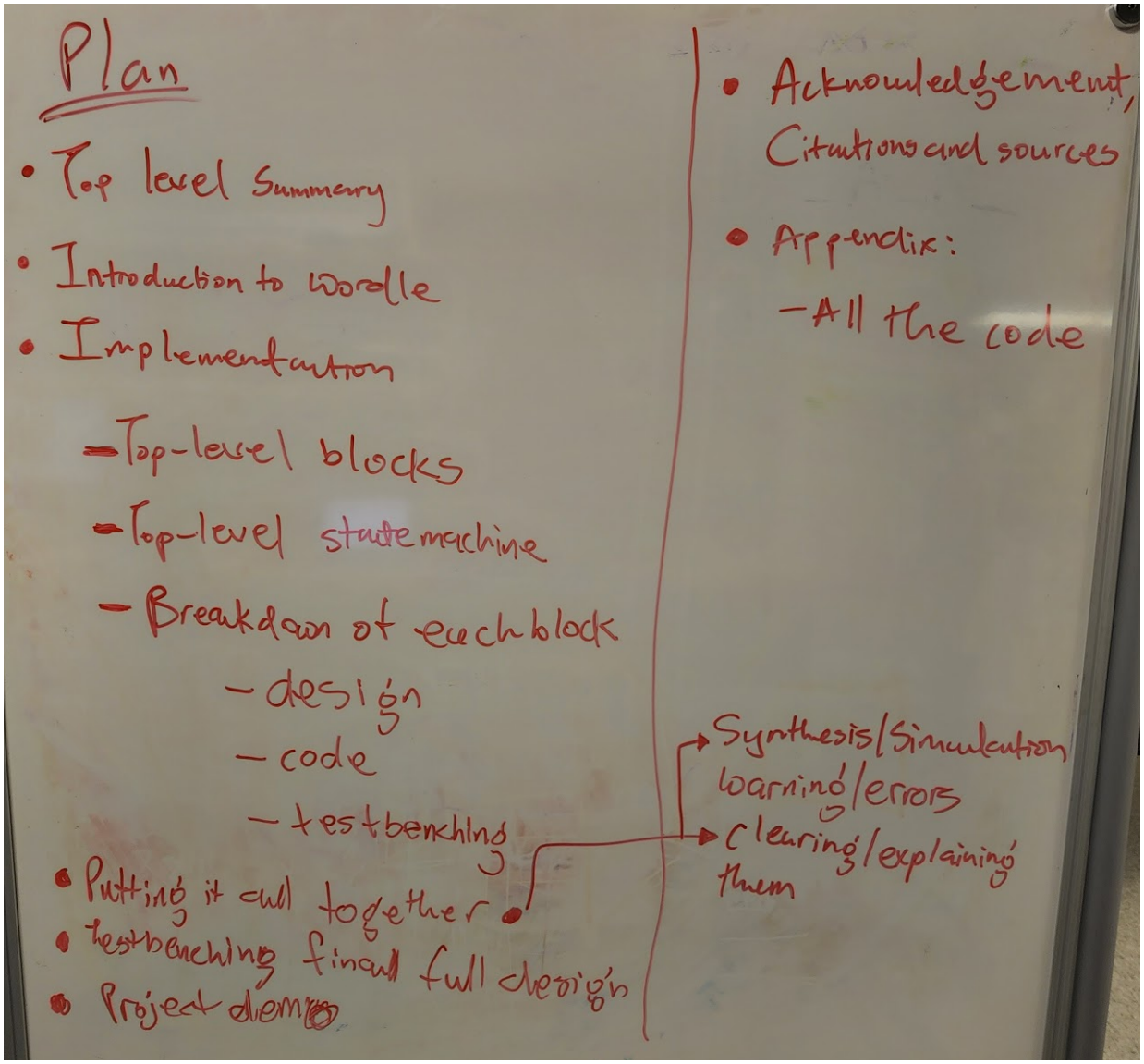
Final Digital Electronics Engineering Project: WORDLE REPLICA

Ikeoluwa Abioye, Lobna Jbeniani, Mubarak Idoko

May 2022

REPORT PLAN:


Plan
• Top level Summary
• Introduction to Wordle
• Implementation
- Top-level blocks
- Top-level statemachine
- Breakdown of each block
- design
- code
- testbenching
• Putting it all together
• testbenching final full design
• Project demo
• Acknowledgement, Citations and sources
• Appendix:
- All the code
Synthesis/Simulation warning/errors
clearing/explaining them

Project Summary

In the past year, WORDLE has become one of the most played online games, with over 300,000 users in the first two months of its launch and over 2,000,000 users today. Given that WORDLE started out with less than 90 players in the fall of 2021, the fast-paced growth of the game has incentivized many to recreate it in one way or another. In this final report, our group describes how we built a WORDLE clone using problem-solving techniques that we learned in ENGS31: Digital Electronics.

We will describe our design process and how we eventually built the WORDLE replica using high-level block diagrams, Register-Transfer Level (RTL) diagrams, a top-level Finite State Machine (FSM), and eventually, VHDL code and Vivado Simulation. In addition to this, we have a demonstration video to show our version of WORDLE working in hardware.

Table of Contents

Project Summary **3**

Introduction **5**

Methods **7**

Methodology Description 7

Deconstructing the problem: High Level Block Diagram 7

Functionality: RTL Diagrams 7

Designing the Process: The Finite State Machine 7

Building and Connecting the Dictionary: ROM/RAM 7

Block Diagram 8

Choice of SCI: 8

The SCI Rx: 8

The SCI Tx: 8

The Game Control Unit: 8

VHDL Code Design and Test Benching: Noteworthy Errors **20**

Results **20**

Check Letter Simulation 20

Load Word Simulation 20

Receiver 21

Receiver Transmitter 22

Check Uppercase Letter (Receiver Transmitter) 23

Word Exists 23

WORDLE Dictionary ROM 24

Conclusion **24**

Acknowledgements **24**

Appendix **25**

VHDL Code Testing **76**

FSM Testbench **95**

WORDLE ROM Testbench **104**

Introduction

First designed and engineered by Josh Wardle, WORDLE has become a household name in a short amount of time. In this paper, we report on how we replicated WORDLE using the concepts and tools of ENGS31:Digital Electronics.

To better understand this paper, it's critical to revisit the rules of the game. In short, the game allows the user 6 trials to guess a 5 word letter. The entered words have to be in the dictionary, which contains over 11,500 words or else the program will not accept them. When a user inputs a word, the program first ensures that each of the entered characters is a letter, then once the user has finished typing all 5 characters and hits the ENTER key, the program checks if the word is in the WORDLE dictionary. At the next stage, the program evaluates each character individually. If the letter is in the correct spot, the program highlights in green. If the letter is in the word but not in the correct spot, the program highlights it in yellow. If the letter is not in the word at all, the program highlights it in gray. The user uses these instructions to inform his or her decision on the next word choice. This process repeats 6 times for each entered word. Two other considerations are that letters can be used more than once (e.g., fleet, green, abbey..etc) and answers are never plural.

In this replica, we uphold the above principles of the game, but given our current display, we have slightly altered the coloring of the letters into a display of ? and ! characters. In essence, when a user inputs a letter that exists in the answer but they input it in the incorrect spot, they see a ! at that spot. When they input a letter that does not exist in the answer word at all, they see a ? at that spot. When the user inputs a letter that exists in the answer word at the correct spot which matches its position in the answer word, they will see a space under that letter.

Figure 1: A Visual Demonstration of the User Interface Differences between WORDLE and WORDLE REPLICA

Step 1: The user gets 6 guesses, both in the original game and our replica.

_ _ _ _ _

_ _ _ _ _

_ _ _ _ _

_ _ _ _ _

_ _ _ _ _

_ _ _ _ _

Step 2: The user types a word—in our replica, the user hits ENTER right after entering the word.

D R I N K

_ _ _ _ _

_ _ _ _ _

_ _ _ _ _

_ _ _ _ _

_ _ _ _ _

Step 3: The word gets evaluated (below is a comparison of what the user might see in the actual game versus our replica).

D R I N K	D R I N K
_ _ _ _ _	_ ? ! ? ?
_ _ _ _ _	_ _ _ _ _
_ _ _ _ _	_ _ _ _ _
_ _ _ _ _	_ _ _ _ _
_ _ _ _ _	_ _ _ _ _

Methods

I. Methodology Description

1. Deconstructing the problem: High Level Block Diagram

We started out by designing a high level block diagram, which allowed us to visualize the different functional blocks of the program. The diagram consisted of three main functional blocks:

a. The Serial Communications Interface (SCI) Receiver
b. The SCI Transmitter:
c. The Game Controller: The game controller is the block that runs the game. In other words, now that the program has received user input via the SCI Receiver and is ready to output data via the SCI Transmitter,the only functional block left to think about is the one controlling the game itself. Within this parent block, we find 5 contained blocks: Word Logic, Check Dictionary, The Dictionary ROM, the Correct Word Check, and the Finite State Machine (FSM).

2. Functionality: RTL Diagrams

3. Designing the Process: The Finite State Machine

4. Building and Connecting the Dictionary: ROM/RAM

We accomplished this process by first converting the WORDLE dictionary, which contains 12,972 words, into ASCII binary code. Building and connecting the WORDLE dictionary was done through the following steps:

-Importing the dictionary onto an EXCEL file

-Converting all letters in the file into lower case letters

-Splitting the rows of words into 5 column, each containing one letter

-Converting each of the letters into ASCII decimal numbers (e.g., a is 97, b is 98..etc). Since everything was lower case, the numbers were between 97 and 122

-Converting each decimal number into binary code. This gave us 40 bits of data for each word

-Concatenating the columns to form one single column containing all 40 bits for each word

-Uploading the rows of binary data into a COE file

At a later stage, we uploaded this dictionary onto our VIVADO working file as a design source.

II. Block Diagram

A. Choice of SCI:

When thinking about the game at a higher level, we found that the best way to implement it was through a Serial Communications Interface (SCI) which would facilitate the exchange of data between the user input, the embedded system, and the program output.

B. The SCI Rx:

At an initial stage of our work, we designed the SCI Rx as a block that takes in user input in the form of ASCII binary code and transmits it to the embedded Game Control system. The input to the Game Control is in the form of Rx_data, which carries the actual data associated with each typed word, and Rx_done, which carries the signal indicating that the user has finished entering the data for the word.

C. The SCI Tx:

With an inverse function to the receiver, the SCI Transmitter takes the output of the embedded Game Control system and outputs it to the user in the form of instructions or feedback on the choice of the characters, letters, and eventually the whole word. The SCI Tx accomplishes this task through two main inputs to its block coming from the Game Control: Tx_data, which carries the actual data associated with the output character (recall ! and ? and space).

D. The Game Control Unit:

Having finalized our choice for an appropriate exchange channel between the different blocks, we decided to look at the embedded game control system, which would first ensure that the user is interacting with the game interface using letter inputs and later on enforce the specific rules of the game through displayed commands to the user. This general understanding of what the game control unit should be able to accomplish led our team to break it into 5 major blocks.

- **Word Logic:** At a high level, the purpose of this block is to check the user input and ensure that the character he or she typed is a letter. To check if it's a letter, we look at the alphabet ASCII code. For lower case letters, the ASCII code is from 97 to 122, with 97 characterizing a and 122 characterizing z. For the uppercase letters, the ASCII code is from 65 to 90, with 65 characterizing A and 90 characterizing Z. In order to best work with ASCII codes, we converted all of the letters in our dictionary into lowercase letters. At this stage, we also ensured to convert the user input into lowercase letters by converting the ASCII number into lowercase (essentially adding 32 if the number is between 65 and 90 inclusive).

Table 1: ASCII Decimal Codes for Lowercase and Uppercase Letters

Lowercase	ASCII Decimal	Uppercase	ASCII Decimal
a	97	A	65
b	98	B	66
c	99	C	67
d	100	D	68
e	101	E	69
f	102	F	70
g	103	G	71
h	104	H	72
i	105	I	73
j	106	J	74
k	107	K	75
l	108	L	76
m	109	M	77
n	110	N	78
o	111	O	79
p	112	P	80
q	113	Q	81
r	114	R	82
s	115	S	83
t	116	T	84
u	117	U	85
v	118	V	86
w	119	W	87
x	120	X	88
y	121	Y	89
z	122	Z	90

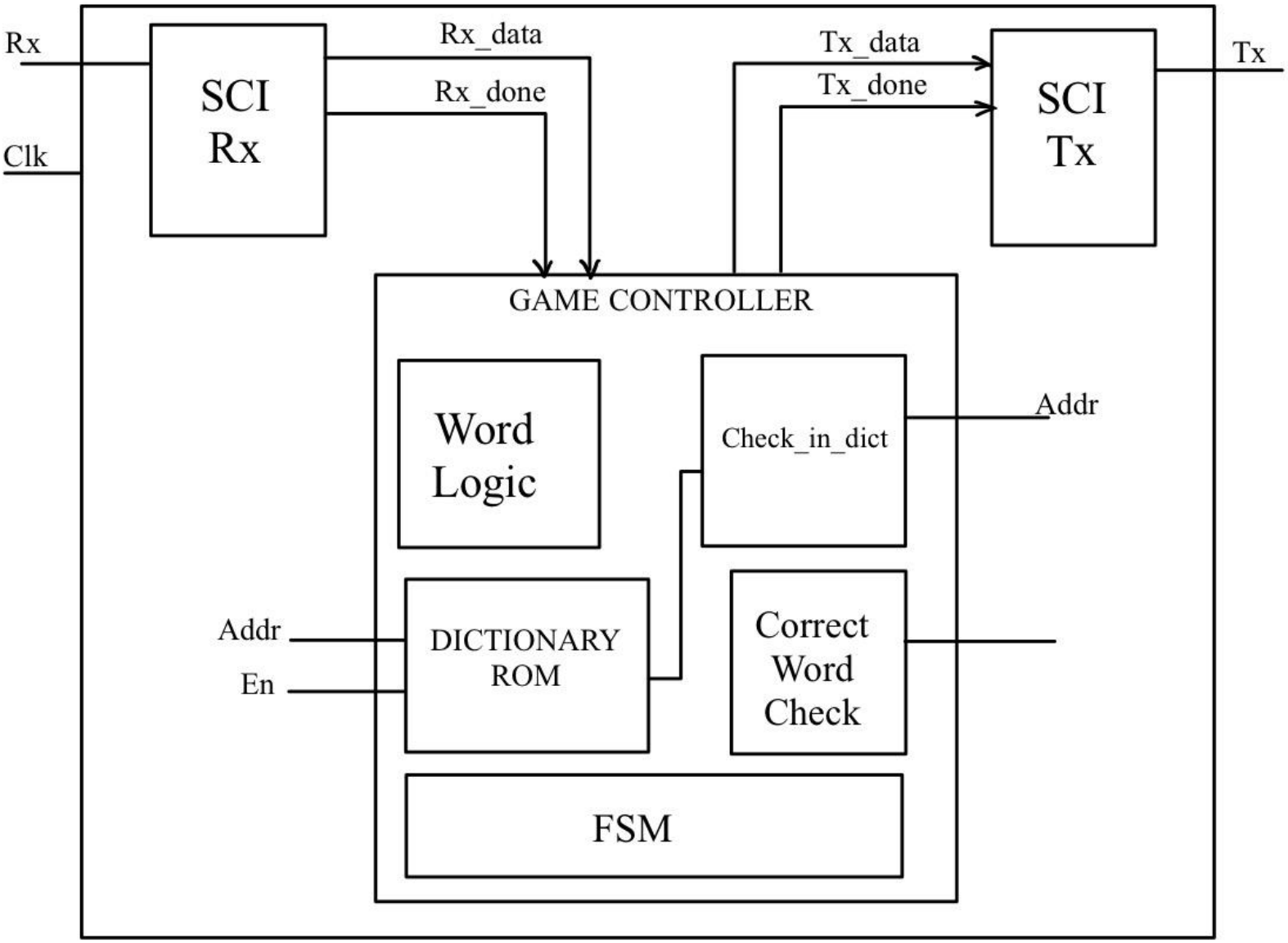
- **Check in Dictionary:** The purpose of this block is to check that the input word (transmitted after the user hits ENTER) exists in the WORDLE dictionary.

- **Dictionary ROM:** The purpose of this block is to hold read only memory (ROM) of all 12,973 WORDLE words in ASCII binary code. Each letter consists of 8 bits of code, and each word consists of 40 total bits.

- **Correct Word Logic:** This block compares the entered word guess to the answer word in the dictionary by evaluating each letter of the guess word and comparing it to letters in the answer word. This block outputs signals that indicate whether or not a letter exists within the answer word (recall ?) is in the correct spot (recall !). This block also outputs whether or not the guess word matches the answer to signal the user's win at any number of trials or loss after 6 unsuccessful trials.

- **The Finite State Machine:** This block in our diagram essentially works as a controller for the GAME CONTROL UNIT. It pulls the previous processes together by initiating a set of states and conditioning transitions between them.

Figure 2: Initial Workflow Block Diagram of WORDLE REPLICA



III. Register Transfer Level (RTL) Diagrams

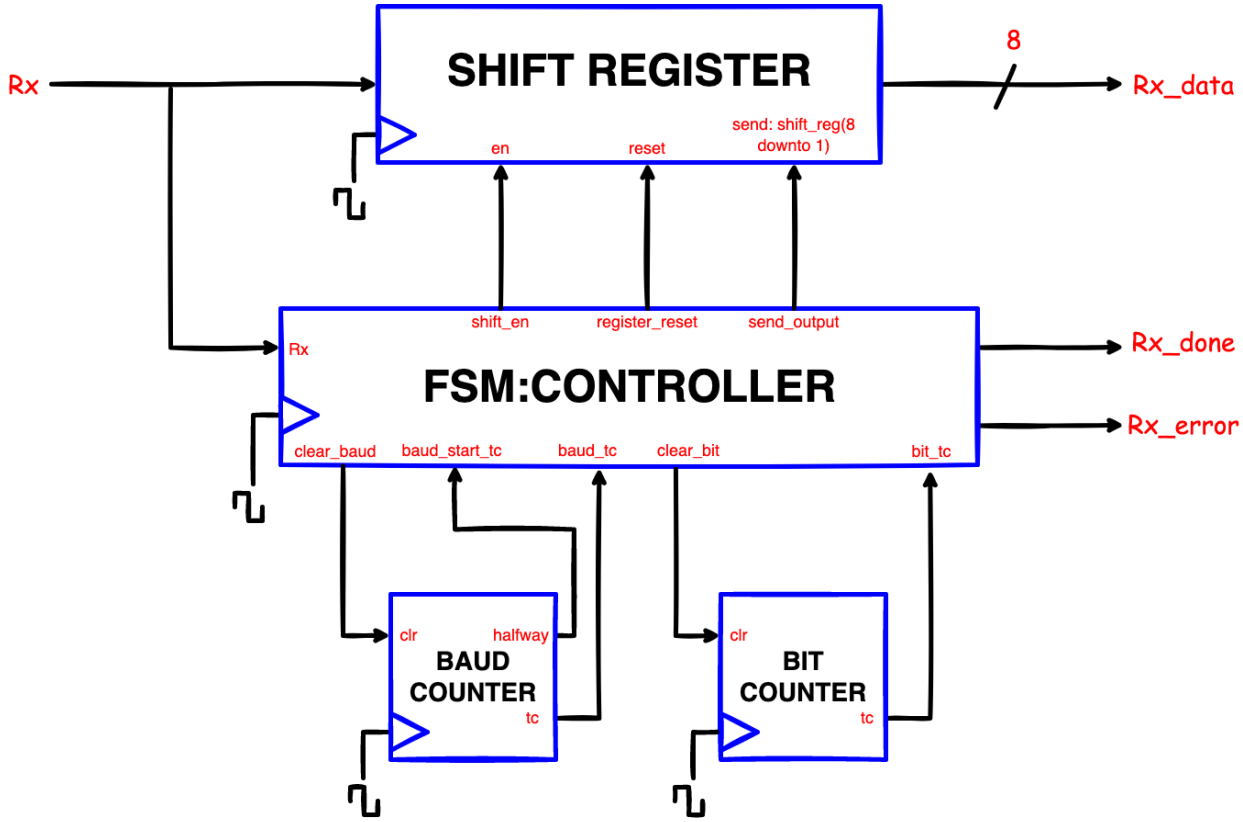
As we moved one more step toward game implementation, it became necessary that we translate our current block chart into a complete and well connected Register Transfer Level (RTL) diagram.

