Assignment - 7 Report

Tanish Gupta , Divyanshu Agarwal

22nd May 2022

1 Introduction

The objective of this assignment was to Design and implement of a serial asynchronous transmitter and connecting the parallel output of the receiver (designed in the previous assignment) to the parallel input of the transmitter to form a loop.

2 Strategy and Logic

After designing the receiver in Assignment 7, we followed a similar approach for transmitter. In the top module, where the components for receiver and transmitter are instantiated, we send the parallel output of receiver to the parallel input of the transmitter, once all the bits are received (this is done by making a new signal "done"). The transmitter then sends these bits serially to the serial input of PC.

3 Code and Working

The main directory contains the design.vhd file, which has the VHDL code logic for instantiating the two components - receiver and transmitter, the two files for receiver and transmitter, one file myType.vhd, which defines the FSM, the multiDisplay.vhd file and the helper.vhd, which are the same files as submitted in assignment 2 and 3 (The helper file contains the logic for single Display of LED, and the multiDisplay file constains the logic for multiple digits display). The directory also has the constraint file and the generated bitstream.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.MyTypes.all;

entity transmitter is
  generic(
    clks_per_bit: integer := 10416
    );
  Port (
    clk: in std_logic;
    start: in std_logic;
    reset: in std_logic;
    inbyte: in std_logic_vector(7 downto 0);
    serial_output: out std_logic
   );
end transmitter;

architecture Behavioral of transmitter is
```

```
signal state: states := idle;
signal clk_count: integer range 0 to clks_per_bit := 0;
signal bit_index: integer range 0 to 7 := 0;
signal data: std_logic_vector(7 downto 0);

begin
    process(clk)
        begin
            if(rising_edge(clk)) then

                if(reset = '1') then
                    state <= reset_check;
                    clk_count <= 0;
--                    data <= '1';
--                    byteToDisplay <= x"0000";
                end if;

                case (state) is

                when reset_check =>
                    if clk_count < clks_per_bit - 1 then
                        clk_count <= clk_count + 1;
                    else
                        serial_output <= '1';
                        state <= idle;
                   end if;

                when idle =>
                    clk_count <= 0;
                    bit_index <= 0;
                    serial_output <= '1';
                    if(start = '1') then
                        data <= inbyte;
                        state <= start_bit;
                    end if;

                when start_bit =>
                    serial_output <= '0';

                    if(clk_count < clks_per_bit - 1) then
                        clk_count <= clk_count + 1;
                        state <= start_bit;
                    else
                        clk_count <= 0;
                        state <= data_bits;
                    end if;

                when data_bits =>

                    serial_output <= data(bit_index);

                    if(clk_count < clks_per_bit - 1) then
                        clk_count <= clk_count + 1;
                        state <= data_bits;
```

```
                    else
                        clk_count <= 0;
                        if(bit_index < 7) then
                            bit_index <= bit_index + 1;
                            state <= data_bits;

                        else
                            bit_index <= 0;
                            state <= stop_bit;
                        end if;
                    end if;


                when stop_bit =>
                    serial_output <= '1';

                    if(clk_count < clks_per_bit - 1) then
                        clk_count <= clk_count + 1;
                        state <= stop_bit;
                    else
                        clk_count <= 0;
                        state <= idle;
                    end if;


                 when others =>
                    state <= idle;

                 end case;

            end if;

    end process;


end Behavioral;
```

4 Simulation and Synthesis

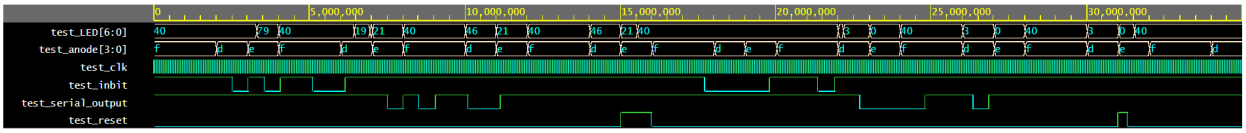

Figure 1: Generated EP Wave

These EPWaves was carefully analysed to check if all outputs were correct!
The Vivado synthesis report is as follows:

