

Lab #7

ECE 4304 Spring 2021

Professor Aly

California State Polytechnic University, Pomona

Ahiezer Lopez

ahiezerlopez@cpp.edu

Bronco ID:012715521

Joe Si

joesi@cpp.edu

Bronco ID: 012636091

Sander Zuckerman

sazuckerman@cpp.edu

Bronco ID: 012644671

4/07/2021

Objective:

Students will design a UART compatible machine utilizing FIFOs, ALUs, Barrel Shifters, and Seven Segments. The number of control bits will be 6 as it will be 3 per barrel shifter. We will have to read buttons each for the two FIFOs as well as a load button for the ALU. Along with the ALU, there are two switches that determine the math operation being executed. The outputs will have 4 LEDs representing whether the FIFOs are full or empty. Lastly, for the Seven Segments we will have 4 of them representing our ALU output and 4 representing our 8 bit inputs A and B.

Materials:

- FPGA (Nexys A7-100T)
- Vivado Software
- Computer
- UART Pmod (optional)

Contributions:

For this lab, everyone had their own part in it. With all parts/ modules already created, the group itself focused on the connection of the UART module and the rest of the code. Having been exposed to how UART works in a previous lecture, this gave us a headstart in this project. Still, we had to research to smooth out any confusion we had regarding UART. We each worked together to connect all the modules accordingly. The most time consuming part of the project was debugging, but through everyone's individual effort in debugging we were able to get it working.

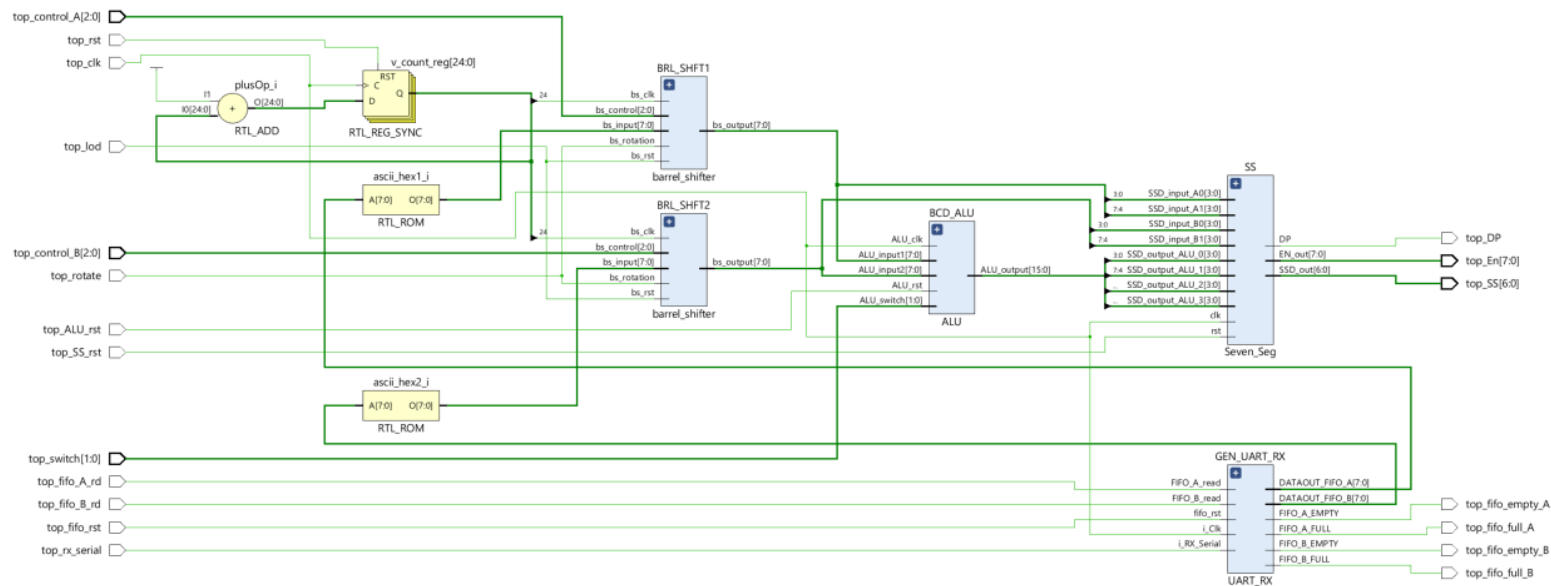
Design Process:

Using the flowchart from below, we implemented our idea on how to approach this lab. We were able to implement the codes from previous labs and the new code provided during class to create our lab. We had multiple issues when designing this lab, one issue was how the connection between the FPGA and the computer through UART was supposed to be done. We had to use the program "Tera Term" to easily send data directly through the USB. This brought on another issue regarding the ASCII data not sending in the correct format. Another issue was created for the transmission of data between the UART and the FIFOs as one FIFO may fill up however, the second FIFO would not. These issues were resolved after hours of debugging and the objective of this lab was completed.

Design:

Using some code from previous lab experiments such as the seven segment code, the Arithmetic Logic Unit, and the barrel shifter we were able to combine these components with what was shown in class. In class, we were provided with the FIFO and the UART_rx code, so all we had to do was wire up / connect these components together correctly. The main design was already created and we just had to correctly wire up these components with the right settings such as the baud rate and stop bit.

