

**Logic Design using FPGAs**

**Assignment**

**Universal Asynchronous Receiver Transmitter**

**Objective:** To understand and modify a Datapath-controller UART design.

**Assignment:**

1. Create a testbench for the given UART design and verify the functionality of the datapath and the controller modules.
2. Modify the design to include an odd parity bit in the transmission data.

**The Verilog code:**

/\*-------------------------------------------------------------------------------------------

UART transmitter design parametrizable (Datapath - Controller)

--------------------------------------------------------------------------------------------\*/

module UART\_TX #( parameter word\_size = 8)( // size of data: 8 bits

// ports connected to data-path

output Serial\_out, // serial output of data channel

input [word\_size-1 : 0] Data\_Bus, // Host data bus holding data word

// ports connected to the controller

input Load\_XMT\_datareg, // used by host to load the data register

input Byte\_ready, // used by host to signal ready

input T\_byte, // used by host to signal start of transmission

input clock, // bit clock of the transmitter

input rst\_b // rst\_bs internal registers and loads the XMT\_shftreg with ones for idle state

);

Control\_Unit M0 (

Load\_XMT\_DR, // loads Data\_Bus into Tx\_datareg - internal\_out

Load\_XMT\_shftreg, // loads Tx\_datareg into XMT\_siftreg - internal\_out

start, // launches shifting of bits in XMT\_shftreg - internal\_out

shift, // shifts bits in XMT\_shiftreg - internal\_out

clear, // clears bit\_count after last bit is sent - internal\_out

Load\_XMT\_datareg, // asserts Load\_XMT\_DR in state idle - port\_in

Byte\_ready, // asserts Load\_XMT\_shftreg in state idle - port\_in

T\_byte, // asserts start signal in state waiting - port\_in

BC\_lt\_BCmax, // indicates status of bit counter - internal\_in

clock, // port\_in

rst\_b // port\_in );

Datapath\_Unit M1 (

Serial\_out, // serial output of data channel - port\_out

BC\_lt\_BCmax, // indicates status of bit counter - internal\_out

Data\_Bus, // data bus holding data\_word - port\_in

Load\_XMT\_DR, // loads Data\_Bus into Tx\_datareg - internal\_in

Load\_XMT\_shftreg, // loads Tx\_datareg into XMT\_shftreg - internal\_in

start, // launches shifting of bits in XMT\_shftreg - internal\_in

shift, // shifts bits in XMT\_shftreg - internal\_in

clear, // clears bit\_count after last bit is sent - internal\_in

clock, // port\_in

rst\_b // port\_in );

endmodule

module Control\_Unit #(

parameter one\_hot\_count = 3, // number of one-hot states

parameter state\_count = one\_hot\_count, // number of bits in state register

parameter idle = 3'b001,

waiting = 3'b010,

sending = 3'b100

)(

output reg Load\_XMT\_DR, // loads Data\_Bus into Tx\_datareg - internal\_out

output reg Load\_XMT\_shftreg, // loads Tx\_datareg into XMT\_shftreg - internal\_out

output reg start, // launches shifting of bits in XMT\_shftreg - internal\_out

output reg shift, // shifts bits in XMT\_shftreg - internal\_out

output reg clear, // clears bit\_count after last bit is sent - internal\_out

input Load\_XMT\_datareg, // asserts Load\_XMT\_DR in state idle - port\_in

input Byte\_ready, // asserts Load\_XMT\_shftreg in state idle - port\_in

input T\_byte, // asserts start signal in state waiting - port\_in

input BC\_lt\_BCmax, // indicates status of bit counter - internal\_in

input clock, // port\_in

input rst\_b // port\_in

);

reg [state\_count-1 : 0] state, next\_state; // state machine controller

always @ (posedge clock, negedge rst\_b)

begin: State\_transition

if (rst\_b == 1'b0)

state <= idle; else

state <= next\_state; end

always @ (state, Load\_XMT\_datareg, Byte\_ready, T\_byte, BC\_lt\_BCmax)

begin: Output\_and\_next\_state

Load\_XMT\_DR = 0; Load\_XMT\_shftreg = 0;

start = 0; shift = 0; clear = 0;

next\_state = idle;

case (state)

idle: if (Load\_XMT\_datareg == 1'b1) begin

Load\_XMT\_DR = 1;

next\_state = idle; end

else if (Byte\_ready == 1'b1) begin

Load\_XMT\_shftreg = 1;

next\_state = waiting; end

waiting: if (T\_byte == 1'b1) begin

start = 1;

next\_state = sending; end

else next\_state = waiting;

sending: if (BC\_lt\_BCmax) begin

shift = 1;

next\_state = sending; end

else begin

clear = 1;

next\_state = idle; end

default: next\_state = idle;

endcase

end

endmodule

module Datapath\_Unit #(

parameter word\_size = 8,

size\_bit\_count = 3, // size of the bit counter. Must count word\_size + 1

all\_ones = {(word\_size+1){1'b1}} // 9 bits of ones

)(

output Serial\_out, // serial output of data channel - port\_out

output BC\_lt\_BCmax, // indicates status of bit counter - internal\_out

input [word\_size-1 : 0] Data\_Bus, // data bus holding data\_word - port\_in

input Load\_XMT\_DR, // loads Data\_Bus into Tx\_datareg - internal\_in

input Load\_XMT\_shftreg, // loads Tx\_datareg into XMT\_shftreg - internal\_in

input start, // launches shifting of bits in XMT\_shftreg - internal\_in

input shift, // shifts bits in XMT\_shftreg - internal\_in

input clear, // clears bit\_count after last bit is sent - internal\_in

input clock, // port\_in

input rst\_b // port\_in

);

reg [word\_size -1 : 0] Tx\_datareg; // transmit data register

reg [word\_size : 0] XMT\_shftreg; // transmit shift register {data, start bit}

reg [size\_bit\_count : 0] bit\_count; // counts the bits that are transmitted

assign Serial\_out = XMT\_shftreg[0];

assign BC\_lt\_BCmax = (bit\_count < word\_size+1);

always @ (posedge clock, negedge rst\_b)

if (rst\_b == 0) begin

XMT\_shftreg <= all\_ones;

bit\_count <= 0;

end

else begin: Register\_Transfers

if (Load\_XMT\_DR == 1'b1)

Tx\_datareg <= Data\_Bus; // get the data word from data bus

if (Load\_XMT\_shftreg == 1'b1)

XMT\_shftreg <= {Tx\_datareg, 1'b1}; // load shift reg and insert start bit

if (start == 1'b1)

XMT\_shftreg [0] <= 0; // signal start of transmission

if (clear == 1'b1)

bit\_count <= 0;

if (shift == 1'b1) begin

XMT\_shftreg <= {1'b1, XMT\_shftreg [word\_size : 1]}; // shift right and fill with 1's

bit\_count <= bit\_count + 1;

end

end

endmodule