A. The Shift Register and FSM Controller

The shift register is connected to the FSM through 3 main channels. 1) When the FSM output shift_en is high, it enables the shift register (en goes high). 2) when the FSM output register_reset goes high, it resets the shift register (reset goes high). When send_output goes high, the send_shift_reg on the register goes high.

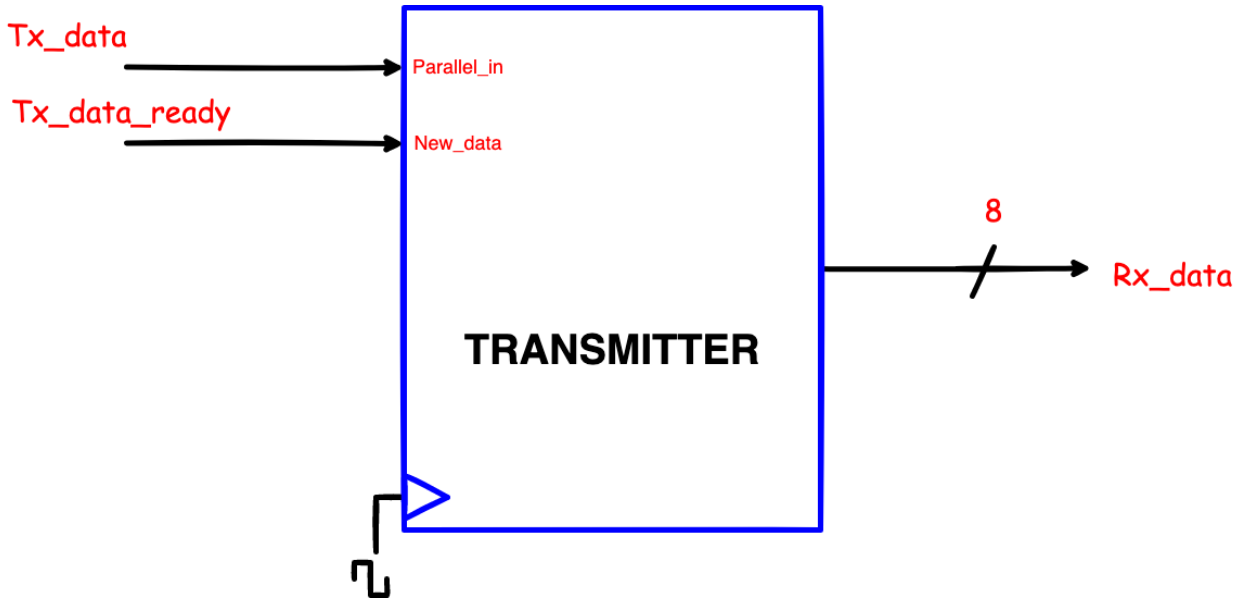
Both the shift register and the FSM receive Rx and a clk signal as inputs. In the shift register, the output is Rx_data, an 8 bit output signal carrying the received letter in binary code. The FSM is connected to both a BAUD COUNTER and a BIT COUNTER. Both counters receive a clk signal and emit baud_tc and bit_tc signals respectively.

The way it all works together is the following: The shift register and the FSM both receive the same Rx signal. Once enabled through the FSM, the register outputs Rx_data, a vector of 8 bits characterizing a letter. The FSM also sends consecutive output to the shift register, indicating that a new bit has been counted. The shift register receives this through the shift_reg, and is able to output 8 bits of data at a time. The FSM input signal from the bit counter is what allows it to recognize if the bit terminal count has been reached. Once it has, the bit counter is cleared to start counting all over again. The FSM also outputs Rx_done, which shows that the user input has been processed, and a letter has been entered, and Rx_error, which indicates that the user has not entered a letter.

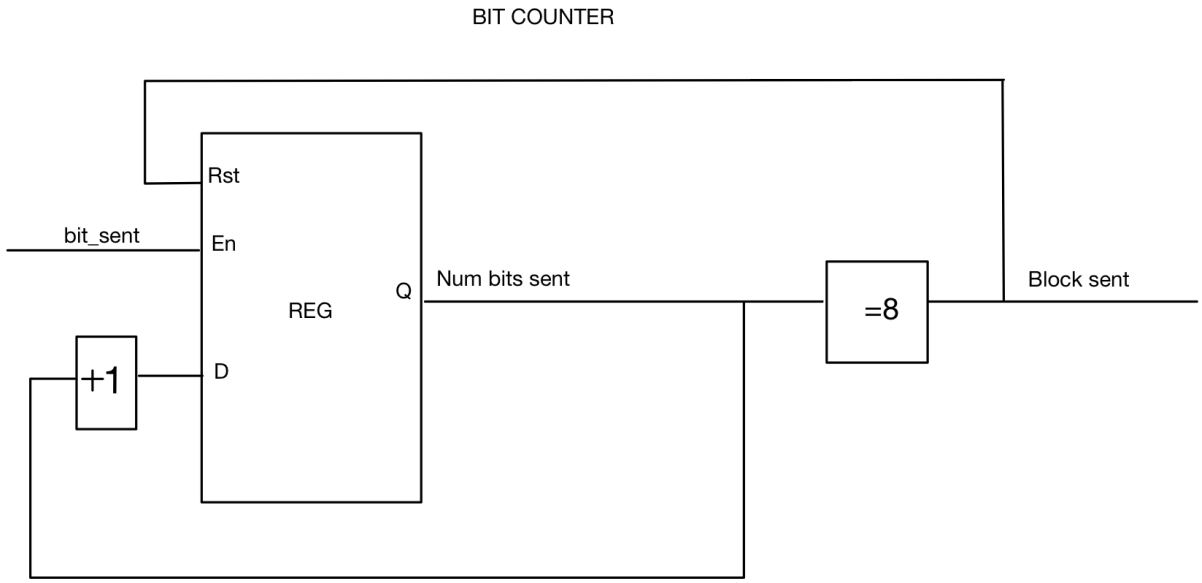


B. The Transmitter

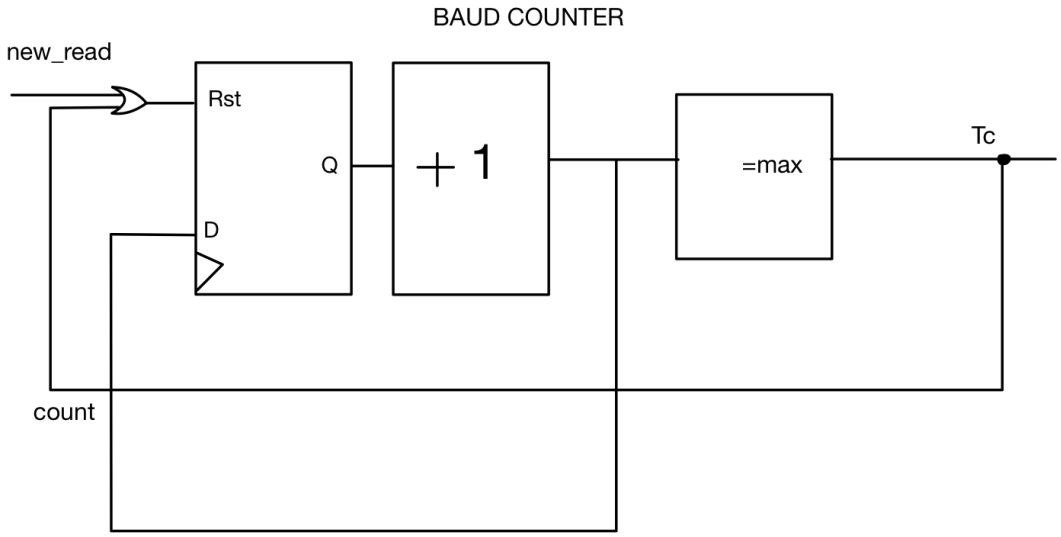
The transmitter receives Tx_data and Tx_data_ready, which are the outputs of the Game Control Unit. More specifically,



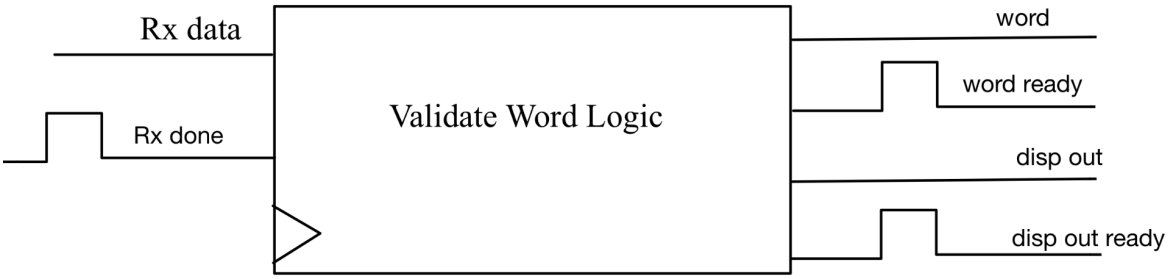
C. Bit Counter



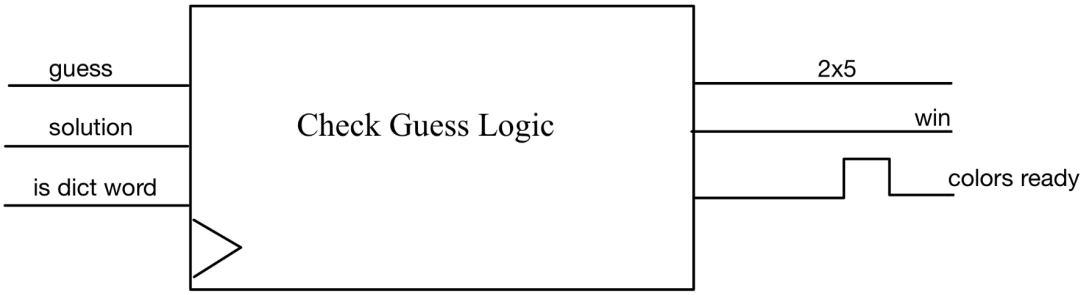
D. Baud Counter



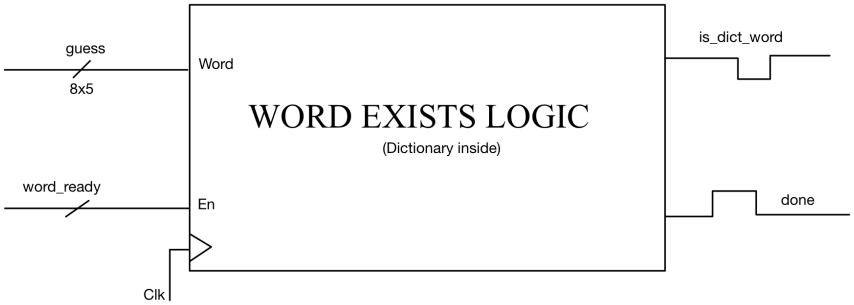
E. Validate Word Logic



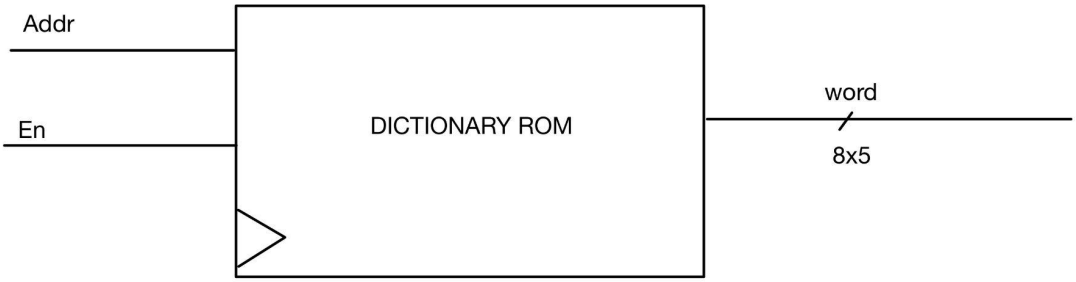
F. Check Guess Logic



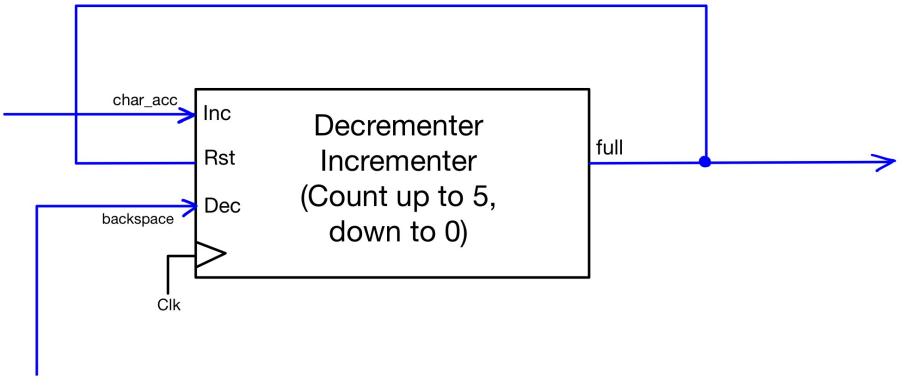
G. Word Exists Logic



H. Dictionary ROM



I. The Letter Counting Decrementer Incrementer



J. The Word Counting Incrementer

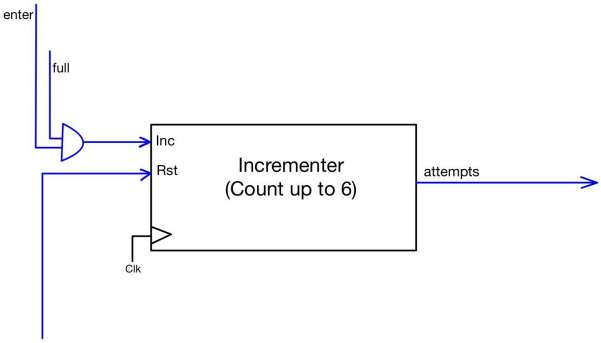


Table 2: RTL Signal Description

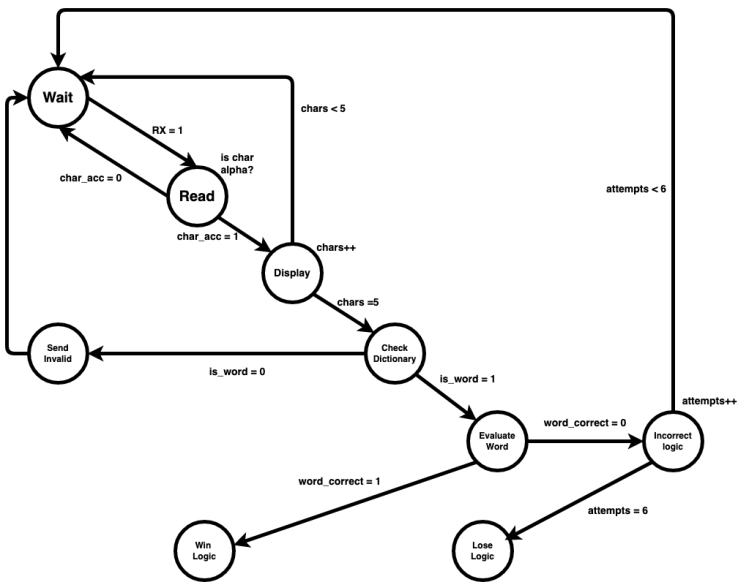
Signal	**Status**	**Number of Bits**	**Associated Blocks**	**Description**
Rx	input	8	Input to shift register	The Rx represents the input bits from the user
Rx_data	output	8	Output from shift register	
Rx_done		1		
Rx_error		1		
Tx_data		8		
Tx_data_ready		1		
char_acc		1		
backspace		1		
full		1		
Clk	input	1	Input to system	

IV. Finite State Machine (FSM)

As we continued to work on the project, the FSM gradually evolved.

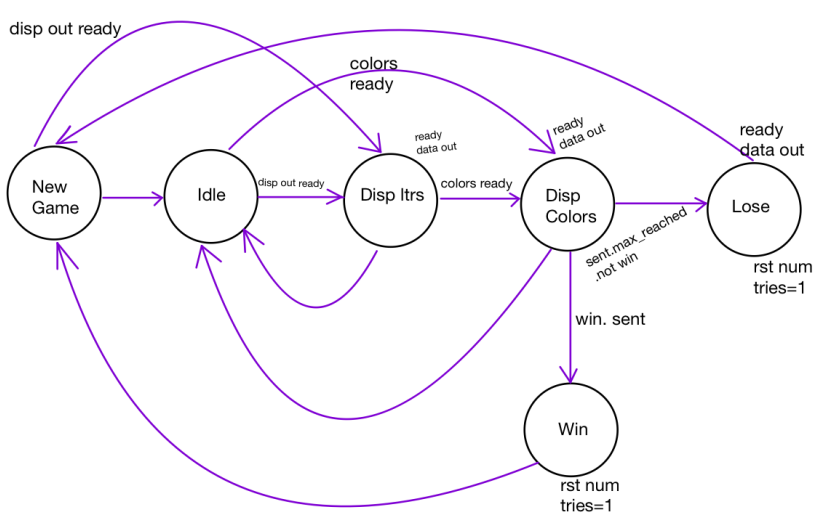
Initial Model

Our first model focused more on the initial stages of the game, including the reading, display and checking if a word is in the dictionary.



Final FSM Model

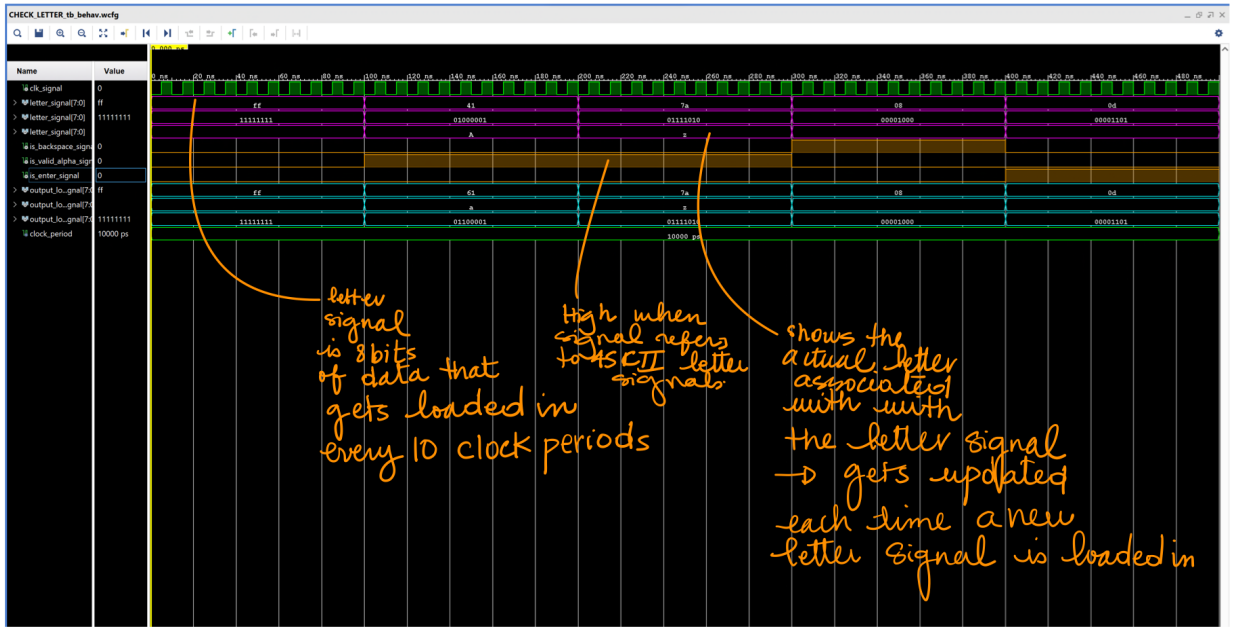
Our second and final model combined properties from the past two models, which allows us to have more of a comprehensive process without getting lost in states that are unnecessary.