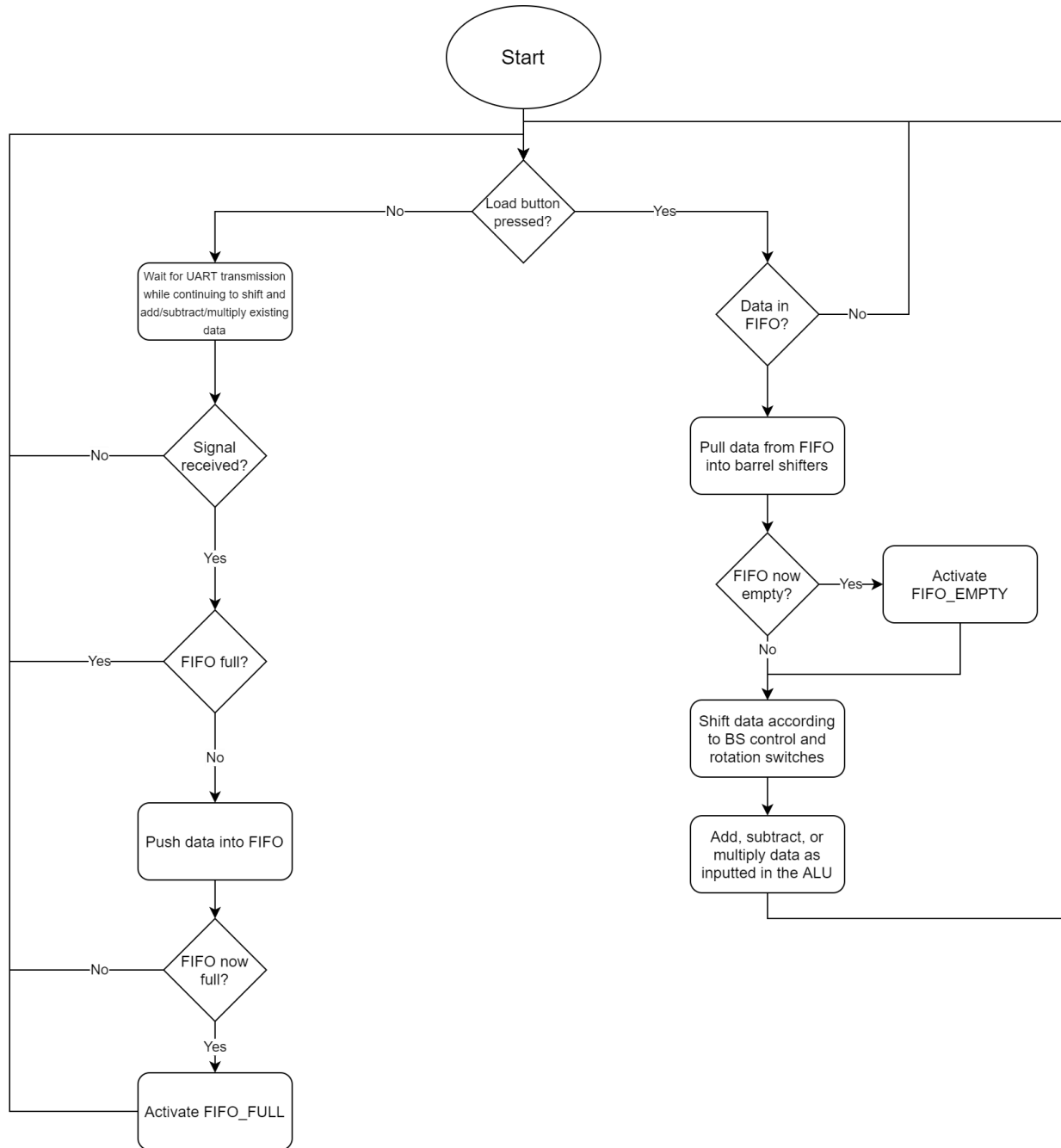
Circuit Diagram:



Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	DSP
synth_1	constrs_1	synth_design Complete!								170	188	0.0	0	0
impl_1	constrs_1	write_bitstream Complete!	5.411	0.000	0.083	0.000	0.000	0.113	0	163	188	0.0	0	0

Implementation:

Logic Flow Diagram



Design:

