



Table of Contents

[Abstract 2](#_30j0zll)

[Introduction 3](#_3znysh7)

[Design 3](#_2et92p0)

[Physical hardware design 3](#_tyjcwt)

[Synthesized hardware design 3](#_1t3h5sf)

[Design Structure 4](#_4d34og8)

[Modules 5](#_2s8eyo1)

[Clock Divider–clk\_dv.v 5](#)

[UART Receiver–UART\_rx.v 5](#)

[Flag Domain Crossing–flag\_x\_domain.v 10](#_wddy84crczvr)

[UART transmitter–UART\_tx.v 10](#_g1r18omc74oa)

[Seven Segment Display–seg\_7.v 13](#_pea0xwh8q4os)

[Simulation and Testing 14](#_17dp8vu)

[Problems 15](#_3rdcrjn)

[Results and Conclusion 15](#_26in1rg)

[Appendix A – RTL Diagrams 16](#_lnxbz9)

[Appendix B – Waveforms 18](#_o1liip29khcm)

[Appendix C – State Machine Diagrams 19](#_k4qyftxwhzoe)

[Appendix D – Pin Tables 20](#_z9rzchshg9wc)

[Table of Figures 22](#_35nkun2)

[References 23](#_1ksv4uv)

# Abstract

This lab is to create the receiving side of a UART. This receiver shall adheres to the following specifications:

1. 9600 baud

2. Eight data bits

3. One start bit

4. One stop bit

5. Even parity bit

There are two parts to this lab. For the first part, the data to be received is hooked to the TX side of the USB UART dongle. When a key is typed in a terminal emulator (putty), this character is received on the RX line. That character’s hex code is then displayed on 7-segment displays on the board. For the second part of the lab, a loopback is performed using the transmitter from the previous lab. The character is to be received by the RX line of the USB UART dongle. Check to see if it is lowercase a – z. If it is lowercase, convert it to uppercase. Then send it via the transmitter back to the terminal emulator. This causes the character typed to appear on the putty screen (capitalized).

# Introduction

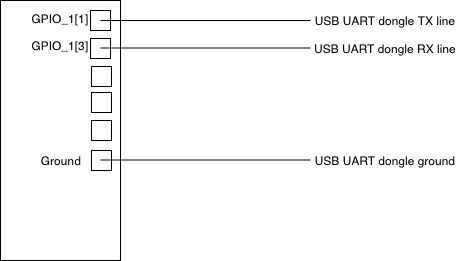
The purpose of this lab is to build the receiving end of a UART to interact with the UART transmitter built in the last lab. This lab is the first this term to handle clock domain crossing. The goal is to effectively transmit and receive data across different clock domains, in a serial UART fashion.

# Design

Here, the physical hardware and synthesized hardware designs are discussed.

## Physical hardware design

The UART USB dongle is used for transmission and reception of the data to and from the DE1-SoC board through GPIO pins. A ground is connected to sync the dongle’s ground with the DE1-SoC’s as well.



*Figure 1: Hardware Connection Diagram*

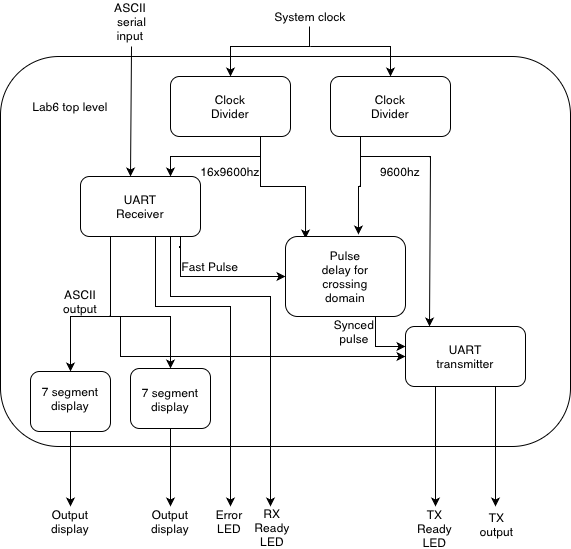
The LED outputs from the transmitter and receiver modules are assigned to LED’s on the board, and the HEX outputs of the seven segment display modules are assigned to HEX0 and HEX1.

## Synthesized hardware design

In this section discuss the major functional modules of your Design hierarchy.

### Design Structure

Here is the block diagram for the completed project:



*Figure 2: Hierarchical Block Diagram*

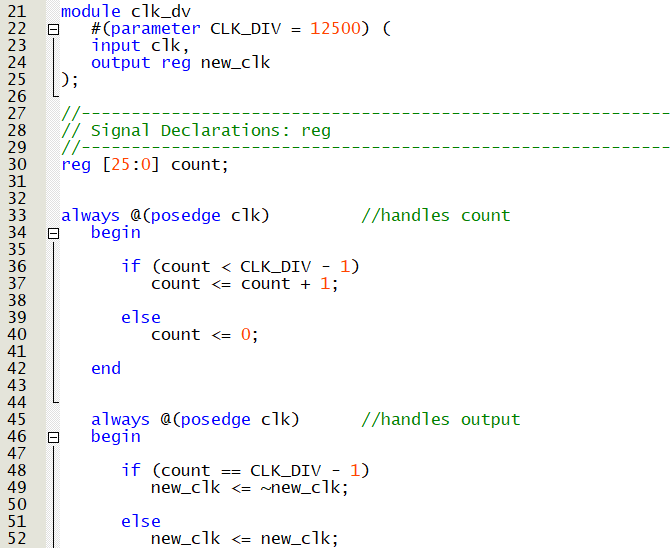
The UART receiver module receives the UART serial transmission from the USB dongle, and the 16x9600hz clock from a clock divider. The receiver module’s outputs are a pulse to signal the transmitter, an error signal, a ready signal, and the 8 bit ASCII code. A separate clock divider creates a 9600hz clock for the transmitter. The pulse delay module receives the pulse from the receiver module and both clocks, and transforms the pulse from the faster clock to pulse as synchronized for the slower clock. The ASCII output from the receiver module is sent to two seven segment display modules whose outputs correspond to seven segment display for the hex of the ASCII character code. The UART transmitter module takes the delayed pulse from the pulse delay module, the 9600hz clock, and the UART receiver’s 8 bit ASCII code output. Its outputs are serial UART output and a ready LED signal.

### Modules

Each module is further described below.