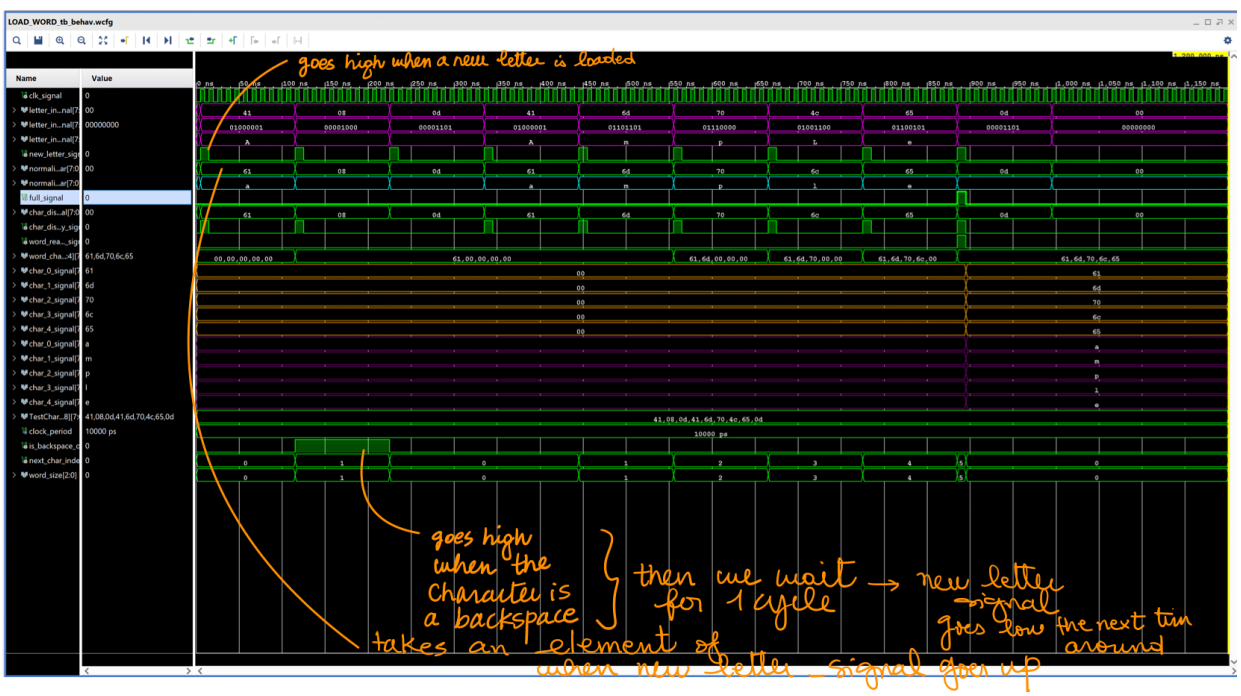
V. VHDL Code Design and Test Benching: Noteworthy Errors

VI. Results

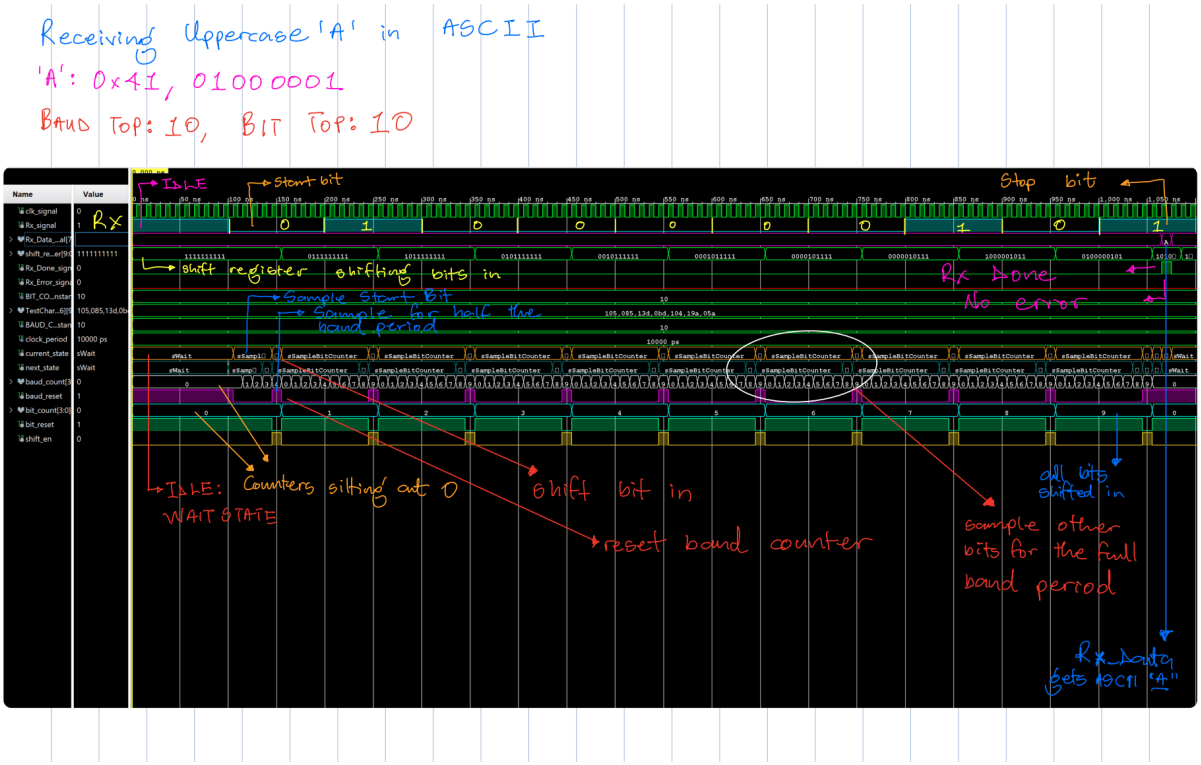
A. Check Letter Simulation

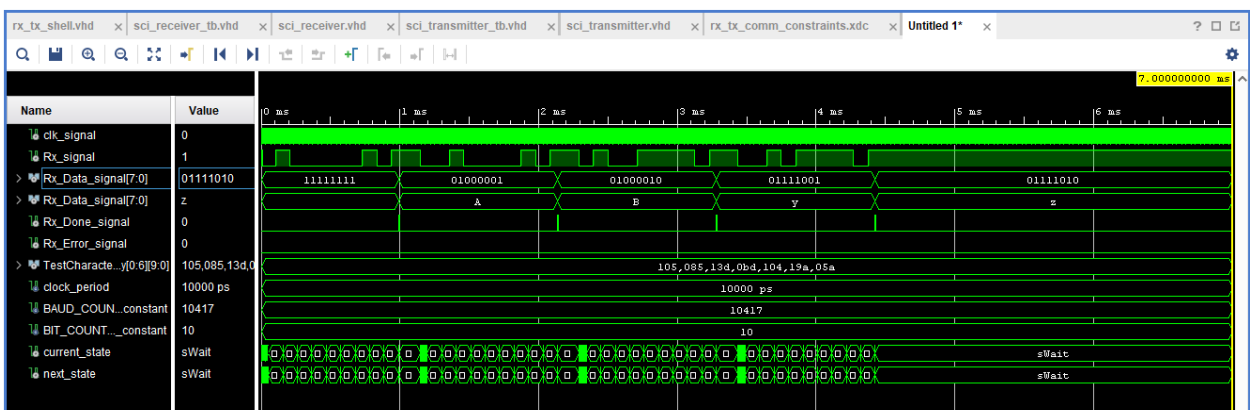


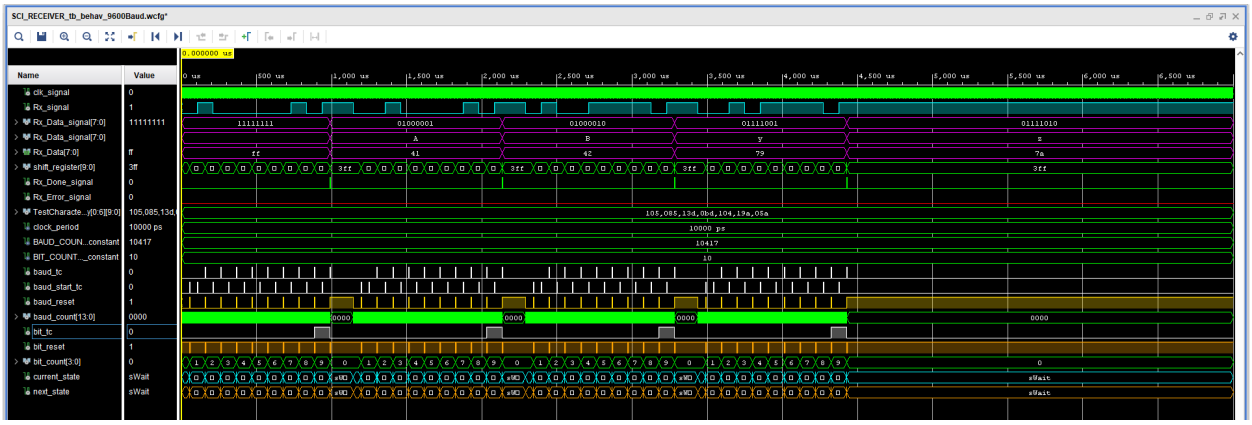
B. Load Word Simulation

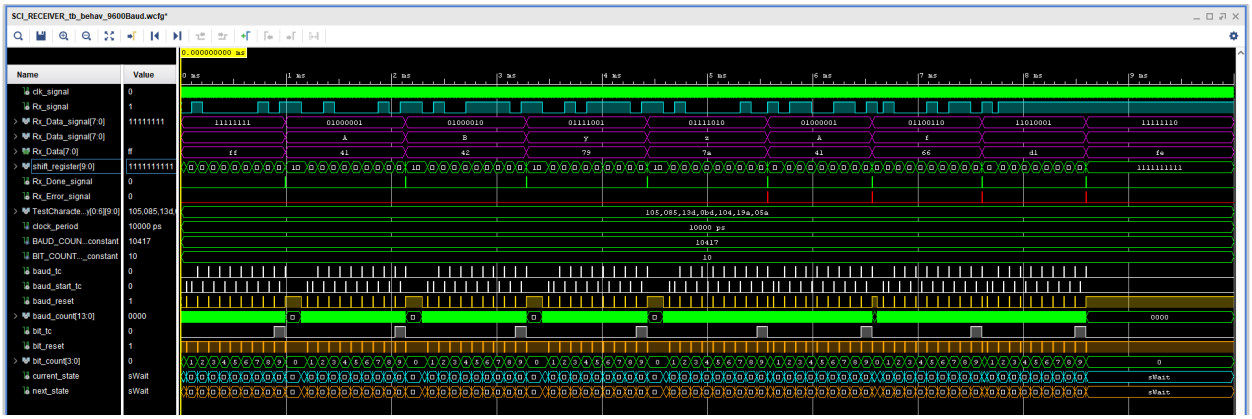


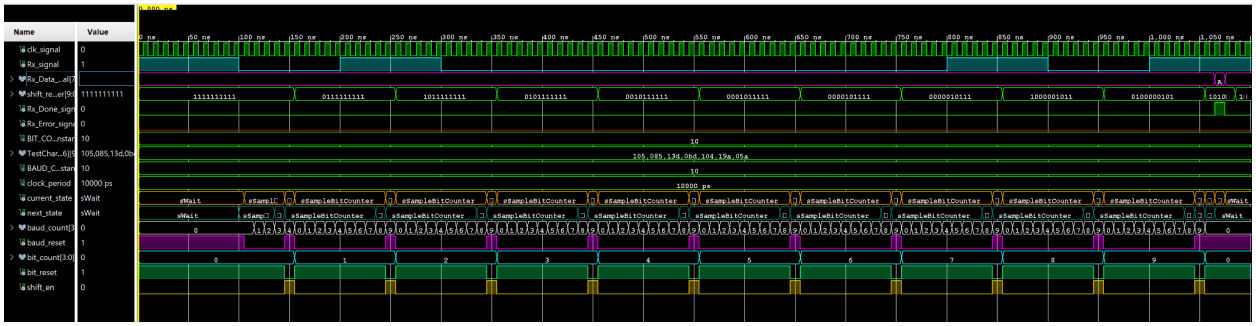
C. Receiver


Receiving Uppercase 'A' in ASCII
'A': 0x41, 01000001
BAUD TOP: 10, BIT TOP: 10
Start bit
Stop bit
IDLE
Rx
shift register shifting bits in
Sample Start Bit
Sample for half the baud period
Rx Done
No error
IDLE: Counters sitting at 0
WAIT STATE
shift bit in
reset baud counter
all bits shifted in
sample other bits for the full baud period
Rx_data gets 0x41 "A"

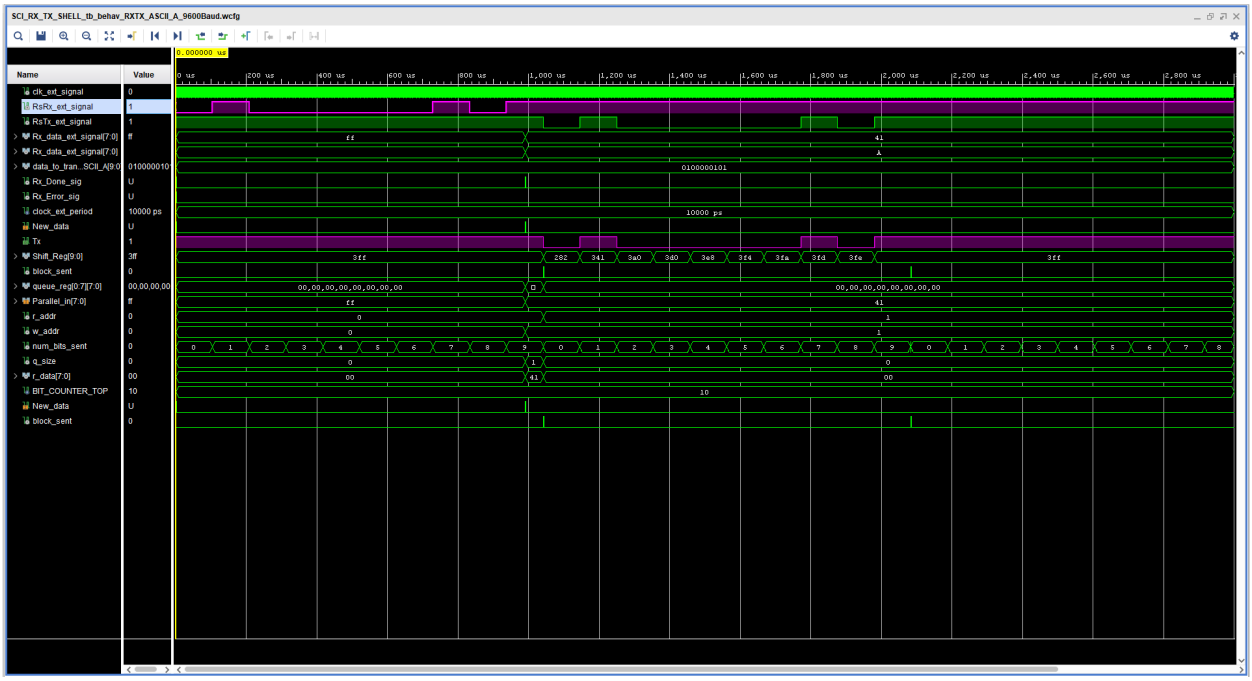

rx_tx_shell.vhd
sci_receiver_tb.vhd
sci_receiver.vhd
sci_transmitter_tb.vhd
sci_transmitter.vhd
rx_tx_comm_constraints.xdc
Untitled 1*
Name
Value
clk_signal 0
Rx_signal 1
Rx_Data_signal[7:0] 01111010
Rx_Data_signal[7:0] z
Rx_Done_signal 0
Rx_Error_signal 0
TestCharacte...y[0:6][9:0] 105,085,13d,0
clock_period 10000 ps
BAUD_COUN...constant 10417
BIT_COUNT..._constant 10
current_state sWait
next_state sWait
7.000000000 ms
0 ms
1 ms
2 ms
3 ms
4 ms
5 ms
6 ms
11111111
01000001
01000010
01111001
01111010
A
B
y
z
105,085,13d,0bd,104,19a,05a
10000 ps
10417
10
sWait


SCI_RECEIVER_tb_behav_9600Baud.wcfg*
Name
Value
clk_signal
Rx_signal
Rx_Data_signal[7:0]
Rx_Data_signal[7:0]
Rx_Data[7:0]
shift_register[9:0]
Rx_Done_signal
Rx_Error_signal
TestCharacte...y[0:6][9:0]
clock_period
BAUD_COUN...constant
BIT_COUNT..._constant
baud_tc
baud_start_tc
baud_reset
baud_count[13:0]
bit_tc
bit_reset
bit_count[3:0]
current_state
next_state

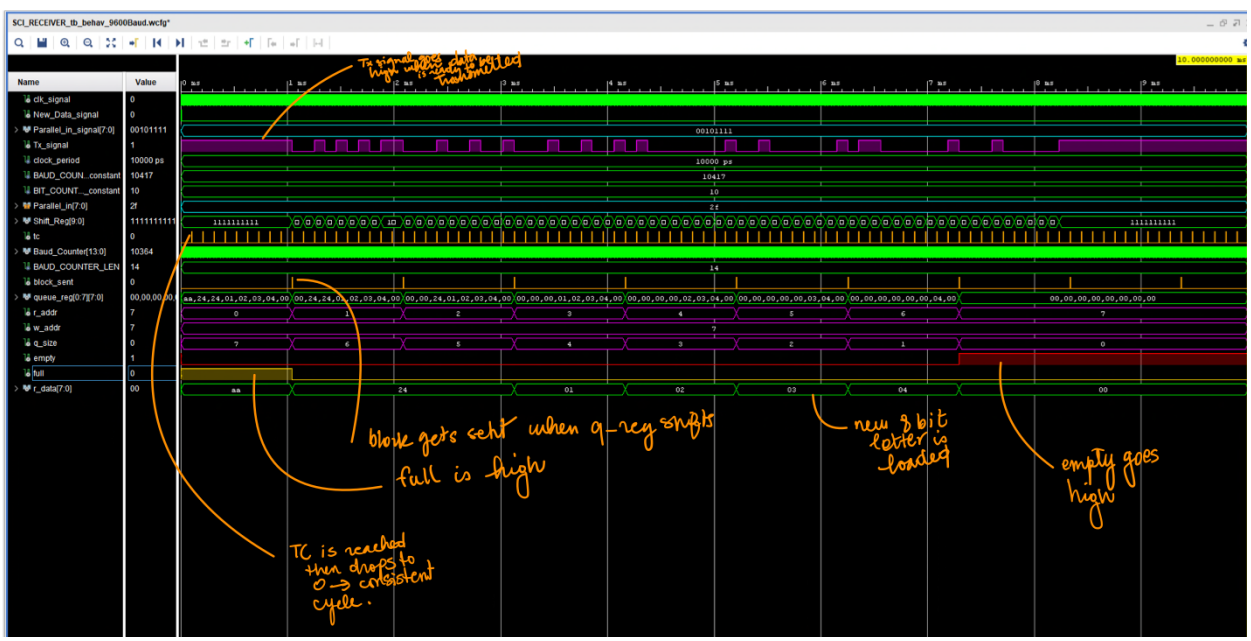

SCI_RECEIVER_tb_behav_9600Baud.wcfg*
Name
Value
clk_signal
Rx_signal
Rx_Data_signal[7:0]
Rx_Data_signal[7:0]
Rx_Data[7:0]
shift_register[9:0]
Rx_Done_signal
Rx_Error_signal
TestCharacte...y[0:6][9:0]
clock_period
BAUD_COUN...constant
BIT_COUNT..._constant
baud_tc
baud_start_tc
baud_reset
baud_count[13:0]
bit_tc
bit_reset
bit_count[3:0]
current_state
next_state


Name
Value
clk_signal
Rx_signal
Rx_Done_signal
Rx_Error_signal
clock_period
current_state
next_state
baud_count[3
baud_reset
bit_count[3:0]
bit_reset
shift_en

