12/9/2023

CST 455 Term Project

LiDAR Display with HPS

Cameron Robinson
CST 455 – SOC
Cameron Robinson

Table of Contents

Abstract

The following paper discusses the system made for the CST 455 term project. This project is a system that reads distance data from a 2D LiDAR module over UART and displays that data on the DE10-Standard's LCD display. The system also determines the direction of the closest object and displays the general direction on the DE10-Standard's seven-segment displays. The UART receiver, seven-segment decoders, and LiDAR CRC are all implemented through hardware modules created using Verilog HDL. Displaying the LiDAR data on the LCD and determining the direction of the closest object is implemented in software on the DE10-Standard's HPS. The hard-core processor is running a Linux operating system. The LiDAR used in this project is the LD19 2D LiDAR.

Overall, there were four custom hardware modules and two software tasks used in this project. The four hardware modules include a UART receiver, clock divider, CRC module, and seven-segment decoder. All of the modules were wrapped in Avalon slaves and implemented into the project through Platform Designer. One of the software tasks parsed the LiDAR data, checked the data for validity, determined the closest object, and displayed the LiDAR data on the LCD. The other task was a UART receiver task that looked for the beginning of a LiDAR packet and then stored the entire packet of raw data into a buffer and sent it to the other task. The only major problem throughout the project was the inconsistency of the LiDAR data collection since using blocking functions resulted in a lot of data being missed. There are many improvements that could be made to the project, but the most significant one would be to use DMA to collect the LiDAR data. This project turned out to be a very good learning experience for designing a project with both custom hardware and custom software components.

Introduction

The primary goal of this project was to implement both hardware and software components on the DE10-Standard to receive distance data from a 2D LiDAR and accurately display it on the DE10-Standard's LCD. Another goal was to further process the LiDAR data to determine the direction and distance to the closest object and display that direction on the DE10-Standard's seven-segment displays. This project used a hard-core processor running a Linux operating system. The system block diagram can be seen in Figure A3. The task block diagram can be seen in Figure A4. Additionally, the Platform Designer modules and connections are shown in Figure A2.

Design

There are six main components to this project: four hardware modules and two software tasks. The hardware modules include a UART receiver, clock divider, CRC calculator, and seven-segment decoder. The software tasks include a 'Main' task (because it runs in main) and a UART task. The UART receiver and clock divider modules were implemented in hardware because it would be a waste of CPU time to implement the timing for those modules in software. The CRC was implemented in hardware to add slightly more complexity to the project since the LD19 manufacturers already provided the software implementation. Lastly, the seven-segment decoder module would have worked well in either hardware or software, however, it seemed important to leave as much CPU time as possible to display the LiDAR data on the LCD, so the decoding was chosen to be done in hardware. Forming the LiDAR data packets, parsing the LiDAR data, determining where the closest object is, and displaying the LiDAR data was all implemented in software. The reasons for these choices were that they made the most efficient use of time since the code was compatible with other projects, and because the LCD display code required a lot of loops and trigonometric functions which would not have been efficient to implement in hardware.

Physical hardware design

The physical hardware used in this project includes a Cyclone V SoC FPGA and HPS, SDRAM, USB Blaster II, a 50 MHz clock, an LCD, and six common anode seven-segment displays. Most of these hardware components are part of the DE10-Standard board and are illustrated in Figure A1, which is a diagram of the board. Other hardware used was a Type A to Type B USB cable, a 12V wall wart power adapter, a 9V power supply, a Mini-B to Type A USB cable, an Ethernet cable, a Power MB V2 power adapter, six wires, and an LD19 LiDAR. The Power MB V2 and the 9V power supply were used to power the LD19, while the other cables parts were for powering and communicating with the DE10-Standard.

The pinout for the LD19 2D LiDAR is illustrated in Figure A33. As seen in the diagram, the LD19 has four pins, a UART TX, PWM, GND, and P5V. For this project, PWM and GND are connected to a ground pin on the DE10-Standard's 2x20 GPIO pin header. The P5V is connected to the Power MB V2 which is powered by the 9V power supply. The UART TX wire on the LD19 is connected to GPIO[6] on the DE10-Standard's 2x20 GPIO pin header. A pinout assignment table for the GPIO is shown in Table A3. A wiring schematic showing how the LD19 and DE10-Standard were connected can be seen in Figure A5.

The physical setup for the demonstration of this project is shown in Figure A34.

Synthesized hardware design

Design Structure

The custom hardware modules created for this project include a UART receiver, clock divider, CRC, and seven-segment decoder. Only one instantiation of the UART, clock divider, and CRC were needed, while six instantiations of the seven-segment decoder module were needed (one for each display). The only hardware module that directly interacts with another hardware module is the clock divider which outputs a clock to the UART module. The clock divider takes in a 50MHz clock and divides it to a 4.17MHz clock which is about 18x faster than the 230400 baud rate that the LD19 transmitter is running at. All the modules are read and written through software. Since the project uses the Cyclone V HPS, a top-level structural module is used to instantiate some of the system connection. The pinout for the top-level module is shown in Table A5, and the code is shown in both Figure A20 and Figure A21.

Modules

My_UART_RX

The UART receiver hardware module is made up of two submodules, a UART receiver FSM, and a load register. The UART FSM has three inputs and two outputs. The inputs are a clock, reset, and an RX data line (RX_in). The outputs are a 9-bit bus containing the received data with an error bit, and a load signal. There are three states to the FSM: Idle, Shift, and Stop. On restart, the module enters the Idle state where it waits for the RX input line to go low. If the line goes low, the module counts to nine to make sure that there is a valid start bit and to align with the center of the signal (the module is running 18x faster than LD19 UART TX baud rate). If the count makes it to nine, the state is changed to Shift, otherwise the module stays in Idle. In the Shift state, the module counts to seventeen before sampling the RX input line for a data bit. It repeats this process seven more times for a total of eight bits of data before switching to the Stop state. In the Stop state, the module again counts to seventeen before sampling for the UART stop bit. If the stop bit is not detected (i.e., the line is low) then the error bit in the RX output bus is set to one, otherwise it is set to zero. At the end of the Stop state, the load output signal is set high to indicate the data is ready. Afterwards, the module returns to the Idle state. A diagram for the state machine is shown in Figure A10. The code for the UART FSM is shown in Figure A24 and the RTL is shown in Figure A7.

The load register module is the component that stores the received data from the UART FSM. This module has seven inputs and one output. The inputs are a clock, reset, 2-bit address, read, chip select, 9-bit RX data bus, and a load signal. The only output is a 32-bit read data bus. Of the 32-bit output, only ten bits are used; the first eight bits are for the UART data, the ninth bit indicates whether the data is valid, and the tenth bit is an error bit. The UART data bits and the error bit are sent from the UART FSM and are only read into the load register when the load input goes high. The data valid bit is also set when load goes high and is only set low when the module is read by software. This means that the valid bit indicates both that data is ready, and that the data has not yet been read. Data is read into software when chip select and read are both high, and the address bus is 2'b00. The code for the load register is shown in Figure A25 and the RTL is shown in Figure A9.

The top-level structural code for My_UART_RX is shown in Figure A22 and the top-level RTL is shown in Figure A6.

My_ClockDivider

The clock divider module has six inputs and one output. The inputs are a clock, reset, chip select, 2-bit address bus, write, and a 32-bit write data bus. The only output is a clock output. In general, all this module does is divide the input clock frequency specified by the value in an internal register called 'div'. Every rising clock edge, an internal count register is incremented; if this register reaches the value of div - 1, then the clock output signal is toggled. If chip select and write are high, and the address bus is set to 2'b00, then the data on the write data bus is stored in div and is used as the new value to divide the input clock. The reset signal resets the counter register to zero, but otherwise does nothing. The code for this module is shown in Figure A23 and the RTL is shown in Figure A11.

My_CRC

The CRC module is made specifically for checking the LD19 LiDAR data. The module contains a lookup table with all of the CRC8 values. This module has seven inputs and one output. The inputs are a clock, reset, 2-bit address bus, read, write, chip select, and 32-bit write data bus. The only output is a 32-bit read data bus. Data is loaded into this module in a series of twelve writes to the module for a total of 47 bytes (4 bytes x 11 + 3 bytes x 1). Writes are performed by setting write and chip select high, and setting the address to 2'b00. While writes are being performed, an internal status register is set to zero. Once all twelve writes are done, the status register is set to one and the module begins calculating the CRC one byte at a time. After calculating the CRC, the status register is set to two to indicate that the CRC is ready to be read. The CRC can be read by setting chip select and read high, and setting the address to 2'b00. A read causes the CRC and status register to be stored in the read data bus, with the first eight bits being the CRC, and bits nine and ten being the status register. This data can be read at any point; however, it is only valid when the status register equals 2'b10. Dropping the reset input low at any point results in all of the internal registers being set to zero. The RTL for this module is shown in Figure A12 and the code is shown in Figure A27.

SevenSegDecoder

The seven-segment decoder module decodes decimal values ranging from 0 – 35. Any 6-bit value outside of this range results in an output of 0x3F to turn off the connected display. This module has six inputs and one output. The inputs are a clock, reset, 2-bit address bus, write, chip select, and a 32-bit write data bus. The only output is a 7-bit bus called 'segs' which is intended to be connected to a seven-segment display. The module has a 6-bit register, data, which holds the current data written to the module. The data register is decoded to its seven-segment display equivalent value which is then assigned to the 'segs' output bus. When the reset signal is low, the data register is set to 0x3F which results in the connected display turning off. If chip select and write are high, and the address bits are zero, then the lowest six bits of the write data input bus are assigned to the data register. The RTL for this module is shown in Figure A8. The code for this module is shown in Figure A26.

Software design

Design Structure

The software in this project handles receiving LiDAR data from the UART, checking the data for validity, displaying the data on the DE10-Standard LCD, determining where the closest object is, and displaying the direction of the closest object. These functions are split between two tasks, the 'UART RX' task and

'Main' task. These tasks communicate using a shared flag variable and shared data buffer. A mutex lock is used to prevent race conditions. A task diagram is shown in Figure A4.

Tasks

Main Task

The 'Main' task runs within main and handles all of the software tasks aside from receiving the LD19 distance data. This task first waits for the data packet from the UART task to be ready before doing anything else. Once the data is ready in the shared buffer, the task copies the data into a separate array and then sends the data to the CRC hardware module to check if there are any errors in the data. If the CRC module returns a value equal to the CRC in the data packet, then the data is valid, otherwise, the data is assumed to be invalid, and the task goes back to waiting for another data packet. If the data is valid, then the task parses the packet into usable data consisting of a start angle, end angle, and twelve distance measurements that were sampled between the two angles. Next, the task loops through the distance measurements and determines if any of them are smaller than the current smallest measurement, or if the distance at which the current smallest measurement was taken has changed. If there is a new smallest distance, then the new distance and the direction of the measurement is recorded and the general direction is sent to the seven-segment decoders. The possible displayed directions are: Forward, Behind, Left, and Right. If there is a measurement in the same direction as the current smallest distance, then the smallest distance is updated to the new value. The final function the 'Main' task performs is displaying the distance data on the DE10-Standard's LCD. The function to display the data only displays objects up to a meter away from the LD19. For each distance measurement, the display function first clears all other pixels in the same direction as the new measurement, and then draws a pixel at the newest measured distance. After displaying all of the new data, the task goes back to waiting for a new data packet before repeating the process again. The code for this task is shown in Figure A28. The seven-segment display function is shown in Figure A31, the LCD display function is shown in Figure A30, and the LD19 data functions are shown in Figure A32.

UART RX Task

The 'UART RX' task's only job is to store a complete packet of LiDAR data that it reads from the UART RX hardware module. The task needs to look for two identifiers that signal the start of a data packet: a header which always has the value 0x54 and a packet version that always has the value 0x2C. Currently, the code also checks that the fourth byte read is equal to 0x0e, which is the MSB for the rotation speed of the LiDAR and is basically a constant. This step is not necessary; however, it should filter out a few more bad packets. The task will continue to loop until all of these values are found. Once they are, the task then loops through and reads the rest of the bytes in the packet without regard for their values. After reading all 47 bytes in the packet, the task stores them in the shared buffer and sets a shared flag variable to indicate that the data is ready, before repeating the entire process again. The code for this task is shown in Figure A29.

Simulation and Testing

Hardware Module Testing

Questa was used to test and simulate all of the hardware modules. Most of the tests were similar in nature: input values with a known output and verify that the module simulation resulted in the expected values.

To test the UART module, the values for the LiDAR header and packet version (0x54 and 0x2C respectively) were input on the RX input line of the module. In order to verify that the UART was working correctly, it was necessary to verify that the state machine transitioned between all of the corrects states, and that the correct data was in the output buffer when the load signal went high. The results of the simulation are illustrated in Figure A13. To test the load register module the same values, 0x54 and 0x2C, were input on the module's data input bus, loaded into the internal register, and then read by setting read and chip select high. To verify that the module was working, it was necessary to check that the correct value was loaded into the register and that the valid data flag (RD_FLAG in the simulation) was high before reading the data, and low after reading the data. The results of this simulation are shown in Figure A14.

To test the CRC, a LiDAR data packet with a valid CRC value was loaded into the CRC module's data register. Since the CRC module works like a shift register, it was important to check throughout the loading and CRC calculation process, that the data register was having data shifted in and out respectively. The test turned out to be successful with the CRC output matching the CRC from the input packet. The simulation results are shown in Figure A15.