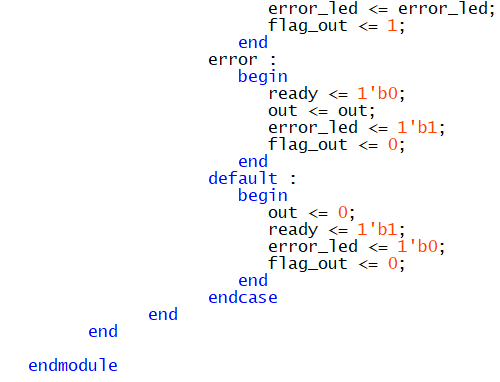
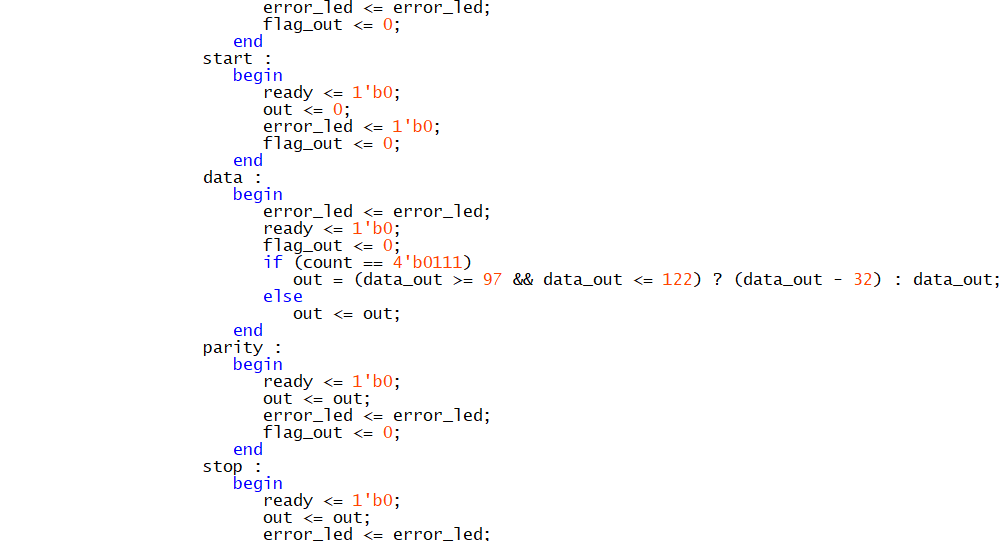
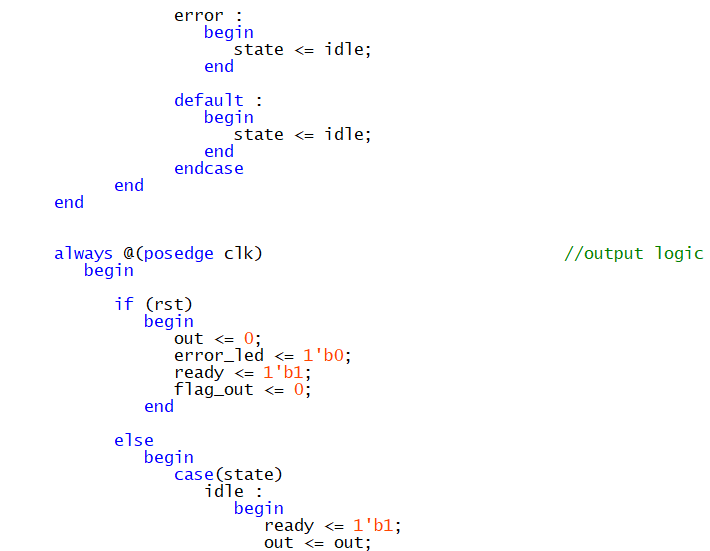
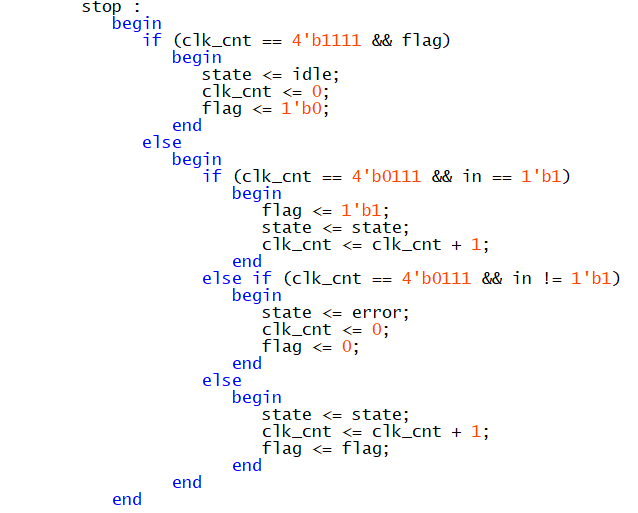
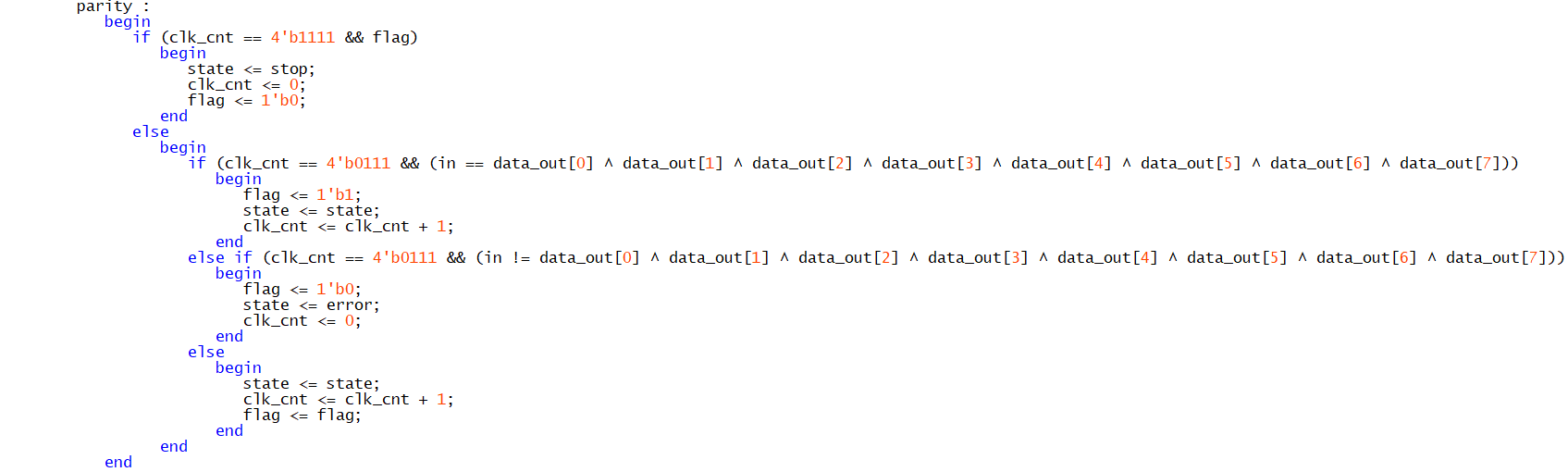
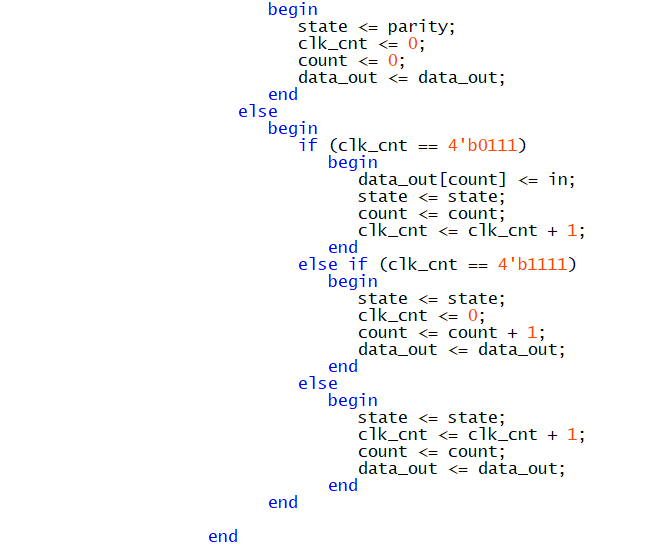
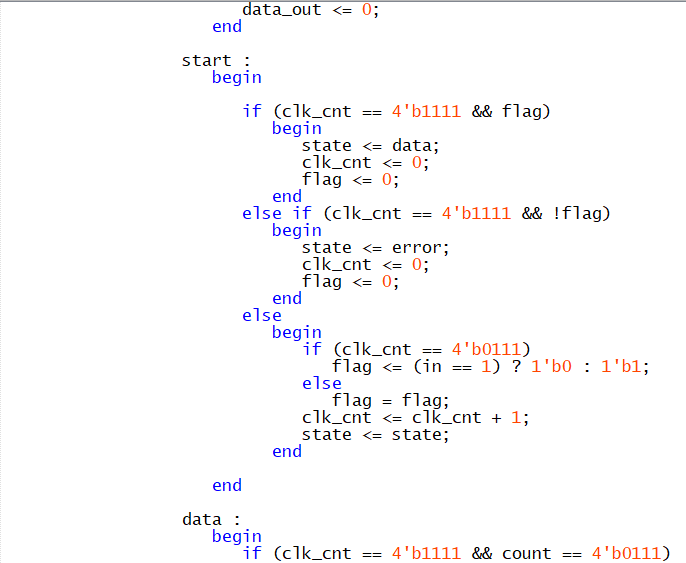
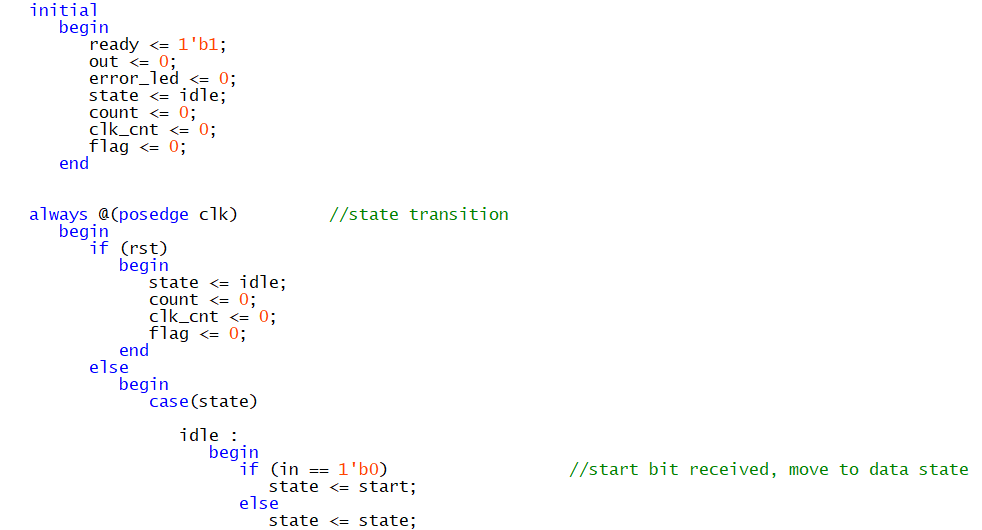
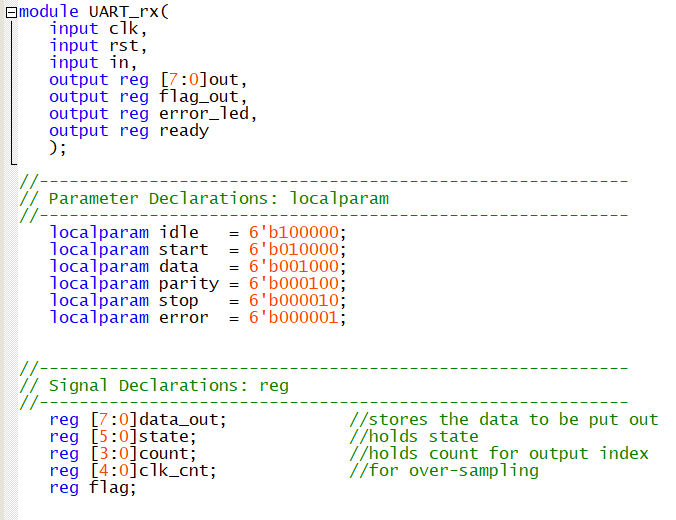
#### Clock Divider–clk\_dv.v