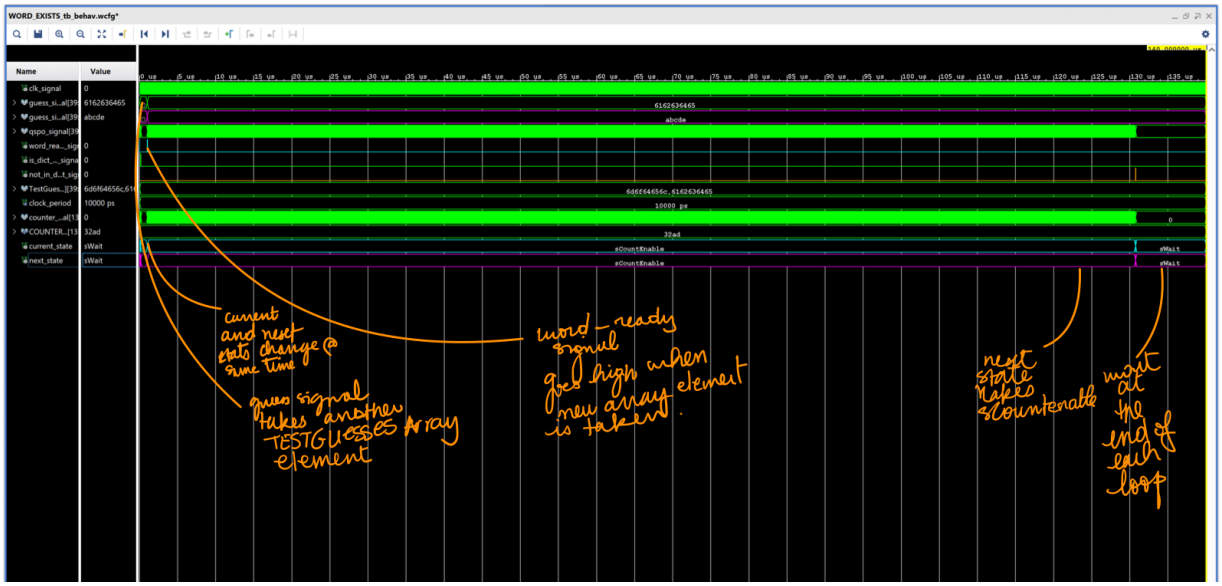
D. Receiver Transmitter



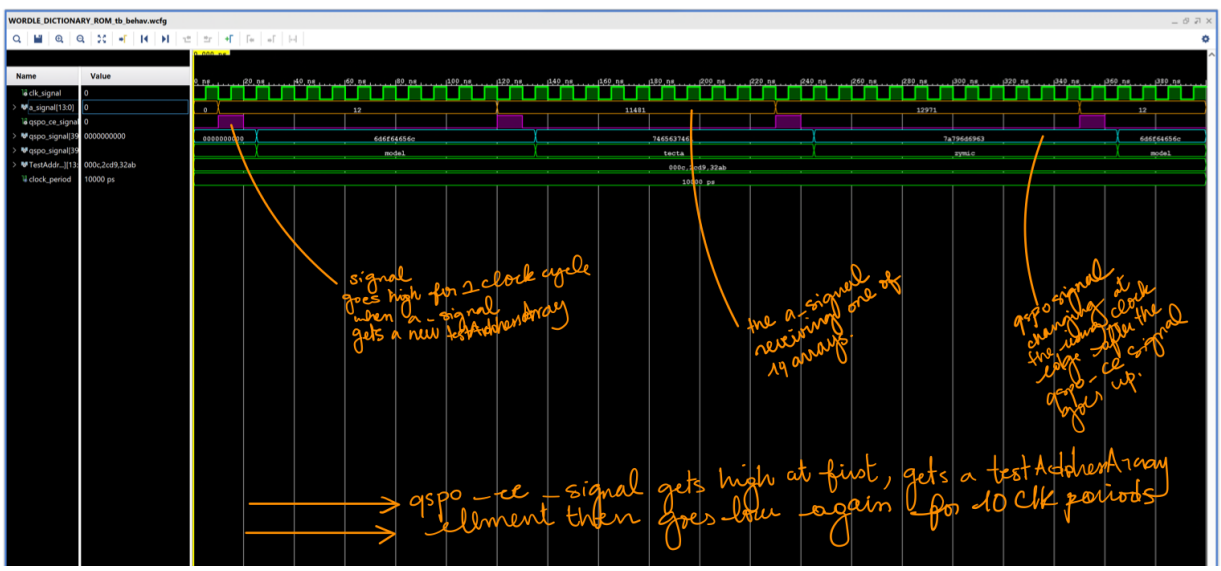
E. Check Uppercase Letter (Receiver Transmitter)



F. Word Exists



G. WORDLE Dictionary ROM



VII. Conclusion

A. Encountered Errors: As we worked through this project, we encountered the following major errors:

- As we built the memory on VIVADO, we made the mistake of making it a distributed memory as opposed to a block memory. This slowed us down toward the end of the project and made debugging somewhat difficult. Once we recognized this as an issue, we found it easier to debug for the rest of the errors.

- As we implemented the FSM in VIVADO, we found that it was not interacting well with our transmitter. The reason for this was that the signals connecting the FSM and the transmitter were not working properly in hardware

VIII. Acknowledgements

ROM/ram. (n.d.). Retrieved June 5, 2022, from http://web.mit.edu/6.111/volume2/www/f2019/handouts/labs/lab3_19/rom_vivado.html

Wordle dictionary. Wordle Dictionary. (n.d.). Retrieved June 5, 2022, from https://wordledictionary.com/

IX. Appendix

SCI Receiver Design

```
--==================================================================
==========
--ENGS 31/ CoSc 56 22S
--Final Project - Wordle
--B.L. Dobbins, E.W. Hansen, Professor Luke
--Mubarak Idoko, Ikeoluwa Abioye, Lobna Jbenaini
   --SCI Receiver

-- TODO: (DO NOT REMOVE) Write to "Update Log" if you modify the code, document
your changes.
-- TODO: (DO NOT REMOVE) Update known issues to account for fixes of newly
discovered bugs.

-- UPDATE LOG:
-- Format: (Name) Date: Notes
--==================================================================
==========
   -- (Mubbie) 05/25/2022: Created the file, implemented all receiver functionality
   -- (Mubbie) 05/28/2022: Wrote testbench and discovered bug with baud counter
--==================================================================
==========

-- KNOWN ISSUES:
   -- Known issues with implementation that have not been fixes
   -- Critical Score (0-5):
```

```
        -- subjective measure
        -- how important fixing this issue is important to the proper functioning of the system
    -- Ideas for fix: thoughts on how the issue could be fixed
-- Format: (Name) Date [Critical Score (0-5)]: Notes [Ideas for fix] {Updates (if any)}
--==============================================================================
===========
    -- (Mubbie) 05/25/2022 [1]:  Baud counter is always running, so there could be delays in the
transmission of the data.
                        -- Delays will barely be noticed outside in depth study of simulation.
                        -- They just irk me so much and I think we are better of fixing it.
                        -- Very non critical issue
                        --
                        -- [Ideas for fix]: add a baud counter enable that only goes high when the Rx
bit goes low and we start transmitting, otherwise stays low
                        -- this will tell the baud counter when or when not to run
                        -- alternatively, we could clear the counter when new data comes in so that it
starts from scratch
                        -- {Updates}: Implemented the baud and bit counter reset that enables/resets
the counters. I believe that fixes the problem. I think I have fixed this.
    -- (Mubbie) 05/25/2022 [5]:  The baud counter does not work properly.
                        -- In simulation, using the correct syntax in lines 278-287 i.e. assert the tc's
when count reaches count_top - 1
                        -- The baud counter disregards this instructions and counts up to count_top
before stopping and resetting
                        -- {Update}: One potential fix is just to use the syntax "count_top - 2"  as in
the code now but this is not correct and could perhaps cause other issues
--==============================================================================
===========



--==============================================================================
===========
--Library Declarations:
--==============================================================================
===========
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use ieee.math_real.all;
```

```
--==========================================================================
===========
--Entity Declaration:
--==========================================================================
===========
entity SCI_RECEIVER is
    -- constants
    generic(
        BAUD_COUNTER_TOP : integer;
        BIT_COUNTER_TOP : integer
    );

    -- ports
    port (
        -- inputs
        clk : in std_logic;
        Rx: in std_logic;

        -- outputs
        Rx_Data : out std_logic_vector(7 downto 0);
        Rx_Done : out std_logic;
        Rx_Error : out std_logic
    );
end entity SCI_RECEIVER;

--==========================================================================
===========
--Architecture Type:
--==========================================================================
===========
architecture behavioral_architecture of SCI_RECEIVER is

--==========================================================================
===========
--State Type Declarations:
--==========================================================================
===========
type state is (sWait, sSampleStartBitCounter, sShiftBit, sSampleBitCounter, sParallelOut,
sRxSuccess, sRxFail);
```

```
--==========================================================================
===========
--Signal Declarations:
--==========================================================================
===========
-- constants: counter periods
constant BAUD_COUNTER_LEN : integer :=
integer(ceil(log2(real(BAUD_COUNTER_TOP))));
constant BIT_COUNTER_LEN : integer := integer(ceil(log2(real(BIT_COUNTER_TOP))));

-- states
signal current_state : state := sWait;
signal next_state : state;

-- state label (useful for simulation in EdaPlayground):
signal current_state_bin, next_state_bin : std_logic_vector(2 downto 0) := "000";
   -- where the state labels map:
      -- sWait -> 000
      -- sSampleStartBitCounter -> 001
      -- sShiftBit -> 010
      -- sSampleBitCounter -> 011
      -- sParallelOut -> 100
      -- sRxSuccess -> 101
      -- sRxFail -> 110

-- others
-- total count monopulse signals:
signal baud_tc : std_logic := '0';
signal baud_start_tc : std_logic := '0';
signal bit_tc : std_logic := '0';

-- counter reset/control signals:
signal baud_reset : std_logic := '1';
signal bit_reset : std_logic := '1';

-- counter variables:
signal baud_count : unsigned(BAUD_COUNTER_LEN - 1 downto 0) := (others => '0');
signal bit_count : unsigned(BIT_COUNTER_LEN - 1 downto 0) := (others => '0');

-- shift register:
```

```
signal shift_register : std_logic_vector((BIT_COUNTER_TOP - 1) downto 0) := (others =>
'1');

-- shift register control:
signal shift_en : std_logic := '0';
signal register_reset : std_logic := '0';

-- rx_output_register
signal Rx_Data_Out : std_logic_vector((BIT_COUNTER_TOP - 3) downto 0) := (others =>
'1');
signal send_output : std_logic := '0';

signal rx_done_output : std_logic := '0';
signal rx_error_output : std_logic := '0';

begin
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
--Update the current state (synchronous):
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
StateUpdate: process(clk)
begin
   -- update state on rising edge of clock:
   if rising_edge(clk) then
      current_state <= next_state;
      current_state_bin <= next_state_bin;
   end if;
end process StateUpdate;

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
--Next State Logic (asynchronous):
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
NextStateLogic: process(current_state, current_state_bin, Rx, baud_start_tc, baud_tc, bit_tc,
shift_register)
begin
   -- define defaults:
   next_state <= current_state;
```

```
next_state_bin <= current_state_bin;

-- update next state:
case current_state is
    when sWait =>
        -- wait for start bit:
        if Rx = '0' then
            next_state <= sSampleStartBitCounter;
            next_state_bin <= "001";
        end if;
    when sSampleStartBitCounter =>
        -- wait to shift the start bit:
        if baud_start_tc = '1' then
            -- transition to bit shift state:
            next_state <= sShiftBit;
            next_state_bin <= "010";
        end if;
    when sShiftBit =>
        -- wait for all bits to be shifted in:
        if bit_tc = '1' then
            -- transition to parallel out state:
            next_state <= sParallelOut;
            next_state_bin <= "100";
        else
            -- sample the next bit
            next_state <= sSampleBitCounter;
            next_state_bin <= "011";
        end if;
    when sSampleBitCounter =>
        -- wait to sample the current bit:
        if baud_tc = '1' then
            next_state <= sShiftBit;
            next_state_bin <= "010";
        end if;
    when sParallelOut =>
        -- validate the data that was received
        if shift_register(0) = '0' and shift_register(9) = '1' then
            -- data is valid:
            next_state <= sRxSuccess;
            next_state_bin <= "101";
```

```
            else
                -- data is invalid:
                next_state <= sRxFail;
                next_state_bin <= "110";
            end if;
        when sRxSuccess =>
            -- return to wait
            next_state <= sWait;
            next_state_bin <= "000";
        when sRxFail =>
            -- return to wait
            next_state <= sWait;
            next_state_bin <= "000";
    end case;
end process NextStateLogic;

--++++++
```

```
            -- enable bit counter, and shift:
            bit_reset <= '0';
            shift_en <= '1';
        when sSampleBitCounter =>
            -- enable baud counter:
            baud_reset <= '0';
        when sRxSuccess =>
            -- output received data and receipt nature signals:
            rx_done_output <= '1';
            rx_error_output <= '0';
            send_output <= '1';
        when sRxFail =>
            -- output received data and receipt nature signals:
            rx_done_output <= '1';
            rx_error_output <= '1';
            send_output <= '1';
        when others =>
            -- do nothing
            -- take default outputs
    end case;
end process OutputLogic;

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Counter Logic:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
CounterLogic: process(clk, baud_count, bit_count)
begin
    -- count
    if rising_edge(clk) then
        -- BAUD COUNTER:
        -- count only when we are supposed to
        if baud_reset = '0' then
            baud_count <= baud_count + 1;
        else
            baud_count <= (others => '0');
        end if;

        -- BIT COUNTER
        -- count only when we are supposed to
        if bit_reset = '0' then
```

```
            if (bit_count < BIT_COUNTER_TOP - 1) then
                bit_count <= bit_count + 1;
            elsif (bit_count = BIT_COUNTER_TOP - 1) then
                bit_count <= (others => '0');
            end if;
        else
            bit_count <= bit_count;
        end if;
    end if;

    -- async total count:
    -- define the default signals:
    baud_tc <= '0';
    baud_start_tc <= '0';
    bit_tc <= '0';

    -- BAUD COUNTER:
    -- load start:
    if (baud_count = ((BAUD_COUNTER_TOP/2) - 2)) then
        baud_start_tc <= '1';
    end if;

    -- load bit:
    if (baud_count = (BAUD_COUNTER_TOP - 2)) then
        baud_tc <= '1';
    end if;

    -- BIT COUNTER
    -- shifted bit:
    if (bit_count = BIT_COUNTER_TOP - 1) then
        bit_tc <= '1';
    end if;
end process CounterLogic;

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Datapath: Shift Register Logic (synchronous):
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
datapath : process(clk)
```

```
begin
        if rising_edge(clk) then
              -- default outputs
               Rx_Done <= '0';
         Rx_Error <= '0';

        -- Shift Register
        if register_reset = '1' then
           shift_register <= (others => '1');
        elsif shift_en = '1' then
           -- shift when we have the signal to do so:
           shift_register <= Rx & shift_register(9 downto 1);
        end if;

        -- output register
        if send_output = '1' then
           Rx_Data_Out <= shift_register(8 downto 1);
        end if;

        -- output signals
        -- rx_done
        Rx_Done <=  rx_done_output;

        -- rx_error
        Rx_Error <= rx_error_output;
   end if;
end process datapath;

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
--Receiver Output Logic:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
Rx_Data <= Rx_Data_Out;

end behavioral_architecture;
```

SCI Transmitter Design

```
--==========================================================================
```

```
===========
--ENGS 31/ CoSc 56 22S
--Final Project - Wordle
--B.L. Dobbins, E.W. Hansen, Professor Luke
--Mubarak Idoko, Ikeoluwa Abioye, Lobna Jbenaini
    --SCI Receiver

-- TODO: (DO NOT REMOVE) Write to "Update Log" if you modify the code, document your changes.
-- TODO: (DO NOT REMOVE) Update known issues to account for fixes of newly discovered bugs.

-- UPDATE LOG:
-- Format: (Name) Date: Notes
--==========================================================================
    -- (Ike) 05/28/2022: Wrote the code
    -- (Mubbie) 05/29/2022: Documented the code
--==========================================================================

-- KNOWN ISSUES:
    -- Known issues with implementation that have not been fixes
    -- Critical Score (0-5):
        -- subjective measure
        -- how important fixing this issue is important to the proper functioning of the system
    -- Ideas for fix: thoughts on how the issue could be fixed
-- Format: (Name) Date [Critical Score (0-5)]: Notes [Ideas for fix] {Updates (if any)}
--==========================================================================
    -- (Mubbie) 05/25/2022 [1]:  Baud counter is always running, so there could be delays in the transmission of the data.
                        -- Delays will barely be noticed outside indepth study of simulation.
                        -- They just irk me so much and I think we are better of fixing it.
                        -- Very non critical issue
                        --
                        -- [Ideas for fix]: add a baud counter enable that only goes high when the Rx bit goes low and we start transmitting, otherwise stays low
                        -- this will tell the baud counter when or when not to run
                        -- alternatively, we could clear the counter when new data comes in so that it
```

```
starts from scratch
--==========================================================================
===========



--==========================================================================
===========
--Library Declarations:
--==========================================================================
===========
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use ieee.math_real.all;


--==========================================================================
===========
--Entity Declaration:
--==========================================================================
===========
ENTITY SCI_Transmitter IS
   -- constants
   generic(
      BAUD_COUNTER_TOP : integer;
      BIT_COUNTER_TOP : integer
   );

   -- ports
   port (
         -- inputs
         clk                  :        in       STD_LOGIC;
         Parallel_in     :        in       STD_LOGIC_VECTOR(7 downto 0);
         New_data        :        in       STD_LOGIC;

         -- outputs
         Tx                   :        out STD_LOGIC
   );
end SCI_Transmitter;
```

```
--==========================================================================
===========
--Architecture Type:
--==========================================================================
===========
ARCHITECTURE behavior of SCI_Transmitter is

--==========================================================================
===========
--Signal Declarations:
--==========================================================================
===========
-- constants: counter periods
constant BAUD_COUNTER_LEN : integer :=
integer(ceil(log2(real(BAUD_COUNTER_TOP))));

signal Shift_Reg : std_logic_vector(9 downto 0) := (others => '1');
signal Baud_Counter : unsigned((BAUD_COUNTER_LEN - 1) downto 0) := (others => '0');
-- 9 bits are needed to represent 391.

signal tc : std_logic := '0';

-- signals for bit counter
signal block_sent : std_logic := '0';
signal num_bits_sent : integer := 0;

-- signals for the queue
type regfile is array(0 to 7) of std_logic_vector((BIT_COUNTER_TOP - 3) downto 0);
signal queue_reg : regfile:= ((others => '0'),(others => '0'),(others => '0'),(others => '0'),(others
=> '0'),(others => '0'),(others => '0'),(others => '0')); --one way to zero out all the elements.

-- other signals
signal r_addr : integer := 0;
signal w_addr : integer := 0;
signal q_size : integer := 0;
signal empty  : std_logic := '0';
signal full   : std_logic := '0';
signal r_data : std_logic_vector(7 downto 0);

BEGIN
```

```
--++++++
```

```
        -- enqueue the data (write to the queue)
        if (new_data = '1' and full = '0') then
                        queue_reg(w_addr) <= Parallel_in;
            q_size <= q_size + 1;
            -- increment the address, accounting for overflow
            if w_addr = 7 then
                        w_addr <= 0;
            else w_addr <= w_addr + 1;
            end if;
        end if;
                        -- dequeue the data
        if (block_sent = '1' and empty = '0') then
            queue_reg(r_addr) <= (others => '0');
            q_size <= q_size - 1;
            -- increment the address, accounting for overflow
            if r_addr = 7 then
                        r_addr <= 0;
            else r_addr <= r_addr + 1;
            end if;
        end if;
    end if;
end process datapath;

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
--Queue Size Logic:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
QueueSizeLogic: process(r_addr, w_addr, queue_reg, empty, full, q_size)
begin
            Empty <= '0';
    Full <= '0';

    -- queue size logic
            if (q_size = 0) then
            Empty <= '1';
            elsif (q_size = 7) then
            Full <= '1';
            end if;
```

```
    r_data <= queue_reg(r_addr);
end process QueueSizeLogic;

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Output:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Tx <= Shift_Reg(0);

end behavior;
```

