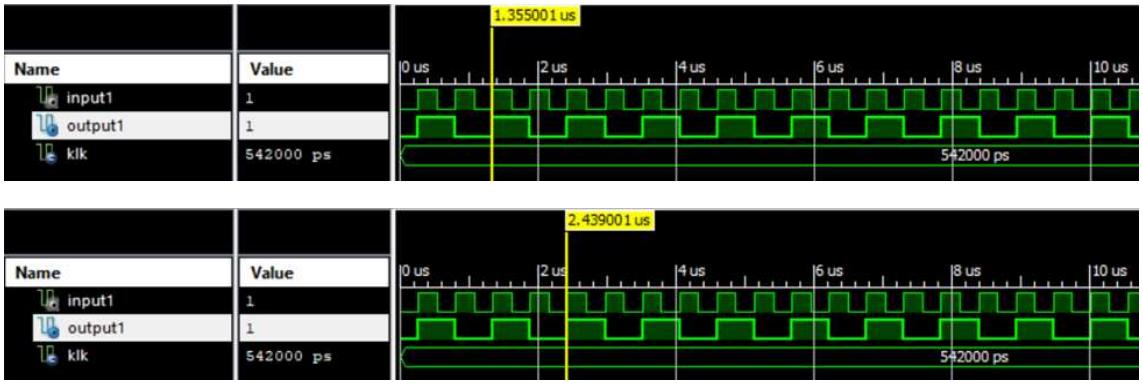


Figure 35. Output of the clock divided by 2 circuit.

$$F = \frac{1}{(2.439001 - 1.35001) \times 10^{-6}} = 922509\ Hz$$

The calculated frequency was determined to be 923kHz, the value corresponds to the expected frequency of 921600 baud. After verifying the clock divider schematic in Xilinx, the schematic was uploaded to the CPLD for practical testing. Since 16 times the baud rate will be used in the receiver, the 923 kHz was implemented. Figure 36 shows the output of the clock divider on the oscilloscope. As shown in Figure 36, the divided clock is 921.4 kHz which is very close to the 921.6 kHz desired frequency.

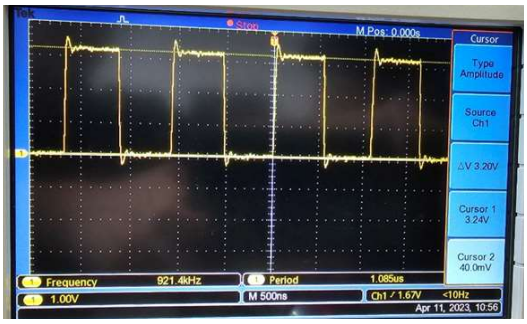
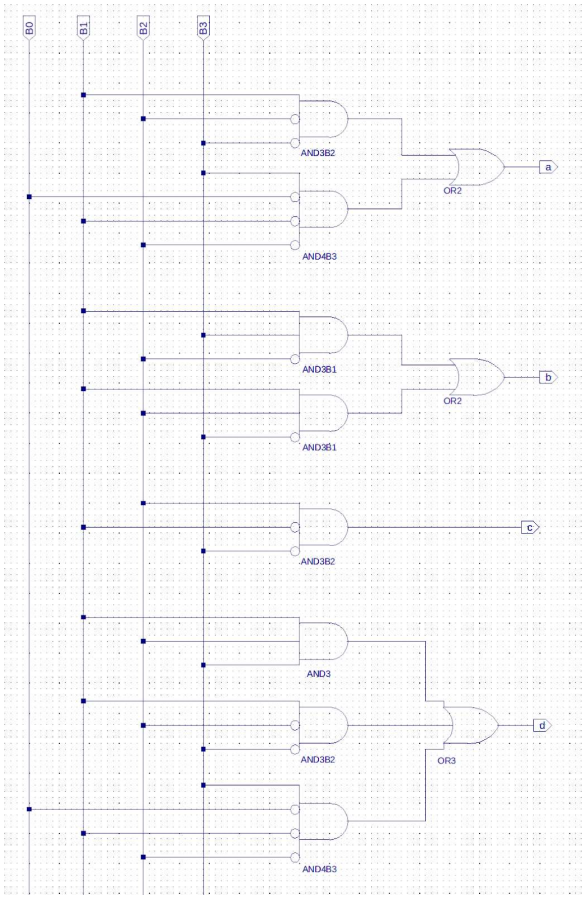


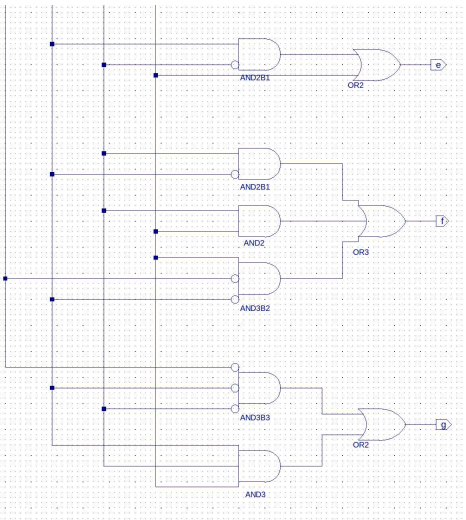
Figure 36. Output of clock divider circuit on oscilloscope.

3.8 Task 14: BCD-7 Segment Decoder

The BCD decoder schematic is constructed in Xilinx ISE according to the optimised equations for each LED segment. Figure 37 illustrates the schematic for each segment. B0 is the MSB and B3 is the LSB.



(a) Segment a, b, c, and d.



(b) Segment e, f, and g.

Figure 37. BCD to 7-segment display decoder.

A test bench was created to verify the operation of this BCD decoder (see Appendix H for code snippet of the test bench). In this test bench, the inputs to B0 to B3 are varied from 0000 to 1001 to test the decoder. The test bench results show that the decoder is working as expected as shown in Figure 38 (see Appendix I for all test bench results).

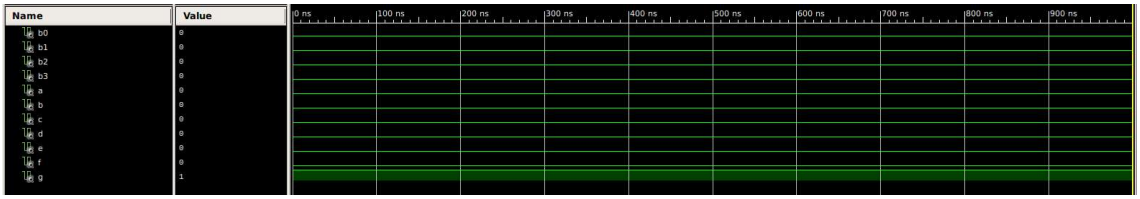


Figure 38. BCD decoder test bench results for 0000 input.

3.9 Task 15: Shift Register Block

Figure 39 shows the schematic used in the test bench to verify the operation of the SR8CE. The RX_Serial pin is the serial data that is shifted into the shift register and converted to parallel data. For experiment purposes, the CE will be set to 1 only for the simulation, while CLR will be 1 at the beginning to reset the shift register and will then be set to 0 for the remainder of the simulation (see Appendix J for test bench).

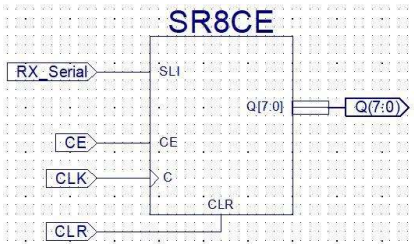


Figure 39. Schematic used for the shift register test bench.

The CLK period is set to 17.36 us to imitate the control logic signal which triggers the shift register every 16 receiver clock cycles. To demonstrate the bit shifting, RX_serial will be 1 for about 1 CLK cycle and 0 for the remainder of the simulation.

Figure 40 depicts the waveforms obtained from the shift register test bench. The Q[7:0] results from Figure 40 show that the logic HIGH bit

shifts in ascending order of Q every rising edge of the CLK. In addition, it can be found that the logic HIGH bit shifts through output Q from Q0 to Q7 by the end of the simulation. These results are sufficient to verify the shift register, SR8CE operates as expected.

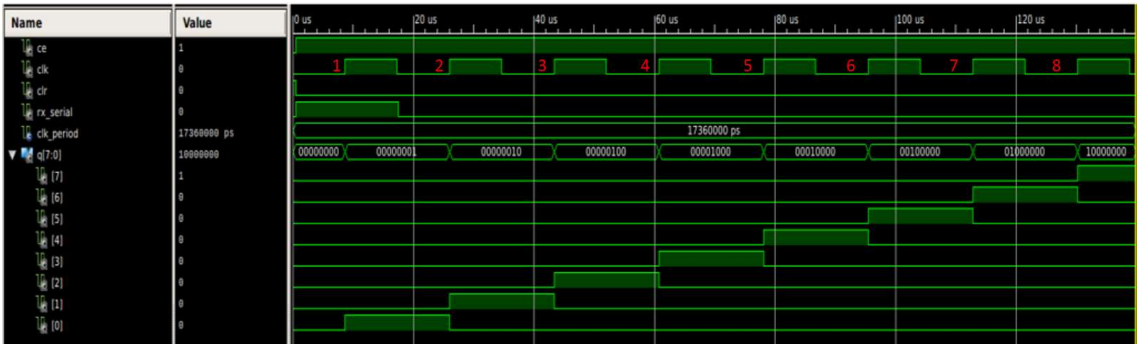


Figure 40. Shift register test bench waveform results.