The clock divider takes the system clock as input and modifies its clock output based on a given parameter to produce a new clock of the desired frequency. The RTL for this module can be found in Appendix A. The module code follows.



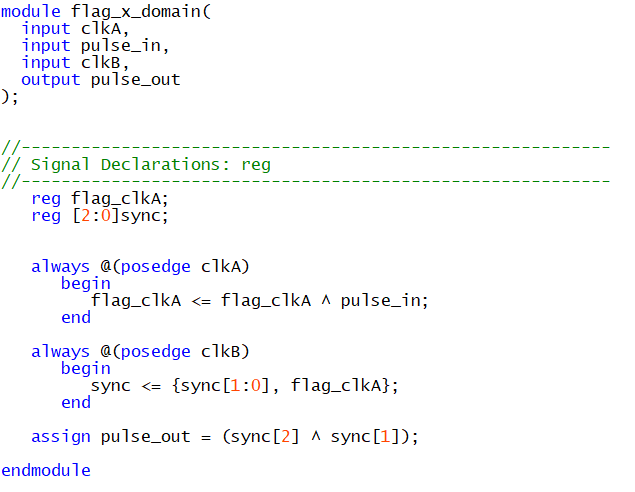
#### UART Receiver–UART\_rx.v

The UART receiver takes the faster clock, serial UART input, and a reset signal as its inputs. When the receiver detects a start bit, a state machine (found in Appendix C) kicks off to transfer the data bits into a temporary register, calculate and compare an even parity bit, and receive a stop bit. If there is an error in the parity bit, the state machine “error” state is entered for one clock cycle, and the error LED output is high until the next start bit is received. Otherwise, the output data register is loaded with the data received and remains there until the next start bit is received. Appendix A contains the RTL diagram for this module, Appendix B contains the result of the simulation waveforms, and Appendix C contains the state machine diagram. The module code follows.