SCI RX TX Design

```
--==================================================================================================
--ENGS 31/ CoSc 56 22S
--Final Project - Wordle
--B.L. Dobbins, E.W. Hansen, Professor Luke
--Mubarak Idoko, Ikeoluwa Abioye, Lobna Jbenaini
    --RX TX Top Level

-- TODO: (DO NOT REMOVE) Write to "Update Log" if you modify the code, document your changes.
-- TODO: (DO NOT REMOVE) Update known issues to account for fixes of newly discovered bugs.

-- UPDATE LOG:
-- Format: (Name) Date: Notes
--==================================================================================================
    -- (Mubbie) 05/28/2022: Wrote top level shell to test SCI receiver and transmitter.
--==================================================================================================

-- KNOWN ISSUES:
    -- Known issues with implementation that have not been fixes
    -- Critical Score (0-5):
```

```
        -- subjective measure
        -- how important fixing this issue is important to the proper functioning of the system
    -- Ideas for fix: thoughts on how the issue could be fixed
-- Format: (Name) Date [Critical Score (0-5)]: Notes [Ideas for fix] {Updates (if any)}
--==============================================================================
===========

--==============================================================================
===========

--==============================================================================
===========
--Library Declarations:
--==============================================================================
===========
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use ieee.math_real.all;
library UNISIM;
use UNISIM.VComponents.all;


--==============================================================================
===========
--Entity Declaration:
--==============================================================================
===========
entity rx_tx_shell is
    port (
        --Inputs:
        -- clock
        clk_ext_port : in std_logic;

        -- Tx and Rx lines
        RsTx_ext_port : out std_logic;
        RsRx_ext_port : in std_logic;

        -- Rx_data_port for testing purposes
        Rx_data_port : out std_logic_vector(7 downto 0)
```

```
    );
end rx_tx_shell;

--=================================================================================
--Architecture Type:
--=================================================================================
architecture behavioral_architecture of rx_tx_shell is

--=================================================================================
--Sub-Component Declarations:
--=================================================================================
--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Receiver Sub-Component:
--+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
component SCI_RECEIVER is
    -- constants
    generic(
        BAUD_COUNTER_TOP : integer;
        BIT_COUNTER_TOP : integer
    );

    -- ports
    port (
        -- inputs
        clk : in std_logic;
        Rx: in std_logic;

        -- outputs
        Rx_Data : out std_logic_vector(7 downto 0);
        Rx_Done : out std_logic;
        Rx_Error : out std_logic
    );
end component SCI_RECEIVER;
```

```
--++++++
```

```
        Tx                          :       out STD_LOGIC
    );
end component SCI_Transmitter;

--==================================================================================
--Signal Declarations:
--==================================================================================
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Receiver:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
signal Rx_Data_sig              :  std_logic_vector(7 downto 0) := (others => '0');
signal Rx_Done_sig              :  std_logic := '0';
signal Rx_Error_sig             :  std_logic := '0';
signal Rx_sig          :  std_logic := '0';


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Check Letter:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
signal CL_output_lower_case_letter : std_logic_vector(7 downto 0) := (others => '1');
signal CL_is_backspace_signal : std_logic := '0';
signal CL_is_valid_alpha_signal : std_logic := '0';
signal CL_is_enter_signal : std_logic := '0';

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Transmitter:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
signal New_Data_Signal : std_logic := '0';

--==================================================================================
--Port Mapping (wiring the component blocks together):
```

```
--==========================================================================
===========
begin
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++
--Wire the receiver sub-component to the shell:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++
receiver : SCI_RECEIVER
    generic map (
        BAUD_COUNTER_TOP => 10417, -- 9600 baud rate
        BIT_COUNTER_TOP => 10 -- 10 bits, 1 start, 8 data, 1 stop
    )
    port map (
        clk => clk_ext_port,
        Rx => RsRx_ext_port,
        Rx_Data => Rx_data_sig,
        Rx_Done => Rx_Done_sig,
        Rx_Error => Rx_Error_sig
    );

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++
--Wire the "check_letter" sub-component to the shell:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++
checkletter : CHECK_LETTER
    PORT MAP (
        -- inputs
        letter => Rx_Data_sig,

        -- outputs
        is_backspace => CL_is_backspace_signal,
        is_valid_alpha => CL_is_valid_alpha_signal,
        is_enter => CL_is_enter_signal,
        output_lower_case_letter => CL_output_lower_case_letter
    );

--==========================================================================
===========
```

```
--New_Data_Stimulus: Prep new data for the transmitter
--==========================================================================
===========
New_Data_Stimulus : process(Rx_Done_sig, CL_is_valid_alpha_signal,
CL_is_backspace_signal, CL_is_enter_signal)
begin
   New_Data_Signal <= (Rx_Done_sig and CL_is_valid_alpha_signal) or (Rx_Done_sig and
CL_is_backspace_signal) or (Rx_Done_sig and CL_is_enter_signal);
end process New_Data_Stimulus;

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
--Wire the transmitter sub-component to the shell:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
transmitter : SCI_Transmitter
   generic map (
      BAUD_COUNTER_TOP => 10417, -- 9600 baud rate
      BIT_COUNTER_TOP => 10 -- 10 bits, 1 start, 8 data, 1 stop
   )
   port map (
      clk => clk_ext_port,
      Parallel_in => CL_output_lower_case_letter,
      New_data => New_Data_Signal,

      -- Tx => Tx_sig
      Tx => RsTx_ext_port
   );

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
--Output for test with LED's:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
Rx_data_port <= Rx_Data_sig;

end behavioral_architecture;
```

Check Guess Logic Design

```
--==============================================================================
--ENGS 31/ CoSc 56 22S
--Final Project - Wordle
--Mubarak Idoko, Ikeoluwa Abioye, Lobna Jbenaini
-- Module Name: CheckGuess - Behavioral

-- TODO: (DO NOT REMOVE) Write to "Update Log" if you modify the code, document your changes.
-- TODO: (DO NOT REMOVE) Update known issues to account for fixes of newly discovered bugs.

-- UPDATE LOG:
-- Format: (Name) Date: Notes
--==============================================================================
   -- (Ike) 06/02/2022: Created the file, implemented checking functionality
--==============================================================================

-- KNOWN ISSUES:

--==============================================================================----------------------------------------------------------------------------------------------------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity CheckGuess is
```

```
    Port ( clk              : in STD_LOGIC;
           guess            : in STD_LOGIC_VECTOR(39 DOWNTO 0);
           solution         : in STD_LOGIC_VECTOR(39 DOWNTO 0);
           is_dict_word     : in STD_LOGIC;

           correct_letters  : out STD_LOGIC_VECTOR(4 DOWNTO 0);
           correct_places   : out STD_LOGIC_VECTOR(4 DOWNTO 0);
           win              : out STD_LOGIC;
           done             : out STD_LOGIC     );
end CheckGuess;

architecture Behavioral of CheckGuess is
--==========================================================================
===========
-------------------------------------------
-- Constants
constant max_ltr_idx : integer := 4;         -- max iterations?
-------------------------------------------

--Signal Declarations:
signal enable_outputs : STD_LOGIC := '0';     -- send outputs out?

signal correct_place_vals   : STD_LOGIC_VECTOR(4 DOWNTO 0);

-- store the words in an array for easy indexing
type  WordType  is array(0 to 4) of STD_LOGIC_VECTOR(7 DOWNTO 0);  -- hold a word, a
characters in each index
signal guess_word :  WordType; -- := ((guess(7 DOWNTO 0)), (guess(15 DOWNTO 8)),
(guess(23 DOWNTO 16)), (guess(31 DOWNTO 24)), (guess(39 DOWNTO 32)));
signal solution_word :  WordType; --:= ((solution(7 DOWNTO 0)), (solution(15 DOWNTO
8)), (solution(23 DOWNTO 16)), (solution(31 DOWNTO 24)), (solution(39 DOWNTO 32)));

-- for check_right_locations
signal place_correct_mid : STD_LOGIC_VECTOR(4 DOWNTO 0);

-- for address counters
signal count_addresses : STD_LOGIC := '0';
signal sol_addr : integer := 0; -- is this also a counter?
signal guess_addr : integer := 0;
signal guess_count_en :  STD_LOGIC := '0';
```

```
-- for check_wrong_locations
signal compare_en          : STD_LOGIC := '1';
signal letters_correct_mid  : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00000";
signal sol_letters_present  : STD_LOGIC_VECTOR(4 DOWNTO 0) := "00000";
signal reset_ltrs_correct   : STD_LOGIC := '0';
--==================================================================
===========
-- state machine signals
type StateType is (idle, checkRightLocs, checkWrongLocs);
signal current_state, next_state : StateType;

--==================================================================
===========

begin

--=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+=+
-- DATAPATH:
--=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
=+=+=+=+=+=+=+=+=+=+=+
-- Wiring
win <= (correct_place_vals(0) and correct_place_vals(1) and correct_place_vals(2) and
correct_place_vals(3) and correct_place_vals(4));
correct_places <= correct_place_vals;
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++
--Fill Array Logic:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++
FillArrays: process (clk)
begin
   if rising_edge(clk) then
      if is_dict_word = '1' then
         guess_word <= ( (guess(39 DOWNTO 32)), (guess(31 DOWNTO 24)), (guess(23
DOWNTO 16)), (guess(15 DOWNTO 8)), (guess(7 DOWNTO 0)) );
         solution_word <= ((solution(39 DOWNTO 32)), (solution(31 DOWNTO 24)),
(solution(23 DOWNTO 16)), (solution(15 DOWNTO 8)), (solution(7 DOWNTO 0)));
      end if;
```

```
    end if;
end process FillArrays;

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Right Locations Logic:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
RightLocations: process (guess_word, solution_word, clk)
begin
    if rising_edge(clk) then
        if enable_outputs = '1' then
            correct_place_vals <= place_correct_mid;
        end if;
    end if;

    comparator: for ltr_start in 0 to max_ltr_idx loop
        place_correct_mid(ltr_start) <= '0';
        if guess_word(ltr_start) = solution_word(ltr_start) then
            place_correct_mid(ltr_start) <= '1';
        end if;
    end loop comparator;
end process RightLocations;

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Addressing Counters Logic:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
AddressCounters: process (clk, sol_addr, guess_addr)
begin
guess_count_en <= '0';
enable_outputs <= '0';

if sol_addr = 4 then
    guess_count_en <= '1';
end if;
if guess_addr = 4 and sol_addr = 4 then
    enable_outputs <= '1';
end if;
```

```
if rising_edge(clk) then
    if count_addresses = '1' then
        sol_addr <= sol_addr + 1;
        if sol_addr = 4 then
            sol_addr <= 0;
        end if;
    else
        sol_addr <= 0;
    end if;
    if guess_count_en = '1' then
        guess_addr <= guess_addr + 1;
        if guess_addr = 4 then
            guess_addr <= 0;
        end if;
    end if;

end if;

end process AddressCounters;


--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Wrong Locations Logic:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
WrongLocations: process (clk, sol_addr, guess_addr, solution_word, guess_word,
place_correct_mid, letters_correct_mid, reset_ltrs_correct, sol_letters_present, compare_en)
    begin

    compare_en <= not( place_correct_mid(sol_addr) or letters_correct_mid(guess_addr) or
sol_letters_present(sol_addr) );

    if rising_edge(clk) then
        if enable_outputs = '1' then
            correct_letters <= letters_correct_mid;
        end if;

        if compare_en = '1' then
```

```
            if guess_word(guess_addr) = solution_word(sol_addr) then
                letters_correct_mid(guess_addr) <= '1';
                sol_letters_present(sol_addr) <= '1';
            end if;
        end if;

        if reset_ltrs_correct = '1' then
            letters_correct_mid <= "00000";
            sol_letters_present <= "00000";
        end if;
    end if;

--    if reset_ltrs_correct = '1' then
--        letters_correct_mid <= "00000";
--        sol_letters_present <= "00000";
--    end if;
end process WrongLocations;

--=+=+=+
```

```
    when checkRightLocs =>
        next_state <= checkWrongLocs;
    when checkWrongLocs =>
        if enable_outputs = '1' then
            next_state <= idle;
        end if;
    -- when others =>  -- gets the current state automatically
end case current_state;
end process NextStateLogic;

OutputLogic: process(current_state)
begin
reset_ltrs_correct <= '0';
--store_words <= '0';
count_addresses <= '0';

case current_state is
    when idle =>
        reset_ltrs_correct <= '1';
    when checkRightLocs =>
    when checkWrongLocs =>
        count_addresses <= '1';

    -- when others =>  -- gets the current state automatically
end case current_state;
end process OutputLogic;



end Behavioral;
```

Check Letter Design

```
--==================================================================
===========
--ENGS 31/ CoSc 56 22S
--Final Project - Wordle
--B.L. Dobbins, E.W. Hansen, Professor Luke
--Mubarak Idoko, Ikeoluwa Abioye, Lobna Jbenaini
    --SCI Receiver Testbench
```

```
-- TODO: (DO NOT REMOVE) Write to "Update Log" if you modify the code, document
your changes.
-- TODO: (DO NOT REMOVE) Update known issues to account for fixes of newly
discovered bugs.

-- UPDATE LOG:
-- Format: (Name) Date: Notes
--==================================================================
===========
   -- (Mubbie) 05/30/2022: Created the file, implemented checking functionality
--==================================================================
===========

-- KNOWN ISSUES:
   -- Known issues with implementation that have not been fixes
   -- Critical Score (0-5):
      -- subjective measure
      -- how important fixing this issue is important to the proper functioning of the system
   -- Ideas for fix: thoughts on how the issue could be fixed
-- Format: (Name) Date [Critical Score (0-5)]: Notes [Ideas for fix] {Updates (if any)}
--==================================================================
===========



--==================================================================
===========



--==================================================================
===========
--Library Declarations:
--==================================================================
===========
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use ieee.math_real.all;
```

```
--==========================================================================
===========
--Entity Declaration:
--==========================================================================
===========
ENTITY Check_Letter IS
   -- ports
   port (
         -- inputs
         letter    :         in        STD_LOGIC_VECTOR(7 downto 0);

         -- outputs
         is_backspace                        :         out STD_LOGIC;
         is_valid_alpha         : out STD_LOGIC;
         is_enter              :      out STD_LOGIC;

         -- lower case output
         output_lower_case_letter :     out STD_LOGIC_VECTOR(7 downto 0)
   );
end entity Check_Letter;

--==========================================================================
===========
--Architecture Type:
--==========================================================================
===========
architecture behavioral_architecture of Check_Letter is


--==========================================================================
===========
--Signal Declarations:
--==========================================================================
===========
-- constants
constant backspace : unsigned(7 downto 0) := "00001000"; -- ASCII backspace
constant enter : unsigned(7 downto 0) := "00001101"; -- ASCII carriage return
constant upper_case_a : unsigned(7 downto 0) := "01000001"; -- ASCII A
constant upper_case_z : unsigned(7 downto 0) := "01011010"; -- ASCII Z
constant lower_case_a : unsigned(7 downto 0) := "01100001"; -- ASCII a
```

```
constant lower_case_z : unsigned(7 downto 0) := "01111010"; -- ASCII z
constant upper_case_to_lower_case : unsigned(7 downto 0) := "00100000"; -- this is 32

begin
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Output Logic:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Output_Logic : process(letter)
begin
   -- default outputs
   is_backspace <= '0';
   is_valid_alpha <= '0';
   is_enter <= '0';
   output_lower_case_letter <= letter;

   -- backspace
   if unsigned(letter) = backspace then
      is_backspace <= '1';
   end if;

   -- enter
   if unsigned(letter) = enter then
      is_enter <= '1';
   end if;

   -- is valid alpha
   if (unsigned(letter) >= upper_case_a and unsigned(letter) <= upper_case_z)
      or (unsigned(letter) >= lower_case_a and unsigned(letter) <= lower_case_z) then
      is_valid_alpha <= '1';
   end if;

   -- lower case output
   if unsigned(letter) >= upper_case_a and unsigned(letter) <= upper_case_z then
      -- add 32 to make it lowercase
      output_lower_case_letter <= std_logic_vector(unsigned(letter) +
upper_case_to_lower_case);
   end if;
end process Output_Logic;
```

```
end behavioral_architecture;
```