Testing the clock divider was a simple two-step process; first check that the output clock level stays high or low for the number of positive input clock edges specified in the 'div' register; and then check that a new divisor can be written to the module and that the output clock frequency will adjust to match it. The simulation results turned out well; the input clock was initially divided by six and then set to be divided by two; in both cases the output clock's frequency was adjusted correctly. The results of the simulation can be seen in Figure A17.

Testing the seven-seg decoder was also simple since every case could be easily tested by writing 37 different input values to the module. Input values ranging from 0 – 35 covered all of the values that could be decoded by the module, and then an extra value outside of that range was input to make sure that 0x7F was output. To verify that the module was working correctly it was necessary to check the output of each input value to make sure that the correct segment values were set. The results of this simulation are shown in Figure A16, however, it is important to note that some letters of the alphabet cannot be displayed on seven-segment displays (X, W, M, etc.); these values were set to 0x7F in the module.

Software Testing

The process for testing the software modules was similar to the hardware modules: inputs with known output values were sent into functions and then the outputs were checked to make sure they were correct. To test the LCD display function, a set of test LiDAR data arrays were sent to the function. To verify the steps within the function, printf() was used to print values to a serial console. To verify that the overall function worked, it was necessary to check that the LCD displayed the distance data correctly. For example, two arrays sent to the display function were circles of different sizes; one with a radius of one meter, and then another with a radius of 0.5 meters. The expected outcome was that the larger circle would touch the top and bottom of the LCD while the smaller circle would be about halfway to the top and bottom since the LCD only displayed range data up to a meter. When the smaller circle was drawn, it was also important to verify that the larger circle was erased since when a new pixel is drawn, all of the pixel in the same direction should be cleared. Another useful testing method for the display task was using a calculator and manually going through the calculation process of the

coordinates for a pixel to make sure that there were no logic errors. An example of a test data packet displayed on the LCD is shown in Figure A19.

Testing the CRC and data packet collection was straightforward. When a packet was received, it was printed to the console. The packet was also sent through both a software CRC and the hardware CRC module to check if the packet was valid and if the calculations for the hardware and software CRCs were the same. If the CRCs did not match each other, or if the packet contained incorrect data (i.e., missing a header), then it was obvious that there was an error. Finding the error was slightly more challenging since it required stepping through the code and checking the values of the variables at each step.

Testing the closest object and direction calculations was the similar to the LCD display function except the testing was done with actual LiDAR data rather than test packets. While the program was running, an object would be brought close to the LiDAR and three different outputs were checked. The calculated direction and distance measurements were sent to the serial console, and they were checked to make sure they were reasonable (i.e., non-zero distance, and a direction value between 0 and 360). The other value checked was the direction displayed on the seven-segment displays which should match the general direction of the object (front, behind, left, right).

Data from the tests where data was printed to the console is shown in Figure A18. This image shows the CRC data, direction data, closest object data, and the LCD display data.

Problems

Throughout this project, the only major issue was the collection of the LiDAR data. Since the LD19 uses UART and only has a transmit line, there is no good way to synchronize the UART receiver with the transmitter without turning the LD19 off and on again. The LD19 constantly sends thousands of bytes a second, and if the UART software task ever misses a header, then the entire rest of the packet is thrown out. This means that not only does the UART software task miss a lot of data, but it also spends a lot of time looking for the start of a packet. This issue caused the direction and distance data on the displays to update very slowly, especially when packets for a specific direction were missed multiple times in a row. The best solution to this problem would have been to use DMA to read the UART hardware module instead of using blocking functions. Unfortunately, with the time constraints of this project, this method was not practical to implement. Another issue with synchronization that occurred was that as time went on, the rate of invalid packets would increase until little to no valid packets were ever found. The only solution found for this issue was to restart the LD19.

Results and Conclusion

Overall, the project turned out to be successful, at least as proof of concept. Although it updated slowly, the LD19 data could be accurately displayed on the LCD and the direction of the closest object could be displayed on the seven-segment displays. This project was a valuable experience for learning about hardware and software codesign. One change that would be made next time is to use DMA to receive the LiDAR data instead of blocking functions since it would result in much less data missed and a lot less CPU time spent looking for valid data.