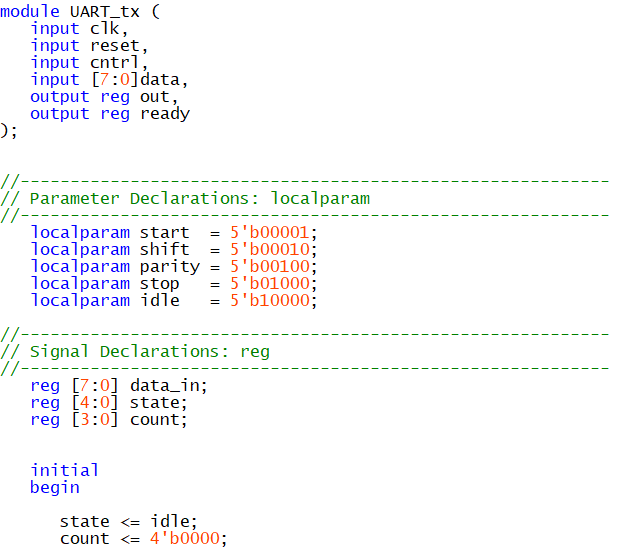
#### Flag Domain Crossing–flag\_x\_domain.v

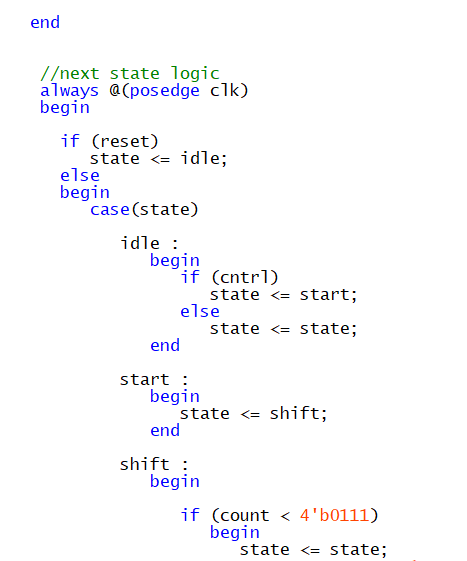
The flag domain crossing module takes in two different clocks, and a pulse. The logic delays the pulse from one clock and synchronizes it with the other, outputting the synchronized pulse. The RTL for this module can be found in Appendix A. The module code follows.

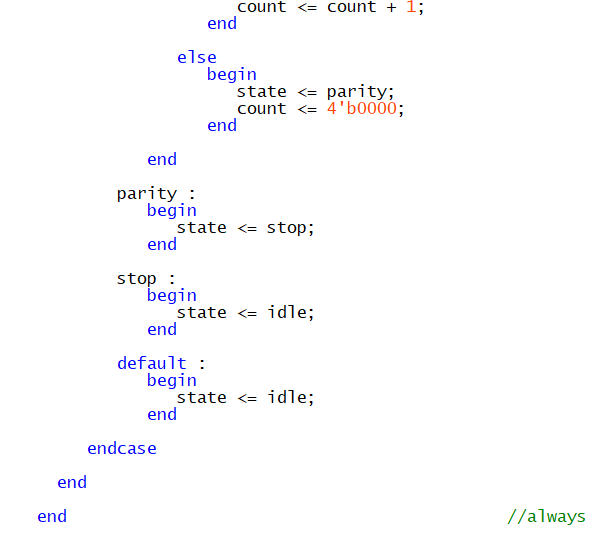


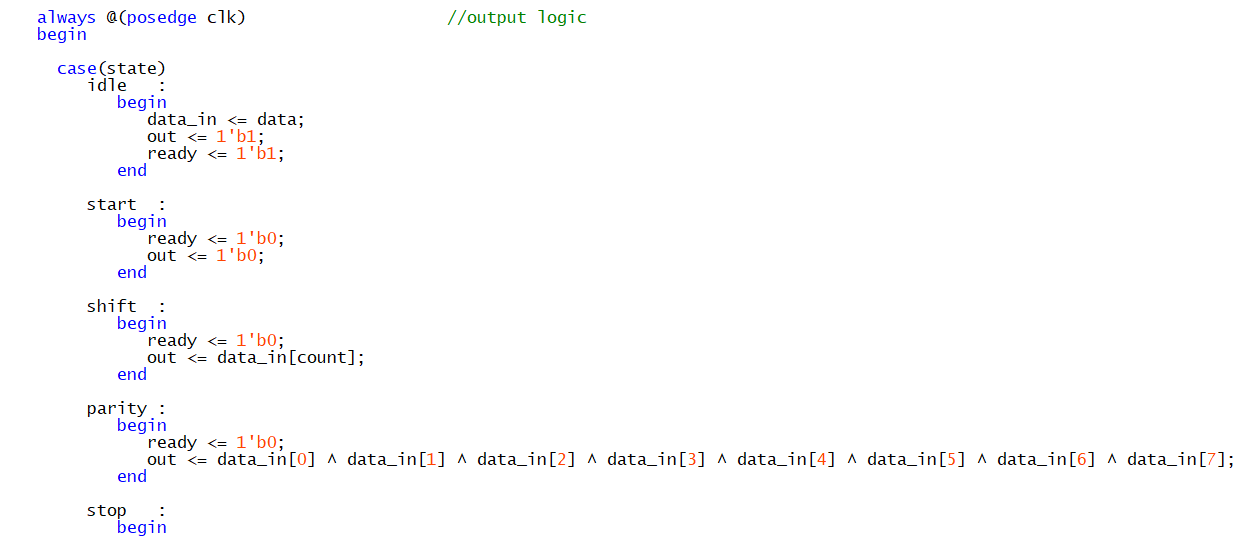
#### UART transmitter–UART\_tx.v

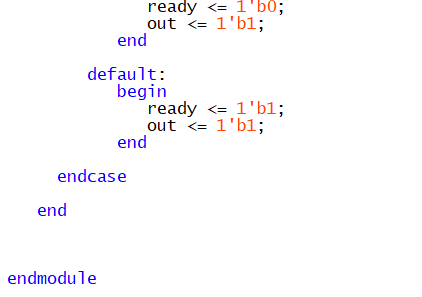
The UART transmitter takes in an 8 bit register for transmission data, a clock, a control signal, and a reset. When control is high, a state machine (diagram in Appendix C) kicks off to output the serial UART output of the data. The ready LED output signal is high when the state machine is in idle, and each bit of UART is output during each transition of the state machine. During the data stage, the register is checked for lowercase letters and converted to uppercase if necessary. Appendix A contains the RTL for this diagram, and Appendix B contains the waveforms from simulation.







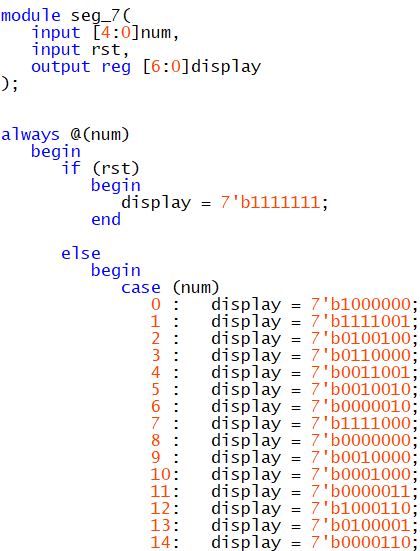


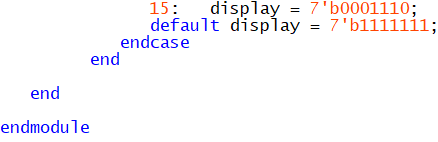


#### 

#### Seven Segment Display–seg\_7.v

The seven segment display module takes in a reset and four bits of data representing a hex number and translates it to a seven bit register output to display the hex number on a seven segment display on the DE1-SoC board, which requires an active low configuration.