3.10 Task 16: Control Logic Block

The control logic block was simulated using a test bench to verify its functionality. The test bench imitates a transmitted UART signal by periodically sending a start bit followed by 8 HIGH or LOW data bits and a stop bit. The input signal also has a baud rate of 57600 similar to the actual transmitter (see Appendix K for the code snippet of the test bench). Figure 41 shows the results of the simulation.

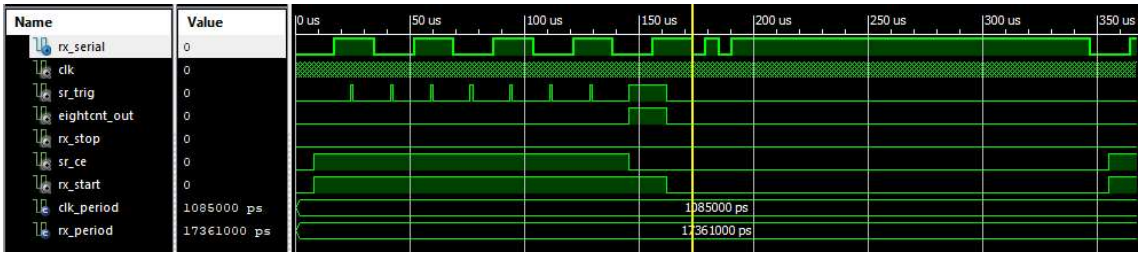


Figure 41. Simulation results of the control logic block test bench.

As shown in Figure 41, the bits sent to the receiver is 0 1010 1010 1, where the first 0 is the start bit and the last 1 is the stop bit. After that, a false start bit is sent to test the receiver response to glitches in the start bit. As shown in Figure 41, the control logic block can accurately position

the shift register trigger (SR_trig) at the centre of each data bit so that the correct logic level of the data bit can be shifted into the shift register. Besides that, the receiver correctly shifts in 8 bits as there are 8 rising edges of the SR_trig. The shift register also correctly enables before the first bit and disables after the last bit. The clear signal triggered by Rx_Stop also effectively resets the whole control block so that the receiver can receive the next signal. Figure 41 also shows that the start bit detector is working as the false start bit after the yellow cursor does not trigger the control logic to read the succeeding bits.

3.11 Task 17: Demonstrate a Working System

The remote display board is tested by sending a 2-digit decimal number from the STM32 UART to the receiver input of the system via the RS485 communication link. The experiment results are tabulated in Table 13.

Table 13. Experiment results of the remote display system.

Sent Value	Receive Value
00	00
11	11
22	22
33	33
44	44
55	55
66	66
77	77
88	88
99	99

As shown in Table 13, the remote display system is able to accurately display the transmitted values. Figure 42 compares the SR_trig with the start bit detector output (see Appendix L for other figures).

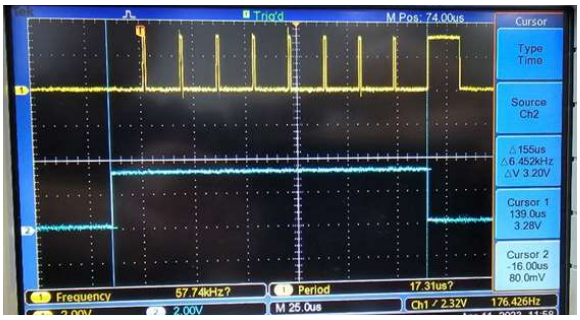


Figure 42. Oscilloscope SR_trig with start bit detector output.

From Figure 42, it can be inferred that the CE of the shift register goes LOW before the last rising edge of the SR_trig. Hence, the 10x counter worked as the last bit (MSB) does not depend on the final rising edge of the SR_trig signal to be shifted into the shift register. Comparatively, the EightCounter_v3 depends on the last rising edge of SR_trig to detect the MSB. Hence, the CE of the shift register would already be LOW before the SR_trig could shift in the last bit.

3.12 Task 18: Completed Radar Speed Detection System

Since there were no available doppler sensor in the lab, a signal generator was used as a substitute. Figure 43 showcases the speed detector system capable of displaying the speed correctly on both the LCD and the 7 segment displays using COMP system mode. As shown in Figure 43, the COMP system was able to accurately determine the frequency of the sine wave generator which is at 404.7 Hz.

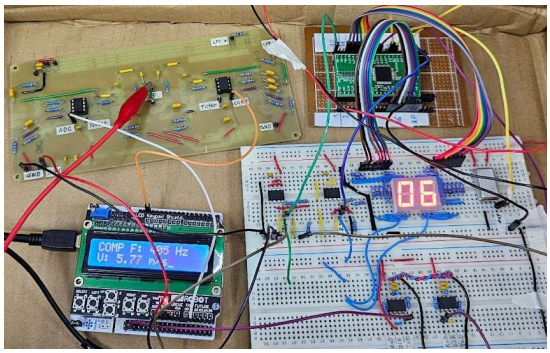


Figure 43. Complete speed detector system in COMP system mode.

The same sine wave signal was also tested with the ADC system as shown in Figure 44 by pressing the left push button on the LCD keypad shield. It can be seen that the ADC system detects a frequency of 404 Hz. For both systems, when the frequency is in the stopband, the LCD will display 0 for both the frequency and speed. The remote display system will also display 0 on both the 7 segment displays. Therefore, the performance of the complete radar speed detection system is very desirable.

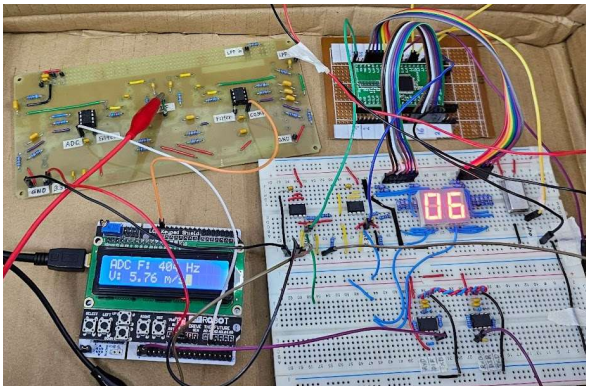


Figure 44. Complete speed detector system in ADC system mode.

4.0 Conclusion

In conclusion, the complete Doppler radar speed detection system has been successfully developed. The PCB signal conditioning circuit, ADC and COMP systems from Semester 1 have been integrated with the remote display board developed in Semester 2 to form a system that is able to accurately calculate the speed of a moving object and display the speed on two 7-segment displays. The system also contains a menu selection to pick between ADC and Comparator for speed detection, and the speed unit that will be shown on the 7-segment display and LCD screen.

Finally, the speed detector system is quite accurate as discussed in section 3.1 and 3.2 with a maximum error of 0.2% for the COMP system and 0.47% for the ADC system. Comparatively, the COMP system has a lower maximum error than the ADC system. The COMP system is also

more cost effective than the ADC system as it does not require high computational power for FFT calculation. Hence, it can be concluded that the COMP system is better than the ADC system.

5.0 References

[1] Giang Lam, N. Vaasa University of Applied Sciences VAMK (Vaasan ammattikorkeakoulu). (2021). VEHICLE SPEED MEASUREMENT USING DOPPLER EFFECT. Theseus. https://www.theseus.fi/bitstream/handle/10024/496044/Thesis.pdf?sequence=2&isAllowed=y

[2] Zin Tun, M., & Thwe Zin, K. (2019). Implementation of Doppler Radar-Based Vehicle Speed Detection System. Ijtsrd. https://www.ijtsrd.com/papers/ijtsrd26653.pdf

[3] Bhuiyan, Mohammad & Rosly, Hasrul & Reaz, Mamun Bin Ibne & Minhad, Khairun & Husain, Hafizah. (2014). Advances on CMOS Shift Registers for Digital Data Storage. TELKOMNIKA Indonesian Journal of Electrical Engineering. 10.11591/telkomnika.v12i5.5207.

Appendix A: Comparator.c Code Snippet

```c
#include "comparator.h"
uint32_t frequency = 0;
uint32_t T1 = 0;
uint32_t T2 = 0;
uint16_t TIM16_OVC = 0;
uint8_t state = 0;
uint32_t ticks = 0;

void rising_edge_trigger(COMP_HandleTypeDef *hcomp, COMP_HandleTypeDef *hcomp1, TIM_HandleTypeDef* htim16){
    //if the trigger callback is caused by comparator 1
    if (hcomp == hcomp1){
        if(state == 0)
        {
            T1 = __HAL_TIM_GET_COUNTER(htim16);
            TIM16_OVC = 0;
            state = 1;
        }
        else if(state == 1)
        {
            T2 = __HAL_TIM_GET_COUNTER(htim16);
            ticks = (T2+(TIM16_OVC*65536)) - T1;
            //ticks = (T2 + (TIM2_OVC * 65536)) - T1;
            frequency = (uint32_t)(1/(CLK_Period*ticks));
            state = 0;
        }
    }
}

void Update_rollover(TIM_HandleTypeDef* htim, TIM_HandleTypeDef* htim16){
    if(htim==htim16){
        TIM16_OVC++;
    }
}

int get_comp_frequency(){
    return frequency;
}
```

Figure A1. Code snippet of comparator.c.