Appendix – A

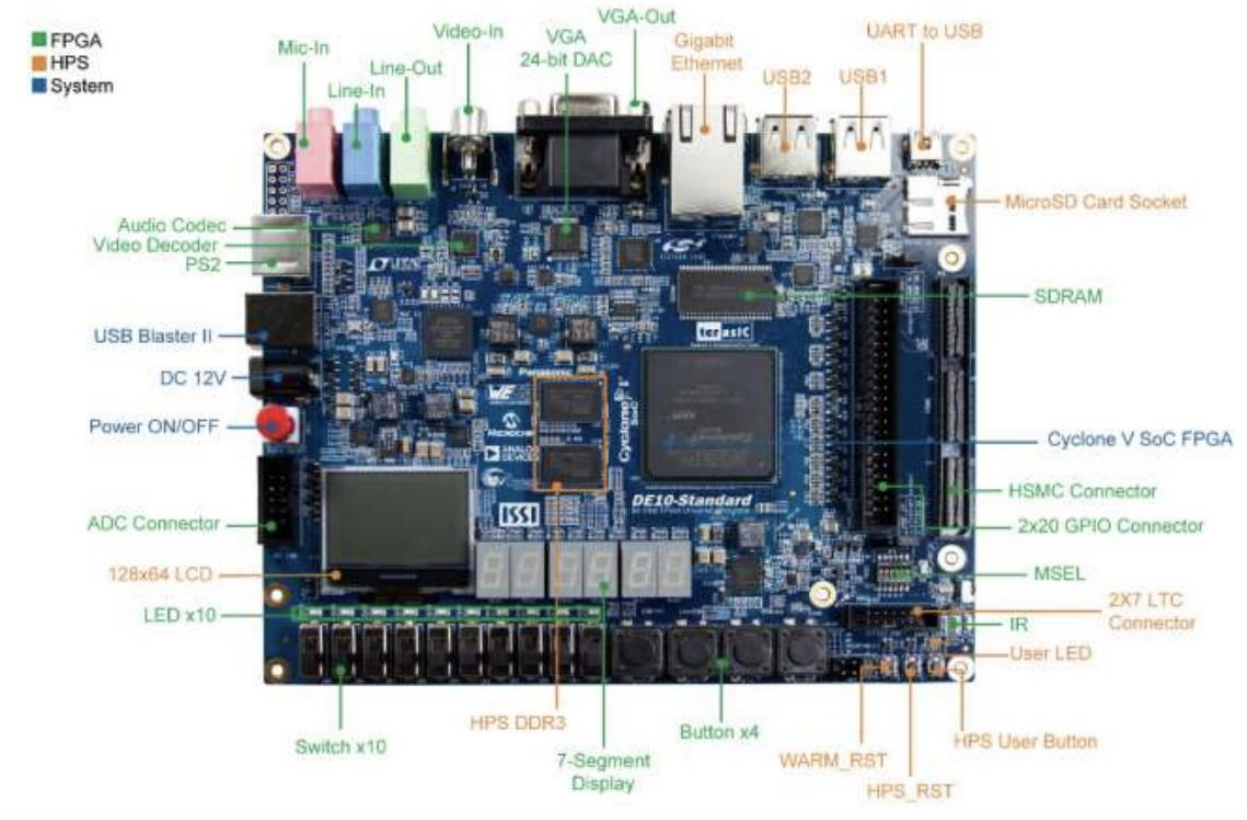


Figure A1: DE10-Standard Diagram

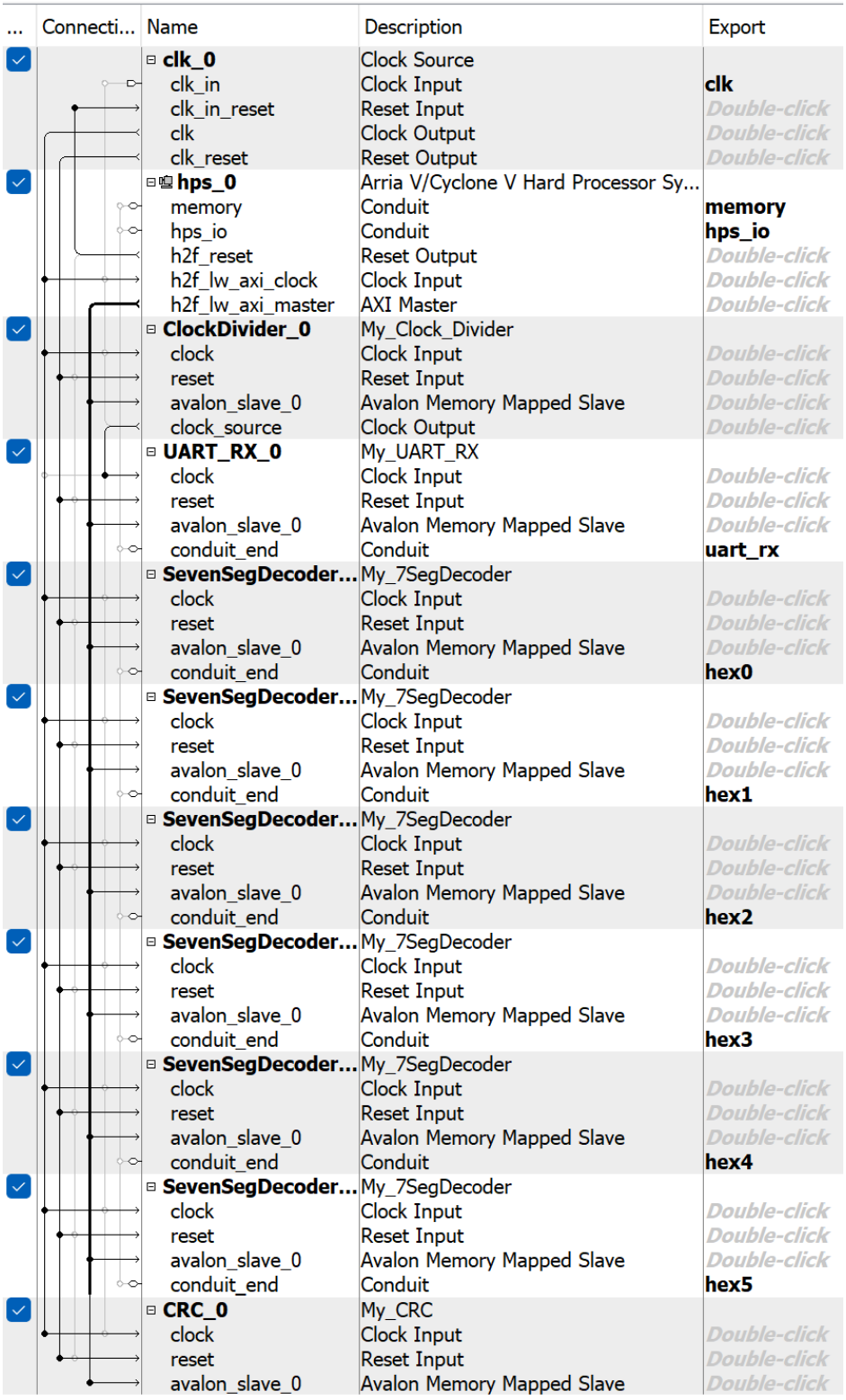


Figure A2: Platform Designer Components

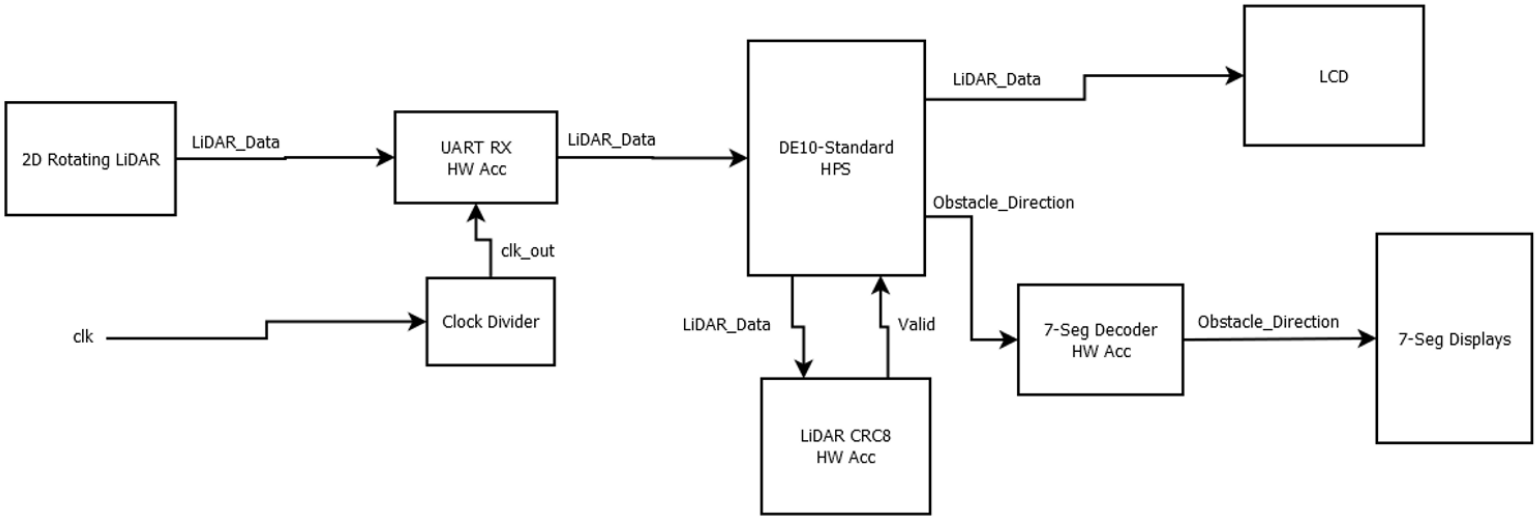


Figure A3: System Block Diagram

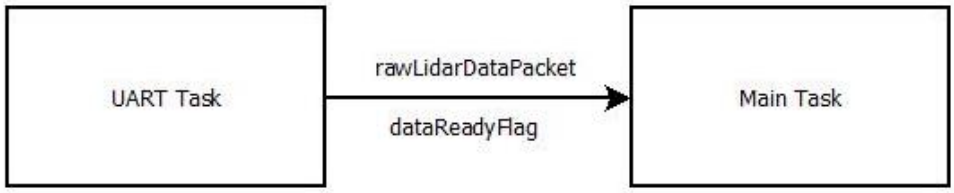


Figure A4: Task Diagram

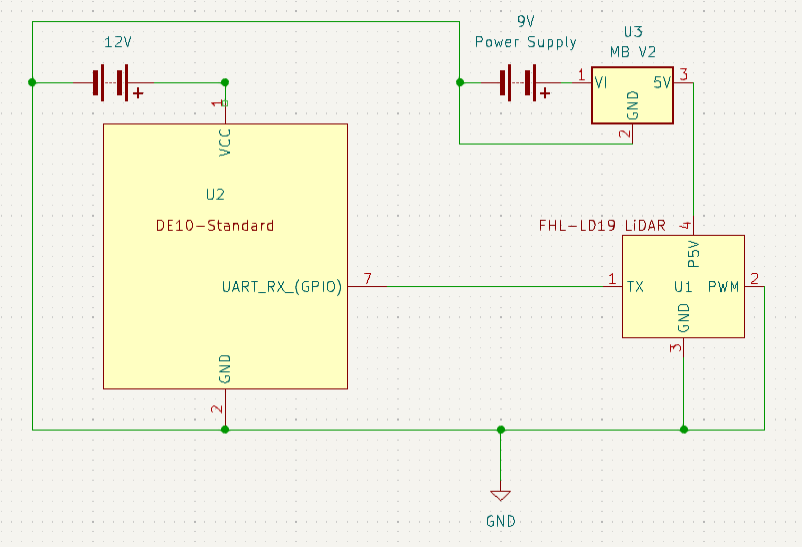


Figure A5: Wiring Schematic

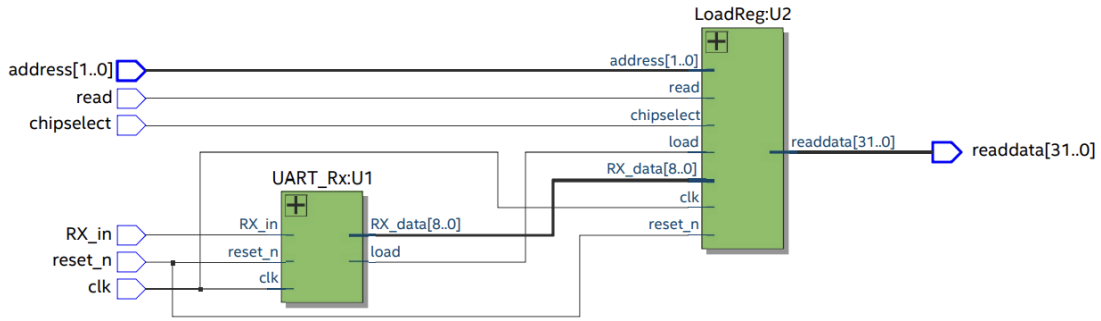


Figure A6: UART RX Top-Level RTL

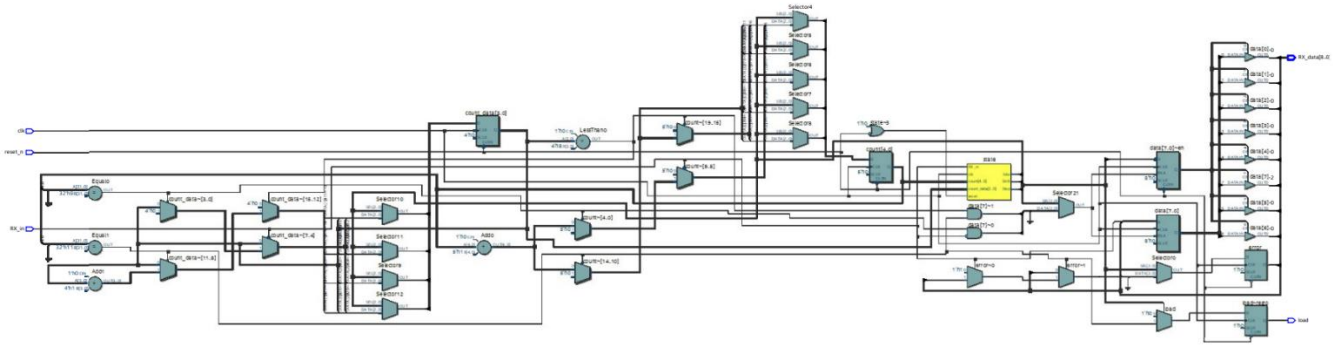


Figure A7: UART RX FSM RTL

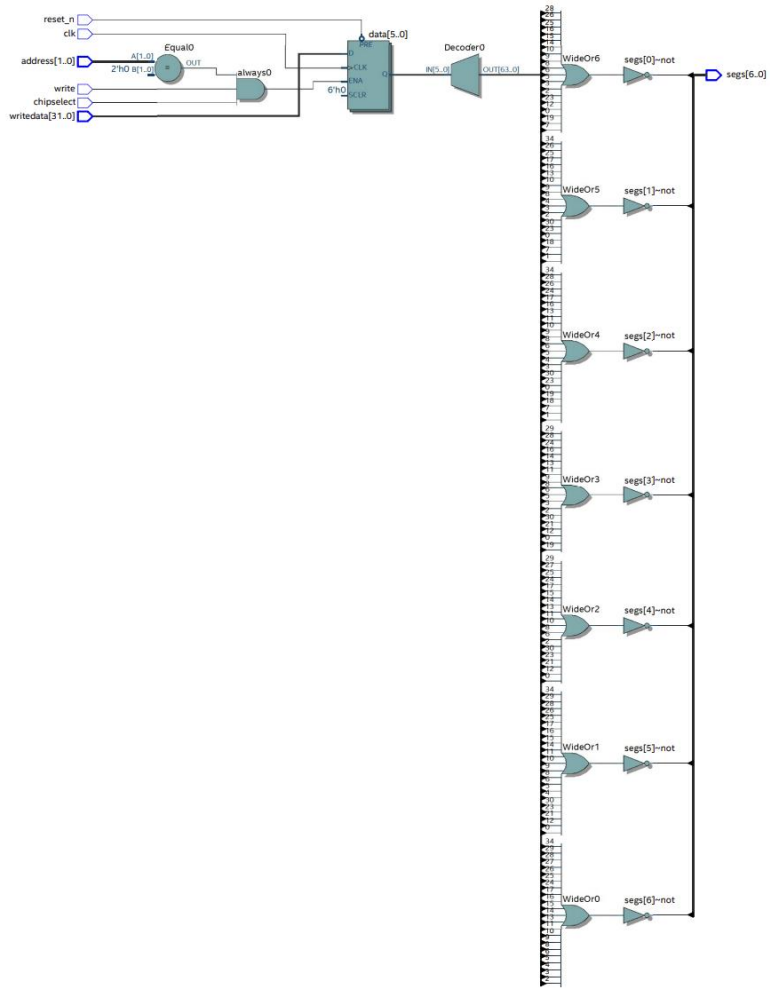


Figure A8: SevenSegDecoder RTL

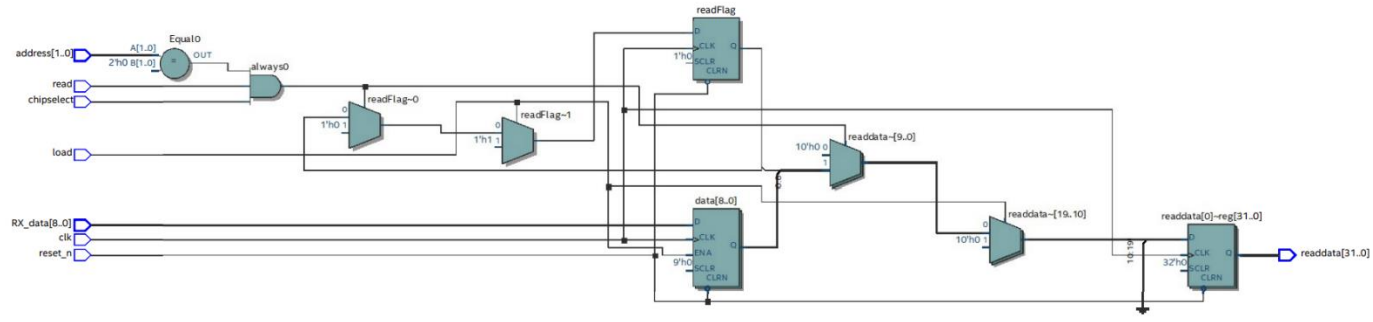


Figure A9: UART Load Register RTL

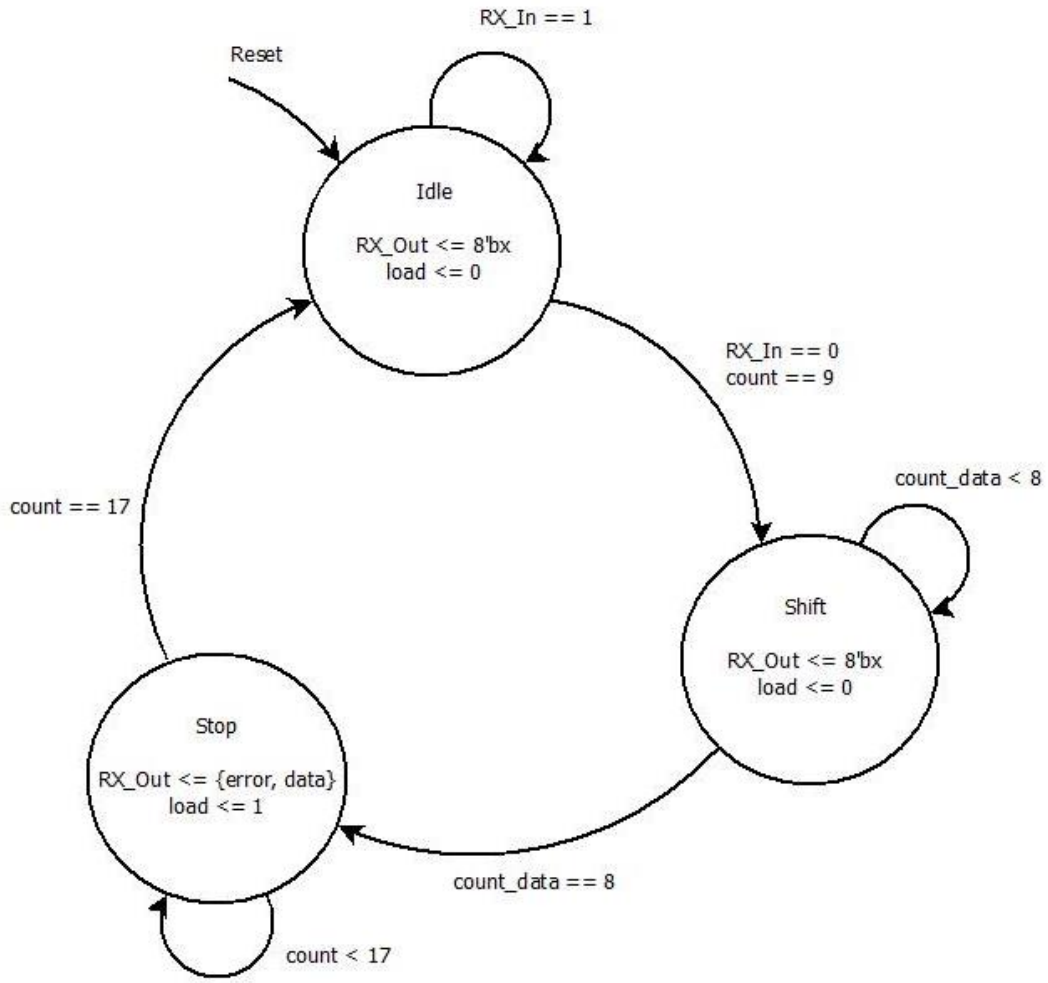


Figure A10: UART FSM State Diagram

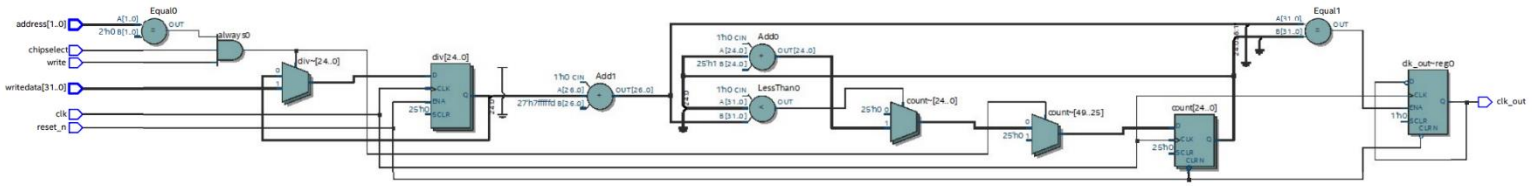


Figure A11: Clock Divider RTL

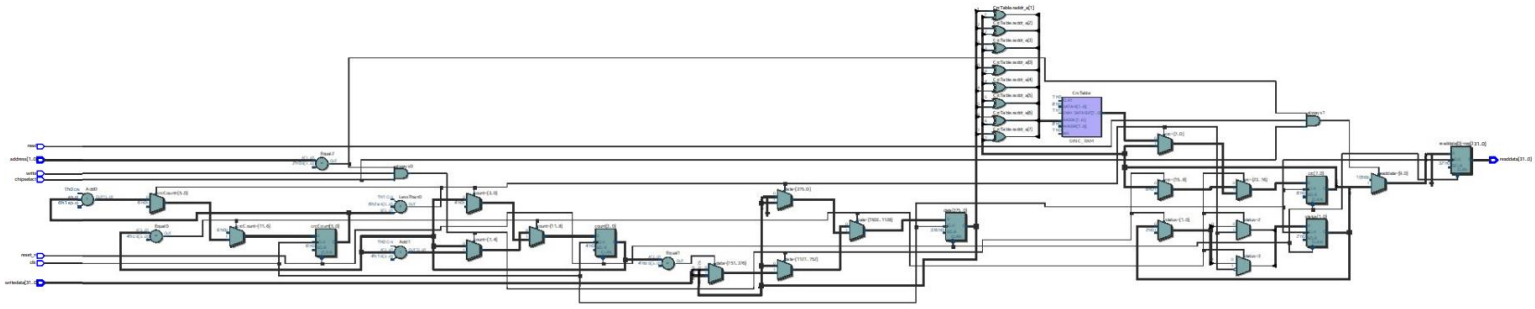


Figure A12: CRC RTL

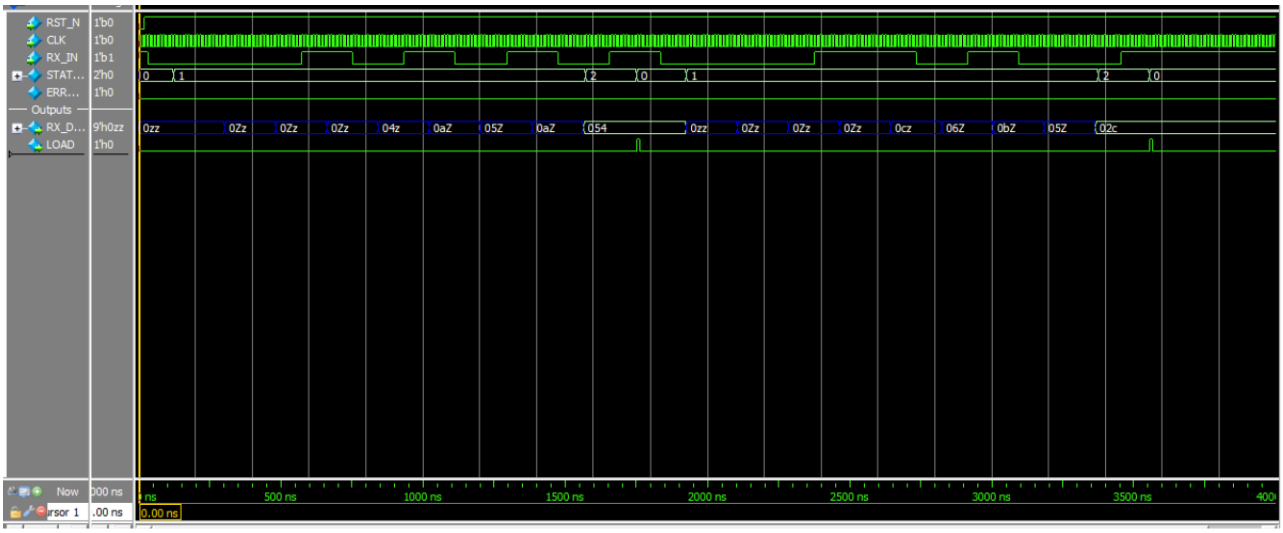


Figure A13: UART_RX Questa Simulation

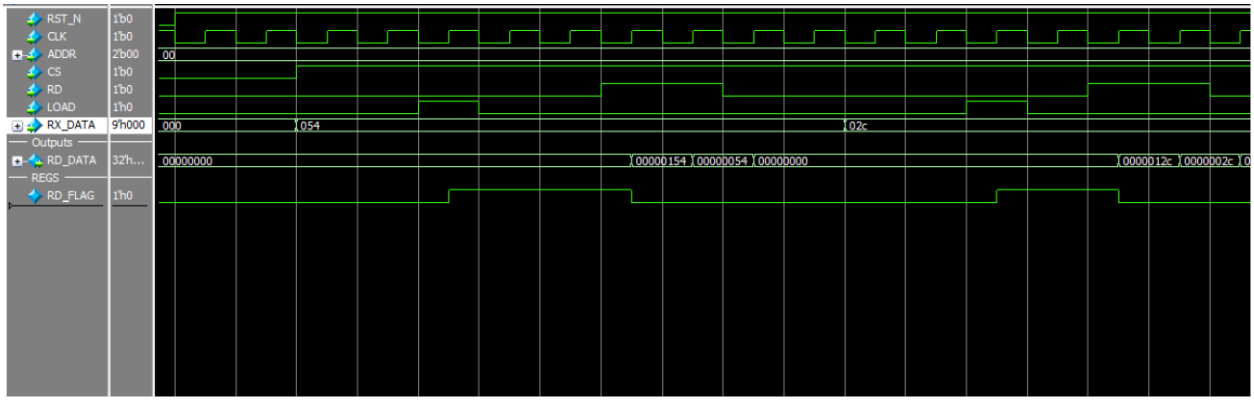


Figure A14: LoadReg Module Questa Simulation

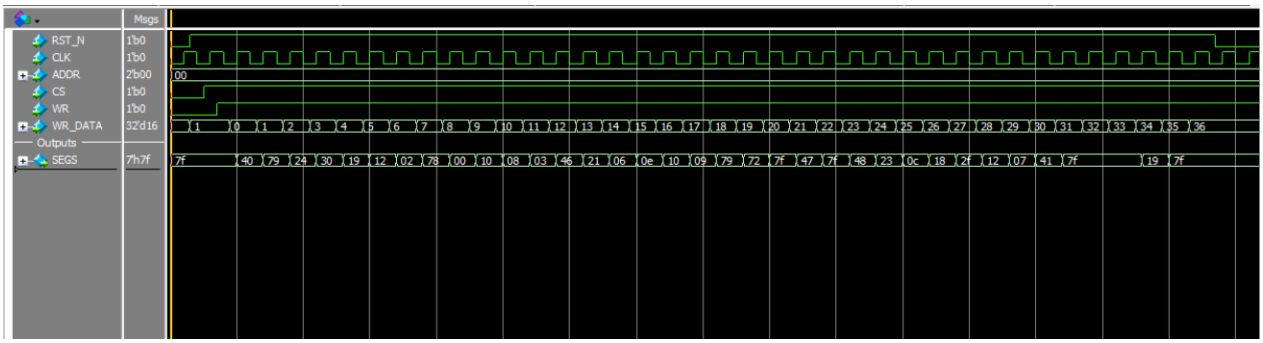