FSM Design

```
--==========================================================================
===========
--ENGS 31/ CoSc 56 22S
--Final Project - Wordle
--Mubarak Idoko, Ikeoluwa Abioye, Lobna Jbenaini
-- Module Name: CheckGuess - Behavioral


-- UPDATE LOG:
-- Format: (Name) Date: Notes
--==========================================================================
===========
   -- (Ike) 06/03/2022: Created the file, implemented checking functionality
--==========================================================================
===========

-- KNOWN ISSUES:

--==========================================================================
===========

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity FSM is
   Port ( clk            : in STD_LOGIC;
```

```
        Rx_data : in STD_LOGIC_VECTOR (7 downto 0);
        Rx_done : in STD_LOGIC;
        Tx_data : out STD_LOGIC_VECTOR (7 downto 0);
        Tx_data_ready : out STD_LOGIC);
end FSM;

architecture Behavioral of FSM is
component load_word is
   port (
      --Inputs:
      clk              : in std_logic;
      letter_in        : in STD_LOGIC_VECTOR(7 downto 0);
      new_letter       : in std_logic;

      --Outputs:
      full             :  out std_logic;
      char_disp_out      :  out std_logic_vector(7 downto 0);
      char_disp_out_ready :  out std_logic;
      word_ready         :  out std_logic;

      -- Output: Word characters (5 letter word)
      char_0           :  out std_logic_vector(7 downto 0);
      char_1           :  out std_logic_vector(7 downto 0);
      char_2           :  out std_logic_vector(7 downto 0);
      char_3           :  out std_logic_vector(7 downto 0);
      char_4           :  out std_logic_vector(7 downto 0)
   );
end component load_word;

component WORD_EXISTS IS
   -- ports
   port (
         -- inputs
         clk_in                  :       in      STD_LOGIC;
         guess                   :       in      STD_LOGIC_VECTOR(39 downto 0);
         word_ready              :       in      STD_LOGIC;

         -- outputs
         is_dict_word    :       out STD_LOGIC;
         not_in_dict     :       out STD_LOGIC
```

```
    );
end component WORD_EXISTS;

component CheckGuess
    Port ( clk              : in STD_LOGIC;
           guess            : in STD_LOGIC_VECTOR(39 DOWNTO 0);
           solution         : in STD_LOGIC_VECTOR(39 DOWNTO 0);
           is_dict_word     : in STD_LOGIC;

           correct_letters  : out STD_LOGIC_VECTOR(4 DOWNTO 0);
           correct_places   : out STD_LOGIC_VECTOR(4 DOWNTO 0);
           win              : out STD_LOGIC;
           done             : out STD_LOGIC     );
end component CheckGuess;

-- wire sol_addr to the dictionary
component wordle_dictionary_rom
  PORT (
    a : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
    clk : IN STD_LOGIC;
    qspo_ce : IN STD_LOGIC;
    qspo : OUT STD_LOGIC_VECTOR(39 DOWNTO 0) );
end component wordle_dictionary_rom;

--==========================================================================
===========
-- Constants
constant max_ltr_idx : integer := 4;
constant max_num_tries : integer := 5;
constant byte_size : integer := 8;
constant max_dict_word : unsigned(13 downto 0) := "11001010101011"; -- 12971

constant black_sym : STD_LOGIC_VECTOR(7 DOWNTO 0) := "00100000";      -- space
constant yellow_sym : STD_LOGIC_VECTOR(7 DOWNTO 0) := "00111111";     -- question
mark
constant backspace : STD_LOGIC_VECTOR(7 DOWNTO 0) := "00001000";
constant delete : STD_LOGIC_VECTOR(7 DOWNTO 0) := "01111111";
constant enter : STD_LOGIC_VECTOR(7 DOWNTO 0) := "00001101";

constant dash : STD_LOGIC_VECTOR(7 DOWNTO 0) := "00101101";
```

```
constant win_out: STD_LOGIC_VECTOR(95 DOWNTO 0) := (47 downto 0 => '0') & enter
& dash & dash & dash & dash & dash;        -- 12 bytes of data
constant lose_out: STD_LOGIC_VECTOR(47 DOWNTO 0) := enter & black_sym &
black_sym & black_sym & black_sym & black_sym;

-------------------------------------------

--Signal Declarations:

--the solution
signal solution_sig : STD_LOGIC_VECTOR(39 DOWNTO 0);
-- the solution counter
signal sol_addr : unsigned(13 downto 0) := "00000000000000";
-- the most recent guess
signal char_0_signal : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
signal char_1_signal : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
signal char_2_signal : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
signal char_3_signal : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
signal char_4_signal : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
signal guess_sig    : STD_LOGIC_VECTOR(39 DOWNTO 0) := (others => '0');
-- to wait for dict (2 clk cycles) before getting guess_sig
signal waitcount : integer := 0;


-- data to send register
signal data_to_send : STD_LOGIC_VECTOR(95 DOWNTO 0);       -- register enough to
hold 12 characters (including CR)
signal cnt          : integer := 0;
signal data_ready   : std_logic := '0';
signal send         : std_logic := '0';
signal sent         : std_logic := '1';
signal data_ready_next   : std_logic := '0';
signal data_ready_mp   : std_logic := '0';



-- state machine signals
type StateType is (newGame, idle, displayLetters, displayColors, win, lose);
signal current_state, next_state : StateType;
```

```
-- load_word signals
signal char_disp_out_sig        : std_logic_vector(7 downto 0);
signal char_disp_out_ready_sig  : std_logic;
signal word_ready_sig       : std_logic;

-- CheckWordExists signals
signal in_dict_sig          : std_logic := '0';
signal not_in_dict_sig      : std_logic := '0';

-- CheckGuess signals
signal cltrs                : STD_LOGIC_VECTOR(4 DOWNTO 0);
signal cplaces              : STD_LOGIC_VECTOR(4 DOWNTO 0);
signal win_all_green        : STD_LOGIC := '0';
signal colors_ready         : STD_LOGIC := '0';

-- count numtries signals
signal num_tries : integer := 0;
signal max_tries_reached : std_logic := '0';
signal rst_tries : std_logic := '0';

-- dictionary signals
signal qspo_signal : STD_LOGIC_VECTOR(39 DOWNTO 0);
signal qspo_ce_signal : STD_LOGIC;
--==========================================================================
===========

begin
--==========================================================================
===========
--Port Mapping (wiring the component blocks together):
--==========================================================================
===========
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
--Wire the load_word sub-component to the CheckWordExists sub-component:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
word_exist_block : WORD_EXISTS
   Port map(
```

```
        -- inputs
        clk_in          => clk,
        guess           => guess_sig,
        word_ready      => word_ready_sig,

        -- outputs
        is_dict_word    => in_dict_sig,
        not_in_dict     => not_in_dict_sig
    );

word_loader : load_word
    port map (
        --Inputs:
        clk             => clk,
        letter_in       => Rx_data,
        new_letter      => Rx_done,

        --Outputs:
        full            => open,
        char_disp_out       => char_disp_out_sig,
        char_disp_out_ready => char_disp_out_ready_sig,
        word_ready          => word_ready_sig,

        -- Output: Word characters (5 letter word)
--      char_0              => guess_sig(7 downto 0),
--      char_1              => guess_sig(15 downto 8),
--      char_2              => guess_sig(23 downto 16),
--      char_3              => guess_sig(31 downto 24),
--      char_4              => guess_sig(39 downto 32)

        char_0              => char_0_signal,
        char_1              => char_1_signal,
        char_2              => char_2_signal,
        char_3              => char_3_signal,
        char_4              => char_4_signal
    );

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Wire the guess_checker sub-component:
```

```
--++++++
```

```
begin
    if rising_edge(clk) then
        sol_addr <= sol_addr + 1;
        if sol_addr = max_dict_word then
            sol_addr <= "00000000000000";
        end if;
    end if;
end process count;


--=+=+=+
```

```
    if rst_tries = '1' then
        num_tries <= 0;
        max_tries_reached <= '0';
    end if;

end process CountTries;

--=+=+=+
```

```
=+=+=+=+=+=+=+=+
StateUpdate: process (clk)
begin
    if rising_edge(clk) then
        current_state <= next_state;
    end if;
end process StateUpdate;


NextStateLogic: process (current_state, waitcount, char_disp_out_ready_sig, colors_ready,
win_all_green, max_tries_reached, sent)
begin
next_state <= current_state;

case current_state is
    when newGame =>
        if char_disp_out_ready_sig = '1' then
            next_state <= displayLetters;
        end if;
    when idle =>
        if char_disp_out_ready_sig = '1' then
            next_state <= displayLetters;
        elsif colors_ready = '1' then
            next_state <= displayColors;
        end if;
    when displayLetters =>
        if sent = '1' then
            if colors_ready = '1' then
                next_state <= displayColors;
            else
                next_state <= idle;
            end if;
        end if;
    when displayColors =>
        if sent = '1' then
            if win_all_green = '1' then
                next_state <= win;
            elsif max_tries_reached = '1' then
                next_state <= lose;
            else next_state <= idle;
```

```
            end if;
        end if;
    when win =>
        if sent = '1' then
            next_state <= newGame;
        end if;
    when lose =>
        if sent = '1' then
            next_state <= newGame;
        end if;
    when others =>  next_state <= newGame;

end case current_state;

end process NextStateLogic;

OutputLogic: process(clk, current_state, char_disp_out_sig, cltrs, cplaces, data_to_send,
solution_sig, qspo_signal)
begin
    rst_tries <= '0';
    data_ready <= '0';
    qspo_ce_signal <= '0';
    data_to_send <= (others => '0');

    if rising_edge(clk) then
        data_ready_next <= data_ready;
    end if;
    data_ready_mp <= not data_ready_next and data_ready;

    case current_state is
        when newGame =>
            qspo_ce_signal <= '1';
            -- send output
            if rising_edge(clk) then
                solution_sig <= qspo_signal;
            end if;
        when idle =>
            -- no output
        when displayLetters =>
            -- send output
```

```
            if char_disp_out_sig = backspace then
                data_to_send <= (79 downto 0 => '0') & delete & backspace;
            else
                data_to_send <= (87 downto 0 => '0') & char_disp_out_sig;
            end if;
            data_ready <= '1';
        when displayColors =>
            -- send output
            for ltr_start in 0 to max_ltr_idx loop
                if cltrs(ltr_start) = '0' and cplaces(ltr_start) = '0' then
                    data_to_send(ltr_start*byte_size + 7 downto ltr_start*byte_size) <= black_sym;
                elsif cltrs(ltr_start) = '1' and cplaces(ltr_start) = '0' then
                    data_to_send(ltr_start*byte_size + 7 downto ltr_start*byte_size) <= yellow_sym;
                else data_to_send(ltr_start*byte_size + 7 downto ltr_start*byte_size) <=
solution_sig(ltr_start*byte_size + 7 downto ltr_start*byte_size);
                end if;
            end loop;
            data_to_send(95 downto 40) <= (47 downto 0 => '0') & enter;
            data_ready <= '1';
        when win =>
            data_to_send <= win_out;
            data_ready <= '1';
            rst_tries <= '1';
        when lose =>
            data_to_send <= enter & solution_sig & lose_out;
            data_ready <= '1';
            rst_tries <= '1';
        when others =>
            -- send nothing
    end case current_state;

end process OutputLogic;


end Behavioral;
```

Load Word Design

```
--=================================================================
===========
```

```
--ENGS 31/ CoSc 56 22S
--Final Project - Wordle
--B.L. Dobbins, E.W. Hansen, Professor Luke
--Mubarak Idoko, Ikeoluwa Abioye, Lobna Jbenaini
    --Load Word Logic for Wordle

-- TODO: (DO NOT REMOVE) Write to "Update Log" if you modify the code, document
your changes.
-- TODO: (DO NOT REMOVE) Update known issues to account for fixes of newly
discovered bugs.

-- UPDATE LOG:
-- Format: (Name) Date: Notes
--==================================================================
===========
    -- (Mubbie) 06/252022: Created the file, implemented functionality
--==================================================================
===========

-- KNOWN ISSUES:
    -- Known issues with implementation that have not been fixes
    -- Critical Score (0-5):
        -- subjective measure
        -- how important fixing this issue is important to the proper functioning of the system
    -- Ideas for fix: thoughts on how the issue could be fixed
-- Format: (Name) Date [Critical Score (0-5)]: Notes [Ideas for fix] {Updates (if any)}
--==================================================================
===========

--==================================================================
===========



--==================================================================
===========
--Library Declarations:
--==================================================================
===========
library IEEE;
use IEEE.std_logic_1164.all;
```

```
use IEEE.numeric_std.all;
use ieee.math_real.all;
library UNISIM;
use UNISIM.VComponents.all;


--==================================================================================
--Entity Declaration:
--==================================================================================
entity load_word is
   port (
      --Inputs:
      clk                 : in std_logic;
      letter_in           : in STD_LOGIC_VECTOR(7 downto 0);
      new_letter          : in std_logic;

      --Outputs:
      full                :  out std_logic;
      char_disp_out       :  out std_logic_vector(7 downto 0);
      char_disp_out_ready :  out std_logic;
      word_ready          :  out std_logic;

      -- Output: Word characters (5 letter word)
      char_0              :  out std_logic_vector(7 downto 0);
      char_1              :  out std_logic_vector(7 downto 0);
      char_2              :  out std_logic_vector(7 downto 0);
      char_3              :  out std_logic_vector(7 downto 0);
      char_4              :  out std_logic_vector(7 downto 0)
   );
end entity load_word;


--==================================================================================
--Architecture Type:
--==================================================================================
architecture behavioral_architecture of load_word is
```

```
--==========================================================================
===========
--Sub-Component Declarations:
--==========================================================================
===========
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++
--Check Letter Sub-Component:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++
component Check_Letter IS
   -- ports
   port (
         -- inputs
         letter    :         in        STD_LOGIC_VECTOR(7 downto 0);

         -- outputs
         is_backspace                    :         out STD_LOGIC;
         is_valid_alpha         : out STD_LOGIC;
         is_enter              :      out STD_LOGIC;

         -- lower case output
         output_lower_case_letter :     out STD_LOGIC_VECTOR(7 downto 0)
   );
end component Check_Letter;

--==========================================================================
===========
--Signal Declarations:
--==========================================================================
===========
-- signal declarations for character nature
signal is_backspace_out : std_logic := '0';
signal is_valid_alpha_out : std_logic := '0';
signal is_enter_out : std_logic := '0';

-- signal declaration for normalized word
signal normalized_char : std_logic_vector(7 downto 0) := (others => '0');
```

```
-- signal declarations for word size
signal word_size : unsigned(2 downto 0) := (others => '0');
signal word_full : std_logic := '0';
signal next_char_index : integer := 0;

-- signal declarations for word characters ( registers)
-- character 0 -> 4
type word_chars is array (0 to 4) of std_logic_vector(7 downto 0);
signal word_chars_reg : word_chars := (others => (others => '0'));
signal char_0_out : std_logic_vector(7 downto 0) := (others => '0');
signal char_1_out : std_logic_vector(7 downto 0) := (others => '0');
signal char_2_out : std_logic_vector(7 downto 0) := (others => '0');
signal char_3_out : std_logic_vector(7 downto 0) := (others => '0');
signal char_4_out : std_logic_vector(7 downto 0) := (others => '0');

-- word output control signals
signal word_ready_out_0 : std_logic := '0';
signal word_ready_out_1 : std_logic := '0';
signal word_ready_out   : std_logic := '0';

-- display output signals
signal char_disp_out_ready_out : std_logic := '0';

begin
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
--Wire the check letter block:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
check_letter_block : Check_Letter
   port map (
      letter => letter_in,                        -- letter into combinational block
      is_backspace => is_backspace_out,           -- letter is backspace
      is_valid_alpha => is_valid_alpha_out,       -- letter is valid alphabetic character
      is_enter => is_enter_out,                   -- letter is enter  (new line)
      output_lower_case_letter => normalized_char     -- lower case letter
   );

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++
```

```
--Word Size Logic:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Word_Size_Logic : process(clk, word_size)
begin
    -- counter for word size
    if rising_edge(clk) then
        -- add one character to the word size
        -- new character that is valid alpha
        if new_letter = '1' and is_valid_alpha_out = '1' then
            if word_size < "101" then
                word_size <= word_size + 1;
            end if;
        elsif new_letter = '1' and is_backspace_out = '1' then
            -- remove one character to the word size
            -- new character that is backspace
            word_size <= word_size - 1;
        elsif word_ready_out = '1' then
            word_size <= "000";
        end if;
    end if;

    -- maximum number of characters
    if word_size = "101" then
        word_full <= '1';
    else
        word_full <= '0';
    end if;

end process Word_Size_Logic;

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Word Array Logic:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Word_Array_Logic : process(clk)
begin
    -- load a character into the word array
    if rising_edge(clk) then
```

```
        -- new character that is valid alpha
        if new_letter = '1' and is_valid_alpha_out = '1' then
            -- load the character into the word array
            word_chars_reg(next_char_index) <= normalized_char;

            -- increment the next character index
            next_char_index <= next_char_index + 1;
        elsif new_letter = '1' and is_backspace_out = '1' then
            -- remove one character from the word array
            next_char_index <= next_char_index - 1;
        elsif word_ready_out = '1' then
            next_char_index <= 0; -- reset for next word use
        end if;
    end if;
end process Word_Array_Logic;

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
--Word Output Logic:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Word_Output_Logic : process(clk, is_enter_out, word_full, word_ready_out_1,
word_ready_out_0, is_backspace_out, new_letter, is_valid_alpha_out)
begin
    if rising_edge(clk) then
        -- pass signal through register
        word_ready_out_1 <= word_ready_out_0;

        -- if the word is ready send the output
        if word_ready_out = '1' then
            -- pass the word characters through the registers
            char_0_out <= word_chars_reg(0);
            char_1_out <= word_chars_reg(1);
            char_2_out <= word_chars_reg(2);
            char_3_out <= word_chars_reg(3);
            char_4_out <= word_chars_reg(4);
        end if;
    end if;

    -- prepare word ready signal monopulse
```

```
    word_ready_out_0 <= is_enter_out and word_full;
    word_ready_out <= not(word_ready_out_1) and word_ready_out_0;

    -- display output signals
    char_disp_out_ready_out <= ((is_enter_out and word_full) or is_backspace_out or
(new_letter and is_valid_alpha_out)) and new_letter;
end process Word_Output_Logic;

--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++
--Block Output:
--++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++
word_ready <= word_ready_out;
char_0 <= char_0_out;
char_1 <= char_1_out;
char_2 <= char_2_out;
char_3 <= char_3_out;
char_4 <= char_4_out;
char_disp_out <= normalized_char;
char_disp_out_ready <= char_disp_out_ready_out;
full <= word_full;

end behavioral_architecture;
```

WORDLE ROM Design

```
-- (c) Copyright 1995-2022 Xilinx, Inc. All rights reserved.
--
-- This file contains confidential and proprietary information
-- of Xilinx, Inc. and is protected under U.S. and
-- international copyright and other intellectual property
-- laws.
--
-- DISCLAIMER
-- This disclaimer is not a license and does not grant any
-- rights to the materials distributed herewith. Except as
-- otherwise provided in a valid license issued to you by
-- Xilinx, and to the maximum extent permitted by applicable
```

-- law: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND
-- WITH ALL FAULTS, AND XILINX HEREBY DISCLAIMS ALL WARRANTIES
-- AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING
-- BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-
-- INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and
-- (2) Xilinx shall not be liable (whether in contract or tort,
-- including negligence, or under any other theory of
-- liability) for any loss or damage of any kind or nature
-- related to, arising under or in connection with these
-- materials, including for any direct, or any indirect,
-- special, incidental, or consequential loss or damage
-- (including loss of data, profits, goodwill, or any type of
-- loss or damage suffered as a result of any action brought
-- by a third party) even if such damage or loss was
-- reasonably foreseeable or Xilinx had been advised of the
-- possibility of the same.
--
-- CRITICAL APPLICATIONS
-- Xilinx products are not designed or intended to be fail-
-- safe, or for use in any application requiring fail-safe
-- performance, such as life-support or safety devices or
-- systems, Class III medical devices, nuclear facilities,
-- applications related to the deployment of airbags, or any
-- other applications that could lead to death, personal
-- injury, or severe property or environmental damage
-- (individually and collectively, "Critical
-- Applications"). Customer assumes the sole risk and
-- liability of any use of Xilinx products in Critical
-- Applications, subject only to applicable laws and
-- regulations governing limitations on product liability.
--
-- THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS
-- PART OF THIS FILE AT ALL TIMES.
--
-- DO NOT MODIFY THIS FILE.