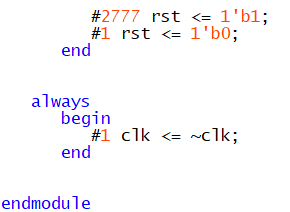
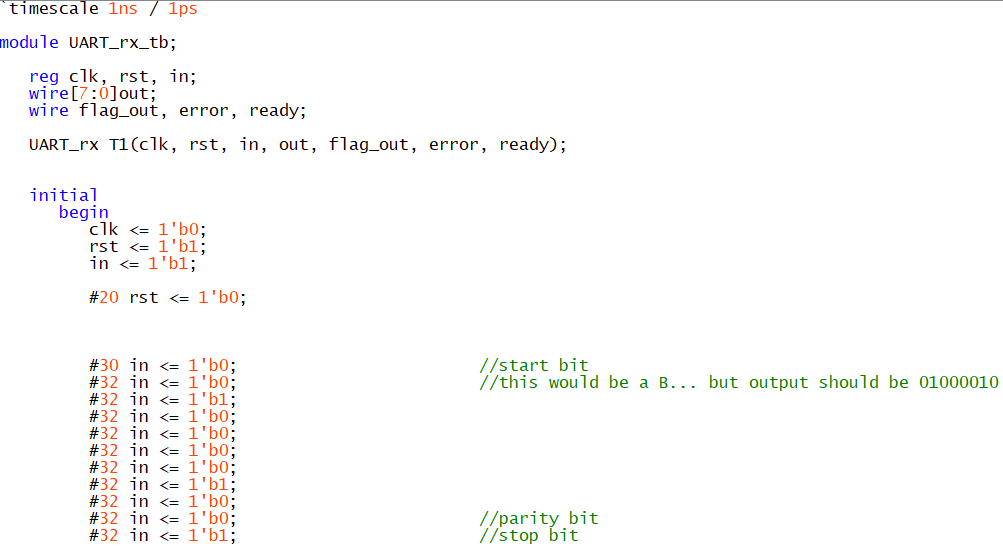


# 

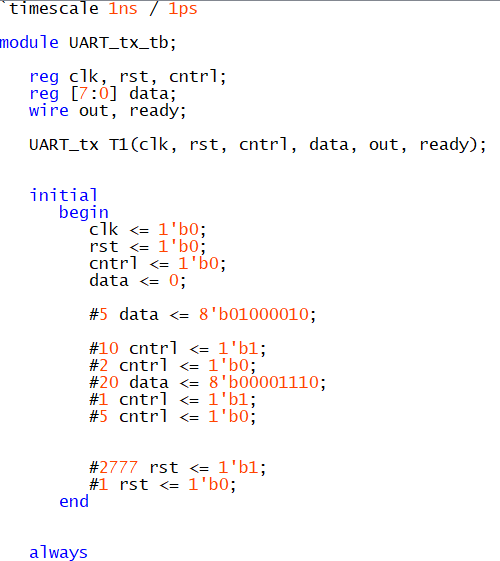
# 

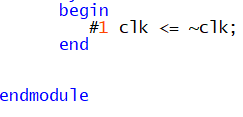
# Simulation and Testing

The test bench for the UART receiver simulates the serial UART transmission of a “B” ASCII character. The test bench is included below, the output waveforms are in Appendix B.



The testbench for the UART transmitter is included below. This emulates the transmission of a “B” and then “00001110.” The waveform outputs of the testbench are included in Appendix B.





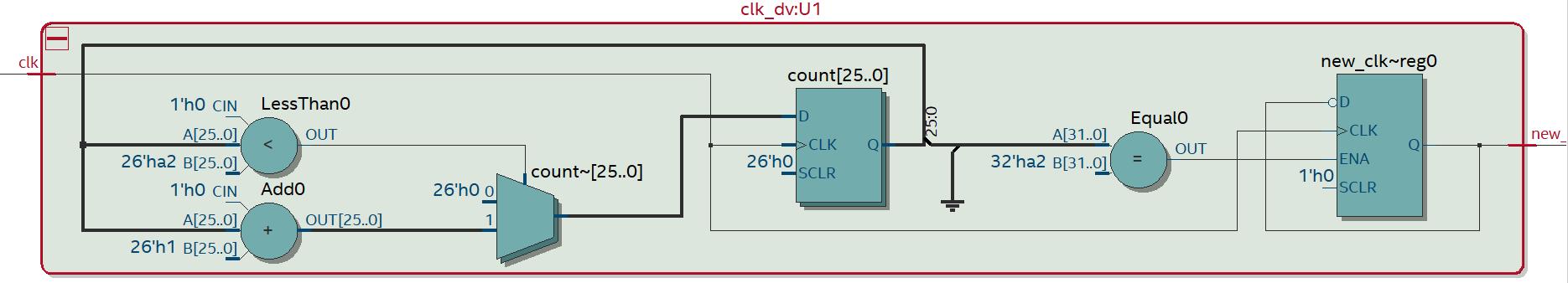
# Problems

One problem when testing the design was incorrect PIN assignments. This is fixed with a careful review of the design, pin assignments, and the DE1-SoC user manuals. One other problem occurred during testing, was the pulse from the receiver, using the fast clock domain, was getting missed by the slower clock domain of the transmitter. Adding and adjusting the flag domain crossing module fixed this error.

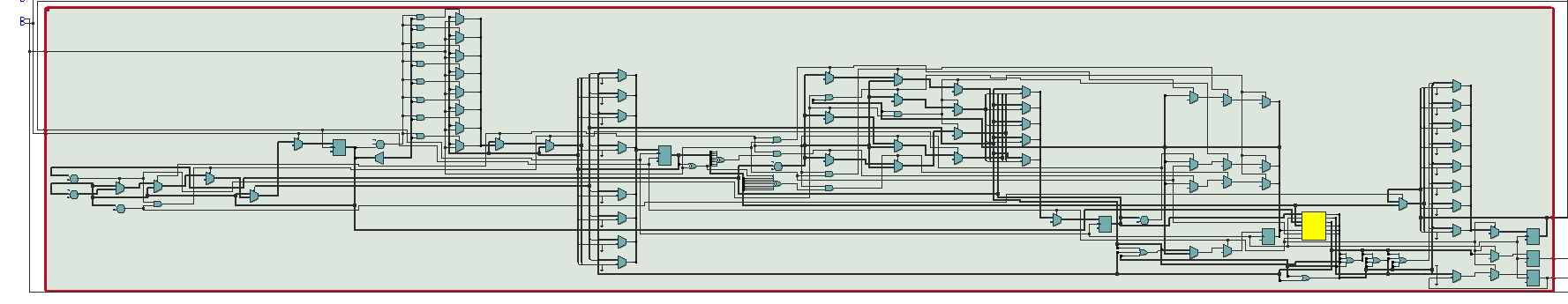
# Results and Conclusion

The result of this lab is a fully operational UART device. It receives and transmits according to UART standards. Clock domain crossing is accomplished–although not perfect, and could be improved. Next time, a more careful examination of the flag domain crossing module and pin assignments will improve the speed and accuracy of the lab.