Figure A16: SevenSegDecoder Questa Simulation

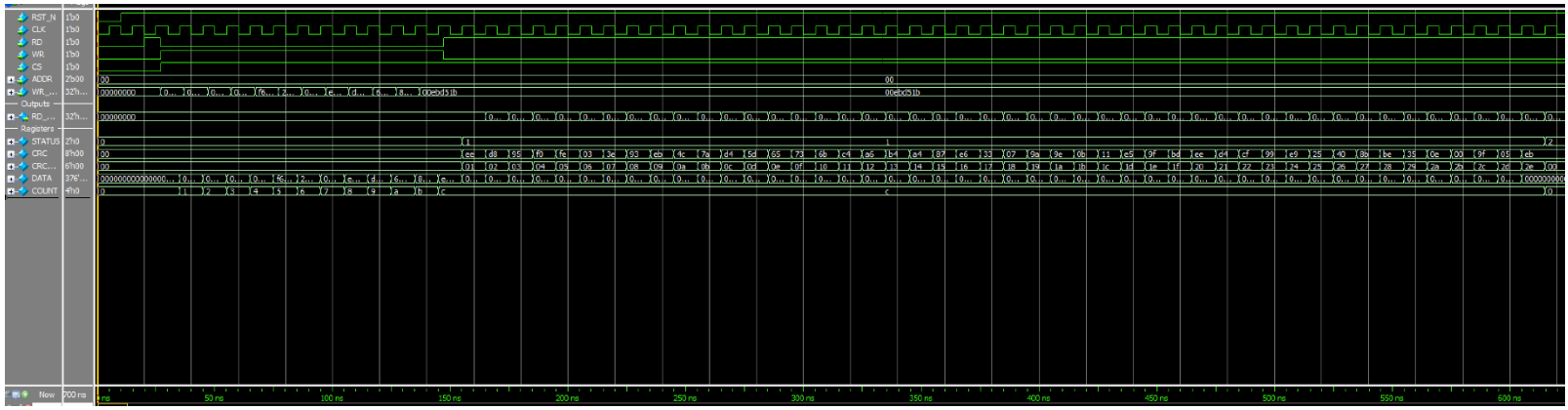


Figure A15: My_CRC Questa Simulation

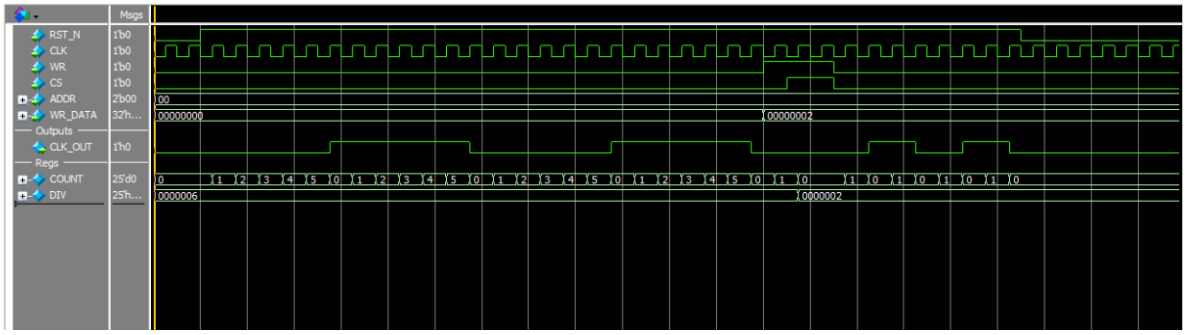


Figure A17: My_ClockDivider Questa Simulation

```
Data Valid
Smallest: 273, Dir: 66.350
StartAngle: 5768, EndAngle: 6635
X: 72, Y: 37, Begin: 1.075, End: 1.158, Delta: 0.014
X: 72, Y: 36, Begin: 1.089, End: 1.158, Delta: 0.014
X: 73, Y: 36, Begin: 1.103, End: 1.158, Delta: 0.014
X: 73, Y: 36, Begin: 1.117, End: 1.158, Delta: 0.014
X: 73, Y: 36, Begin: 1.131, End: 1.158, Delta: 0.014
X: 74, Y: 36, Begin: 1.144, End: 1.158, Delta: 0.014
X: 75, Y: 37, Begin: 1.158, End: 1.158, Delta: 0.014
Packet CRC: 240
Software CRC: 209
Hardware CRC: 209
HW CRC Correct
Data Invalid
Packet CRC: 44
Software CRC: 4
Hardware CRC: 4
HW CRC Correct
Data Invalid
Packet CRC: 239
Software CRC: 36
Hardware CRC: 36
HW CRC Correct
Data Invalid
Packet CRC: 68
Software CRC: 218
Hardware CRC: 218
HW CRC Correct
Data Invalid
Packet CRC: 5
Software CRC: 250
Hardware CRC: 250
HW CRC Correct
Data Invalid
Packet CRC: 242
Software CRC: 107
Hardware CRC: 107
HW CRC Correct
```