-- IP VLNV: xilinx.com:ip:dist_mem_gen:8.0
-- IP Revision: 12

LIBRARY ieee;

```
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

LIBRARY dist_mem_gen_v8_0_12;
USE dist_mem_gen_v8_0_12.dist_mem_gen_v8_0_12;

ENTITY wordle_dictionary_rom IS
  PORT (
    a : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
    clk : IN STD_LOGIC;
    qspo_ce : IN STD_LOGIC;
    qspo : OUT STD_LOGIC_VECTOR(39 DOWNTO 0)
  );
END wordle_dictionary_rom;

ARCHITECTURE wordle_dictionary_rom_arch OF wordle_dictionary_rom IS
  ATTRIBUTE DowngradeIPIdentifiedWarnings : STRING;
  ATTRIBUTE DowngradeIPIdentifiedWarnings OF wordle_dictionary_rom_arch:
ARCHITECTURE IS "yes";
  COMPONENT dist_mem_gen_v8_0_12 IS
    GENERIC (
      C_FAMILY : STRING;
      C_ADDR_WIDTH : INTEGER;
      C_DEFAULT_DATA : STRING;
      C_DEPTH : INTEGER;
      C_HAS_CLK : INTEGER;
      C_HAS_D : INTEGER;
      C_HAS_DPO : INTEGER;
      C_HAS_DPRA : INTEGER;
      C_HAS_I_CE : INTEGER;
      C_HAS_QDPO : INTEGER;
      C_HAS_QDPO_CE : INTEGER;
      C_HAS_QDPO_CLK : INTEGER;
      C_HAS_QDPO_RST : INTEGER;
      C_HAS_QDPO_SRST : INTEGER;
      C_HAS_QSPO : INTEGER;
      C_HAS_QSPO_CE : INTEGER;
      C_HAS_QSPO_RST : INTEGER;
      C_HAS_QSPO_SRST : INTEGER;
      C_HAS_SPO : INTEGER;
```

```
      C_HAS_WE : INTEGER;
      C_MEM_INIT_FILE : STRING;
      C_ELABORATION_DIR : STRING;
      C_MEM_TYPE : INTEGER;
      C_PIPELINE_STAGES : INTEGER;
      C_QCE_JOINED : INTEGER;
      C_QUALIFY_WE : INTEGER;
      C_READ_MIF : INTEGER;
      C_REG_A_D_INPUTS : INTEGER;
      C_REG_DPRA_INPUT : INTEGER;
      C_SYNC_ENABLE : INTEGER;
      C_WIDTH : INTEGER;
      C_PARSER_TYPE : INTEGER
    );
    PORT (
      a : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
      d : IN STD_LOGIC_VECTOR(39 DOWNTO 0);
      dpra : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
      clk : IN STD_LOGIC;
      we : IN STD_LOGIC;
      i_ce : IN STD_LOGIC;
      qspo_ce : IN STD_LOGIC;
      qdpo_ce : IN STD_LOGIC;
      qdpo_clk : IN STD_LOGIC;
      qspo_rst : IN STD_LOGIC;
      qdpo_rst : IN STD_LOGIC;
      qspo_srst : IN STD_LOGIC;
      qdpo_srst : IN STD_LOGIC;
      spo : OUT STD_LOGIC_VECTOR(39 DOWNTO 0);
      dpo : OUT STD_LOGIC_VECTOR(39 DOWNTO 0);
      qspo : OUT STD_LOGIC_VECTOR(39 DOWNTO 0);
      qdpo : OUT STD_LOGIC_VECTOR(39 DOWNTO 0)
    );
  END COMPONENT dist_mem_gen_v8_0_12;
  ATTRIBUTE X_CORE_INFO : STRING;
  ATTRIBUTE X_CORE_INFO OF wordle_dictionary_rom_arch: ARCHITECTURE IS
"dist_mem_gen_v8_0_12,Vivado 2018.3.1";
  ATTRIBUTE CHECK_LICENSE_TYPE : STRING;
  ATTRIBUTE CHECK_LICENSE_TYPE OF wordle_dictionary_rom_arch :
ARCHITECTURE IS "wordle_dictionary_rom,dist_mem_gen_v8_0_12,{}";
```

```
  ATTRIBUTE CORE_GENERATION_INFO : STRING;
  ATTRIBUTE CORE_GENERATION_INFO OF wordle_dictionary_rom_arch:
ARCHITECTURE IS "wordle_dictionary_rom,dist_mem_gen_v8_0_12,{x_ipProduct=Vivado
2018.3.1,x_ipVendor=xilinx.com,x_ipLibrary=ip,x_ipName=dist_mem_gen,x_ipVersion=8.0,
x_ipCoreRevision=12,x_ipLanguage=VHDL,x_ipSimLanguage=VHDL,C_FAMILY=artix7,C
_ADDR_WIDTH=14,C_DEFAULT_DATA=0,C_DEPTH=16384,C_HAS_CLK=1,C_HAS_D
=0,C_HAS_DPO=0,C_HAS_DPRA=0,C_HAS_I_CE=0,C_HAS_QDPO=0,C_HAS_QDPO_
CE=0,C_HAS_QDPO_CLK=0,C_HAS_QDPO_RST=0,C_HAS_QDPO_SRST=0,C_HAS_Q
SPO=1,C_HAS_QSPO_CE=1,C_HAS_QSPO_RST=0,C_HAS_QSPO_SRST=0,C_HAS_SP
O=0,C_HAS_WE=0,C_M" &
"EM_INIT_FILE=wordle_dictionary_rom.mif,C_ELABORATION_DIR=./,C_MEM_TYPE=
0,C_PIPELINE_STAGES=0,C_QCE_JOINED=0,C_QUALIFY_WE=0,C_READ_MIF=1,C_
REG_A_D_INPUTS=1,C_REG_DPRA_INPUT=0,C_SYNC_ENABLE=1,C_WIDTH=40,C_
PARSER_TYPE=1}";
BEGIN
  U0 : dist_mem_gen_v8_0_12
    GENERIC MAP (
      C_FAMILY => "artix7",
      C_ADDR_WIDTH => 14,
      C_DEFAULT_DATA => "0",
      C_DEPTH => 16384,
      C_HAS_CLK => 1,
      C_HAS_D => 0,
      C_HAS_DPO => 0,
      C_HAS_DPRA => 0,
      C_HAS_I_CE => 0,
      C_HAS_QDPO => 0,
      C_HAS_QDPO_CE => 0,
      C_HAS_QDPO_CLK => 0,
      C_HAS_QDPO_RST => 0,
      C_HAS_QDPO_SRST => 0,
      C_HAS_QSPO => 1,
      C_HAS_QSPO_CE => 1,
      C_HAS_QSPO_RST => 0,
      C_HAS_QSPO_SRST => 0,
      C_HAS_SPO => 0,
      C_HAS_WE => 0,
      C_MEM_INIT_FILE => "wordle_dictionary_rom.mif",
      C_ELABORATION_DIR => "./",
      C_MEM_TYPE => 0,
```

```
    C_PIPELINE_STAGES => 0,
    C_QCE_JOINED => 0,
    C_QUALIFY_WE => 0,
    C_READ_MIF => 1,
    C_REG_A_D_INPUTS => 1,
    C_REG_DPRA_INPUT => 0,
    C_SYNC_ENABLE => 1,
    C_WIDTH => 40,
    C_PARSER_TYPE => 1
  )
  PORT MAP (
    a => a,
    d => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 40)),
    dpra => STD_LOGIC_VECTOR(TO_UNSIGNED(0, 14)),
    clk => clk,
    we => '0',
    i_ce => '1',
    qspo_ce => qspo_ce,
    qdpo_ce => '1',
    qdpo_clk => '0',
    qspo_rst => '0',
    qdpo_rst => '0',
    qspo_srst => '0',
    qdpo_srst => '0',
    qspo => qspo
  );
END wordle_dictionary_rom_arch;
```

X. VHDL Code Testing

SCI Receiver Testbench

```
--==================================================================
===========
--ENGS 31/ CoSc 56 22S
--Final Project - Wordle
--B.L. Dobbins, E.W. Hansen, Professor Luke
--Mubarak Idoko, Ikeoluwa Abioye, Lobna Jbenaini
  --SCI Receiver Testbench
```

```
-- TODO: (DO NOT REMOVE) Write to "Update Log" if you modify the code, document
your changes.
-- TODO: (DO NOT REMOVE) Update known issues to account for fixes of newly
discovered bugs.

-- UPDATE LOG:
-- Format: (Name) Date: Notes
--==================================================================
===========
   -- (Mubbie) 05/25/2022: Created the file, implemented testbench functionality
   -- (Mubbie) 05/28/2022: Finished implementing testbench functionality
--==================================================================
===========

-- KNOWN ISSUES:
   -- Known issues with implementation that have not been fixes
   -- Critical Score (0-5):
      -- subjective measure
      -- how important fixing this issue is important to the proper functioning of the system
   -- Ideas for fix: thoughts on how the issue could be fixed
-- Format: (Name) Date [Critical Score (0-5)]: Notes [Ideas for fix] {Updates (if any)}
--==================================================================
===========

--==================================================================
===========



--==================================================================
===========
--Library Declarations:
--==================================================================
===========
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use ieee.math_real.all;
```

```
--==========================================================================
===========
--Entity Declaration:
--==========================================================================
===========
entity SCI_RECEIVER_tb is
   end SCI_RECEIVER_tb;


--==========================================================================
===========
--Architecture Type:
--==========================================================================
===========
architecture testbench of SCI_RECEIVER_tb is

--==========================================================================
===========
--Component Declaration:
--==========================================================================
===========
component SCI_RECEIVER is
   -- constants
   generic(
      BAUD_COUNTER_TOP : integer;
      BIT_COUNTER_TOP : integer
   );

   -- ports
   port (
      -- inputs
      clk : in std_logic;
      Rx: in std_logic;

      -- outputs
      Rx_Data : out std_logic_vector(7 downto 0);
      Rx_Done : out std_logic;
      Rx_Error : out std_logic
   );
end component SCI_RECEIVER;
```

```
--==========================================================================
===========
--Signal and Constant Declarations:
--==========================================================================
===========
-- inputs
signal clk_signal : std_logic := '0';
signal Rx_signal : std_logic := '1'; -- start at 1 i.e. not receiving (testbench transmitter is idle
initially, so receiver is just waiting)

-- outputs
signal Rx_Data_signal : std_logic_vector(7 downto 0) := (others => '1'); -- all ones initially
signal Rx_Done_signal : std_logic := '0';
signal Rx_Error_signal : std_logic := '0';

-- constants
constant clock_period : time := 10 ns; -- 100 MHz clock

-- BAUD AND BIT COUNTER TOP
-- TODO: Modify as necessary for test
constant BAUD_COUNTER_TOP_constant : integer := 10417; -- 96 Baud Rate
constant BIT_COUNTER_TOP_constant : integer := 10; -- shift in 10 bits: 1 start bit, 8 data
bits, 1 stop bit

--==========================================================================
===========
--Test Signal Declarations:
--==========================================================================
===========
type RegisterFile is array (0 to 6) of std_logic_vector(9 downto 0);
signal TestCharactersArray : RegisterFile :=
   (
      "0100000101",
      "0010000101",
      "0100111101",
      "0010111101",
      "0100000100",
      "0110011010",
```

```
        "0001011010"
    );

-- TESTBENCH
begin
-- unit under test
uut : SCI_RECEIVER
    GENERIC MAP (
        -- generics
        BAUD_COUNTER_TOP => BAUD_COUNTER_TOP_constant,
        BIT_COUNTER_TOP => BIT_COUNTER_TOP_constant
    )
    PORT MAP(
        -- inputs
        clk => clk_signal,
        Rx => Rx_signal,

        -- outputs
        Rx_Data => Rx_Data_signal,
        Rx_Done => Rx_Done_signal,
        Rx_Error => Rx_Error_signal
    );

--==========================================================================
===========
--Clock Generation Process:
--==========================================================================
===========
clk_proc : process
BEGIN
  clk_signal <= '0';
  wait for clock_period/2;   --100 MHz clock

  clk_signal <= '1';
  wait for clock_period/2;
END PROCESS clk_proc;

--==========================================================================
===========
--Stimulus Process:
```

```
--==========================================================================
===========
stimulus_process: process
begin
   -- iteratively set Rx to the next character in the array chose and wait for the sampling to
complete
   -- then send the next character
   -- this is done without using a loop
   wait for clock_period*10;

   -- SEND DATA PACKETS FOR TEST
   -- FIRST 7 are valid
   -- LAST 3 are not

   -- send the valid data packets
   for j in 0 to 3 loop
      for i in 0 to 9 loop
         -- send the bits for the packet
         Rx_signal <= TestCharactersArray(j)(9-i);
         wait for clock_period*BAUD_COUNTER_TOP_constant; -- wait for the baud time
      end loop;
      wait for clock_period*BAUD_COUNTER_TOP_constant;
   end loop;

   -- send the invalid data packets
   Rx_signal <= '1';
   wait for clock_period*20;
   for j in 4 to 6 loop
      for i in 0 to 9 loop
         -- send the bits for the packet
         Rx_signal <= TestCharactersArray(j)(9-i);
         if i = 9 then
            wait for clock_period*5250; -- wait for less than the baud time to prevent confusion
with start bit
         else
            wait for clock_period*BAUD_COUNTER_TOP_constant; -- wait for the baud time
         end if;
      end loop;
      Rx_signal <= '1';
      wait for clock_period*BAUD_COUNTER_TOP_constant;
```

```
    end loop;

    -- wait
    wait;
end process stimulus_process;

end testbench;
```

SCI Transmitter Testbench

```
--==================================================================
===========
--ENGS 31/ CoSc 56 22S
--Final Project - Wordle
--B.L. Dobbins, E.W. Hansen, Professor Luke
--Mubarak Idoko, Ikeoluwa Abioye, Lobna Jbenaini
    --SCI Receiver Testbench

-- TODO: (DO NOT REMOVE) Write to "Update Log" if you modify the code, document
your changes.
-- TODO: (DO NOT REMOVE) Update known issues to account for fixes of newly
discovered bugs.

-- UPDATE LOG:
-- Format: (Name) Date: Notes
--==================================================================
===========
    -- (Mubbie) 05/29/2022: Wrote the files
--==================================================================
===========

-- KNOWN ISSUES:
    -- Known issues with implementation that have not been fixes
    -- Critical Score (0-5):
        -- subjective measure
        -- how important fixing this issue is important to the proper functioning of the system
    -- Ideas for fix: thoughts on how the issue could be fixed
-- Format: (Name) Date [Critical Score (0-5)]: Notes [Ideas for fix] {Updates (if any)}
--==================================================================
```

```
===========

--=============================================================================
===========

--=============================================================================
===========
--Library Declarations:
--=============================================================================
===========
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use ieee.math_real.all;


--=============================================================================
===========
--Entity Declaration:
--=============================================================================
===========
entity SCI_RX_TX_SHELL_tb is
   end SCI_RX_TX_SHELL_tb;


--=============================================================================
===========
--Architecture Type:
--=============================================================================
===========
architecture testbench of SCI_RX_TX_SHELL_tb is


--=============================================================================
===========
--Component Declaration:
--=============================================================================
===========
component rx_tx_shell is
   port (
```

```
        --Inputs:
        -- clock
        clk_ext_port : in std_logic;

        -- Tx and Rx lines
        RsTx_ext_port : out std_logic;
        RsRx_ext_port : in std_logic;

        --
        Rx_data_port : out std_logic_vector(7 downto 0)
    );
end component rx_tx_shell;

--==================================================================
===========
--Signal and Constant Declarations:
--==================================================================
===========
-- inputs
signal clk_ext_signal : std_logic := '0';
signal RsRx_ext_signal : std_logic := '1';

-- outputs
signal RsTx_ext_signal : std_logic := '1';
signal Rx_data_ext_signal : std_logic_vector(7 downto 0) := (others => '1');

-- data to be transmitted: including start and stop bit
signal data_to_transmit_ASCII_A : std_logic_vector(9 downto 0) := "0100000101";

-- constants
constant clock_ext_period : time := 10 ns; -- 100 MHz clock

begin
-- unit under test
uut: rx_tx_shell
    port map (
        clk_ext_port => clk_ext_signal,
        RsTx_ext_port => RsTx_ext_signal,
        RsRx_ext_port => RsRx_ext_signal,
        Rx_data_port => Rx_data_ext_signal
```

```
    );

--==========================================================================
===========
--Clock Generation Process:
--==========================================================================
===========
clk_proc : process
BEGIN
  clk_ext_signal <= '0';
  wait for clock_ext_period/2;   --100 MHz clock

  clk_ext_signal <= '1';
  wait for clock_ext_period/2;
END PROCESS clk_proc;

--==========================================================================
===========
--Stimulus Process:
--==========================================================================
===========
stimulus_process: process
begin
    -- wait before starting
    wait for clock_ext_period*10;

    -- send the data packet, bit by bit
    for i in 0 to 9 loop
        RsRx_ext_signal <= data_to_transmit_ASCII_A(9-i);
        wait for clock_ext_period*10417; -- 65000 baudrate
    end loop;
    RsRx_ext_signal <= '1'; -- return to idle state

    wait;
end process stimulus_process;

end testbench;
```

SCI RX TX Testbench

```
--==========================================================================
===========
--ENGS 31/ CoSc 56 22S
--Final Project - Wordle
--B.L. Dobbins, E.W. Hansen, Professor Luke
--Mubarak Idoko, Ikeoluwa Abioye, Lobna Jbenaini
    --SCI Receiver Testbench

-- TODO: (DO NOT REMOVE) Write to "Update Log" if you modify the code, document
your changes.
-- TODO: (DO NOT REMOVE) Update known issues to account for fixes of newly
discovered bugs.

-- UPDATE LOG:
-- Format: (Name) Date: Notes
--==========================================================================
===========
    -- (Mubbie) 05/29/2022: Wrote the files
--==========================================================================
===========

-- KNOWN ISSUES:
    -- Known issues with implementation that have not been fixes
    -- Critical Score (0-5):
        -- subjective measure
        -- how important fixing this issue is important to the proper functioning of the system
    -- Ideas for fix: thoughts on how the issue could be fixed
-- Format: (Name) Date [Critical Score (0-5)]: Notes [Ideas for fix] {Updates (if any)}
--==========================================================================
===========

--==========================================================================
===========

--==========================================================================
===========
```

```
--Library Declarations:
--==========================================================================
===========
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use ieee.math_real.all;


--==========================================================================
===========
--Entity Declaration:
--==========================================================================
===========
entity SCI_RX_TX_SHELL_tb is
   end SCI_RX_TX_SHELL_tb;


--==========================================================================
===========
--Architecture Type:
--==========================================================================
===========
architecture testbench of SCI_RX_TX_SHELL_tb is


--==========================================================================
===========
--Component Declaration:
--==========================================================================
===========
component rx_tx_shell is
   port (
      --Inputs:
      -- clock
      clk_ext_port : in std_logic;

      -- Tx and Rx lines
      RsTx_ext_port : out std_logic;
      RsRx_ext_port : in std_logic;
```

```
        --
        Rx_data_port : out std_logic_vector(7 downto 0)
    );
end component rx_tx_shell;

--==================================================================
===========
--Signal and Constant Declarations:
--==================================================================
===========
-- inputs
signal clk_ext_signal : std_logic := '0';
signal RsRx_ext_signal : std_logic := '1';

-- outputs
signal RsTx_ext_signal : std_logic := '1';
signal Rx_data_ext_signal : std_logic_vector(7 downto 0) := (others => '1');

-- data to be transmitted: including start and stop bit
signal data_to_transmit_ASCII_A : std_logic_vector(9 downto 0) := "0100000101";

-- constants
constant clock_ext_period : time := 10 ns; -- 100 MHz clock

begin
-- unit under test
uut: rx_tx_shell
    port map (
        clk_ext_port => clk_ext_signal,
        RsTx_ext_port => RsTx_ext_signal,
        RsRx_ext_port => RsRx_ext_signal,
        Rx_data_port => Rx_data_ext_signal
    );

--==================================================================
===========
--Clock Generation Process:
--==================================================================
===========
```

```
clk_proc : process
BEGIN
  clk_ext_signal <= '0';
  wait for clock_ext_period/2;   --100 MHz clock

  clk_ext_signal <= '1';
  wait for clock_ext_period/2;
END PROCESS clk_proc;

--==========================================================================
===========
--Stimulus Process:
--==========================================================================
===========
stimulus_process: process
begin
    -- wait before starting
    wait for clock_ext_period*10;

    -- send the data packet, bit by bit
    for i in 0 to 9 loop
        RsRx_ext_signal <= data_to_transmit_ASCII_A(9-i);
        wait for clock_ext_period*10417; -- 65000 baudrate
    end loop;
    RsRx_ext_signal <= '1'; -- return to idle state

    wait;
end process stimulus_process;

end testbench;
```

Check Guess Logic Testbench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```vhdl
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity check_guess_tb is
end check_guess_tb;

architecture testbench of check_guess_tb is

component CheckGuess
   Port ( clk            : in STD_LOGIC;
          guess          : in STD_LOGIC_VECTOR(39 DOWNTO 0);
          solution       : in STD_LOGIC_VECTOR(39 DOWNTO 0);
          is_dict_word   : in STD_LOGIC;

          correct_letters  : out STD_LOGIC_VECTOR(4 DOWNTO 0);
          correct_places   : out STD_LOGIC_VECTOR(4 DOWNTO 0);
          win            : out STD_LOGIC;
          done           : out STD_LOGIC     );
end component;

-- signals
SIGNAL clk            : STD_LOGIC := '0';
SIGNAL guess          : STD_LOGIC_VECTOR(39 DOWNTO 0);
SIGNAL solution       : STD_LOGIC_VECTOR(39 DOWNTO 0);
SIGNAL is_dict_word   : STD_LOGIC;

SIGNAL correct_letters  : STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL correct_places   : STD_LOGIC_VECTOR(4 DOWNTO 0);
SIGNAL done           : STD_LOGIC;
SIGNAL win            : STD_LOGIC;

constant clk_period : time := 20 ns;
```

```
begin
uut: CheckGuess port map(
    clk            => clk,
    guess          => guess,
    solution       => solution,
    is_dict_word   => is_dict_word,
    correct_letters => correct_letters,
    correct_places  => correct_places,
    win            => win,
    done           => done    );

stim_proc : process
begin
    wait for clk_period/2;
    clk <= not clk;
end process stim_proc;

testing : process
begin
wait for clk_period/2;
solution <= "01100111011100100110010101100001011101 00";    -- great
guess <= "01100001011100000111000001101100011001 01";       -- apple
is_dict_word <= '1';
wait for clk_period;
is_dict_word <= '0';

wait for clk_period*100;

guess <= "01101000011001010110010101100100011100 11";       -- heeds
is_dict_word <= '1';
wait for clk_period;
is_dict_word <= '0';

wait for clk_period*50;

guess <= "01100111011100100110000101110100011001 01";       -- grate
is_dict_word <= '1';
wait for clk_period;
is_dict_word <= '0';
```

```
wait for clk_period*50;

guess <= "01100111011100100110010101110000101110100";        -- great
is_dict_word <= '1';
wait for clk_period;
is_dict_word <= '0';

wait for clk_period*50;

wait;
end process testing;

end testbench;
```

Check Letter Testbench

```
--==================================================================
===========
--ENGS 31/ CoSc 56 22S
--Final Project - Wordle
--B.L. Dobbins, E.W. Hansen, Professor Luke
--Mubarak Idoko, Ikeoluwa Abioye, Lobna Jbenaini
    --SCI Receiver Testbench

-- TODO: (DO NOT REMOVE) Write to "Update Log" if you modify the code, document
your changes.
-- TODO: (DO NOT REMOVE) Update known issues to account for fixes of newly
discovered bugs.