Appendix B: 7-Segment to Xilinx Connections

Table B1. Seven segment 1 (MSB DIGIT) connections.

Display segment	Xilinx XC2C64A pin	XILINX XC2C64A CARRIER PCB pin
a	31	1_12
b	32	1_11
c	33	1_10
d	34	1_9
e	36	1_3
f	37	1_2
g	38	1_1

Table B2. Seven segment 2 (LSB DIGIT) connections.

Display segment	Xilinx XC2C64A pin	XILINX XC2C64A CARRIER PCB pin
a	5	-
b	6	4_2
c	8	4_7
d	12	4_11
e	13	4_13
f	14	4_14
g	16	4_15

Appendix C: Shift Register

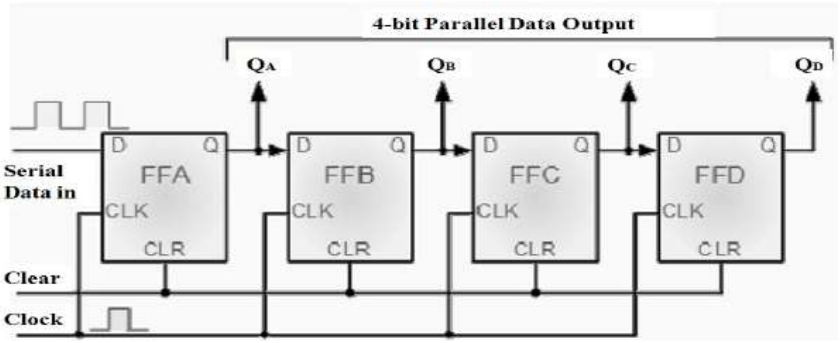


Figure C1. 4-bit serial-in parallel-out shift register [3].

Table C1. SR8CE output shifting pattern.

Current Output	Next Output
Q0	Q1
Q1	Q2
Q2	Q3
Q3	Q4
Q4	Q5
Q5	Q6
Q6	Q7
Q7	Shifted out

University of Nottingham
UK | CHINA | MALAYSIA

Appendix D: Start Bit Detector

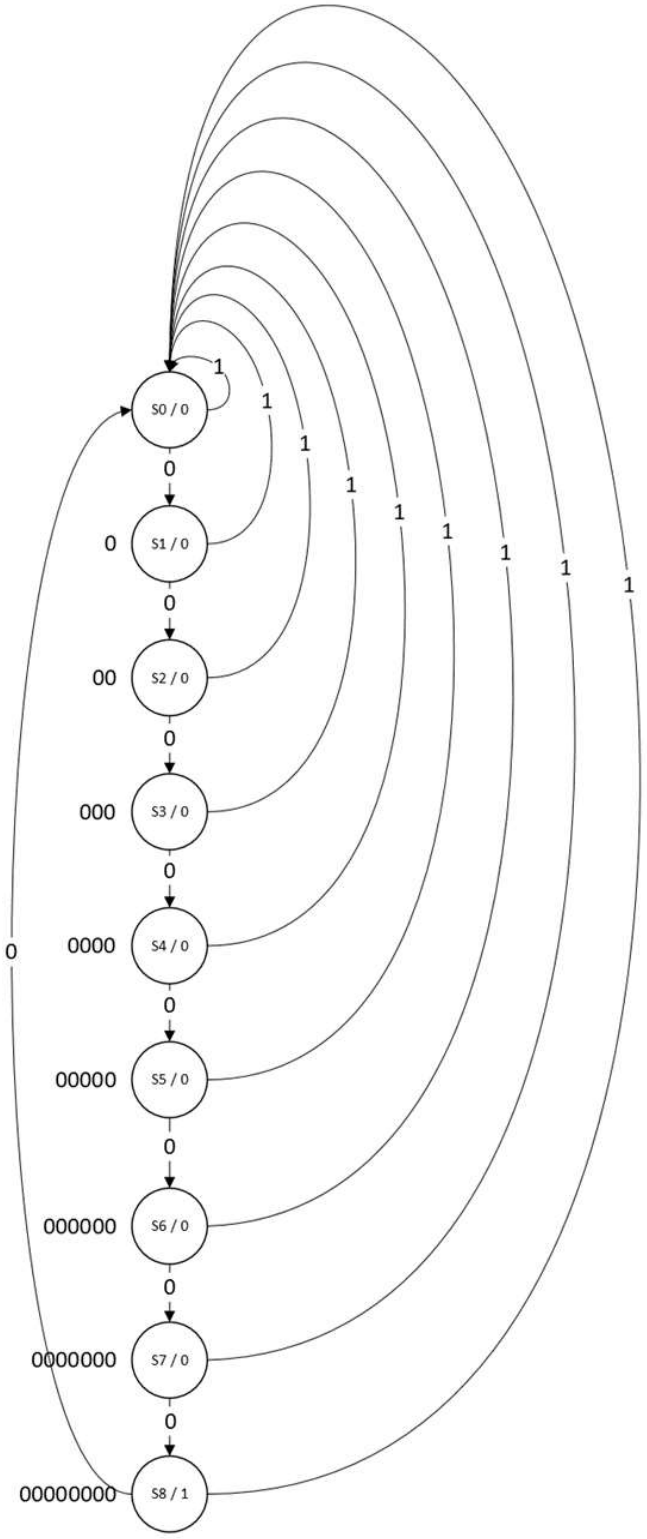


Figure D1. State machine of the start bit detector (StartBD_v3_hold).

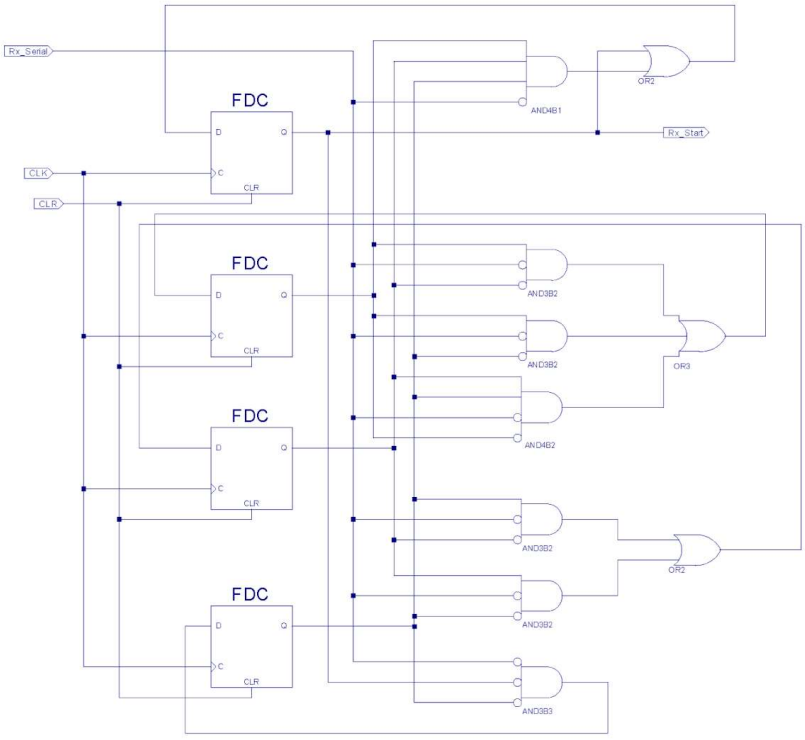


Figure D2. StartBD_v3_hold schematic.

Appendix E: 8-Bit Counter

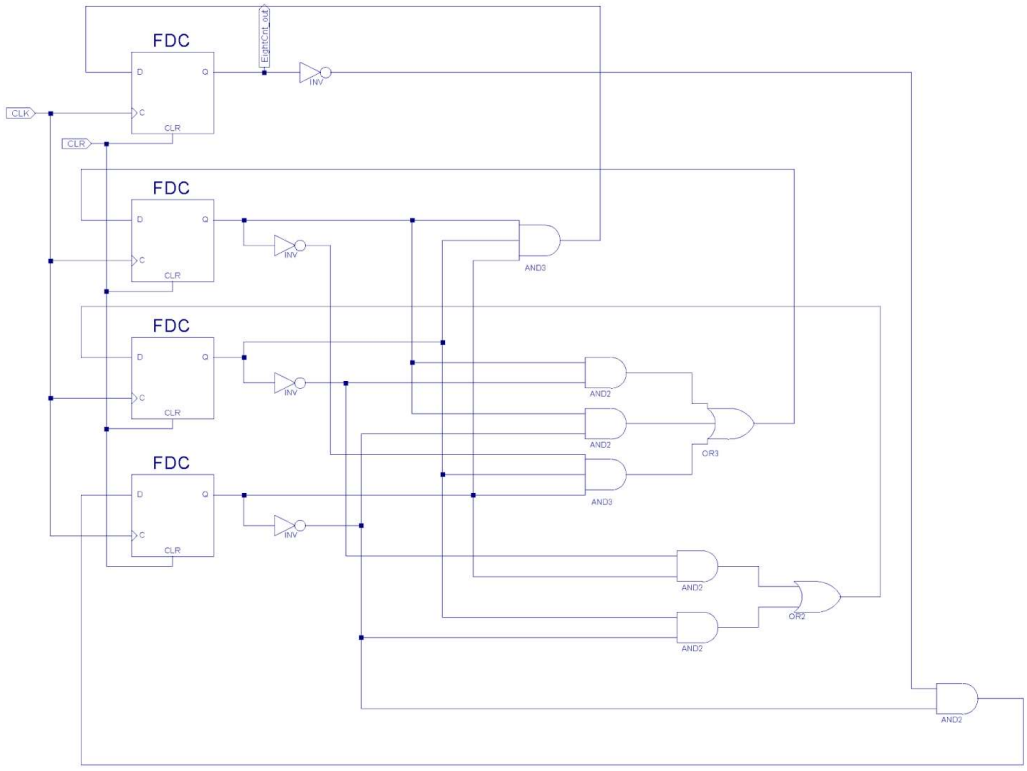


Figure E1. EightCounter_v3 schematic.