Figure A18: Console Output of Task Test

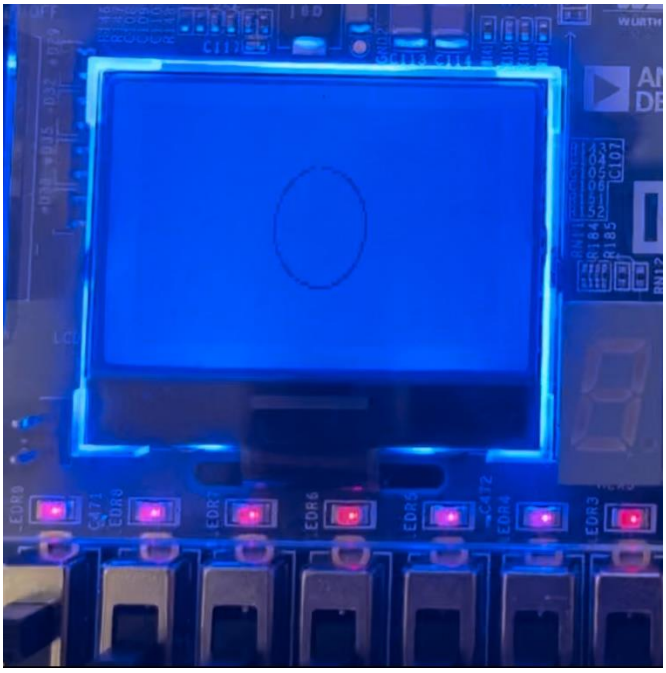


Figure A19: LCD Display Test Example

```
module Term_Project_HPS(

    ////////// CLOCK //////////
    input                       CLOCK_50,

    ////////// Seg7 //////////
    output            [6:0]     HEX0,
    output            [6:0]     HEX1,
    output            [6:0]     HEX2,
    output            [6:0]     HEX3,
    output            [6:0]     HEX4,
    output            [6:0]     HEX5,

    ////////// HPS //////////
    inout                       HPS_CONV_USB_N,
    output          [14:0]      HPS_DDR3_ADDR,
    output           [2:0]      HPS_DDR3_BA,
    output                      HPS_DDR3_CAS_N,
    output                      HPS_DDR3_CKE,
    output                      HPS_DDR3_CK_N,
    output                      HPS_DDR3_CK_P,
    output                      HPS_DDR3_CS_N,
    output           [3:0]      HPS_DDR3_DM,
    inout           [31:0]      HPS_DDR3_DQ,
    inout            [3:0]      HPS_DDR3_DQS_N,
    inout            [3:0]      HPS_DDR3_DQS_P,
    output                      HPS_DDR3_ODT,
    output                      HPS_DDR3_RAS_N,
    output                      HPS_DDR3_RESET_N,
    input                       HPS_DDR3_RZQ,
    output                      HPS_DDR3_WE_N,
    output                      HPS_ENET_GTX_CLK,
    inout                       HPS_ENET_INT_N,
    output                      HPS_ENET_MDC,
    inout                       HPS_ENET_MDIO,
    input                       HPS_ENET_RX_CLK,
    input             [3:0]     HPS_ENET_RX_DATA,
    input                       HPS_ENET_RX_DV,
    output            [3:0]     HPS_ENET_TX_DATA,
    output                      HPS_ENET_TX_EN,
    inout             [3:0]     HPS_FLASH_DATA,
    output                      HPS_FLASH_DCLK,
    output                      HPS_FLASH_NCSO,
    inout                       HPS_GSENSOR_INT,
    inout                       HPS_I2C1_SCLK,
    inout                       HPS_I2C1_SDAT,
    inout                       HPS_I2C2_SCLK,
    inout                       HPS_I2C2_SDAT,
    inout                       HPS_I2C_CONTROL,
    inout                       HPS_KEY,
    inout                       HPS_LCM_BK,
    inout                       HPS_LCM_D_C,
    inout                       HPS_LCM_RST_N,
    output                      HPS_LCM_SPIM_CLK,
    input                       HPS_LCM_SPIM_MISO,
    output                      HPS_LCM_SPIM_MOSI,
    output                      HPS_LCM_SPIM_SS,
    inout                       HPS_LED,
    inout                       HPS_LTC_GPIO,
    output                      HPS_SD_CLK,
    inout                       HPS_SD_CMD,
    inout             [3:0]     HPS_SD_DATA,
    output                      HPS_SPIM_CLK,
    input                       HPS_SPIM_MISO,
    output                      HPS_SPIM_MOSI,
    output                      HPS_SPIM_SS,
    input                       HPS_UART_RX,
    output                      HPS_UART_TX,
    input                       HPS_USB_CLKOUT,
    inout             [7:0]     HPS_USB_DATA,
    input                       HPS_USB_DIR,
    input                       HPS_USB_NXT,
    output                      HPS_USB_STP,

    ////////// GPIO //////////
    input                       GPIO_Pin7

);
```

Figure A20: Top-Level Module Code Part 1

```
//=======================================================
//  Structural coding
//=======================================================
    TermProject u0 (
        .clk_clk                       (CLOCK_50),                  //          clk.clk
        .hex0_export                   (HEX0),                      //         hex0.export
        .hex1_export                   (HEX1),                      //         hex1.export
        .hex2_export                   (HEX2),                      //         hex2.export
        .hex3_export                   (HEX3),                      //         hex3.export
        .hex4_export                   (HEX4),                      //         hex4.export
        .hex5_export                   (HEX5),                      //         hex5.export
        .hps_io_hps_io_emac1_inst_TX_CLK (HPS_ENET_GTX_CLK),        //         hps_io.hps_io_emac1_inst_TX_CLK
        .hps_io_hps_io_emac1_inst_TXD0   (HPS_ENET_TX_DATA[0]),     //                .hps_io_emac1_inst_TXD0
        .hps_io_hps_io_emac1_inst_TXD1   (HPS_ENET_TX_DATA[1]),     //                .hps_io_emac1_inst_TXD1
        .hps_io_hps_io_emac1_inst_TXD2   (HPS_ENET_TX_DATA[2]),     //                .hps_io_emac1_inst_TXD2
        .hps_io_hps_io_emac1_inst_TXD3   (HPS_ENET_TX_DATA[3]),     //                .hps_io_emac1_inst_TXD3
        .hps_io_hps_io_emac1_inst_RXD0   (HPS_ENET_RX_DATA[0]),     //                .hps_io_emac1_inst_RXD0
        .hps_io_hps_io_emac1_inst_MDIO   (HPS_ENET_MDIO),           //                .hps_io_emac1_inst_MDIO
        .hps_io_hps_io_emac1_inst_MDC    (HPS_ENET_MDC),            //                .hps_io_emac1_inst_MDC
        .hps_io_hps_io_emac1_inst_RX_CTL (HPS_ENET_RX_DV),          //                .hps_io_emac1_inst_RX_CTL
        .hps_io_hps_io_emac1_inst_TX_CTL (HPS_ENET_TX_EN),          //                .hps_io_emac1_inst_TX_CTL
        .hps_io_hps_io_emac1_inst_RX_CLK (HPS_ENET_RX_CLK),         //                .hps_io_emac1_inst_RX_CLK
        .hps_io_hps_io_emac1_inst_RXD1   (HPS_ENET_RX_DATA[1]),     //                .hps_io_emac1_inst_RXD1
        .hps_io_hps_io_emac1_inst_RXD2   (HPS_ENET_RX_DATA[2]),     //                .hps_io_emac1_inst_RXD2
        .hps_io_hps_io_emac1_inst_RXD3   (HPS_ENET_RX_DATA[3]),     //                .hps_io_emac1_inst_RXD3
        .hps_io_hps_io_sdio_inst_CMD     (HPS_SD_CMD),              //                .hps_io_sdio_inst_CMD
        .hps_io_hps_io_sdio_inst_D0      (HPS_SD_DATA[0]),          //                .hps_io_sdio_inst_D0
        .hps_io_hps_io_sdio_inst_D1      (HPS_SD_DATA[1]),          //                .hps_io_sdio_inst_D1
        .hps_io_hps_io_sdio_inst_CLK     (HPS_SD_CLK),              //                .hps_io_sdio_inst_CLK
        .hps_io_hps_io_sdio_inst_D2      (HPS_SD_DATA[2]),          //                .hps_io_sdio_inst_D2
        .hps_io_hps_io_sdio_inst_D3      (HPS_SD_DATA[3]),          //                .hps_io_sdio_inst_D3
        .hps_io_hps_io_usb1_inst_D0      (HPS_USB_DATA[0]),         //                .hps_io_usb1_inst_D0
        .hps_io_hps_io_usb1_inst_D1      (HPS_USB_DATA[1]),         //                .hps_io_usb1_inst_D1
        .hps_io_hps_io_usb1_inst_D2      (HPS_USB_DATA[2]),         //                .hps_io_usb1_inst_D2
        .hps_io_hps_io_usb1_inst_D3      (HPS_USB_DATA[3]),         //                .hps_io_usb1_inst_D3
        .hps_io_hps_io_usb1_inst_D4      (HPS_USB_DATA[4]),         //                .hps_io_usb1_inst_D4
        .hps_io_hps_io_usb1_inst_D5      (HPS_USB_DATA[5]),         //                .hps_io_usb1_inst_D5
        .hps_io_hps_io_usb1_inst_D6      (HPS_USB_DATA[6]),         //                .hps_io_usb1_inst_D6
        .hps_io_hps_io_usb1_inst_D7      (HPS_USB_DATA[7]),         //                .hps_io_usb1_inst_D7
        .hps_io_hps_io_usb1_inst_CLK     (HPS_USB_CLKOUT),          //                .hps_io_usb1_inst_CLK
        .hps_io_hps_io_usb1_inst_STP     (HPS_USB_STP),             //                .hps_io_usb1_inst_STP
        .hps_io_hps_io_usb1_inst_DIR     (HPS_USB_DIR),             //                .hps_io_usb1_inst_DIR
        .hps_io_hps_io_usb1_inst_NXT     (HPS_USB_NXT),             //                .hps_io_usb1_inst_NXT
        .hps_io_hps_io_spim0_inst_CLK    (HPS_LCM_SPIM_CLK),        //               .hps_io_spim0_inst_CLK
        .hps_io_hps_io_spim0_inst_MOSI   (HPS_LCM_SPIM_MOSI),       //               .hps_io_spim0_inst_MOSI
        .hps_io_hps_io_spim0_inst_MISO   (HPS_LCM_SPIM_MISO),       //               .hps_io_spim0_inst_MISO
        .hps_io_hps_io_spim0_inst_SS0    (HPS_LCM_SPIM_SS),         //               .hps_io_spim0_inst_SS0
        .hps_io_hps_io_uart0_inst_RX     (HPS_UART_RX),             //                .hps_io_uart0_inst_RX
        .hps_io_hps_io_uart0_inst_TX     (HPS_UART_TX),             //                .hps_io_uart0_inst_TX
        .hps_io_hps_io_gpio_inst_GPIO35  (HPS_ENET_INT_N),          //                .hps_io_gpio_inst_GPIO35
        .memory_mem_a                  (HPS_DDR3_ADDR),             //          memory_0.mem_a
        .memory_mem_ba                 (HPS_DDR3_BA),               //                .mem_ba
        .memory_mem_ck                 (HPS_DDR3_CK_P),             //                .mem_ck
        .memory_mem_ck_n               (HPS_DDR3_CK_N),             //                .mem_ck_n
        .memory_mem_cke                (HPS_DDR3_CKE),              //                .mem_cke
        .memory_mem_cs_n               (HPS_DDR3_CS_N),             //                .mem_cs_n
        .memory_mem_ras_n              (HPS_DDR3_RAS_N),            //                .mem_ras_n
        .memory_mem_cas_n              (HPS_DDR3_CAS_N),            //                .mem_cas_n
        .memory_mem_we_n               (HPS_DDR3_WE_N),             //                .mem_we_n
        .memory_mem_reset_n            (HPS_DDR3_RESET_N),          //                .mem_reset_n
        .memory_mem_dq                 (HPS_DDR3_DQ),               //                .mem_dq
        .memory_mem_dqs                (HPS_DDR3_DQS_P),            //                .mem_dqs
        .memory_mem_dqs_n              (HPS_DDR3_DQS_N),            //                .mem_dqs_n
        .memory_mem_odt                (HPS_DDR3_ODT),              //                .mem_odt
        .memory_mem_dm                 (HPS_DDR3_DM),               //                .mem_dm
        .memory_oct_rzqin              (HPS_DDR3_RZQ),              //                .oct_rzqin
        .uart_rx_conduit               (GPIO_Pin7)                  //          uart_rx.uart_rx
    );
```

Figure A21: Top-Level Module Code Part 2

```
module My_UART_RX(
    input clk,
    input reset_n,
    input [1:0] address,
    input read,
    input chipselect,
    input RX_in,
    output [31:0] readdata
);

wire ld;              //load wire b/w UART and load reg
wire [8:0] dataRX;    //RX data bus b/w UART and load reg

UART_Rx U1(clk, reset_n, RX_in, dataRX, ld);
LoadReg U2(clk, reset_n, address, read, chipselect, dataRX, ld, readdata);

endmodule
```

Figure A22: UART Top-Level Code

```
module My_ClockDivider
(
   input clk,
   input reset_n,
   input chipselect,
   input [1:0] address,
   input write,
   input [31:0] writedata,
   output reg clk_out
);

   reg [24:0] count;                        //Register to count input clock edges
   reg [24:0] div = 25'd6;                  //Register to divide clock

   always @(posedge clk or negedge reset_n)
      begin
         if(~reset_n)
            begin
               count <= 25'b0;
               div <= div;
            end
         else if(chipselect && write && (address == 2'b00))
            begin
               count <= 25'b0;
               div <= writedata[24:0];
            end
         else if(count < div - 1)
            begin
               count <= count + 1'b1;
               div <= div;
            end
         else
            begin
               count <= 25'b0;
               div <= div;
            end
      end


   always @(posedge clk or negedge reset_n)
      begin
         if(~reset_n)
            begin
               clk_out <= 1'b0;
            end
         else if(count == div - 1)
            begin
               clk_out <= ~clk_out;
            end
         else
            begin
               clk_out <= clk_out;
            end
      end

endmodule
```