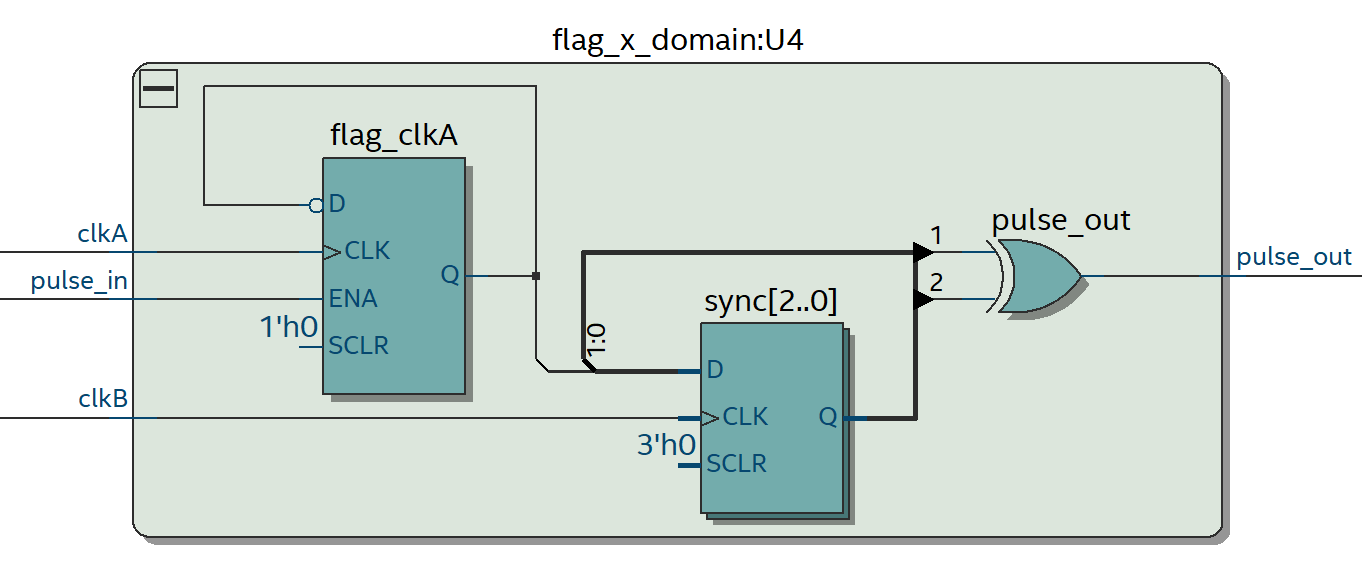
# Appendix A – RTL Diagrams



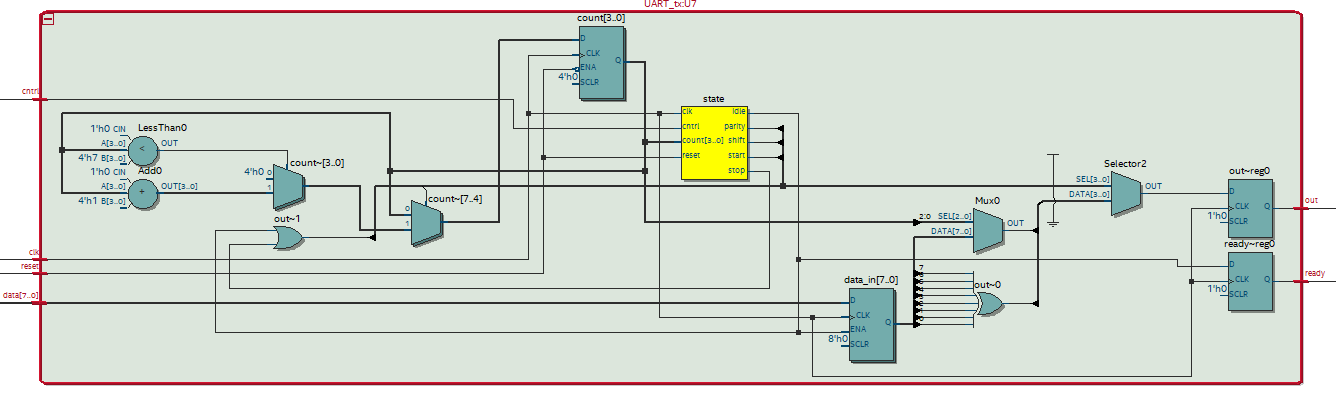
*Figure 3: Clock Divider RTL Diagram*



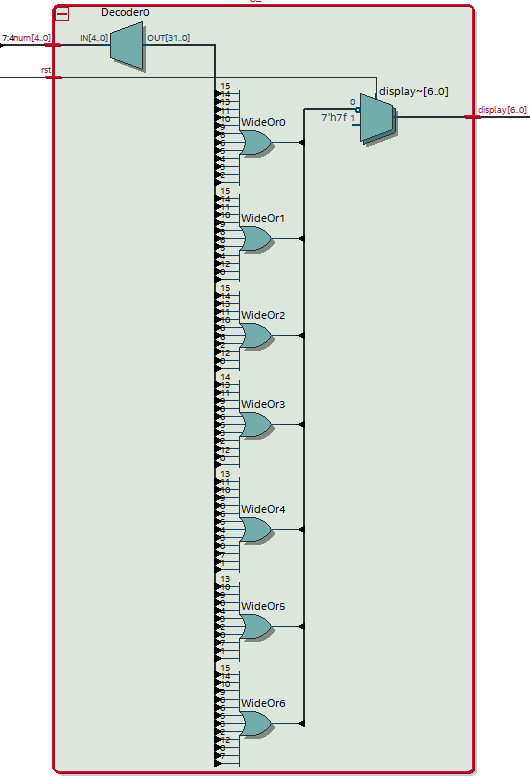
*Figure 4: UART Receiver RTL Diagram*

**

*Figure 5: Flag Domain Crossing RTL Diagram*

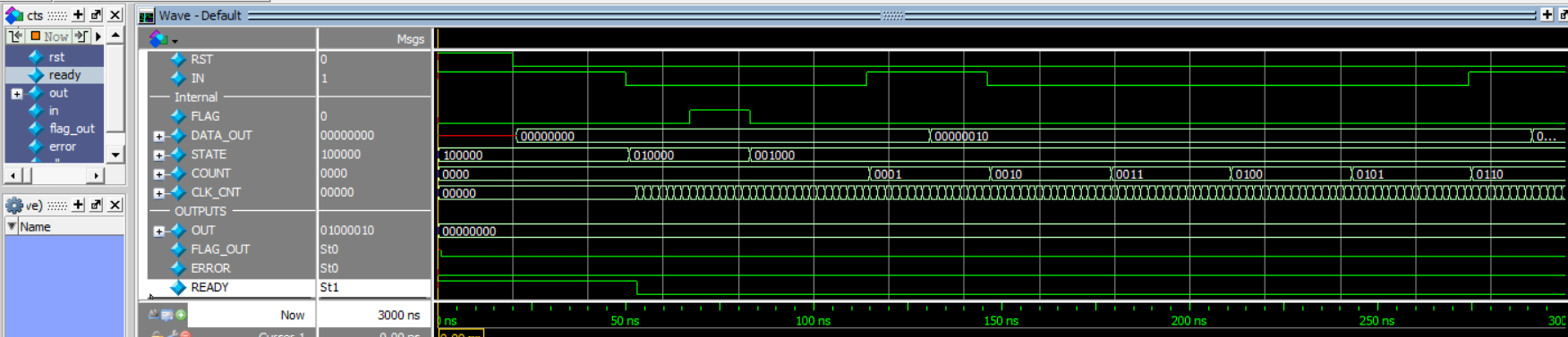
**

*Figure 6: UART Transmitter RTL Diagram*