```
Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.
----------------------------------------------------------------------------------------------------
| Tool Version : Vivado v.2019.1 (lin64) Build 2552052 Fri May 24 14:47:09 MDT 2019
| Date         : Sat May 21 22:19:47 2022
| Host         : divyanshu-HP-ENVY-x360-Convertible-13-bd0xxx
                 running 64-bit Ubuntu 20.04.2 LTS
| Command      : report_utilization -file design_utilization_synth.rpt
                 -pb design_utilization_synth.pb
| Design       : \design
| Device       : 7a35tcpg236-2
| Design State : Synthesized
----------------------------------------------------------------------------------------------------

Utilization Design Information

Table of Contents
-----------------

1. Slice Logic
1.1 Summary of Registers by Type
2. Memory
3. DSP
4. IO and GT Specific
5. Clocking
6. Specific Feature
7. Primitives
8. Black Boxes
9. Instantiated Netlists

1. Slice Logic
--------------

+----------------------------+------+-------+-----------+-------+
|          Site Type         | Used | Fixed | Available | Util% |
+----------------------------+------+-------+-----------+-------+
| Slice LUTs*                |  125 |     0 |     20800 |  0.60 |
|   LUT as Logic             |  125 |     0 |     20800 |  0.60 |
|   LUT as Memory            |    0 |     0 |      9600 |  0.00 |
| Slice Registers            |  117 |     0 |     41600 |  0.28 |
|   Register as Flip Flop    |  109 |     0 |     41600 |  0.26 |
|   Register as Latch        |    8 |     0 |     41600 |  0.02 |
```

| F7 Muxes | 0 | 0 | 16300 | 0.00 |
| F8 Muxes | 0 | 0 | 8150 | 0.00 |

1.1 Summary of Registers by Type

Total	Clock Enable	Synchronous	Asynchronous
0	-	-	-
0	-	-	Set
0	-	-	Reset
0	-	Set	-
0	-	Reset	-
0	Yes	-	-
0	Yes	-	Set
8	Yes	-	Reset
4	Yes	Set	-
105	Yes	Reset	-

2. Memory

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	0	0	50	0.00
RAMB36/FIFO*	0	0	50	0.00
RAMB18	0	0	100	0.00

* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile, that tile can still accommodate a RAMB18E1

3. DSP

Site Type	Used	Fixed	Available	Util%
DSPs	0	0	90	0.00

4. IO and GT Specific

Site Type	Used	Fixed	Available	Util%
Bonded IOB	15	0	106	14.15
Bonded IPADs	0	0	10	0.00
Bonded OPADs	0	0	4	0.00
PHY_CONTROL	0	0	5	0.00
PHASER_REF	0	0	5	0.00
OUT_FIFO	0	0	20	0.00
IN_FIFO	0	0	20	0.00
IDELAYCTRL	0	0	5	0.00
IBUFDS	0	0	104	0.00
GTPE2_CHANNEL	0	0	2	0.00
PHASER_OUT/PHASER_OUT_PHY	0	0	20	0.00
PHASER_IN/PHASER_IN_PHY	0	0	20	0.00
IDELAYE2/IDELAYE2_FINEDELAY	0	0	250	0.00
IBUFDS_GTE2	0	0	2	0.00
ILOGIC	0	0	106	0.00
OLOGIC	0	0	106	0.00

5. Clocking

Site Type	Used	Fixed	Available	Util%
BUFGCTRL	1	0	32	3.13
BUFIO	0	0	20	0.00
MMCME2_ADV	0	0	5	0.00
PLLE2_ADV	0	0	5	0.00
BUFMRCE	0	0	10	0.00
BUFHCE	0	0	72	0.00
BUFR	0	0	20	0.00

6. Specific Feature

Site Type	Used	Fixed	Available	Util%
BSCANE2	0	0	4	0.00
CAPTUREE2	0	0	1	0.00
DNA_PORT	0	0	1	0.00
EFUSE_USR	0	0	1	0.00
FRAME_ECCE2	0	0	1	0.00
ICAPE2	0	0	2	0.00
PCIE_2_1	0	0	1	0.00
STARTUPE2	0	0	1	0.00
XADC	0	0	1	0.00

7. Primitives

Ref Name	Used	Functional Category
FDRE	105	Flop & Latch
LUT2	48	LUT
LUT6	31	LUT
LUT1	26	LUT
CARRY4	26	CarryLogic
LUT4	25	LUT
LUT5	19	LUT
OBUF	12	IO
LUT3	12	LUT
LDCE	8	Flop & Latch
FDSE	4	Flop & Latch
IBUF	3	IO
BUFG	1	Clock

8. Black Boxes

Ref Name	Used

9. Instantiated Netlists

Ref Name	Used