Appendix F: 10x Counter

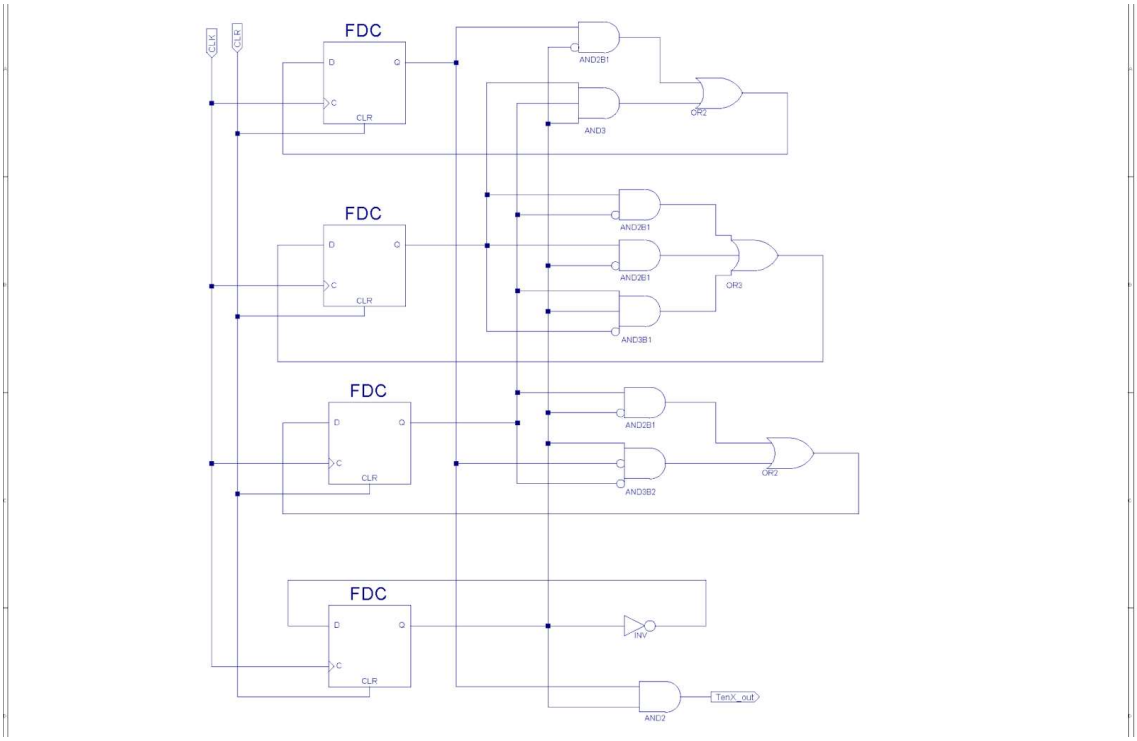


Figure F1. Schematic of the 10x counter.

Appendix G: RS485 Transmitter Input and Receiver Output

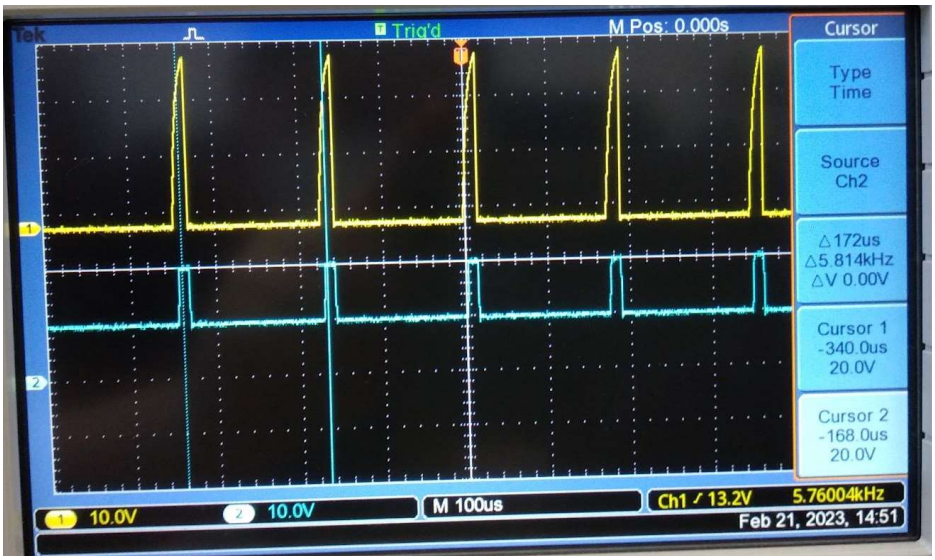


Figure G1. The RS485 software output (yellow) and RS485 receiver output (blue) when 0 is sent from the STM32 UART.

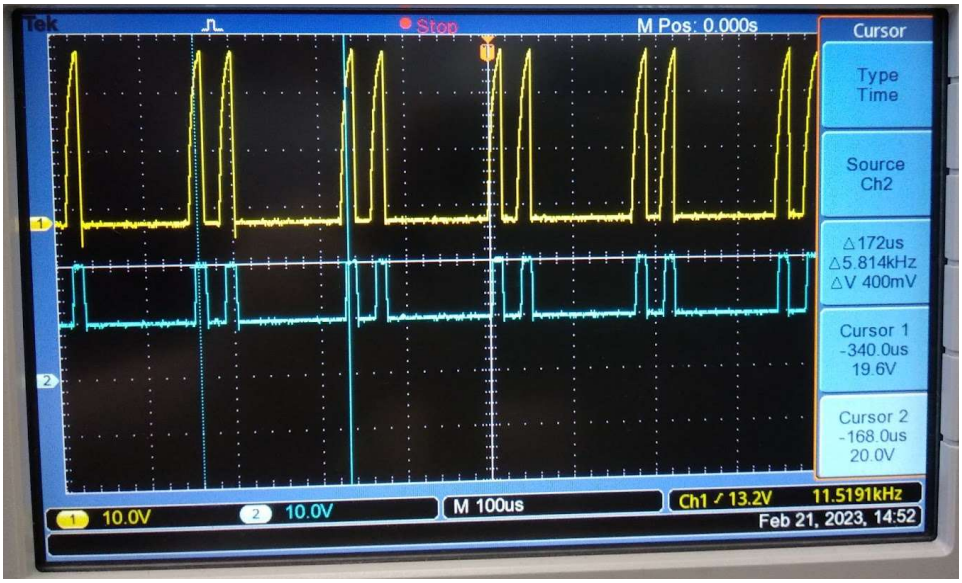


Figure G2. The RS485 software output (yellow) and RS485 receiver output (blue) when 1 is sent from the STM32 UART.

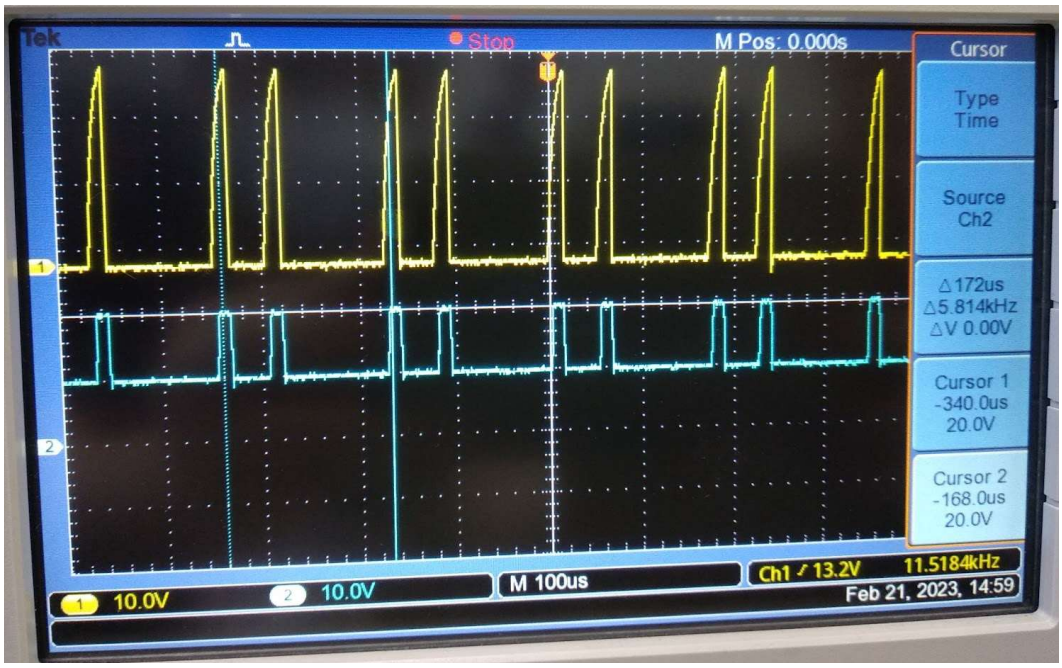


Figure G3. The RS485 software output (yellow) and RS485 receiver output (blue) when 2 is sent from the STM32 UART.