-- UPDATE LOG:
-- Format: (Name) Date: Notes
--==================================================================
===========
    -- (Mubbie) 05/31/2022: Created the file, implemented testbench functionality
--==================================================================
```

```
===========

-- KNOWN ISSUES:
   -- Known issues with implementation that have not been fixes
   -- Critical Score (0-5):
      -- subjective measure
      -- how important fixing this issue is important to the proper functioning of the system
   -- Ideas for fix: thoughts on how the issue could be fixed
-- Format: (Name) Date [Critical Score (0-5)]: Notes [Ideas for fix] {Updates (if any)}
--=============================================================================
===========

--=============================================================================
===========



--=============================================================================
===========
--Library Declarations:
--=============================================================================
===========
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use ieee.math_real.all;


--=============================================================================
===========
--Entity Declaration:
--=============================================================================
===========
entity CHECK_LETTER_tb is
   end CHECK_LETTER_tb;

--=============================================================================
===========
--Architecture Type:
--=============================================================================
===========
```

```
architecture testbench of CHECK_LETTER_tb is


--==============================================================================
===========
--Component Declaration:
--==============================================================================
===========
component Check_Letter IS
   -- ports
   port (
         -- inputs
         letter    :         in        STD_LOGIC_VECTOR(7 downto 0);

         -- outputs
         is_backspace                        :         out STD_LOGIC;
         is_valid_alpha          : out STD_LOGIC;
         is_enter                :      out STD_LOGIC;

         -- lower case output
         output_lower_case_letter :     out STD_LOGIC_VECTOR(7 downto 0)
   );
end component Check_Letter;

--==============================================================================
===========
--Signal and Constant Declarations:
--==============================================================================
===========
-- inputs
signal clk_signal : std_logic := '0';
signal letter_signal : std_logic_vector(7 downto 0) := (others => '1');

-- outputs
signal is_backspace_signal : std_logic := '0';
signal is_valid_alpha_signal : std_logic := '0';
signal is_enter_signal : std_logic := '0';
signal output_lower_case_letter_signal : std_logic_vector(7 downto 0) := (others => '1');

-- constants
```

```
constant clock_period : time := 10 ns; -- 100 MHz clock

-- TESTBENCH
begin
-- unit under test
uut: CHECK_LETTER
   PORT MAP (
      -- inputs
      letter => letter_signal,

      -- outputs
      is_backspace => is_backspace_signal,
      is_valid_alpha => is_valid_alpha_signal,
      is_enter => is_enter_signal,
      output_lower_case_letter => output_lower_case_letter_signal
   );

--==========================================================================
===========
--Clock Generation Process:
--==========================================================================
===========
clk_proc : process
BEGIN
  clk_signal <= '0';
  wait for clock_period/2;   --100 MHz clock

  clk_signal <= '1';
  wait for clock_period/2;
END PROCESS clk_proc;


--==========================================================================
===========
--Stimulus Process:
--==========================================================================
===========
stimulus_process: process
begin
   wait for clock_period*10;
```

```
    -- send uppercase A
    letter_signal <= "01000001";
    wait for clock_period*10;

    -- send lowercase z
    letter_signal <= "01111010";
    wait for clock_period*10;

    -- send backspace
    letter_signal <= "00001000";
    wait for clock_period*10;

    -- send enter
    letter_signal <= "00001101";
    wait for clock_period*10;

    wait;
end process stimulus_process;


end testbench;
```

FSM Testbench

```
--==================================================================
===========
--ENGS 31/ CoSc 56 22S
--Final Project - Wordle
--B.L. Dobbins, E.W. Hansen, Professor Luke
--Mubarak Idoko, Ikeoluwa Abioye, Lobna Jbenaini
    --SCI Receiver Testbench

-- TODO: (DO NOT REMOVE) Write to "Update Log" if you modify the code, document
your changes.
-- TODO: (DO NOT REMOVE) Update known issues to account for fixes of newly
discovered bugs.
```

```
-- UPDATE LOG:
-- Format: (Name) Date: Notes
--==================================================================================
    -- (Mubbie) 06/03/2022: Created the file, implemented testbench functionality
--==================================================================================

-- KNOWN ISSUES:
    -- Known issues with implementation that have not been fixes
    -- Critical Score (0-5):
        -- subjective measure
        -- how important fixing this issue is important to the proper functioning of the system
    -- Ideas for fix: thoughts on how the issue could be fixed
-- Format: (Name) Date [Critical Score (0-5)]: Notes [Ideas for fix] {Updates (if any)}
--==================================================================================

--==================================================================================



--==================================================================================
--Library Declarations:
--==================================================================================
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use ieee.math_real.all;


--==================================================================================
--Entity Declaration:
--==================================================================================
entity FSM_tb is
```

```
    end FSM_tb;


--==============================================================================
===========
--Architecture Type:
--==============================================================================
===========
architecture testbench of FSM_tb is


--==============================================================================
===========
--Component Declaration:
--==============================================================================
===========
component FSM is
    port ( clk            : in STD_LOGIC;
           Rx_data : in STD_LOGIC_VECTOR (7 downto 0);
           Rx_done : in STD_LOGIC;
           Tx_data : out STD_LOGIC_VECTOR (7 downto 0);
           Tx_data_ready : out STD_LOGIC
         );
end component FSM;

--==============================================================================
===========
--Signal and Constant Declarations:
--==============================================================================
===========
-- inputs
signal clk_signal : std_logic := '0';
signal Rx_data_signal : STD_LOGIC_VECTOR (7 downto 0) := (others => '1');
signal Rx_done_signal : STD_LOGIC := '0';

-- outputs
signal Tx_data_signal : STD_LOGIC_VECTOR (7 downto 0) := (others => '1');
signal Tx_data_ready_signal : STD_LOGIC := '0';

-- constants
```

```
constant clock_period : time := 10 ns; -- 100 MHz clock

--==========================================================================
===========
--Test Signal Declarations:
--==========================================================================
===========
type RegisterFile is array (0 to 21) of std_logic_vector(7 downto 0);
signal TestInputsArray : RegisterFile :=
   (
      "01000001", -- A
      "01101101", -- m
      "00001000", -- \b
      "00001000", -- \b
      "01000001", -- A
      "00110010", -- 2
      "00001101", -- \r
      "01001101", -- M
      "00110100", -- 4
      "00001101", -- \r
      "00001101", -- \r
      "01110000", -- p
      "01001100", -- L
      "00100001", -- !
      "00101010", -- *
      "00001000", -- \b
      "00001101", -- \r
      "01101100", -- l
      "01100101", -- e
      "00001000", -- \b
      "01000101", -- E
      "00001101"  -- \r
   );

begin
-- unit under test
uut : FSM
   PORT MAP(
      -- inputs
      clk => clk_signal,
```

```
        Rx_data => Rx_data_signal,
        Rx_done => Rx_done_signal,

        -- outputs
        Tx_data => Tx_Data_signal,
        Tx_data_ready => Tx_data_ready_signal
    );

--==========================================================================
===========
--Clock Generation Process:
--==========================================================================
===========
clk_proc : process
BEGIN
  clk_signal <= '0';
  wait for clock_period/2;   --100 MHz clock

  clk_signal <= '1';
  wait for clock_period/2;
END PROCESS clk_proc;

--==========================================================================
===========
--Stimulus Process:
--==========================================================================
===========
stimulus_process: process
begin
    -- iteratively write to the input signals
    wait for clock_period/2;
    wait for clock_period*4;

    -- send the data packets
    for i in 0 to 21 loop
        -- send character
        Rx_done_signal <= '1';
        Rx_data_signal <= TestInputsArray(i);
        wait for clock_period;
        Rx_done_signal <= '0';
```

```
        wait for clock_period*1000;
    end loop;

    Rx_data_signal <= (others => '1');

    wait;

end process stimulus_process;

end testbench;
```

Load Word Testbench

```
--==================================================================
===========
--ENGS 31/ CoSc 56 22S
--Final Project - Wordle
--B.L. Dobbins, E.W. Hansen, Professor Luke
--Mubarak Idoko, Ikeoluwa Abioye, Lobna Jbenaini
    --SCI Receiver Testbench

-- TODO: (DO NOT REMOVE) Write to "Update Log" if you modify the code, document
your changes.
-- TODO: (DO NOT REMOVE) Update known issues to account for fixes of newly
discovered bugs.

-- UPDATE LOG:
-- Format: (Name) Date: Notes
--==================================================================
===========
    -- (Mubbie) 05/25/2022: Created the file, implemented testbench functionality
--==================================================================
===========

-- KNOWN ISSUES:
```

```
    -- Known issues with implementation that have not been fixes
    -- Critical Score (0-5):
        -- subjective measure
        -- how important fixing this issue is important to the proper functioning of the system
    -- Ideas for fix: thoughts on how the issue could be fixed
-- Format: (Name) Date [Critical Score (0-5)]: Notes [Ideas for fix] {Updates (if any)}
--==================================================================================

--==================================================================================



--==================================================================================
--Library Declarations:
--==================================================================================
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use ieee.math_real.all;



--==================================================================================
--Entity Declaration:
--==================================================================================
entity LOAD_WORD_tb is
    end LOAD_WORD_tb;


--==================================================================================
--Architecture Type:
--==================================================================================
architecture testbench of LOAD_WORD_tb is
```

```
--==========================================================================
===========
--Component Declaration:
--==========================================================================
===========
component load_word is
   port (
      --Inputs:
      clk              : in std_logic;
      letter_in         : in STD_LOGIC_VECTOR(7 downto 0);
      new_letter        : in std_logic;

      --Outputs:
      full             :  out std_logic;
      char_disp_out      :  out std_logic_vector(7 downto 0);
      char_disp_out_ready :  out std_logic;
      word_ready         :  out std_logic;

      -- Output: Word characters (5 letter word)
      char_0            :  out std_logic_vector(7 downto 0);
      char_1            :  out std_logic_vector(7 downto 0);
      char_2            :  out std_logic_vector(7 downto 0);
      char_3            :  out std_logic_vector(7 downto 0);
      char_4            :  out std_logic_vector(7 downto 0)
   );
end component load_word;


--==========================================================================
===========
--Signal and Constant Declarations:
--==========================================================================
===========
-- inputs
signal clk_signal : std_logic := '0';
signal letter_in_signal : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
signal new_letter_signal : std_logic := '0';

-- outputs
```

```
signal full_signal : std_logic := '0';
signal char_disp_out_signal : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
signal char_disp_out_ready_signal : std_logic := '0';
signal word_ready_signal : std_logic := '0';
signal char_0_signal : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
signal char_1_signal : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
signal char_2_signal : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
signal char_3_signal : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');
signal char_4_signal : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');

-- constants
constant clock_period : time := 10 ns; -- 100 MHz clock

--==========================================================================
===========
--Test Signal Declarations:
--==========================================================================
===========
type RegisterFile is array (0 to 8) of std_logic_vector(7 downto 0);
signal TestCharactersArray : RegisterFile :=
   (
      "01000001", -- A
      "00001000", -- backspace
      "00001101", -- enter
      "01000001", -- A
      "01101101", -- m
      "01110000", -- p
      "01001100", -- L
      "01100101", -- e
      "00001101"  -- enter
   );

-- TESTBENCH
begin
-- unit under test
uut: load_word
   PORT MAP (
      -- inputs
      clk              => clk_signal,
      letter_in          => letter_in_signal,
```

```
        new_letter          => new_letter_signal,

        -- outputs
        full                => full_signal,
        char_disp_out       => char_disp_out_signal,
        char_disp_out_ready => char_disp_out_ready_signal,
        word_ready          => word_ready_signal,

         -- Output: Word characters (5 letter word)
        char_0              => char_0_signal,
        char_1              => char_1_signal,
        char_2              => char_2_signal,
        char_3              => char_3_signal,
        char_4              => char_4_signal
    );


--==========================================================================
===========
--Clock Generation Process:
--==========================================================================
===========
clk_proc : process
BEGIN
  clk_signal <= '0';
  wait for clock_period/2;   --100 MHz clock

  clk_signal <= '1';
  wait for clock_period/2;
END PROCESS clk_proc;


--==========================================================================
===========
--Stimulus Process:
--==========================================================================
===========
stimulus_process: process
begin
    -- iteratively write to the input signals
```

```
    wait for clock_period/2;

    -- send the data packets
    for i in 0 to 8 loop
        -- send character
        new_letter_signal <= '1';
        letter_in_signal <= TestCharactersArray(i);
        wait for clock_period;
        new_letter_signal <= '0';
        wait for clock_period*10;
    end loop;

    letter_in_signal <= "00000000";

    wait;
end process stimulus_process;

end testbench;
```

WORDLE ROM Testbench

```
--==================================================================
==========
--ENGS 31/ CoSc 56 22S
--Final Project - Wordle
--B.L. Dobbins, E.W. Hansen, Professor Luke
--Mubarak Idoko, Ikeoluwa Abioye, Lobna Jbenaini
    --Worlde ROM Testbench

-- TODO: (DO NOT REMOVE) Write to "Update Log" if you modify the code, document
your changes.
-- TODO: (DO NOT REMOVE) Update known issues to account for fixes of newly
discovered bugs.

-- UPDATE LOG:
-- Format: (Name) Date: Notes
--==================================================================
==========
```

```
    -- (Mubbie) 05/25/2022: Created the file, implemented testbench functionality
--==============================================================================
===========

-- KNOWN ISSUES:
    -- Known issues with implementation that have not been fixes
    -- Critical Score (0-5):
        -- subjective measure
        -- how important fixing this issue is important to the proper functioning of the system
    -- Ideas for fix: thoughts on how the issue could be fixed
-- Format: (Name) Date [Critical Score (0-5)]: Notes [Ideas for fix] {Updates (if any)}
--==============================================================================
===========


--==============================================================================
===========



--==============================================================================
===========
--Library Declarations:
--==============================================================================
===========
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use ieee.math_real.all;


--==============================================================================
===========
--Entity Declaration:
--==============================================================================
===========
entity WORDLE_DICTIONARY_ROM_tb is
    end WORDLE_DICTIONARY_ROM_tb;


--==============================================================================
===========
```

```
--Architecture Type:
--==========================================================================
===========
architecture testbench of WORDLE_DICTIONARY_ROM_tb is

--==========================================================================
===========
--Component Declaration:
--==========================================================================
===========
component wordle_dictionary_rom IS
PORT (
    a : IN STD_LOGIC_VECTOR(13 DOWNTO 0);
    clk : IN STD_LOGIC;
    qspo_ce : IN STD_LOGIC;
    qspo : OUT STD_LOGIC_VECTOR(39 DOWNTO 0)
  );
END component wordle_dictionary_rom;


--==========================================================================
===========
--Signal and Constant Declarations:
--==========================================================================
===========
-- inputs
signal clk_signal : std_logic := '0';
signal a_signal : STD_LOGIC_VECTOR(13 DOWNTO 0) := (others => '0');
signal qspo_ce_signal : STD_LOGIC := '0';

-- outputs
signal qspo_signal : STD_LOGIC_VECTOR(39 DOWNTO 0) := (others => '0');

-- constants
constant clock_period : time := 10 ns; -- 100 MHz clock

--==========================================================================
===========
--Test Signal Declarations:
--==========================================================================
```

```
===========
type RegisterFile is array (0 to 2) of std_logic_vector(13 downto 0);
signal TestAddressesArray : RegisterFile :=
    (
        "00000000001100",
        "10110011011001",
        "11001010101011"
    );


-- TESTBENCH
begin
uut : wordle_dictionary_rom
    PORT MAP (
        a => a_signal,
        clk => clk_signal,
        qspo_ce => qspo_ce_signal,
        qspo => qspo_signal
    );
--==========================================================================
===========
--Clock Generation Process:
--==========================================================================
=======
clk_proc : process
BEGIN
  clk_signal <= '0';
  wait for clock_period/2;   --100 MHz clock

  clk_signal <= '1';
  wait for clock_period/2;
END PROCESS clk_proc;


--==========================================================================
===========
--Stimulus Process:
--==========================================================================
===========
stimulus_process: process
```

```
begin
   -- loop over each address to test
   wait for clock_period;

   -- send in each address
   for i in 0 to 2 loop
      qspo_ce_signal <= '1';
      a_signal <= TestAddressesArray(i);
      wait for clock_period;
      qspo_ce_signal <= '0';
      wait for clock_period*10;
   end loop;

end process stimulus_process;

end testbench;
```

```
--=====================================================================
==========
--ENGS 31/ CoSc 56 22S
--Final Project - Wordle
--B.L. Dobbins, E.W. Hansen, Professor Luke
--Mubarak Idoko, Ikeoluwa Abioye, Lobna Jbenaini
    --Worlde ROM Testbench

-- TODO: (DO NOT REMOVE) Write to "Update Log" if you modify the
code, document your changes.
-- TODO: (DO NOT REMOVE) Update known issues to account for fixes of
newly discovered bugs.

-- UPDATE LOG:
-- Format: (Name) Date: Notes
--=====================================================================
==========
    -- (Mubbie) 05/25/2022: Created the file, implemented testbench
```

```
functionality
--=================================================================
==========

-- KNOWN ISSUES:
    -- Known issues with implementation that have not been fixes
    -- Critical Score (0-5):
        -- subjective measure
        -- how important fixing this issue is important to the proper
functioning of the system
    -- Ideas for fix: thoughts on how the issue could be fixed
-- Format: (Name) Date [Critical Score (0-5)]: Notes [Ideas for fix]
{Updates (if any)}
--=================================================================
==========

--=================================================================
==========



--=================================================================
==========
--Library Declarations:
--=================================================================
==========
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use ieee.math_real.all;



--=================================================================
==========
--Entity Declaration:
--=================================================================
==========
entity WORD_EXISTS_tb is
    end WORD_EXISTS_tb;
```

```
--=====================================================================
==========
--Architecture Type:
--=====================================================================
==========
architecture testbench of WORD_EXISTS_tb is

--=====================================================================
==========
--Component Declaration:
--=====================================================================
==========
component WORD_EXISTS IS
    -- ports
    port (
            -- inputs
            clk_in                :     in    STD_LOGIC;
            guess               :      in    STD_LOGIC_VECTOR(39
downto 0);
            word_ready          : in    STD_LOGIC;

            -- outputs
            is_dict_word    :    out STD_LOGIC;
            not_in_dict     :    out STD_LOGIC
    );
end component WORD_EXISTS;

--=====================================================================
==========
--Signal and Constant Declarations:
--=====================================================================
==========
-- inputs
signal clk_signal : std_logic := '0';
signal guess_signal : std_logic_vector(39 downto 0) := (others =>
'0');
signal word_ready_signal : std_logic := '0';

-- outputs
```

```
signal is_dict_word_signal : std_logic := '0';
signal not_in_dict_signal : std_logic := '0';

-- constants
constant clock_period : time := 10 ns; -- 100 MHz clock

--=====================================================================
==========
--Test Signal Declarations:
--=====================================================================
==========
type RegisterFile is array (0 to 1) of std_logic_vector(39 downto 0);
signal TestGuessesArray : RegisterFile :=
    (
        "0110110101101111011001000110010101101100",
        "0110000101100010011000110110010001100101"
    );

begin
uut : WORD_EXISTS
    PORT MAP (
        -- inputs
        clk_in              => clk_signal,
        guess              => guess_signal,
        word_ready         => word_ready_signal,

        -- outputs
        is_dict_word     => is_dict_word_signal,
        not_in_dict      => not_in_dict_signal
    );

--=====================================================================
==========
--Clock Generation Process:
--=====================================================================
==========
clk_proc : process
BEGIN
  clk_signal <= '0';
```

```
  wait for clock_period/2;   --100 MHz clock

  clk_signal <= '1';
  wait for clock_period/2;
END PROCESS clk_proc;

--=====================================================================
==========
--Stimulus Process:
--=====================================================================
==========
stimulus_process: process
begin
    -- loop over each address to test
    wait for clock_period/2;

    -- send in each address
    for i in 0 to 1 loop
        word_ready_signal <= '1';
        guess_signal <= TestGuessesArray(i);
        wait for clock_period;
        word_ready_signal <= '0';
        wait for clock_period*100;
    end loop;

    wait;
end process stimulus_process;

end testbench;
```