Top Module

```
21 library IEEE;
22 use IEEE.STD_LOGIC_1164.ALL;
23 use IEEE.math_real.all;
24 use IEEE.std_logic_unsigned.all;
25
26 -- Uncomment the following library declaration if using
27 -- arithmetic functions with Signed or Unsigned values
28 --use IEEE.NUMERIC_STD.ALL;
29
30 -- Uncomment the following library declaration if instantiating
31 -- any Xilinx leaf cells in this code.
32 --library UNISIM;
33 --use UNISIM.VComponents.all;
34
35 entity top is
36     generic ( WIDTH: integer:= 8; -- for every 8 bit signal/input/output
37               top_g_WIDTH: integer:=8; --for width of fifo
38               top_g_DEPTH: integer:=4; -- for depth of fifo
39               top_g_CLKS_PER_BIT : integer:= 869 -- for UART
40             );Port (
41
42         top_clk           : in std_logic;
43         top_rst           : in std_logic;-- reset for barrel shifter clock
44         top_SS_rst        : in std_logic;-- reset for seven segment
45         top_ALU_rst       : in std_logic;-- reset for ALU
46         top_fifo_rst      : in std_logic;-- reset for FIFOs
47         top_switch        : in std_logic_vector(1 downto 0); -- switches for ALU (+,-,*)
48         top_lod           : in std_logic;--loads onto ALU
49         top_fifo_A_rd     : in std_logic;-- reads input from uart onto fifo A
50         top_fifo_B_rd     : in std_logic;-- reads input from uart onto fifo B
51         top_control_A     : in std_logic_vector(2 downto 0);-- will control number of bits rotating
52         top_control_B     : in std_logic_vector(2 downto 0);-- will control number of bits rotating
53         top_rotate        : in std_logic; -- will determine the direction of rotation
54         top_rx_serial     : in std_logic; -- input from UART
55         top_fifo_empty_A  : out std_logic; -- empty for FIFO A
56         top_fifo_empty_B  : out std_logic; -- empty for FIFO B
57         top_fifo_full_A   : out std_logic; -- full for FIFO A
58         top_fifo_full_B   : out std_logic; -- full for FIFO A
59         top_En            : out std_logic_vector(7 downto 0);
60         top_SS            : out std_logic_vector(6 downto 0);
61         top_DP            : out std_logic
62     );
63
```

```

66 | architecture Behavioral of top is
67 |
68 | component ALU
69 |     generic(width : integer:=8);
70 |     Port (
71 |         ALU_clk      : in std_logic;
72 |         ALU_rst      : in std_logic;
73 |         ALU_input1   : in std_logic_vector(width-1 downto 0);
74 |         ALU_input2   : in std_logic_vector(width-1 downto 0);
75 |         ALU_switch   : in std_logic_vector(1 downto 0);
76 |         ALU_output   : out std_logic_vector(2*width-1 downto 0)
77 |     );
78 | end component;
79 |
80 | component Seven_Seg
81 |     Port (
82 |         clk          : in std_logic;
83 |         rst          : in std_logic;
84 |         SSD_input_A0 : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for input
85 |         SSD_input_A1 : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for input
86 |         SSD_input_B0 : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for output of rotation
87 |         SSD_input_B1 : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for output of rotation
88 |         SSD_output_ALU_0 : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for control of rotation
89 |         SSD_output_ALU_1 : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for control of rotation
90 |         SSD_output_ALU_2 : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for control of rotation
91 |         SSD_output_ALU_3 : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for control of rotation
92 |         EN_out         : out std_logic_vector(7 downto 0);
93 |         DP             : out std_logic;
94 |         SSD_out        : out STD_LOGIC_VECTOR(6 downto 0)
95 |     );
96 | end component;
97 |
98 | component barrel_shifter is
99 |     generic(width: integer:=8
100 |     );
101 |     Port (
102 |         bs_clk      : in std_logic;
103 |         bs_rst      : in std_logic;
104 |         bs_input    : in std_logic_vector(width-1 downto 0); -- barrel shifter input
105 |         bs_control  : in std_logic_vector(integer(LOG2(real(width)))-1 downto 0); -- control
106 |         bs_rotation : in std_logic; -- will rotate the input left or right

```

```

107         bs_output    : out std_logic_vector(width-1 downto 0) -- barrel shifter output
108
109     );
110 end component;
111
112
113 component UART_RX is
114     generic (
115         g_CLKS_PER_BIT : integer := 869;      -- Needs to be set correctly
116         width : integer:= 8;
117         depth : integer:=4
118     );
119     port (
120         i_Clk      : in  std_logic;
121         FIFO_RST    : in  std_logic;
122         i_RX_Serial : in  std_logic;
123         FIFO_A_read : in  std_logic;
124         FIFO_B_read : in  std_logic;
125         FIFO_A_EMPTY : out std_logic;
126         FIFO_B_EMPTY : out std_logic;
127         FIFO_A_FULL  : out std_logic;
128         FIFO_B_FULL  : out std_logic;
129         DATAOUT_FIFO_A: out std_logic_vector(7 downto 0);
130         DATAOUT_FIFO_B: out std_logic_vector(7 downto 0)
131     );
132 end component;
133
134
135 signal ALU_input1: std_logic_vector(WIDTH-1 downto 0);
136 signal ALU_input2: std_logic_vector(WIDTH-1 downto 0);
137 signal ALU_out_buf: std_logic_vector(2*width-1 downto 0);
138
139 signal BS_input1: std_logic_vector(WIDTH-1 downto 0);
140 signal BS_input2: std_logic_vector(WIDTH-1 downto 0);
141
142 signal BIT_16_out_buf: std_logic_vector(2*width-1 downto 0);
143
144 signal top_output_BS1 : std_logic_vector(2*width-1 downto 0);
145 signal top_output_BS2 : std_logic_vector(2*width-1 downto 0);
146
147 signal delay_sig      : std_logic;
148
149 signal input_B : std_logic_vector(WIDTH-1 downto 0);

```

<

```

149 signal input_B : std_logic_vector(WIDTH-1 downto 0);
150 signal full_A : std_logic;
151
152 signal uart_out : std_logic_vector(WIDTH-1 downto 0);
153
154 signal ascii_hex1: std_logic_vector(width-1 downto 0);
155 signal ascii_hex2: std_logic_vector(width-1 downto 0);
156
157 begin
158
159 DELAY: process(top_clk, top_rst)
160     variable v_count: std_logic_vector(24 downto 0);
161     begin
162         if(rising_edge(top_clk))then
163             if(top_rst ='1')then
164                 v_count := (others => '0');
165             else
166                 v_count := v_count + 1;
167             end if;
168         end if;
169         delay_sig <= v_count(24);
170     end process;
171
172 BRL_SHFT1: barrel_shifter
173     port map(
174         bs_clk      => delay_sig,
175         bs_rst      => top_lod,
176         bs_input    => ascii_hex1,
177         bs_control  => top_control_A,
178         bs_rotation => top_rotate,
179         bs_output   => ALU_input1
180     );
181
182 BRL_SHFT2: barrel_shifter
183     port map(
184         bs_clk      => delay_sig,
185         bs_rst      => top_lod,
186         bs_input    => ascii_hex2,
187         bs_control  => top_control_B,
188         bs_rotation => top_rotate,
189         bs_output   => ALU_input2
190     );
191