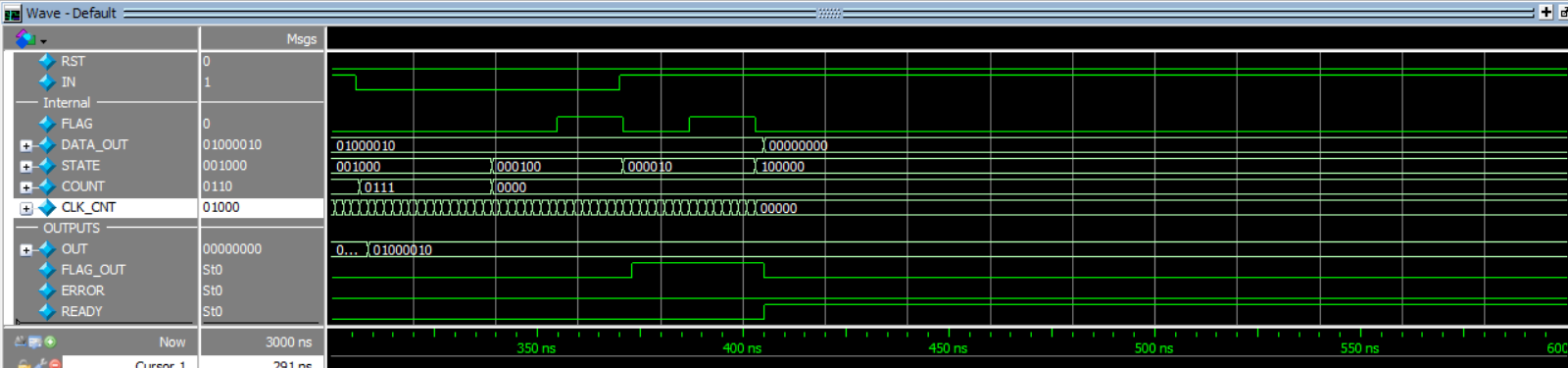


*Figure 7: Seven Segment RTL Diagram*

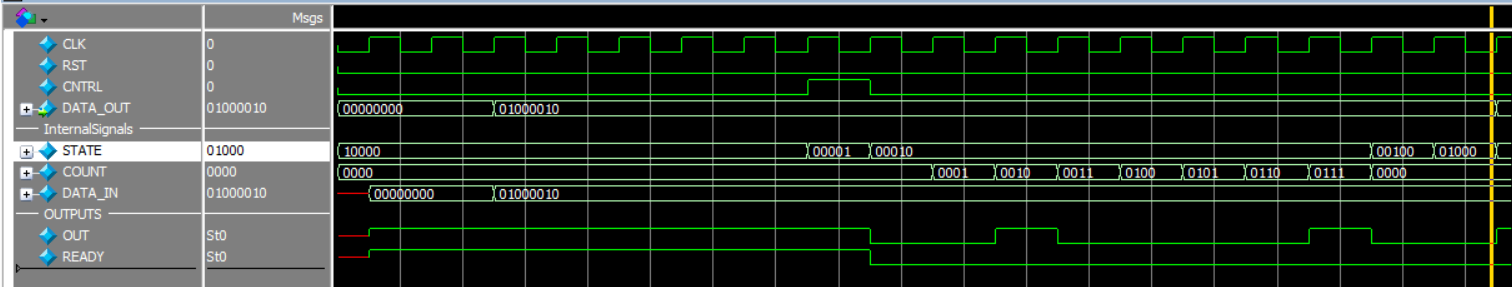
# Appendix B – Waveforms

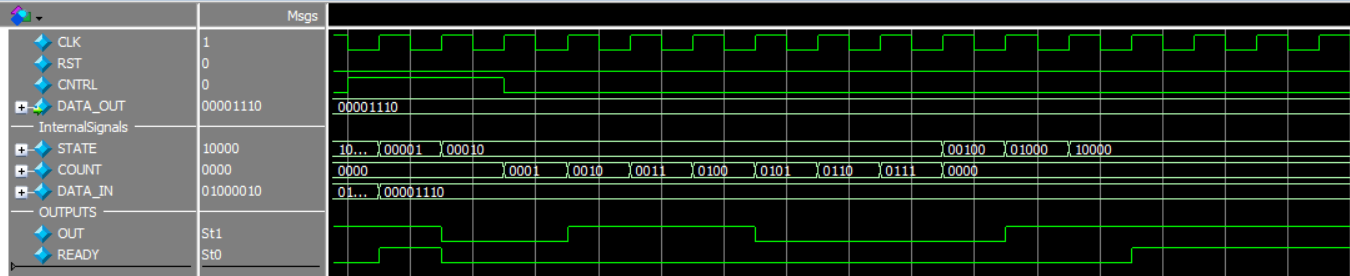


*Figure 8: UART rx testbench waveforms part one*

**

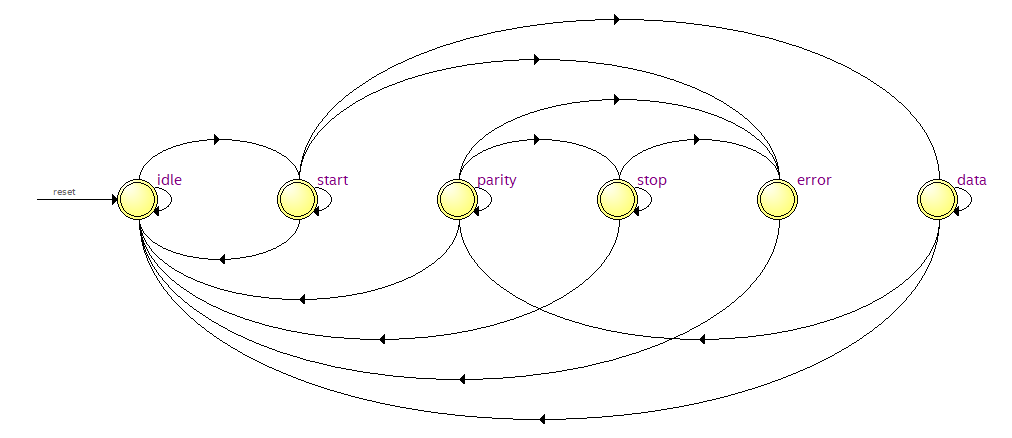
*Figure 9: UART rx testbench waveforms part two*

