This lab is in continuation of the previous lab, in which I asked you to work on the serial port implementation and to look at a JTAG implementation.

In this lab, we are going to complete these two exercises and extend the work.

Task 1. Verifying the working TX part of the serial port implementation provided. If you update your repository (git pull) you will find working code, but before doing that try following hints to make it work:
A. In last lab, you may have verified that the tick signal is counting as per specification (i.e. produces a tick at 163 clock cycles). This means that the FSM that controls the RX or TX part of UART should make transitions if other signals are correct.
B. Now feed data to TX part. To do this you have to turn wr_uart signal ON and supply data on w_data lines of UART. FIFO unit related to TX part of UART uses wr_uart and w_data signal. Is FIFO FSM working properly? Is the next state assigned properly? (see Ans1 below) Does FSM switch on “wr”=1, “rd”=0? (see ans2 below). Is there any difference between “wr” signal being a pulse and “wr” being “1” for a considerable amount of time? (Ans3) Verify that the TX unit receives “tx_start” and “din” signal correctly.
C. Now look at uart_tx module. Its input signals are correct as you verified in B. There are several typos in this module related to bit widths. Can you find them? (Ans4).
D. After these changes, apply a pulse on wr_uart and set w_data to “A”. Can you trace “tx” output of UART module to see “A” propagated on it. How many clock cycles it needs to send “A”?
E. Connect two uart modules together, i.e. rx of one is connected to tx of the other and vice-versa. Send “A” on one of the uart and detect it on the other side.

Ans1. No. Add the following in appropriate place:

```
else begin
        wcurr <= wnext;
        rcurr <= rnext;
        full_reg <= full_next;
        empty_reg <= empty_next;
    end
```

Ans2. No. Change “2’b0:” to “2’b10:” as second case comparison. Also change “if (rnext == wcurr) empty_next = 1'b1;” to “if (rsucc == wcurr) empty_next = 1'b1;” for correct working of FIFO.
Ans3. Yes. The FSM as coded will switch for multiple cycles.
Ans4. Following lines need correction in bit-width fields.
For urx.v:

```
reg [3:0] n_reg, n_next;
reg [3:0] b_reg, b_next;
if (reset) {state_reg, s_reg, n_reg, b_reg} <= {idle, 1'b0, 1'b0, 1'b0}
idle: if (~rx) {state_next, s_next} = {start, 3'b0};
```

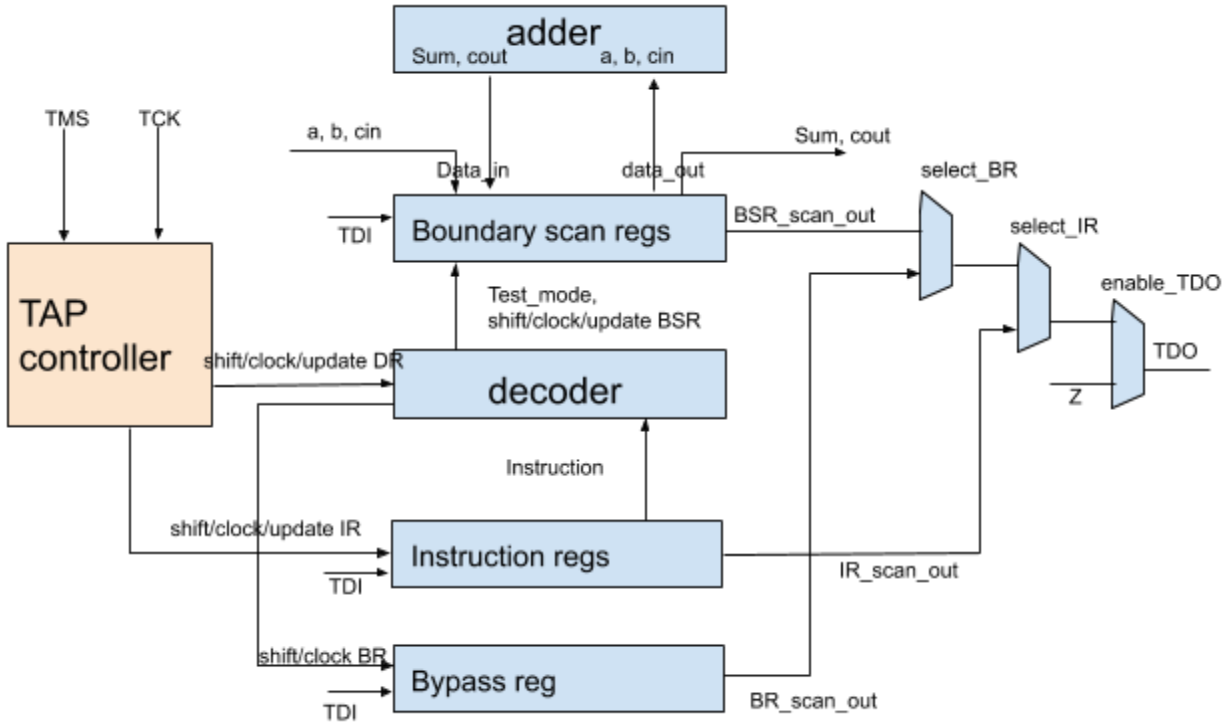
```
if (s_reg == 7) {state_next, s_next, n_next} = {data, 3'b0, 3'b0};
```

For utx.v

```
if (s_reg == 15) {state_next, s_next, n_next} = {data, 4'b0, 4'b0};
```

Task 2. Verifying the working of single bit full bit adder with JTAG port. This code is provided in jtag folder.
A. Using code provided in adder.v, make a block diagram using subparts (M0, M1, … M5) and connect them with signals as specified. (Ans1)
B. Add $dumpfile and $dumpvars command to produce a .vcd file for the simulation.
C. Show that sum of two numbers comes out correctly. Modify array_of_adder_test_patterns to see if the full adder works properly.
D. Exercise: connect two of these adder_with_taps by threading a TDI line through them. Test if bypass of data works properly.

Ans1.



Task3. Embedding JTAG TAP controller inside a serial port. Modify uart module to produce uart_with_tap along the lines of adder_with_tap. How many bits do you need in boundary scan register? Which clock signal you are going to use for the tap controller and for the serial port code?
A. Verify that you can transmit “A” and “B” as in Task1, without affecting TMS, TDI etc.
B. Verify you can use bypass data through JTAG parts as done in Task2.

Task 4.

A. Exercise: Modify the module adder_with_TAP to generate a 14 bit identifier when IDCODE instruction is executed.

B. Exercise: Chain two of the adder_with_TAP to generate a 28 bit scan for reading the idcode of the two adders.