```

```

194 BCD_ALU: ALU
195     generic map( width => width)
196     port map(
197         ALU_clk    => top_clk,
198         ALU_rst    => top_ALU_rst,
199         ALU_input1 => ALU_input1,
200         ALU_input2 => ALU_input2,
201         ALU_switch => top_switch(1 downto 0),
202         ALU_output => ALU_out_buf
203     );
204
205 GEN_CONV1: process(BS_input1)
206     begin
207         case BS_input1 is
208             when x"30" => ascii_hex1 <= x"00";
209             when x"31" => ascii_hex1 <= x"01";
210             when x"32" => ascii_hex1 <= x"02";
211             when x"33" => ascii_hex1 <= x"03";
212             when x"34" => ascii_hex1 <= x"04";
213             when x"35" => ascii_hex1 <= x"05";
214             when x"36" => ascii_hex1 <= x"06";
215             when x"37" => ascii_hex1 <= x"07";
216             when x"38" => ascii_hex1 <= x"08";
217             when x"39" => ascii_hex1 <= x"09";
218             when others => ascii_hex1 <= (others => 'U');
219         end case;
220     end process;
221
222 GEN_CONV2: process(BS_input2)
223     begin
224         case BS_input2 is
225             when x"30" => ascii_hex2 <= x"00";
226             when x"31" => ascii_hex2 <= x"01";
227             when x"32" => ascii_hex2 <= x"02";
228             when x"33" => ascii_hex2 <= x"03";
229             when x"34" => ascii_hex2 <= x"04";
230             when x"35" => ascii_hex2 <= x"05";
231             when x"36" => ascii_hex2 <= x"06";
232             when x"37" => ascii_hex2 <= x"07";
233             when x"38" => ascii_hex2 <= x"08";
234             when x"39" => ascii_hex2 <= x"09";
235             when others => ascii_hex2 <= (others => 'U');
236         end case;

```

```

237         end process;
238
239     SS: Seven_Seg
240     port map(
241         clk      => top_clk,
242         rst      => top_SS_rst,
243         SSD_input_A0  => ALU_input1(3 downto 0),
244         SSD_input_A1  => ALU_input1(7 downto 4),
245         SSD_input_B0  => ALU_input2(3 downto 0),
246         SSD_input_B1  => ALU_input2(7 downto 4),
247         SSD_output_ALU_0 => ALU_out_buf(3 downto 0),
248         SSD_output_ALU_1 => ALU_out_buf(7 downto 4),
249         SSD_output_ALU_2 => ALU_out_buf(11 downto 8),
250         SSD_output_ALU_3 => ALU_out_buf(15 downto 12),
251         EN_out   => top_En,
252         DP       => top_DP,
253         SSD_out  => top_SS
254     );
255
256
257     GEN_UART_RX: UART_RX
258     generic map(g_CLKS_PER_BIT => top_g_CLKS_PER_BIT, width => top_g_WIDTH, depth => top_g_DEPTH)
259     port map(
260         i_Clk      => top_clk,
261         FIFO_RST   => top_fifo_rst,
262         i_RX_Serial => top_rx_serial,
263         FIFO_A_read => top_fifo_A_rd,
264         FIFO_B_read => top_fifo_B_rd,
265         FIFO_A_EMPTY => top_fifo_empty_A,
266         FIFO_B_EMPTY => top_fifo_empty_B,
267         FIFO_A_FULL  => top_fifo_full_A,
268         FIFO_B_FULL  => top_fifo_full_B,
269         DATAOUT_FIFO_A => BS_input1,
270         DATAOUT_FIFO_B => BS_input2
271     );
272
273
274
275     end Behavioral;

```

UART Rx Module:

```
27 library ieee;
28 use ieee.std_logic_1164.ALL;
29 use ieee.numeric_std.all;
30
31 entity UART_RX is
32   generic (
33     g_CLKS_PER_BIT : integer := 869;
34     width : integer := 8;
35     depth : integer := 4
36   );
37   port (
38     i_Clk      : in  std_logic;
39     fifo_rst   : in  std_logic;
40     i_RX_Serial : in  std_logic;
41     FIFO_A_read : in  std_logic;
42     FIFO_B_read : in  std_logic;
43     FIFO_A_FULL : out std_logic;
44     FIFO_A_EMPTY : out std_logic;
45     FIFO_B_FULL : out std_logic;
46     FIFO_B_EMPTY : out std_logic;
47     DATAOUT_FIFO_A : out std_logic_vector(7 downto 0);
48     DATAOUT_FIFO_B : out std_logic_vector(7 downto 0)
49   );
50 end UART_RX;
51
52
53 architecture rtl of UART_RX is
54
55   type t_SM_Main is (s_Idle, s_RX_Start_Bit, s_RX_Data_Bits,
56                     s_RX_Stop_Bit, s_Cleanup);
57   signal r_SM_Main : t_SM_Main := s_Idle;
58
59   signal r_RX_Data_R : std_logic := '0';
60   signal r_RX_Data   : std_logic := '0';
61
62   signal r_Clk_Count : integer range 0 to g_CLKS_PER_BIT-1 := 0;
63   signal r_Bit_Index : integer range 0 to 7 := 0; -- 8 Bits Total
64   signal r_RX_Byte   : std_logic_vector(7 downto 0) := (others => '0');
65   signal o_RX_Byte   : std_logic_vector(7 downto 0);
66   signal r_RX_DV     : std_logic := '0';
67   signal o_RX_DV     : std_logic := '0';
68   signal FAB         : std_logic;
69   signal FBB         : std_logic;
```

```

70 : signal o_RX_DVB: std_logic;
71 : signal o_RX_DVA: std_logic;
72 :
73 : component STD_FIFO is
74 :     Generic (
75 :         constant DATA_WIDTH : positive := 8;
76 :         constant FIFO_DEPTH : positive := 256
77 :     );
78 :     Port (
79 :         CLK      : in  STD_LOGIC;
80 :         RST      : in  STD_LOGIC;
81 :         WriteEn   : in  STD_LOGIC;
82 :         DataIn    : in  STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
83 :         ReadEn    : in  STD_LOGIC;
84 :         DataOut   : out STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
85 :         Empty     : out STD_LOGIC;
86 :         Full      : out STD_LOGIC
87 :     );
88 : end component;
89 :
90 :
91 : begin
92 :     FIFO_A_FULL <= FAB;
93 :     FIFO_B_FULL <= FBB;
94 :
95 :
96 :     GEN_FIFO_A: STD_FIFO
97 :         generic map (DATA_WIDTH => width, FIFO_DEPTH => depth )
98 :         port map(
99 :             CLK      => i_Clk,
100 :             RST      => fifo_rst,
101 :             WriteEn  => o_RX_DVA,
102 :             DataIn   => o_RX_Byte,
103 :             ReadEn   => FIFO_A_read,
104 :             DataOut  => DATAOUT_FIFO_A,
105 :             Empty    => FIFO_A_EMPTY,
106 :             Full     => FAB
107 :
108 :         );
109 :
110 :     COND: process (FAB, FBB)
111 :     begin
112 :         if (FAB = '1') then
113 :             o_RX_DVB <= o_RX_DV;