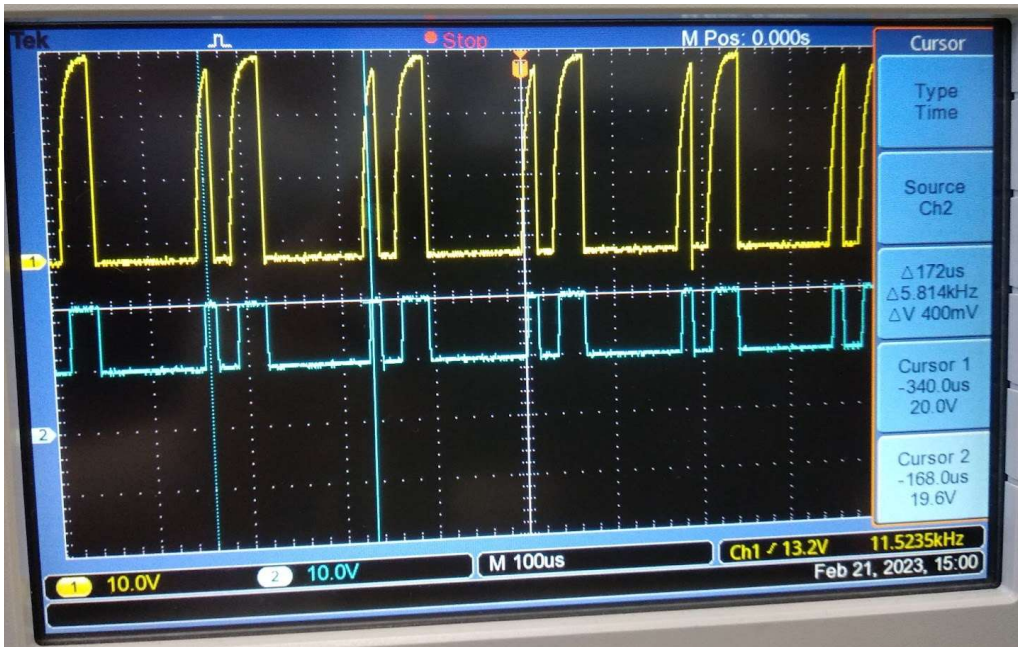


Figure G4. The RS485 software output (yellow) and RS485 receiver output (blue) when 3 is sent from the STM32 UART.

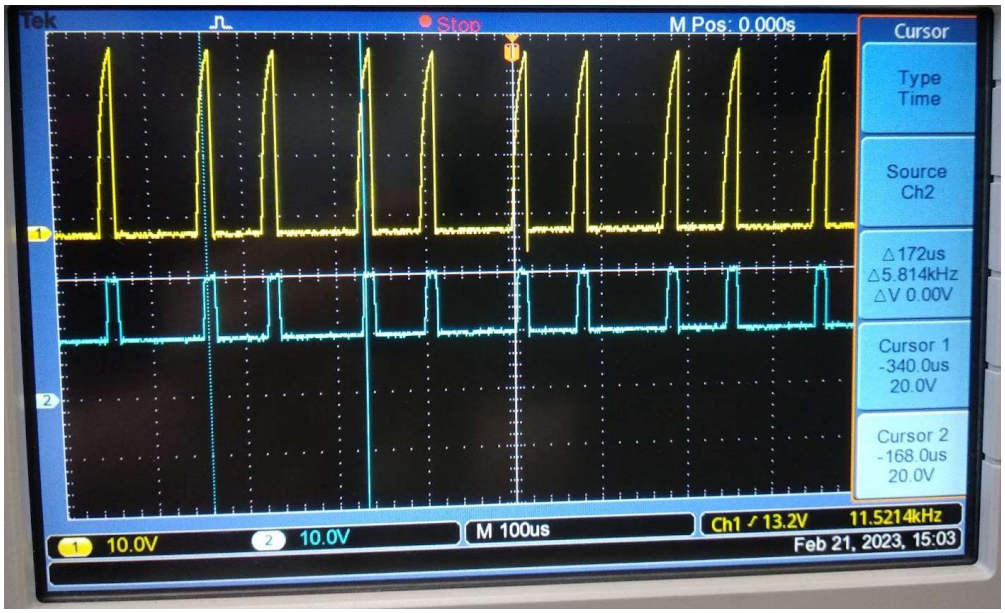


Figure G5. The RS485 software output (yellow) and RS485 receiver output (blue) when 4 is sent from the STM32 UART.

Figure G6. The RS485 software output (yellow) and RS485 receiver output (blue) when 5 is sent from the STM32 UART.

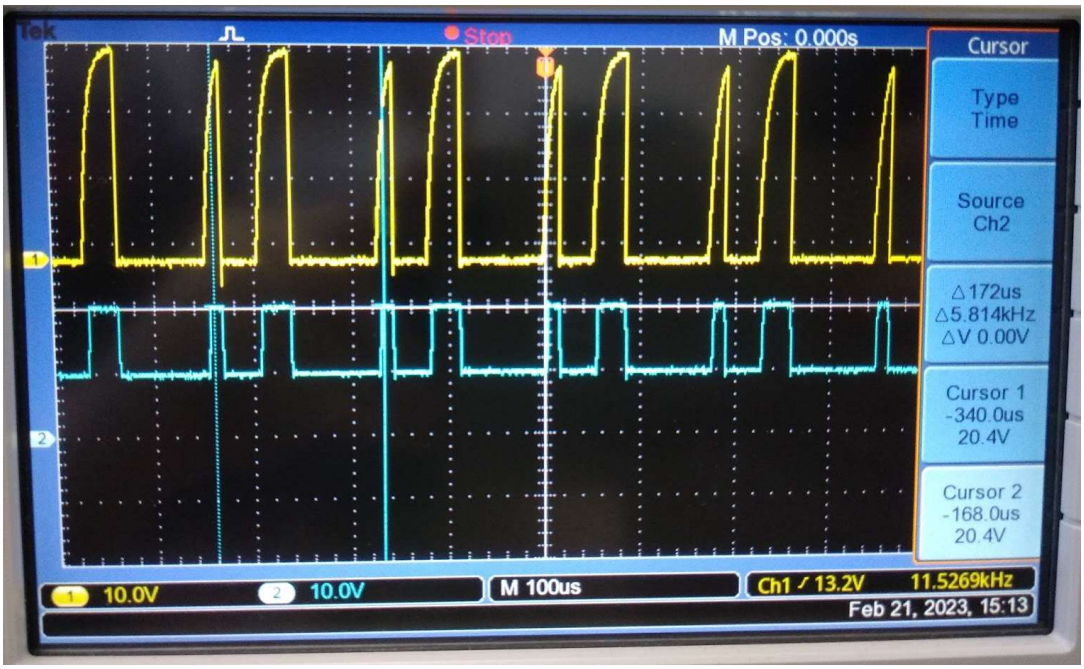


Figure G7. The RS485 software output (yellow) and RS485 receiver output (blue) when 6 is sent from the STM32 UART.

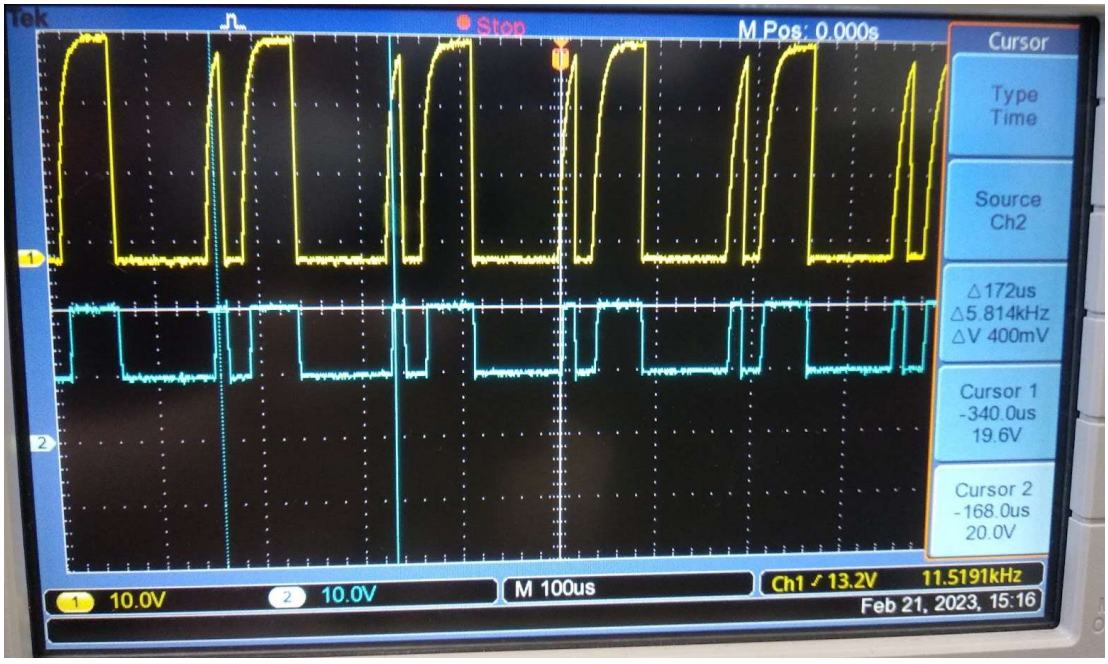


Figure G8. The RS485 software output (yellow) and RS485 receiver output (blue) when 7 is sent from the STM32 UART.