Figure A23: Clock Divider Code

```
module UART_Rx(
    input clk,
    input reset_n,
    input RX_in,
    output [8:0] RX_data,
    output reg load
);

    //-------------------------------------------------------
    // Signal Declarations: reg
    //-------------------------------------------------------
    reg [1:0] state;                //reg to hold current state
    reg [4:0] count;                //reg to count between bit samples
    reg [3:0] count_data;           //reg to count the number of data bits read
    reg [7:0] data;                 //reg to hold the read data bits
    reg error;                      //reg that goes high if parity is wrong or stop bit is not read

    parameter Idle = 2'b00, Shift = 2'b01, Stop = 2'b10;

    always @(posedge clk or negedge reset_n)
       begin
          if(~reset_n)
             begin
                count <= 5'b0;
                count_data <= 4'b0;
                data <= 8'bz;
                state <= Idle;
                error <= 1'b0;
                load <= 1'b0;
             end
          else
             begin
                case(state)
                   Idle :
                      begin
                         error <= 1'b0;
                         load <= 1'b0;
                         if(~RX_in)
                            begin
                               if(count == 9) //Changed from 8 to 9 for 18x clk
                                  begin
                                     state <= Shift;
                                     count <= 5'b0;
                                     count_data <= 4'b0;
                                     data <= 8'bz;
                                  end
                               else
                                  begin
                                     state <= Idle;
                                     count <= count + 1'b1;
                                  end
                            end
                         else
                            begin
                               state <= Idle;
                               count <= 5'b0;
                            end

                      end
                   Shift :
                      begin
                         load <= 1'b0;
                         if(count_data < 8)
                            begin
                               if(count == 17)//Changed to 17 for an 18x clk since it divides better
                                  begin
                                     count <= 5'b0;
                                     count_data <= count_data + 1'b1;
                                     data <= {RX_in, data[7:1]};          //Data comes LSB first
                                     state <= Shift;
                                  end
                               else
                                  begin
                                     data <= data;
                                     state <= Shift;
                                     count <= count + 1'b1;
                                  end
                            end
                         else
                            begin
                               state <= Stop;
                               count_data <= 4'b0;
                               count <= 5'b0;
                            end
                      end
                   Stop :
                      begin
                         if(count == 17)//Changed to 17 for an 18x clk since it divides better
                            begin
                               count <= 5'b0;
                               state <= Idle;
                               load <= 1'b1;
                               if(RX_in != 1'b1)
                                  begin
                                     error <= 1'b1;
                                  end
                            end
                         else
                            begin
                               state <= Stop;
                               count <= count + 1'b1;
                               load <= 1'b0;
                            end
                      end
                   default :
                      begin
                         count <= 5'b0;
                         count_data <= 4'b0;
                         data <= 8'bz;
                         state <= Idle;
                         error <= 1'b0;
                         load <= 1'b0;
                      end
                endcase
             end
       end

       assign RX_data = {error, data};

endmodule
```

Figure A24: UART FSM Code

```
module LoadReg(
    input clk,
    input reset_n,
    input [1:0] address,
    input read,
    input chipselect,
    input [8:0] RX_data,
    input load,
    output reg [31:0] readdata
);
    reg readFlag;      //register to track if the data has been read
    reg [8:0] data;    //register to store loaded data

    always @(posedge clk or negedge reset_n)
        begin
            if(~reset_n)
            begin
                readdata <= 32'b0;
                readFlag <= 1'b0;
                data <= 9'b0;
            end
            else if(load)
            begin
                readdata <= 32'b0;
                data <= RX_data;
                readFlag <= 1'b1;
            end
            else if(chipselect && read && (address == 2'b00))
            begin
                readdata <= {22'b0, data[8], readFlag, data[7:0]};
                data <= data;
                readFlag <= 0;
            end
            else
            begin
                readdata <= 32'b0;
                data <= data;
                readFlag <= readFlag;
            end
        end

endmodule
```

Figure A25: UART LoadReg Code

```
module SevenSegDecoder(
   input clk,
   input reset_n,
   input [1:0] address,
   input write,
   input [31:0] writedata,
   input chipselect,
   output reg [6:0] segs
);
   reg [5:0] data;      //Reg to store input number

   always @(posedge clk or negedge reset_n)
   begin
      if(reset_n == 1'b0)
      begin
         data <= 6'h3F;
      end
      else if(chipselect && write && address == 2'b0
      begin
         data <= writedata[5:0];
      end
      else
      begin
         data <= data;
      end
   end

   always @(data)    //decode to 7-seg
   begin
      case(data)
         0:
            begin
               segs <= 7'b1000000;
            end
         1:
            begin
               segs <= 7'b1111001;
            end
         2:
            begin
               segs <= 7'b0100100;
            end
         3:
            begin
               segs <= 7'b0110000;
            end
         4:
            begin
               segs <= 7'b0011001;
            end
         5:
            begin
               segs <= 7'b0010010;
            end
         6:
            begin
               segs <= 7'b0000010;
            end
         7:
            begin
               segs <= 7'b1111000;
            end
         8:
            begin
               segs <= 7'b0000000;
            end
         9:
            begin
               segs <= 7'b0010000;
            end
         10:
            begin
               segs <= 7'b0001000;//A
            end
         11:
            begin
               segs <= 7'b0000011;//b
            end
         12:
            begin
               segs <= 7'b1000110;//c
            end
         13:
            begin
               segs <= 7'b0100001;//d
            end
         14:
            begin
               segs <= 7'b0000110;//E
            end
         15:
            begin
               segs <= 7'b0001110;//F
            end
```

Other cases

```
            end
         16:
            begin
               segs <= 7'b0010000;//g
            end
         17:
            begin
               segs <= 7'b0001001;//H
            end
         18:
            begin
               segs <= 7'b1111001;//I
            end
         19:
            begin
               segs <= 7'b1110010;//j
            end
         20:
            begin
               segs <= 7'b1111111;//K
            end
         21:
            begin
               segs <= 7'b1000111;//L
            end
         22:
            begin
               segs <= 7'b1111111;//M
            end
         23:
            begin
               segs <= 7'b1001000;//N
            end
         24:
            begin
               segs <= 7'b0100011;//o
            end
         25:
            begin
               segs <= 7'b0001100;//P
            end
         26:
            begin
               segs <= 7'b0011000;//Q
            end
         27:
            begin
               segs <= 7'b0101111;//R
            end
         28:
            begin
               segs <= 7'b0010010;//S
            end
         29:
            begin
               segs <= 7'b0000111;//t
            end
         30:
            begin
               segs <= 7'b1000001;//U
            end
         31:
            begin
               segs <= 7'b1111111;//V
            end
         32:
            begin
               segs <= 7'b1111111;//W
            end
         33:
            begin
               segs <= 7'b1111111;//X
            end
         34:
            begin
               segs <= 7'b0011001;//Y
            end
         35:
            begin
               segs <= 7'b1111111;//Z
            end
         default:
            begin
               segs <= 7'b1111111;
            end
      endcase
   end

endmodule
```

Figure A26: SevenSegDecoder Code

```
module My_CRC(
    input clk,
    input reset_n,
    input [1:0] address,
    input read,
    input write,
    input chipselect,
    input [31:0] writedata,
    output reg [31:0] readdata
);

reg [7:0] crcTable [255:0];    //Array to hold crc lookup table
reg [7:0] crc;                 //Reg to hold calculated crc
reg [375:0] data;              //Reg to hold LiDAR data packet (47 bytes)
reg [3:0] count;               //Count reg for receiving data
reg [5:0] crcCount;            //Count reg for calculating crc
reg [1:0] status;              //CRC status reg: 0 - not ready, 1 - calculating, 2 - Data ready


always @(posedge clk or negedge reset_n)
begin
    if(reset_n == 1'b0)
    begin
        crcCount <= 6'b0;
        crc <= 8'b0;
        data <= 376'b0;
        count <= 4'b0;
        status <= 2'b0;
    end
    else if(count == 4'hc)
    begin
        //calc CRC using data
        if(crcCount >= 6'h2E)//Only calc CRC with first 46 bytes
        begin
            crcCount <= 6'b0;
            data <= data;
            crc <= crc;
            status <= 2'b10;
            count <= 4'b0;
        end
        else
        begin
            crc <= crcTable[((data[7:0] ^ crc) & 8'hff)];
            data <= {8'b0, data[375:8]};
            crcCount <= crcCount + 1'b1;
            status <= 2'b01;
            count <= count;
        end
    end
    else if(chipselect && write && (address == 2'b0))
    begin
        crcCount <= 6'b0;
        count <= count + 1'b1;
        crc <= 8'b0;
        status <= 2'b0;
        if(count == 4'hb)
            begin
                data <= {writedata[23:0], data[375:24]};//47 bytes in total, 4 bytes at a time. Only want 3 bytes on last.
            end
        else
            begin
                data <= {writedata, data[375:32]};//Once finished: data[7:0] = 0x54, data[15:8] = 0x2C, ...
            end
    end
    else
    begin
        data <= data;
        crcCount <= 6'b0;
        count <= count;
        status <= status;
        crc <= crc;
    end
end

always @(posedge clk or negedge reset_n)
begin
    if(reset_n == 1'b0)
    begin
        readdata <= 32'b0;
    end
    else if (chipselect && read && (address == 2'b0))
    begin
        readdata <= {22'b0, status, crc};
    end
    else
    begin
        readdata <= 32'b0;
    end
end

endmodule
```

Figure A27: CRC Code

```
uint8_t redisplay = 0;
int retVal = 0;                                     //var to store main return value
int fd = 0;                                         //file descriptor
uint16_t smallestDistance = 0xFFFF;                 //Smallest distance detected by LiDAR
float direction = 0;                                //Direction of current measurement
float smallestDir = 0;                              //Direction of smallest distance
uint8_t lcd_buffer[1024] = {0};                     //Array for storing LCD values
LCD_Dev lcd = {128, 64, 1024, lcd_buffer, 0, 0, 0}; //Struct for LCD
uint8_t waitingForPacket = 1;                       //Flag for switching b/w waiting for LiDAR data and displaying it
void * crc_addr = NULL;                             //Base address ptr for CRC module

uint16_t dataPacket[14] = {0};                      //Parsed distance and direction data
uint8_t crc = 0;                                    //var to hold calculated SW crc
uint8_t crc_hw = 0;                                 //var to hol calculated HW crc
uint8_t dataValid = 0;                              //Flag for whether data crc matched calculated crc
uint8_t lpacket[47] = {0};                          //Raw packet data buffer
pthread_t uart_id;                                  //Thread for uart task
```

...

Main Task Loop:

```
while(retVal != 1)
{
    while(!dataValid)
    {
        while(waitingForPacket)
        {
            pthread_mutex_lock(&lock);
            if(packetReady)
            {
                for(uint8_t i = 0; i < PACKET_LENGTH; ++i)
                {
                    lpacket[i] = sharedBuffer[i];
                }
                packetReady = 0;
                waitingForPacket = 0;
            }
            pthread_mutex_unlock(&lock);
        }


        //Send frame to CRC and read results (if invalid, throw out and restart)
        //Get SW CRC
        crc = CalCRC8(lpacket, 46);

        //Get HW CRC
        for(uint8_t i = 0; i < 47; i+=4)
        {
            if(i < 44)
                *(uint32_t*)crc_addr = ( lpacket[i+3] << 24)|( lpacket[i+2] << 16)|( lpacket[i+1] << 8)|(lpacket[i]);
            else
                *(uint32_t*)crc_addr = ( 0x00 << 24)|( lpacket[i+2] << 16)|( lpacket[i+1] << 8)|(lpacket[i]);
        }

        while((*(uint32_t*)crc_addr & 0x200) != 0x200);

        crc_hw = (0xff & (*(uint16_t*)crc_addr));

        printf("Packet CRC: %d\r\nSoftware CRC: %d\r\nHardware CRC: %d\r\n", lpacket[46], crc, crc_hw);

        if(crc_hw == crc)
            printf("HW CRC Correct\r\n");

        //Make sure packet data is valid
        if(crc == lpacket[46])
        {
            printf("Data Valid \r\n");
            dataValid = 1;
        }
        else
        {
            printf("Data Invalid \r\n");
            dataValid = 0;
        }
        waitingForPacket = 1;

    }

    //Parse frame and only keep distance data, start angle, and stop angle
    getLiDARMeasurements(lpacket, dataPacket);

    //Check if smallest distance is smaller than stored smallest (init to max int)
            //if smaller, store new smallest distance and the angle
    redisplay = 0;
    for(uint8_t i = 2; i < 14; ++i)
    {
        direction = (float)(dataPacket[1] - dataPacket[0]);
        if(direction < 0)
            direction += 360.0f;
        direction = (direction / 1100.0f) * (i - 2) + (float)(dataPacket[0])/100.0f; //deltaAngle * measurementNumber + startAngle = currentAngle

        if(dataPacket[i] <= smallestDistance)
        {
            smallestDistance = dataPacket[i];
            smallestDir = direction;
            redisplay = 1;
        }
        else if(round(smallestDir) == round(direction))
        {
            smallestDistance = dataPacket[i];
        }

    }
    printf("Smallest: %d, Dir: %0.3f\r\n", smallestDistance, direction);

    //Send direction of smallest distance to 7SegDecoders
    if(redisplay)
    {
        displaySmallestDistance(direction);
    }

    //Store new distances in array
    myLCD_DisplayData(&lcd, dataPacket[0], dataPacket[1], &dataPacket[2]);
    dataValid = 0;

    usleep(1000 * 25);  //Sleep .025 seconds
}
```