```

```

114         else
115             o_RX_DVA <= o_RX_DV;
116         end if;
117
118     end process;
119
120     GEN_FIFO_B: STD_FIFO
121         generic map (DATA_WIDTH => width, FIFO_DEPTH => depth )
122         port map(
123             CLK      => i_Clk,
124             RST      => FIFO_RST,
125             WriteEn  => o_RX_DVB,
126             DataIn   => o_RX_Byte,
127             ReadEn   => FIFO_B_read,
128             DataOut  => DATAOUT_FIFO_B,
129             Empty    => FIFO_B_EMPTY,
130             Full     => FBB
131         );
132
133
134     -- Purpose: Double-register the incoming data.
135     -- This allows it to be used in the UART RX Clock Domain.
136     -- (It removes problems caused by metastability)
137     p_SAMPLE : process (i_Clk)
138     begin
139         if rising_edge(i_Clk) then
140             r_RX_Data_R <= i_RX_Serial;
141             r_RX_Data   <= r_RX_Data_R;
142         end if;
143     end process p_SAMPLE;
144
145
146     -- Purpose: Control RX state machine
147     p_UART_RX : process (i_Clk)
148     begin
149         if rising_edge(i_Clk) then
150
151             case r_SM_Main is
152
153                 when s_Idle =>
154                     r_RX_DV      <= '0';
155                     r_Clk_Count <= 0;

```

```

155         r_Clk_Count <= 0;
156         r_Bit_Index <= 0;
157
158         if r_RX_Data = '0' then           -- Start bit detected
159             r_SM_Main <= s_RX_Start_Bit;
160         else
161             r_SM_Main <= s_Idle;
162         end if;
163
164
165         -- Check middle of start bit to make sure it's still low
166         when s_RX_Start_Bit =>
167             if r_Clk_Count = (g_CLKS_PER_BIT-1)/2 then
168                 if r_RX_Data = '0' then
169                     r_Clk_Count <= 0; -- reset counter since we found the middle
170                     r_SM_Main <= s_RX_Data_Bits;
171                 else
172                     r_SM_Main <= s_Idle;
173                 end if;
174             else
175                 r_Clk_Count <= r_Clk_Count + 1;
176                 r_SM_Main <= s_RX_Start_Bit;
177             end if;
178
179
180         -- Wait g_CLKS_PER_BIT-1 clock cycles to sample serial data
181         when s_RX_Data_Bits =>
182             if r_Clk_Count < g_CLKS_PER_BIT-1 then
183                 r_Clk_Count <= r_Clk_Count + 1;
184                 r_SM_Main <= s_RX_Data_Bits;
185             else
186                 r_Clk_Count <= 0;
187                 r_RX_Byte(r_Bit_Index) <= r_RX_Data;
188
189                 -- Check if we have sent out all bits
190                 if r_Bit_Index < 7 then
191                     r_Bit_Index <= r_Bit_Index + 1;
192                     r_SM_Main <= s_RX_Data_Bits;
193                 else
194                     r_Bit_Index <= 0;
195                     r_SM_Main <= s_RX_Stop_Bit;
196                 end if;
197             end if;
198

```

```

200      -- Receive Stop bit.  Stop bit = 1
201      when s_RX_Stop_Bit =>
202          -- Wait g_CLKS_PER_BIT-1 clock cycles for Stop bit to finish
203          if r_Clk_Count < g_CLKS_PER_BIT-1 then
204              r_Clk_Count <= r_Clk_Count + 1;
205              r_SM_Main <= s_RX_Stop_Bit;
206          else
207              r_RX_DV <= '1';
208              r_Clk_Count <= 0;
209              r_SM_Main <= s_Cleanup;
210          end if;
211
212
213      -- Stay here 1 clock
214      when s_Cleanup =>
215          r_SM_Main <= s_Idle;
216          r_RX_DV <= '0';
217
218
219      when others =>
220          r_SM_Main <= s_Idle;
221
222      end case;
223  end if;
224  end process p_UART_RX;
225
226  o_RX_DV <= r_RX_DV;
227  o_RX_Byte <= r_RX_Byte;
228
229  end rtl;