Figure G9. The RS485 software output (yellow) and RS485 receiver output (blue) when 8 is sent from the STM32 UART.

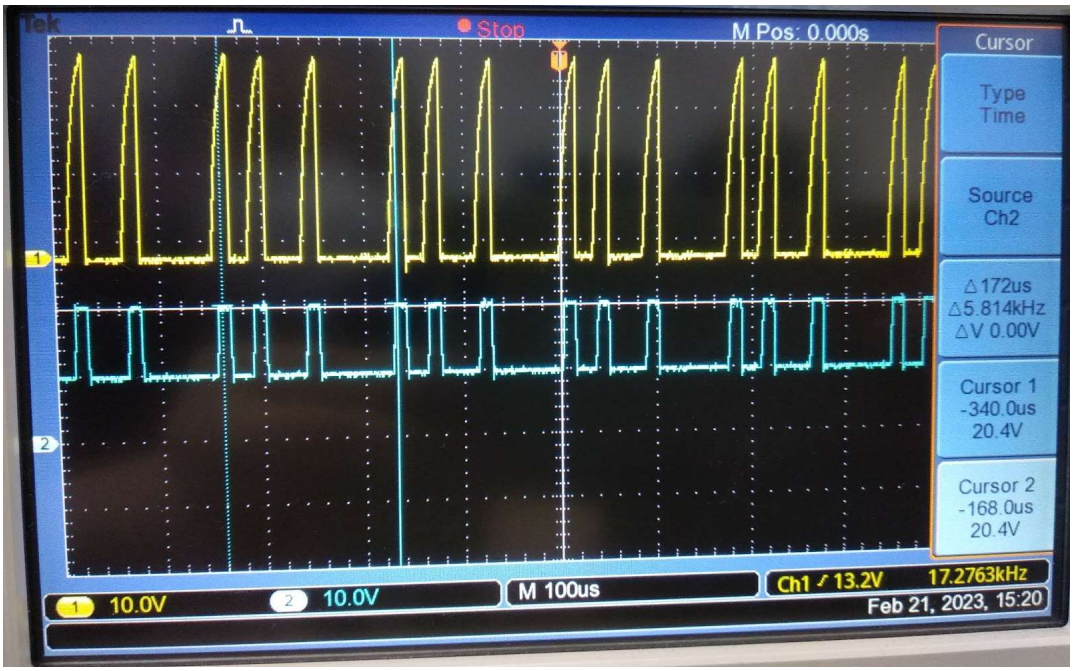


Figure G10. The RS485 software output (yellow) and RS485 receiver output (blue) when 9 is sent from the STM32 UART.

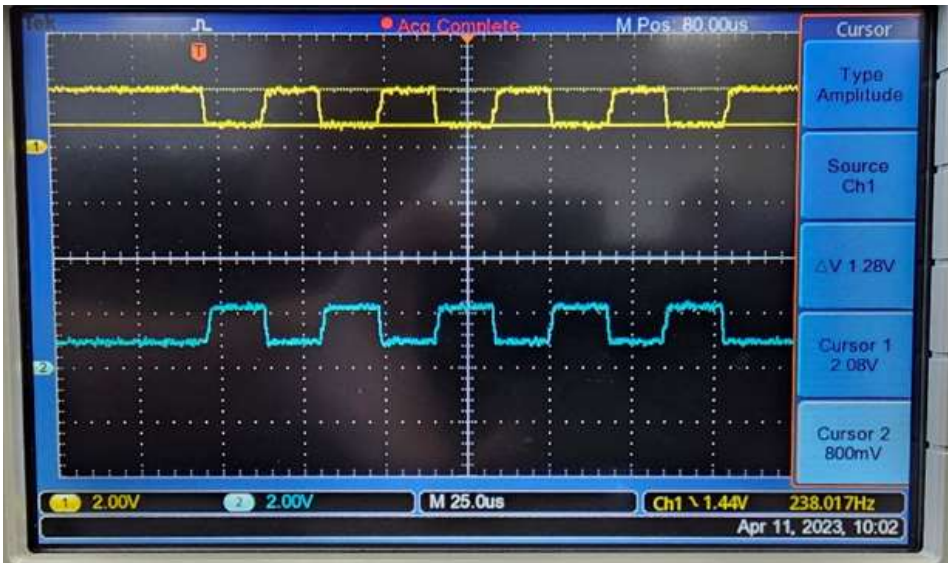


Figure G11. Differential A (yellow) and Differential B (blue).

Figure G12. Differential A (yellow) and Differential B (blue).

University of Nottingham
UK | CHINA | MALAYSIA

Appendix H: BCD to 7-Segment Decoder

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
LIBRARY UNISIM;
USE UNISIM.Vcomponents.ALL;
ENTITY BCDto7Seg_v3_BCDto7Seg_v3_sch_tb IS
END BCDto7Seg_v3_BCDto7Seg_v3_sch_tb;
ARCHITECTURE behavioral OF BCDto7Seg_v3_BCDto7Seg_v3_sch_tb IS

   COMPONENT BCDto7Seg_v3
   PORT( B0 :  IN STD_LOGIC;
         B1   :  IN STD_LOGIC;
         B2   :  IN STD_LOGIC;
         B3   :  IN STD_LOGIC;
         a :  OUT   STD_LOGIC;
         b :  OUT   STD_LOGIC;
         c :  OUT   STD_LOGIC;
         d :  OUT   STD_LOGIC;
         e :  OUT   STD_LOGIC;
         f :  OUT   STD_LOGIC;
         g :  OUT   STD_LOGIC);
   END COMPONENT;

   SIGNAL B0   :  STD_LOGIC;
   SIGNAL B1   :  STD_LOGIC;
   SIGNAL B2   :  STD_LOGIC;
   SIGNAL B3   :  STD_LOGIC;
   SIGNAL a :  STD_LOGIC;
   SIGNAL b :  STD_LOGIC;
   SIGNAL c :  STD_LOGIC;
   SIGNAL d :  STD_LOGIC;
   SIGNAL e :  STD_LOGIC;
   SIGNAL f :  STD_LOGIC;
   SIGNAL g :  STD_LOGIC;


BEGIN

   UUT: BCDto7Seg_v3 PORT MAP(
      B0 => B0,
      B1 => B1,
      B2 => B2,
      B3 => B3,
      a => a,
      b => b,
      c => c,
      d => d,
      e => e,
      f => f,
      g => g
   );


-- *** Test Bench - User Defined Section ***
   tb : PROCESS
   BEGIN

      B0 <= '1'; -- MSB
      B1 <= '0';
      B2 <= '0';
      B3 <= '1'; -- LSB

      WAIT; -- will wait forever
   END PROCESS;
-- *** End Test Bench - User Defined Section ***