Figure A28: Main Task Loop Code and Local Variables

```
/**********************************************************************
* Purpose: Read bytes of lidar data in from UART module and store in a packet.
*               Once complete, stores packet in a shared buffer.
* Precondition:
*          Thread is started by main.
*
* Postcondition:
*          Reads bytes in and stores in 47-byte buffer
**********************************************************************/
static void* uart_task(void* arg)
{
    uint8_t lidarPacket[47] = {0};      //array to store raw packet data
    uint16_t lidarByte;                 //data read from UART. 8-bits of data + 2-bits for status
    uint8_t packetIndex = 0;            //Index var for storing 47-byte packet of lidar data

    while(1)
    {
        lidarPacket[0] = 0;
        lidarPacket[1] = 0;
        //Read LiDAR Data from UART RX (into header)
        while((lidarPacket[0] != PACKET_HEADER) || (lidarPacket[1] != PACKET_VERSION) || (lidarPacket[3] != 0xe))
        {
            while((!(lidarByte & 0x100) || (lidarByte & 0x200)) || ((lidarByte & 0x0ff) != PACKET_HEADER))//Blocking. Waiting for data to be ready
            {
                //Read header into lidarByte
                lidarByte = (uint32_t)(*(uint16_t*)uart_addr & 0x3FF);

            }

            lidarPacket[0] = (uint8_t)(0xFF & lidarByte);

            do
            {
                lidarByte = (uint32_t)(*(uint16_t*)uart_addr & 0x3FF);
            }
            while(!(lidarByte & 0x100) || (lidarByte & 0x200));

            lidarPacket[1] = (uint8_t)(0xFF & lidarByte);

            do
            {
                lidarByte = (uint16_t)(*(uint32_t*)uart_addr & 0x3FF);
            }
            while(!(lidarByte & 0x100) || (lidarByte & 0x200));

            lidarPacket[2] = (uint8_t)(0xFF & lidarByte);

            do
            {
                lidarByte = (uint16_t)(*(uint32_t*)uart_addr & 0x3FF);
            }
            while(!(lidarByte & 0x100) || (lidarByte & 0x200));

            lidarPacket[3] = (uint8_t)(0xFF & lidarByte);
        }

        for(packetIndex = 4; packetIndex < 47; packetIndex++)
        {
            do
            {
                lidarByte = (uint16_t)((*(uint32_t*)uart_addr) & 0x3FF);
            }
            while(!(lidarByte & 0x100) || (lidarByte & 0x200));//While not valid

            lidarPacket[packetIndex] = (uint8_t)(0xFF & lidarByte);

        }

        pthread_mutex_lock(&lock);
        for(uint8_t j = 0; j < 47; ++j)
        {
            sharedBuffer[j] = lidarPacket[j];
        }
        packetReady = 1;
        pthread_mutex_unlock(&lock);

    }

    return NULL;
}
```

Figure A29: UART Software Task Code

```c
/*********************************************************************
* Purpose: Displays LD19 distance data on DE10-Standard's LCD
* Precondition:
*          Function is called and passed pointer to lcd struct,
*           start angle of data in 0.01 deg, end angle in 0.01 deg,
*           and a pointer to a buffer holding the ditance data
*
* Postcondition:
*          Data in buffer is displayed on LCD in directions specified by start and
*           end angle. Displays up to a max distance specified in header file.
*********************************************************************/
void myLCD_DisplayData(LCD_Dev* lcd, uint16_t startAngle, uint16_t endAngle, uint16_t* data)
{
    printf("StartAngle: %d, EndAngle: %d\r\n", startAngle, endAngle);
    float beginAngle = ((float)(startAngle) / 100.0f) * DEG_TO_RAD;     //Convert from .01 deg to deg. Var for angle to begin displaying
    float stopAngle = ((float)(endAngle) / 100.0f) * DEG_TO_RAD;        //Convert from .01 deg to deg. Var for end angle of display
    float deltaAngle = stopAngle - beginAngle;                          //Var for change in angle b/w measurements
    uint16_t x = 0;                                                     //Var for x coord of LCD
    uint16_t y = 0;                                                     //Var for y coord of LCD

    if(deltaAngle < 0)      //Only place where delta is - is when completing a rotation
        deltaAngle += 360.0f;

    deltaAngle = deltaAngle/11.0f;

    for(uint8_t i = 0; i < 12; ++i)
    {
        //Clear lines
        for(uint8_t r = 1; r < 33; ++r)
        {
            x = round(r * sinf(beginAngle) + (lcd->width/2));//switch sin and cos since 0 deg is vertical
            y = round(r * cosf(beginAngle) + (lcd->height/2));
            DRAW_Pixel2(lcd, x, y, WHITE);
        }

        if(data[i] <= MAX_DISPLAY_DIST)
        {
            //Get location of obj
            x = (uint16_t)(round(fabs((lcd->width/2) + ((float)(data[i]) * sinf(beginAngle)) * ((float)(lcd->height) / (float)(MAX_DISPLAY_DIST)))));
            y = (uint16_t)(round(fabs((lcd->height/2) + ((float)(data[i]) * cosf(beginAngle)) * ((float)(lcd->height) / (float)(MAX_DISPLAY_DIST)))));
            printf("X: %d, Y: %d, Begin: %0.3f, End: %0.3f, Delta: %0.3f\r\n", x, y, beginAngle, stopAngle, deltaAngle);
            //Draw obj at location
            DRAW_Pixel2(lcd, x, y, BLACK);
        }
        beginAngle += deltaAngle;
    }

}
```

Figure A30: LCD Display Function

```
/*****************************************************************************
* Purpose: Sends values to 7-seg decoders to indicate direction of smallest distance
*
* Precondition:
*          Function is called and passed direction of smallest distance
*
* Postcondition:
*          Depending on direction, sends hex values for Front, Behind,
*           Left, and Right to the 7-seg decoders.
*****************************************************************************/
void displaySmallestDistance(float dir)
{
    if((dir > 315.0f && dir < 360.0f) || (dir > 0.0f && dir < 45.0f))
    {
        //Forward
        *(uint32_t*) hex5_addr = fwd_hex[0];
        *(uint32_t*) hex4_addr = fwd_hex[1];
        *(uint32_t*) hex3_addr = fwd_hex[2];
        *(uint32_t*) hex2_addr = fwd_hex[3];
        *(uint32_t*) hex1_addr = fwd_hex[4];
        *(uint32_t*) hex0_addr = fwd_hex[5];
    }
    else if(dir <= 315.0f && dir >= 225.0f)
    {
        //Left
        *(uint32_t*) hex5_addr = left_hex[0];
        *(uint32_t*) hex4_addr = left_hex[1];
        *(uint32_t*) hex3_addr = left_hex[2];
        *(uint32_t*) hex2_addr = left_hex[3];
        *(uint32_t*) hex1_addr = left_hex[4];
        *(uint32_t*) hex0_addr = left_hex[5];
    }
    else if(dir >= 45.0f && dir <= 135.0f)
    {
        //Right
        *(uint32_t*) hex5_addr = right_hex[0];
        *(uint32_t*) hex4_addr = right_hex[1];
        *(uint32_t*) hex3_addr = right_hex[2];
        *(uint32_t*) hex2_addr = right_hex[3];
        *(uint32_t*) hex1_addr = right_hex[4];
        *(uint32_t*) hex0_addr = right_hex[5];
    }
    else if(dir > 135.0f && dir < 225.0f)
    {
        //Backward
        *(uint32_t*) hex5_addr = behind_hex[0];
        *(uint32_t*) hex4_addr = behind_hex[1];
        *(uint32_t*) hex3_addr = behind_hex[2];
        *(uint32_t*) hex2_addr = behind_hex[3];
        *(uint32_t*) hex1_addr = behind_hex[4];
        *(uint32_t*) hex0_addr = behind_hex[5];
    }
    else
    {
        //Error
        *(uint32_t*) hex5_addr = error_hex[0];
        *(uint32_t*) hex4_addr = error_hex[1];
        *(uint32_t*) hex3_addr = error_hex[2];
        *(uint32_t*) hex2_addr = error_hex[3];
        *(uint32_t*) hex1_addr = error_hex[4];
        *(uint32_t*) hex0_addr = error_hex[5];
    }

}
```

Figure A31: Direction Display Code

```
static const uint8_t CrcTable[256] = {
    0x00, 0x4d, 0x9a, 0xd7, 0x79, 0x34, 0xe3, 0xae, 0xf2, 0xbf, 0x68, 0x25,
    0x8b, 0xc6, 0x11, 0x5c, 0xa9, 0xe4, 0x33, 0x7e, 0xd0, 0x9d, 0x4a, 0x07,
    0x5b, 0x16, 0xc1, 0x8c, 0x22, 0x6f, 0xb8, 0xf5, 0x1f, 0x52, 0x85, 0xc8,
    0x66, 0x2b, 0xfc, 0xb1, 0xed, 0xa0, 0x77, 0x3a, 0x94, 0xd9, 0x0e, 0x43,
    0xb6, 0xfb, 0x2c, 0x61, 0xcf, 0x82, 0x55, 0x18, 0x44, 0x09, 0xde, 0x93,
    0x3d, 0x70, 0xa7, 0xea, 0x3e, 0x73, 0xa4, 0xe9, 0x47, 0x0a, 0xdd, 0x90,
    0xcc, 0x81, 0x56, 0x1b, 0xb5, 0xf8, 0x2f, 0x62, 0x97, 0xda, 0x0d, 0x40,
    0xee, 0xa3, 0x74, 0x39, 0x65, 0x28, 0xff, 0xb2, 0x1c, 0x51, 0x86, 0xcb,
    0x21, 0x6c, 0xbb, 0xf6, 0x58, 0x15, 0xc2, 0x8f, 0xd3, 0x9e, 0x49, 0x04,
    0xaa, 0xe7, 0x30, 0x7d, 0x88, 0xc5, 0x12, 0x5f, 0xf1, 0xbc, 0x6b, 0x26,
    0x7a, 0x37, 0xe0, 0xad, 0x03, 0x4e, 0x99, 0xd4, 0x7c, 0x31, 0xe6, 0xab,
    0x05, 0x48, 0x9f, 0xd2, 0x8e, 0xc3, 0x14, 0x59, 0xf7, 0xba, 0x6d, 0x20,
    0xd5, 0x98, 0x4f, 0x02, 0xac, 0xe1, 0x36, 0x7b, 0x27, 0x6a, 0xbd, 0xf0,
    0x5e, 0x13, 0xc4, 0x89, 0x63, 0x2e, 0xf9, 0xb4, 0x1a, 0x57, 0x80, 0xcd,
    0x91, 0xdc, 0x0b, 0x46, 0xe8, 0xa5, 0x72, 0x3f, 0xca, 0x87, 0x50, 0x1d,
    0xb3, 0xfe, 0x29, 0x64, 0x38, 0x75, 0xa2, 0xef, 0x41, 0x0c, 0xdb, 0x96,
    0x42, 0x0f, 0xd8, 0x95, 0x3b, 0x76, 0xa1, 0xec, 0xb0, 0xfd, 0x2a, 0x67,
    0xc9, 0x84, 0x53, 0x1e, 0xeb, 0xa6, 0x71, 0x3c, 0x92, 0xdf, 0x08, 0x45,
    0x19, 0x54, 0x83, 0xce, 0x60, 0x2d, 0xfa, 0xb7, 0x5d, 0x10, 0xc7, 0x8a,
    0x24, 0x69, 0xbe, 0xf3, 0xaf, 0xe2, 0x35, 0x78, 0xd6, 0x9b, 0x4c, 0x01,
    0xf4, 0xb9, 0x6e, 0x23, 0x8d, 0xc0, 0x17, 0x5a, 0x06, 0x4b, 0x9c, 0xd1,
    0x7f, 0x32, 0xe5, 0xa8};


/*************************************************************************
* Purpose: Calculates crc using received lidar packet
* Precondition:
*          Function is called. Pointer to buffer of raw packet data, and
*           length of packet - 1 is passed.
*
* Postcondition:
*          Returns calculated crc
*************************************************************************/
uint8_t CalCRC8(uint8_t *packet, uint8_t len)
{

uint8_t crc = 0;            //Var to store crc
uint8_t packetIndex;        //Index of packet

    for (packetIndex = 0; packetIndex < len; packetIndex++)
    {
        crc = CrcTable[(crc ^ *packet++) & 0xff];
    }

    return crc;

}

/*************************************************************************
* Purpose: Parses raw lidar packet into a packet containing direction
*           and angle data.
* Precondition:
*          Function called and passed pointers to raw packet and data packet.
*           Assumed that packet size is 47 bytes (not adjustable on LD19 anyways)
*
* Postcondition:
*          Parsed packet data stored in passed data packet
*************************************************************************/
void getLiDARMeasurements(uint8_t* packet, uint16_t* dataPacket)
{
    uint8_t packetIndex = 2;    //Index of packet

    if(packet != NULL && dataPacket != NULL)
    {
        dataPacket[0] = (packet[5] << 8)|packet[4];     //Start angle
        dataPacket[1] = (packet[43] << 8)|packet[42];       //End angle

        for(uint8_t i = 6; i < 42; i+=3 )//don't want 8, 11, 14, ... (These are intensity, not distance)
        {
            dataPacket[packetIndex] = (packet[i + 1] << 8) | packet[i];//7-6, 10-9, 13-12, 16-15, 19-18, 22-21, 25-24, 28-27, 31-30, 34-33, 37-36, 40-39
            packetIndex++;
        }
    }

}
```