```

Barrel Shifter Module:

```
21  library ieee;
22  use ieee.std_logic_1164.ALL;
23  use ieee.numeric_std.all;
24  use ieee.std_logic_arith.all;
25  use ieee.std_logic_unsigned.all;
26
27  entity UART_RX is
28  generic (
29      g_CLKS_PER_BIT : integer := 87    -- Needs to be set correctly
30  );
31  port (
32      i_Clk      : in  std_logic;
33      i_reset    : in  std_logic;
34      i_RX_Serial : in  std_logic;
35  --  ReadEn      : in  std_logic;
36  --  Output      : out std_logic_vector(7 downto 0);
37  --  DATAOUT_FIFO_A: out std_logic_vector(7 downto 0);
38  --  DATAOUT_FIFO_B: out std_logic_vector(7 downto 0);
39  --  Empty_A     : out std_logic;
40  --  Empty_B     : out std_logic;
41  --  Full_A      : out std_logic;
42  --  Full_B      : out std_logic
43      DP        : out std_logic;
44      EN        : out std_logic_vector(7 downto 0);
45      SS        : out std_logic_vector(6 downto 0)
46  );
47  end UART_RX;
48
49
50  architecture rtl of UART_RX is
51
52      type t_SM_Main is (s_Idle, s_RX_Start_Bit, s_RX_Data_Bits,
53                          s_RX_Stop_Bit, s_Cleanup);
54      signal r_SM_Main : t_SM_Main := s_Idle;
55
56      signal r_RX_Data_R : std_logic := '0';
57      signal r_RX_Data   : std_logic := '0';
58
59      signal r_Clk_Count : integer range 0 to g_CLKS_PER_BIT-1 := 0;
60      signal r_Bit_Index : integer range 0 to 7 := 0;  -- 8 Bits Total
```



```

61 |
62 | architecture Behavioral of barrel_shifter is
63 |
64 |     signal I_temp: std_logic_vector(width-1 downto 0);
65 |     signal C_temp: std_logic_vector(integer(LOG2(real(width)))-1 downto 0);
66 |
67 |     type state_type is (
68 |         s0,
69 |         s1,
70 |         s2,
71 |         s3,
72 |         s4
73 |     );
74 |     signal ps_state: state_type; -- present state
75 |
76 | begin
77 | process(bs_clk, bs_rst)
78 | begin
79 |     if(bs_rst = '1') then
80 |         ps_state <= s0;
81 |     elsif(rising_edge(bs_clk)) then
82 |         case ps_state is
83 |         when s0 => --initialize values
84 |             I_temp <= bs_input;
85 |             C_temp <= bs_control;
86 |             if(bs_rotation = '1') then --if '1' rotate right --if '0' rotate left
87 |                 ps_state <= s2;
88 |             else
89 |                 ps_state <= s3;
90 |             end if;
91 |         when s1 => -- will determine which state to go to next depending on direction of rotation
92 |             C_temp <= bs_control;
93 |             if(bs_rotation = '1') then --if '1' rotate right --if '0' rotate left
94 |                 ps_state <= s2;
95 |             else
96 |                 ps_state <= s3;
97 |             end if;
98 |         when s2 => -- will rotate right each time C_temp reaches 0
99 |             if(C_temp = x"0") then
100 |                 ps_state <= s4;
101 |             else
102 |                 C_temp <= C_temp -1;
103 |                 I_temp <= I_Temp(0) & I_temp(width-1 downto 1);

```

```

104 :           ps_state <= s2;
105 ⊞       end if;
106 ⊞       when s3 => --will rotate left each time C_temp reaches 0
107 ⊞           if(C_temp = x"0")then
108 :               ps_state <= s4;
109 :           else
110 :               C_temp <= C_temp -1;
111 :               I_temp <= I_temp(width-2 downto 0) & I_Temp(width-1);
112 :               ps_state <= s3;
113 ⊞           end if;
114 ⊞       when s4 => -- I_temp is set to output while it goes back to state 1
115 :           bs_output <= I_temp;
116 ⊞       ps_state <= s1;
117 :       when others => bs_output <= (others => '0');
118 ⊞       end case;
119 ⊞   end if;
120 ⊞ end process;
121 ⊞ end Behavioral;
122 :

```

Seven-Segment Display Module:

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24  use IEEE.numeric_std.all;
25  use IEEE.std_logic_unsigned.all;
26
27  -- Uncomment the following library declaration if using
28  -- arithmetic functions with Signed or Unsigned values
29  --use IEEE.NUMERIC_STD.ALL;
30
31  -- Uncomment the following library declaration if instantiating
32  -- any Xilinx leaf cells in this code.
33  --library UNISIM;
34  --use UNISIM.VComponents.all;
35
36  entity Seven_Seg is
37      Port (
38          clk          : in std_logic;
39          rst          : in std_logic;
40          SSD_input_A0  : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for input
41          SSD_input_A1  : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for input
42          SSD_input_B0  : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for output of rotation
43          SSD_input_B1  : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for output of rotation
44          SSD_output_ALU_0 : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for control of rotation
45          SSD_output_ALU_1 : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for control of rotation
46          SSD_output_ALU_2 : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for control of rotation
47          SSD_output_ALU_3 : in STD_LOGIC_VECTOR(3 downto 0); -- seven seg for control of rotation
48          EN_out        : out std_logic_vector(7 downto 0);
49          DP            : out std_logic;
50          SSD_out        : out STD_LOGIC_VECTOR(6 downto 0)
51      );
52  end Seven_Seg;
53
54  architecture Behavioral of Seven_Seg is
55
56
57      signal counter_out: std_logic_vector(2 downto 0);
58      signal dout_out: std_logic_vector(5 downto 0);
59      signal E: std_logic_vector(7 downto 0);
60
61
62  begin
```