END;
```

Figure H1. Code snippet of BCD to 7 segment decoder test bench.

Appendix I: BCD to 7-Segment Decoder Test Bench Results

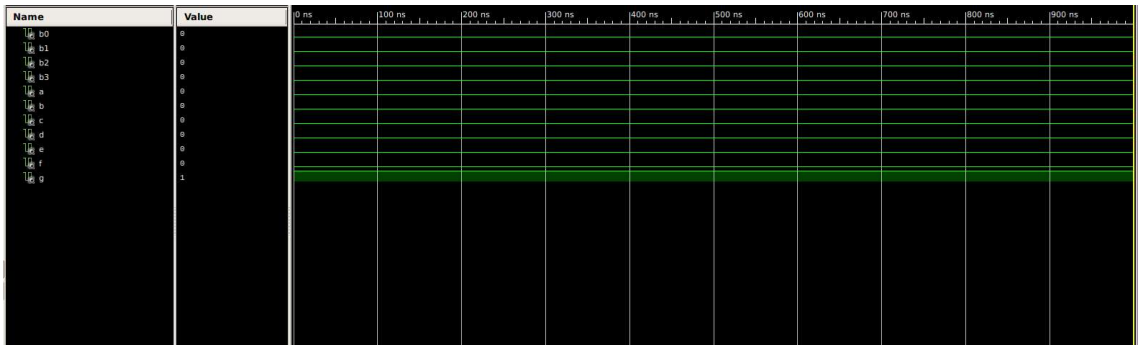


Figure I1. BCD decoder output with input 0000.

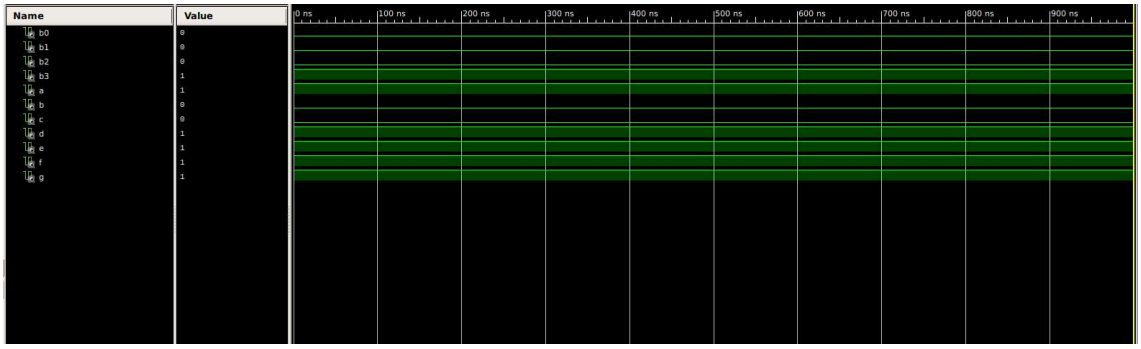


Figure I2. BCD decoder output with input 0001.

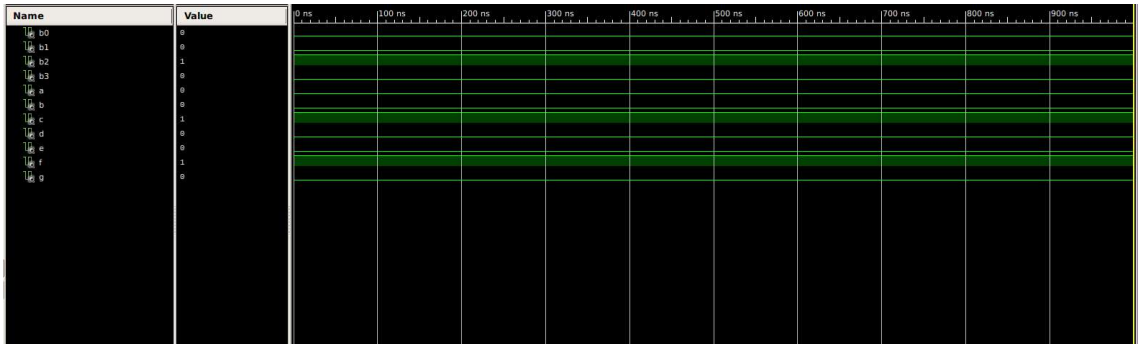


Figure I3. BCD decoder output with input 0010.

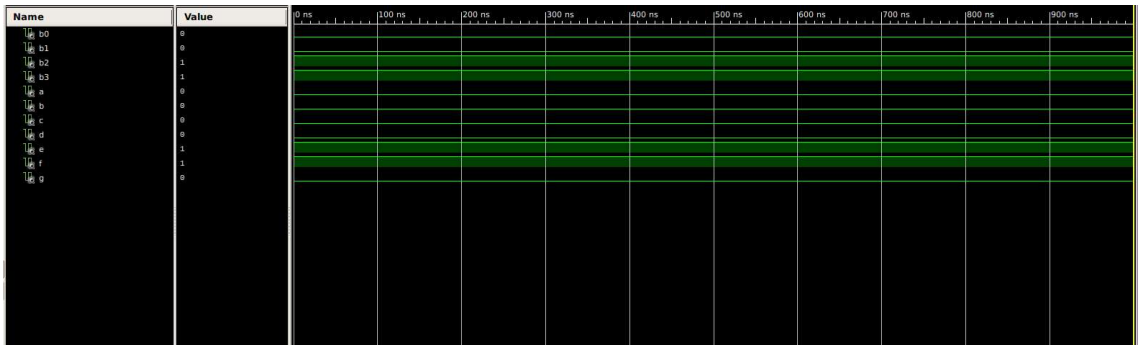


Figure I4. BCD decoder output with input 0011.

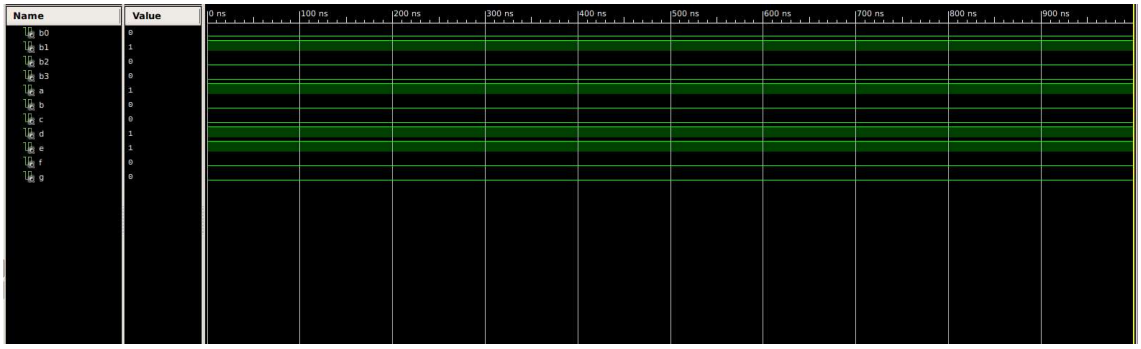


Figure I5. BCD decoder output with input 0100.

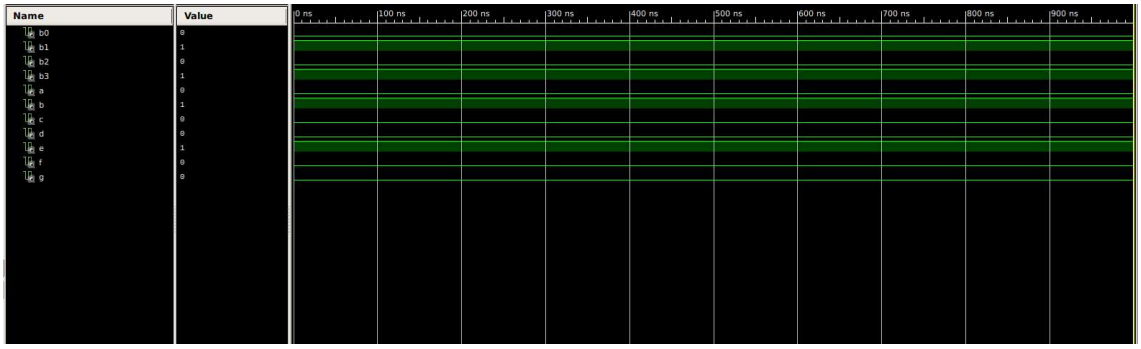


Figure I6. BCD decoder output with input 0101.

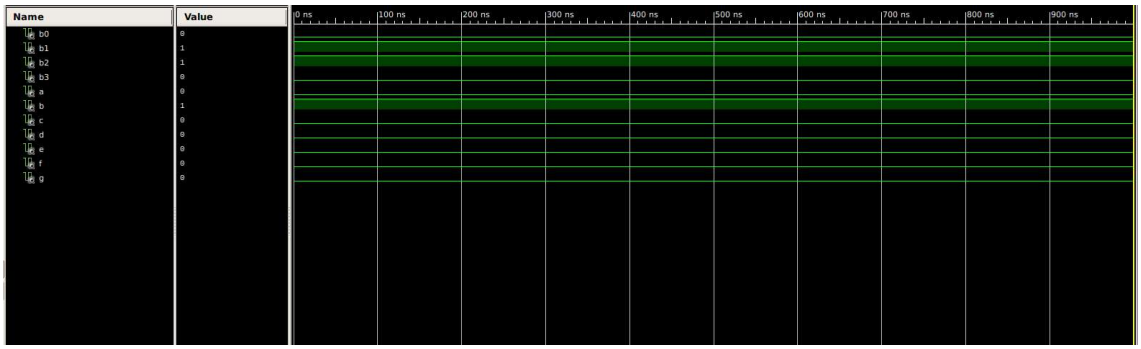


Figure I7. BCD decoder output with input 0110.

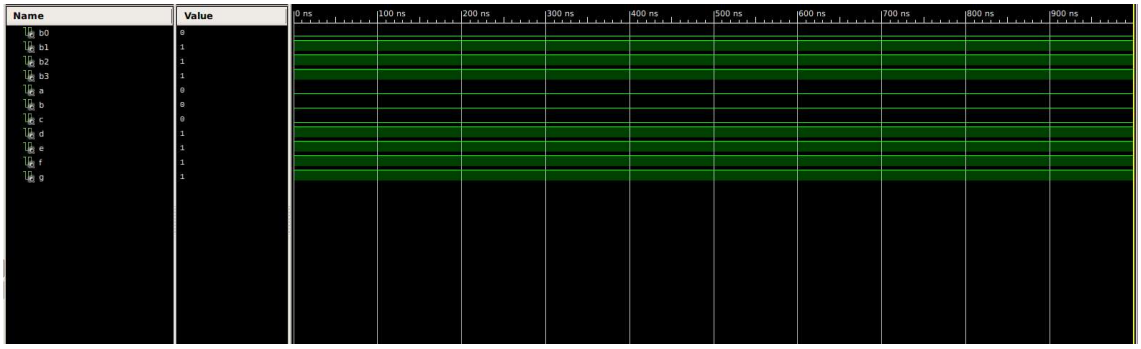


Figure I8. BCD decoder output with input 0111.

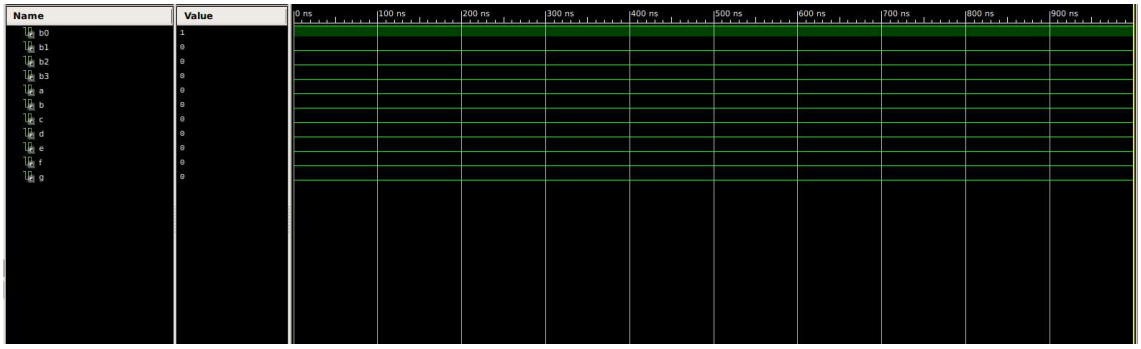


Figure I9. BCD decoder output with input 1000.

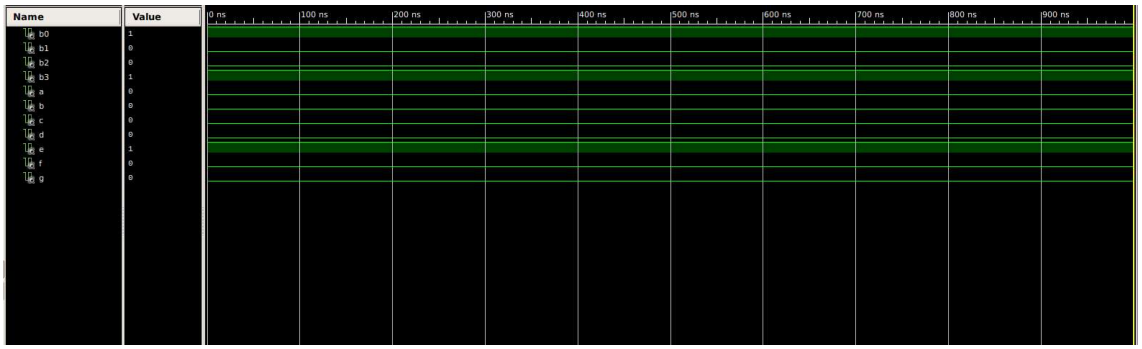


Figure I10. BCD decoder output with input 1001.

University of Nottingham
UK | CHINA | MALAYSIA

Appendix J: Shift Register Test Bench