Figure A32: LD19 CRC and Data Parse Functions

Signal Name	FPGA Pin No.	Description	I/O Standard
HEX0[0]	PIN_W17	Seven Segment Digit 0[0]	3.3V
HEX0[1]	PIN_V18	Seven Segment Digit 0[1]	3.3V
HEX0[2]	PIN_AG17	Seven Segment Digit 0[2]	3.3V
HEX0[3]	PIN_AG16	Seven Segment Digit 0[3]	3.3V
HEX0[4]	PIN_AH17	Seven Segment Digit 0[4]	3.3V
HEX0[5]	PIN_AG18	Seven Segment Digit 0[5]	3.3V
HEX0[6]	PIN_AH18	Seven Segment Digit 0[6]	3.3V
HEX1[0]	PIN_AF16	Seven Segment Digit 1[0]	3.3V
HEX1[1]	PIN_V16	Seven Segment Digit 1[1]	3.3V
HEX1[2]	PIN_AE16	Seven Segment Digit 1[2]	3.3V
HEX1[3]	PIN_AD17	Seven Segment Digit 1[3]	3.3V
HEX1[4]	PIN_AE18	Seven Segment Digit 1[4]	3.3V
HEX1[5]	PIN_AE17	Seven Segment Digit 1[5]	3.3V
HEX1[6]	PIN_V17	Seven Segment Digit 1[6]	3.3V
HEX2[0]	PIN_AA21	Seven Segment Digit 2[0]	3.3V
HEX2[1]	PIN_AB17	Seven Segment Digit 2[1]	3.3V
HEX2[2]	PIN_AA18	Seven Segment Digit 2[2]	3.3V
HEX2[3]	PIN_Y17	Seven Segment Digit 2[3]	3.3V
HEX2[4]	PIN_Y18	Seven Segment Digit 2[4]	3.3V
HEX2[5]	PIN_AF18	Seven Segment Digit 2[5]	3.3V
HEX2[6]	PIN_W16	Seven Segment Digit 2[6]	3.3V
HEX3[0]	PIN_Y19	Seven Segment Digit 3[0]	3.3V
HEX3[1]	PIN_W19	Seven Segment Digit 3[1]	3.3V
HEX3[2]	PIN_AD19	Seven Segment Digit 3[2]	3.3V
HEX3[3]	PIN_AA20	Seven Segment Digit 3[3]	3.3V
HEX3[4]	PIN_AC20	Seven Segment Digit 3[4]	3.3V
HEX3[5]	PIN_AA19	Seven Segment Digit 3[5]	3.3V
HEX3[6]	PIN_AD20	Seven Segment Digit 3[6]	3.3V
HEX4[0]	PIN_AD21	Seven Segment Digit 4[0]	3.3V
HEX4[1]	PIN_AG22	Seven Segment Digit 4[1]	3.3V
HEX4[2]	PIN_AE22	Seven Segment Digit 4[2]	3.3V
HEX4[3]	PIN_AE23	Seven Segment Digit 4[3]	3.3V
HEX4[4]	PIN_AG23	Seven Segment Digit 4[4]	3.3V
HEX4[5]	PIN_AF23	Seven Segment Digit 4[5]	3.3V
HEX4[6]	PIN_AH22	Seven Segment Digit 4[6]	3.3V
HEX5[0]	PIN_AF21	Seven Segment Digit 5[0]	3.3V
HEX5[1]	PIN_AG21	Seven Segment Digit 5[1]	3.3V
HEX5[2]	PIN_AF20	Seven Segment Digit 5[2]	3.3V
HEX5[3]	PIN_AG20	Seven Segment Digit 5[3]	3.3V
HEX5[4]	PIN_AE19	Seven Segment Digit 5[4]	3.3V
HEX5[5]	PIN_AF19	Seven Segment Digit 5[5]	3.3V
HEX5[6]	PIN_AB21	Seven Segment Digit 5[6]	3.3V

Table A1: 7-Segment Display Pinouts

Signal Name	FPGA Pin No.	Description	I/O Standard
CLOCK_50	PIN_AF14	50 MHz clock input	3.3V
CLOCK2_50	PIN_AA16	50 MHz clock input	3.3V
CLOCK3_50	PIN_Y26	50 MHz clock input	3.3V
CLOCK4_50	PIN_K14	50 MHz clock input	3.3V
HPS_CLOCK1_25	PIN_D25	25 MHz clock input	3.3V
HPS_CLOCK2_25	PIN_F25	25 MHz clock input	3.3V

Table A2: Clock Pinouts

Signal Name	FPGA Pin No.	Description	I/O Standard
GPIO[0]	PIN_W15	GPIO Connection 0[0]	3.3V
GPIO[1]	PIN_AK2	GPIO Connection 0[1]	3.3V
GPIO[2]	PIN_Y16	GPIO Connection 0[2]	3.3V
GPIO[3]	PIN_AK3	GPIO Connection 0[3]	3.3V
GPIO[4]	PIN_AJ1	GPIO Connection 0[4]	3.3V
GPIO[5]	PIN_AJ2	GPIO Connection 0[5]	3.3V
GPIO[6]	PIN_AH2	GPIO Connection 0[6]	3.3V
GPIO[7]	PIN_AH3	GPIO Connection 0[7]	3.3V
GPIO[8]	PIN_AH4	GPIO Connection 0[8]	3.3V
GPIO[9]	PIN_AH5	GPIO Connection 0[9]	3.3V
GPIO[10]	PIN_AG1	GPIO Connection 0[10]	3.3V
GPIO[11]	PIN_AG2	GPIO Connection 0[11]	3.3V
GPIO[12]	PIN_AG3	GPIO Connection 0[12]	3.3V
GPIO[13]	PIN_AG5	GPIO Connection 0[13]	3.3V
GPIO[14]	PIN_AG6	GPIO Connection 0[14]	3.3V
GPIO[15]	PIN_AG7	GPIO Connection 0[15]	3.3V
GPIO[16]	PIN_AG8	GPIO Connection 0[16]	3.3V
GPIO[17]	PIN_AF4	GPIO Connection 0[17]	3.3V
GPIO[18]	PIN_AF5	GPIO Connection 0[18]	3.3V
GPIO[19]	PIN_AF6	GPIO Connection 0[19]	3.3V
GPIO[20]	PIN_AF8	GPIO Connection 0[20]	3.3V
GPIO[21]	PIN_AF9	GPIO Connection 0[21]	3.3V
GPIO[22]	PIN_AF10	GPIO Connection 0[22]	3.3V
GPIO[23]	PIN_AE7	GPIO Connection 0[23]	3.3V
GPIO[24]	PIN_AE9	GPIO Connection 0[24]	3.3V
GPIO[25]	PIN_AE11	GPIO Connection 0[25]	3.3V
GPIO[26]	PIN_AE12	GPIO Connection 0[26]	3.3V
GPIO[27]	PIN_AD7	GPIO Connection 0[27]	3.3V

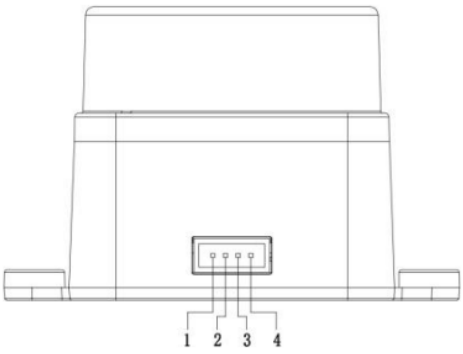
Table A3: GPIO Pinouts

Signal Name	FPGA Pin No.	Description	I/O Standard
SW[0]	PIN_AB30	Slide Switch[0]	Depend on JP3
SW[1]	PIN_Y27	Slide Switch[1]	Depend on JP3
SW[2]	PIN_AB28	Slide Switch[2]	Depend on JP3
SW[3]	PIN_AC30	Slide Switch[3]	Depend on JP3
SW[4]	PIN_W25	Slide Switch[4]	Depend on JP3
SW[5]	PIN_V25	Slide Switch[5]	Depend on JP3
SW[6]	PIN_AC28	Slide Switch[6]	Depend on JP3
SW[7]	PIN_AD30	Slide Switch[7]	Depend on JP3
SW[8]	PIN_AC29	Slide Switch[8]	Depend on JP3
SW[9]	PIN_AA30	Slide Switch[9]	Depend on JP3

Table A4: Switch Pinouts

CLOCK_50	Unknown	PIN_AF14	3B	B3B_N0	3.3-V LVTTL		16mA (default)
HEX0[0]	Unknown	PIN_W17	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX0[1]	Unknown	PIN_V18	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX0[2]	Unknown	PIN_AG17	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX0[3]	Unknown	PIN_AG16	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX0[4]	Unknown	PIN_AH17	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX0[5]	Unknown	PIN_AG18	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX0[6]	Unknown	PIN_AH18	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX1[0]	Unknown	PIN_AF16	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX1[1]	Unknown	PIN_V16	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX1[2]	Unknown	PIN_AE16	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX1[3]	Unknown	PIN_AD17	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX1[4]	Unknown	PIN_AE18	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX1[5]	Unknown	PIN_AE17	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX1[6]	Unknown	PIN_V17	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX2[0]	Unknown	PIN_AA21	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX2[1]	Unknown	PIN_AB17	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX2[2]	Unknown	PIN_AA18	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX2[3]	Unknown	PIN_Y17	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX2[4]	Unknown	PIN_Y18	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX2[5]	Unknown	PIN_AF18	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX2[6]	Unknown	PIN_W16	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX3[0]	Unknown	PIN_Y19	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX3[1]	Unknown	PIN_W19	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX3[2]	Unknown	PIN_AD19	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX3[3]	Unknown	PIN_AA20	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX3[4]	Unknown	PIN_AC20	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX3[5]	Unknown	PIN_AA19	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX3[6]	Unknown	PIN_AD20	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX4[0]	Unknown	PIN_AD21	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX4[1]	Unknown	PIN_AG22	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX4[2]	Unknown	PIN_AE22	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX4[3]	Unknown	PIN_AE23	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX4[4]	Unknown	PIN_AG23	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX4[5]	Unknown	PIN_AF23	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX4[6]	Unknown	PIN_AH22	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX5[0]	Unknown	PIN_AF21	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX5[1]	Unknown	PIN_AG21	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX5[2]	Unknown	PIN_AF20	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX5[3]	Unknown	PIN_AG20	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX5[4]	Unknown	PIN_AE19	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX5[5]	Unknown	PIN_AF19	4A	B4A_N0	3.3-V LVTTL		16mA (default)
HEX5[6]	Unknown	PIN_AB21	4A	B4A_N0	3.3-V LVTTL		16mA (default)
GPIO_Pin7	Unknown	PIN_AH2	3A	B3A_N0	3.3-V LVTTL		16mA (default)

Table A5: Project Pinout



port number	signal name	type	description	minimum	typical	maximum
1	Tx	output	LiDAR data output	0V	3.3V	3.5V
2	PWM	input	motor control	0V	-	3.3V
3	GND	power supply	negative	-	0V	-
4	P5V	power supply	positive	4.5V	5V	5.5V

Figure A33: LD19 Pinout

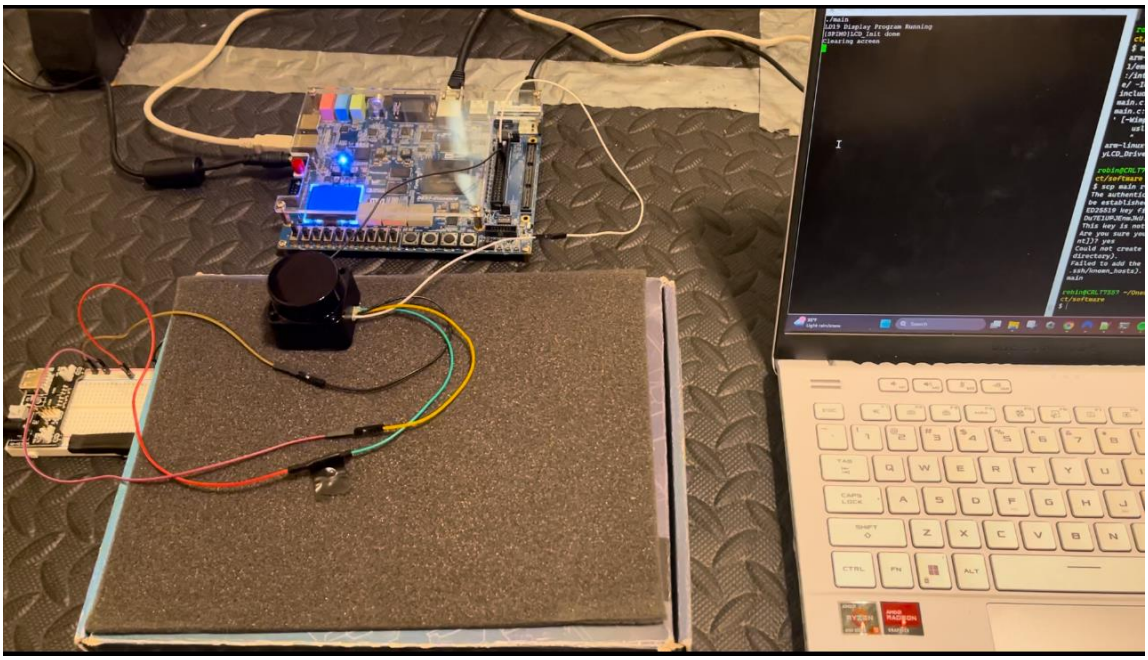


Figure A34: Project Setup

Table of Figures

List of Tables

References

[1] *DE10-Standard User Manual,* Terasic Inc, Hsinchu, Taiwan, 2018.

[2] LD19 Development Manual v2.5, youyeetoo Geek Shop, 2022.