```

63 :
64 ⊞ process(clk, rst)
65 :     variable v_count: std_logic_vector(18 downto 0);
66 :     begin
67 :         if(rising_edge(clk))then -- need to fix the reset here
68 :             if(rst ='1')then
69 :                 v_count := (others => '0');
70 :             else
71 :                 v_count := v_count + 1;
72 :             end if;
73 :         end if;
74 :         counter_out <= v_count(18 downto 16);
75 :     end process;
76 :
77 :
78 ⊞ process(counter_out, SSD_input_A0, SSD_input_A1,SSD_input_B0,SSD_input_B1,SSD_output_ALU_0,SSD_output_ALU_1,SSD_output_ALU_2,SSD_output_ALU_3)
79 : begin
80 :     case counter_out is
81 :         when "000" => E <= "000000001";
82 :             dout_out <= '1' & SSD_input_A0(3 downto 0) & '1';
83 :         when "001" => E <= "000000010";
84 :             dout_out <= '1' & SSD_input_A1(3 downto 0) & '1';
85 :         when "010" => E <= "000000100";
86 :             dout_out <= '1' & SSD_input_B0(3 downto 0) & '1';
87 :         when "011" => E <= "000001000";
88 :             dout_out <= '1' & SSD_input_B1(3 downto 0) & '1';
89 :         when "100" => E <= "000100000";
90 :             dout_out <= '1' & SSD_output_ALU_0(3 downto 0) & '1' ;
91 :         when "101" => E <= "001000000";
92 :             dout_out <= '1' & SSD_output_ALU_1(3 downto 0) & '1' ;
93 :         when "110" => E <= "010000000";
94 :             dout_out <= '1' & SSD_output_ALU_2(3 downto 0) & '1' ;
95 :         when "111" => E <= "100000000";
96 :             dout_out <= '1' & SSD_output_ALU_3(3 downto 0) & '1' ;
97 :         when others => E <= "111111111";
98 :     end case;
99 : end process;
100 :
101 : En_out <= not E;
102 :
103 ⊞ process(dout_out) is -- a reset for the seven segment decoder, shouldnt be the same as the reset above
104 : begin
105 :     case dout_out(4 downto 1) is
106 :         when "0000" => SSD_out(6 downto 0) <= "0000001";
107 :         when "0001" => SSD_out(6 downto 0) <= "1001111";
108 :         when "0010" => SSD_out(6 downto 0) <= "0010010";
109 :         when "0011" => SSD_out(6 downto 0) <= "0000110";
110 :         when "0100" => SSD_out(6 downto 0) <= "1001100";
111 :         when "0101" => SSD_out(6 downto 0) <= "0100100";
112 :         when "0110" => SSD_out(6 downto 0) <= "0100000";
113 :         when "0111" => SSD_out(6 downto 0) <= "0001111";
114 :         when "1000" => SSD_out(6 downto 0) <= "0000000";
115 :         when "1001" => SSD_out(6 downto 0) <= "0000100";
116 :         when "1010" => SSD_out(6 downto 0) <= "0001000";
117 :         when "1011" => SSD_out(6 downto 0) <= "1100000";
118 :         when "1100" => SSD_out(6 downto 0) <= "0110001";
119 :         when "1101" => SSD_out(6 downto 0) <= "1000010";
120 :         when "1110" => SSD_out(6 downto 0) <= "0110000";
121 :         when "1111" => SSD_out(6 downto 0) <= "0111000";
122 :         when others => SSD_out(6 downto 0) <= "1111111";
123 :     end case;
124 : end process;
125 :
126 : DP <= dout_out(0);
127 ⊞ end Behavioral;

```

ALU Module:

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24  use IEEE.NUMERIC_STD.ALL;
25  use IEEE.std_logic_unsigned.all;
26
27  -- Uncomment the following library declaration if using
28  -- arithmetic functions with Signed or Unsigned values
29  --use IEEE.NUMERIC_STD.ALL;
30
31  -- Uncomment the following library declaration if instantiating
32  -- any Xilinx leaf cells in this code.
33  --library UNISIM;
34  --use UNISIM.VComponents.all;
35
36  entity ALU is
37      generic(width : integer:=8);
38      Port (
39          ALU_clk      : in std_logic;
40          ALU_rst      : in std_logic;
41          ALU_input1   : in std_logic_vector(width-1 downto 0);
42          ALU_input2   : in std_logic_vector(width-1 downto 0);
43          ALU_switch   : in std_logic_vector(1 downto 0);
44          ALU_output   : out std_logic_vector(2*width-1 downto 0)
45      );
46  end ALU;
47
48  architecture Behavioral of ALU is
49
50  component MUL
51      generic(mul_width: integer:= 8);
52      Port (
53          mul_clk: in std_logic;
54          mul_rst: in std_logic;
55          go:     in std_logic;
56          M:      in std_logic_vector(mul_width-1 downto 0);
57          Q:      in std_logic_vector(mul_width-1 downto 0);
58          Product: out std_logic_vector((mul_width*2)-1 downto 0)
59      );
60  end component;
61
62  component FA
63      Port (
```

```

64         Cin    : in std_logic;
65         A      : in std_logic;
66         B      : in std_logic;
67         sum    : out std_logic;
68         cout   : out std_logic
69
70
71     );
72 end component;
73
74
75
76 signal Internal_carry: std_logic_vector(2*width downto 0);
77 signal sum_out :std_logic_vector(2*width-1 downto 0):= (others => '0');
78 signal input_1 : std_logic_vector(2*width-1 downto 0);
79 signal input_2 : std_logic_vector(2*width-1 downto 0);
80
81
82 signal mul_output: std_logic_vector(2*width-1 downto 0);
83 signal ALU_go: std_logic;
84 signal ALU_gol: std_logic;
85
86 signal div_out : std_logic_vector(2*width-1 downto 0);
87
88 begin
89 input_1 <= x"00" & ALU_input1;
90 input_2 <= x"ff" & not ALU_input2 when ALU_switch = "01" else x"00" & ALU_input2;
91
92 Internal_carry(0) <= ALU_switch(0);
93
94 GEN_WRAPPER: for i in 0 to width-1 generate
95
96     SUBorADD: FA port map (
97         A    => input_1(i),
98         Cin => Internal_carry(i),
99         B    => input_2(i),
100        Sum  => sum_out(i),
101        Cout => Internal_carry(i+1)
102    );
103 end generate;
104
105