*Figure 10: UART tx testbench waveforms part one*

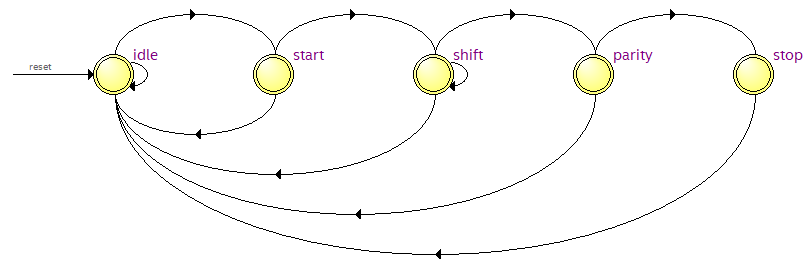
**

*Figure 11: UART tx testbench waveforms part two*

# Appendix C – State Machine Diagrams

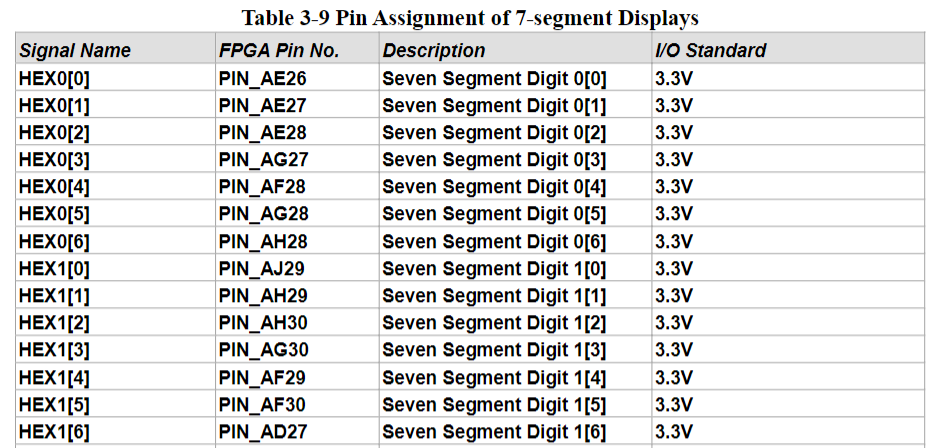


*Figure 12: UART rx State Diagram*

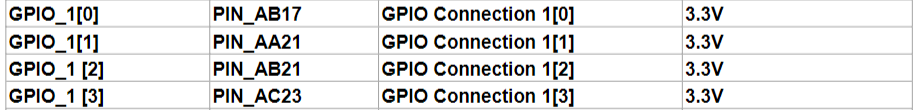


*Figure 13: UART tx State Diagram*

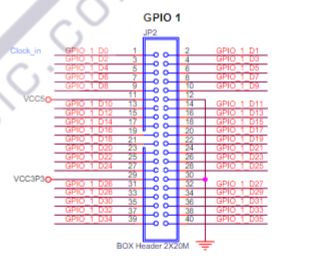
# Appendix D – Pin Tables



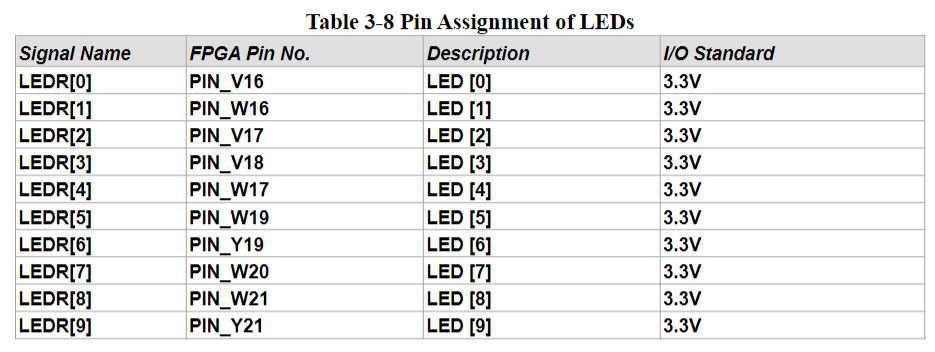
*Figure 14: Seven Segment Display Pin Assignment table from DE1-SoC User Manual*



*Figure 15: Relevant GPIO\_1 Pin Assignment table from DE1-SoC User Manual*



*Figure 16: Pinout Diagram from DE1-SoC User Manual*



*Figure 17: LED Pin Assignment table from DE1-SoC User Manual*

# Table of Figures

[Figure 1: Hardware Connection Diagram](#_3dy6vkm) 3

Figure 2: Hierarchical Block Diagram 4

[Figure 3:](#_3dy6vkm) Clock Divider RTL Diagram16

Figure 4: Hierarchical Block Diagram 16

[Figure](#_3dy6vkm) 5: Flag Domain Crossing RTL Diagram16

Figure 6: UART Transmitter RTL Diagram 17

[Figure 7:](#_3dy6vkm) Seven Segment RTL Diagram 17

Figure 8: UART rx testbench waveforms part one 18

[Figure 9:](#_3dy6vkm) UART rx testbench waveforms part two 18

Figure 10: UART tx testbench waveforms part one 18

[Figure 11:](#_3dy6vkm) UART tx testbench waveforms part two 18

Figure 12: UART rx State Diagram 19

[Figure](#_3dy6vkm) 13: UART tx State Diagram 19

Figure 14: Seven Segment Display Pin Assignment table from DE1-SoC User Manual 20

[Figure 15:](#_3dy6vkm) Relevant GPIO\_1 Pin Assignment table from DE1-SoC User Manual 20

Figure 16: Pinout Diagram from DE1-SoC User Manual 20

[Figure 17:](#_3dy6vkm) LED Pin Assignment table from DE1-SoC User Manual 21

# References

[1] *DE1-SoC User Manual* (2014). Accessed: January 28, 2023. [Online]. Available: https://inst-fs-iad-prod.inscloudgate.net/files/3607afc4-522a-4f44-a5c1-1580f22dd310/DE1-SoC\_User\_manual.pdf?download=1&token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzUxMiJ9.eyJpYXQiOjE2NzQxOTc2NTcsInVzZXJfaWQiOiIxMjgzOTAwMDAwMDAwMzc5MjYiLCJyZXNvdXJjZSI6Ii9maWxlcy8zNjA3YWZjNC01MjJhLTRmNDQtYTVjMS0xNTgwZjIyZGQzMTAvREUxLVNvQ19Vc2VyX21hbnVhbC5wZGYiLCJqdGkiOiI4YzdhYjE1ZC03NTY5LTQ1YmUtYTQ4NC0xNzc4YjZmZjI1N2MiLCJob3N0Ijoib2l0Lmluc3RydWN0dXJlLmNvbSIsIm9yaWdpbmFsX3VybCI6Imh0dHBzOi8vYTEyODM5LTMwMjY0ODguY2x1c3RlcjkxLmNhbnZhcy11c2VyLWNvbnRlbnQuY29tL2NvdXJzZXMvMTI4Mzl-MTc2OTUvZmlsZXMvMTI4Mzl-.rAUBUNddvg1bZIEEdqQl46VLR4ma64u9MRfpNPFygfuL28WGrF8huwyLI-6aA0d8GSLMLjCmXed53\_VN1RCaog

[2] *ALTERA Cyclone V SoC Development & Education Board (DE1-SoC)* (2014). Accessed: January 28, 2023. [Online]. Available: https://oit.instructure.com/courses/17695/files/3026486/download?download\_frd=1