```
IBRARY ieee;
SE ieee.std_logic_1164.ALL;
SE ieee.numeric_std.ALL;
IBRARY UNISIM;
SE UNISIM.Vcomponents.ALL;
NTITY SR_testing_SR_testing_sch_tb IS
ND SR_testing_SR_testing_sch_tb;
RCHITECTURE behavioral OF SR_testing_SR_testing_sch_tb IS

  COMPONENT SR_testing
  PORT( CE :  IN STD_LOGIC;
        CLK  :  IN STD_LOGIC;
        CLR  :  IN STD_LOGIC;
        Q :  OUT   STD_LOGIC_VECTOR (7 DOWNTO 0);
        RX_Serial  :  IN STD_LOGIC);
  END COMPONENT;

  SIGNAL CE   :  STD_LOGIC;
  SIGNAL CLK  :  STD_LOGIC;
  SIGNAL CLR  :  STD_LOGIC;
  SIGNAL Q :  STD_LOGIC_VECTOR (7 DOWNTO 0);
  SIGNAL RX_Serial  :  STD_LOGIC;

  constant clk_period : time := 17.36 us;

EGIN

  UUT: SR_testing PORT MAP(
     CE => CE,
     CLK => CLK,
     CLR => CLR,
     Q => Q,
     RX_Serial => RX_Serial
  );

  clk_process :process
  begin
     CLK <= '0';
     wait for clk_period/2;
     CLK <= '1';
     wait for clk_period/2;
  end process;

- *** Test Bench - User Defined Section ***
  tb : PROCESS
  BEGIN
     -- reset the SR
     RX_Serial <= '0';
     CLR <= '1';
     CE <= '0';
     wait for 0.5 us;

     -- set the SR ready for taking input
     CLR <= '0';
     CE <= '1';

     -- send 1 logic high bit
     RX_Serial <= '1';
     wait for 17 us;
     RX_Serial <= '0';

     WAIT; -- will wait forever
  END PROCESS;
- *** End Test Bench - User Defined Section ***

ND;
```

Figure J1. Code snippet of shift register test bench.

University of Nottingham
UK | CHINA | MALAYSIA

Appendix K: Control Logic Block Test Bench

```
rx_serial_process : process
begin
   rx_serial <= '0'; --start bit
   wait for rx_period;
   rx_serial <= '1'; -- 1
   wait for rx_period;
   rx_serial <= '0'; -- 2
   wait for rx_period;
   rx_serial <= '1'; -- 3
   wait for rx_period;
   rx_serial <= '0'; -- 4
   wait for rx_period;
   rx_serial <= '1'; -- 5
   wait for rx_period;
   rx_serial <= '0'; -- 6
   wait for rx_period;
   rx_serial <= '1'; -- 7
   wait for rx_period;
   rx_serial <= '0'; -- 8
   wait for rx_period;
   rx_serial <= '1'; -- stop bit
   wait for rx_period;
```

(a)

```
rx_serial <= '0'; --start bit
wait for rx_period/3;
rx_serial <= '1'; --start bit
wait for rx_period/3;
rx_serial <= '0'; --start bit
wait for rx_period/3;
rx_serial <= '1'; -- 1
wait for rx_period;
rx_serial <= '1'; -- 2
wait for rx_period;
rx_serial <= '1'; -- 3
wait for rx_period;
rx_serial <= '1'; -- 4
wait for rx_period;
rx_serial <= '1'; -- 5
wait for rx_period;
rx_serial <= '1'; -- 6
wait for rx_period;
rx_serial <= '1'; -- 7
wait for rx_period;
rx_serial <= '1'; -- 8
wait for rx_period;
rx_serial <= '1'; -- stop bit
wait for rx_period;

end process;
```

(b)

Figure K1. Code snippet of the control logic block test bench.

Appendix L: Working System

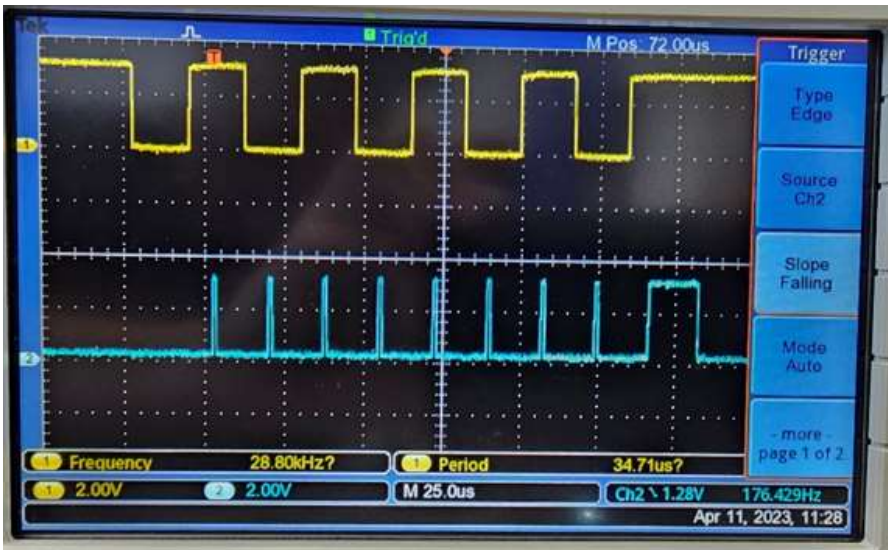


Figure L1. Oscilloscope Rx_serial with shift register CLK (SR_Trig).

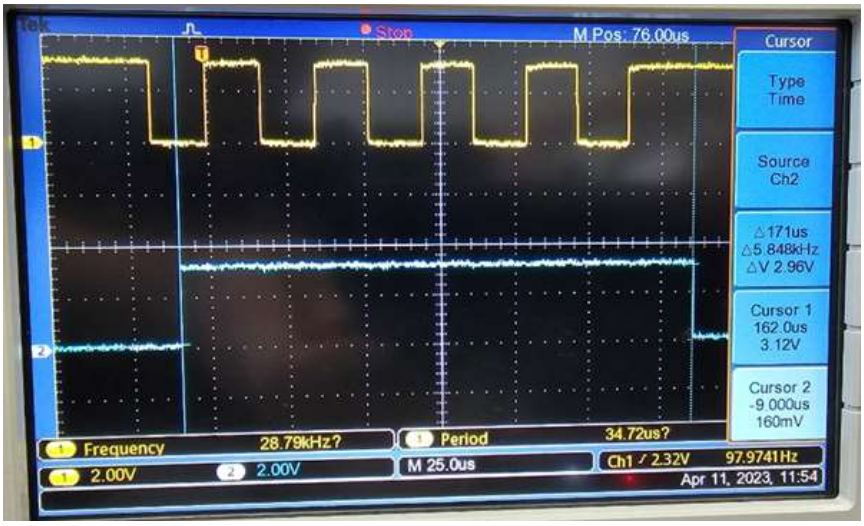


Figure L2. Oscilloscope Rx_serial with start bit detector output.