```

```

106
107
108     ALU_go <= 'U' when ALU_switch = "00" else
109             'U' when ALU_switch = "01" else
110             '1' when ALU_switch = "10" else
111             'U' when ALU_switch = "11";
112  ⊞ MULTITPLY: MUL generic map(mul_width => width)
113     port map(
114         mul_clk => ALU_clk,
115         mul_rst => ALU_rst,
116         go      => ALU_go,
117         M       => ALU_input1,
118         Q       => ALU_input2,
119         Product => mul_output
120
121     ⊞ );
122
123     ALU_gol <= 'U' when ALU_switch = "00" else
124             'U' when ALU_switch = "01" else
125             'U' when ALU_switch = "10" else
126             '1' when ALU_switch = "11";
127
128
129
130
131  ⊞ OUTPUT: process(ALU_switch, ALU_clk)
132     begin
133  ⊞     case ALU_switch is
134         when "00" => ALU_output <= sum_out;
135         when "01" => ALU_output <= sum_out;
136         when "10" => ALU_output <= mul_output;
137         when "11" => ALU_output <= div_out;
138         when others => ALU_output <= (others => '0');
139  ⊞     end case;
140  ⊞     end process;
141
142  ⊞ end Behavioral;

```

FIFO Module:

```
21 library IEEE;
22 USE IEEE.STD_LOGIC_1164.ALL;
23 USE IEEE.NUMERIC_STD.ALL;
24
25 entity STD_FIFO is
26   Generic (
27     constant DATA_WIDTH : positive := 8;
28     constant FIFO_DEPTH : positive := 256
29   );
30   Port (
31     CLK      : in  STD_LOGIC;
32     RST      : in  STD_LOGIC;
33     WriteEn  : in  STD_LOGIC;
34     DataIn   : in  STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
35     ReadEn   : in  STD_LOGIC;
36     DataOut  : out STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
37     Empty    : out STD_LOGIC;
38     Full     : out STD_LOGIC
39   );
40 end STD_FIFO;
41
42 architecture Behavioral of STD_FIFO is
43
44   begin
45
46     -- Memory Pointer Process
47     fifo_proc : process (CLK)
48       type FIFO_Memory is array (0 to FIFO_DEPTH - 1) of STD_LOGIC_VECTOR (DATA_WIDTH - 1 downto 0);
49       variable Memory : FIFO_Memory;
50
51       variable Head : natural range 0 to FIFO_DEPTH - 1;
52       variable Tail : natural range 0 to FIFO_DEPTH - 1;
53
54       variable Looped : boolean;
55     begin
56       if rising_edge(CLK) then
57         if RST = '1' then
58           Head := 0;
59           Tail := 0;
60
61           Looped := false;
62
63           Full  <= '0';
64           Empty <= '1';
65         else
66           if (ReadEn = '1') then
67             if ((Looped = true) or (Head /= Tail)) then
68               -- Update data output
69               DataOut <= Memory(Tail);
70
71               -- Update Tail pointer as needed
72               if (Tail = FIFO_DEPTH - 1) then
73                 Tail := 0;
74
75                 Looped := false;
```



```

76     else
77         Tail := Tail + 1;
78     end if;
79
80
81     end if;
82 end if;
83
84 if (WriteEn = '1') then
85     if ((Looped = false) or (Head /= Tail)) then
86         -- Write Data to Memory
87         Memory(Head) := DataIn;
88
89         -- Increment Head pointer as needed
90         if (Head = FIFO_DEPTH - 1) then
91             Head := 0;
92
93             Looped := true;
94         else
95             Head := Head + 1;
96         end if;
97     end if;
98 end if;
99
100 -- Update Empty and Full flags
101 if (Head = Tail) then
102     if Looped then
103         Full <= '1';
104     else
105         Empty <= '1';
106     end if;
107 else
108     Empty <= '0';
109     Full <= '0';
110 end if;
111 end if;
112 end if;
113 end process;
114
115 end Behavioral;
116

```

Analysis:

Through a large amount of problems, we were finally able to create our design compatible with UART. The main problem that occurred was the transferring of data between UART and the FPGA. Initially we had used other code found online to help with the connection, however it was difficult to do so. To solve the problem we used Tera Term which automatically connects the computer inputs to the FPGA. Our connections were simple as all code was either created from previous labs or provided from class. We have small edits such as the 7 segment display and a conversion between ASCII to binary.

Conclusion:

In this lab, we were successfully able to create a UART compatible machine utilizing FIFOs, ALUs, Barrel Shifters, and Seven Segments. Using previous code such as the ALU, Barrel Shifter and seven segments from previous labs, and the code provided during class, we were able to connect these modules with ease. Our main issue was UART as we did have trouble connecting and transferring the data correctly into the FPGA, but with endless debugging we were able to get the right ASCII input we needed to fully say our code is working.