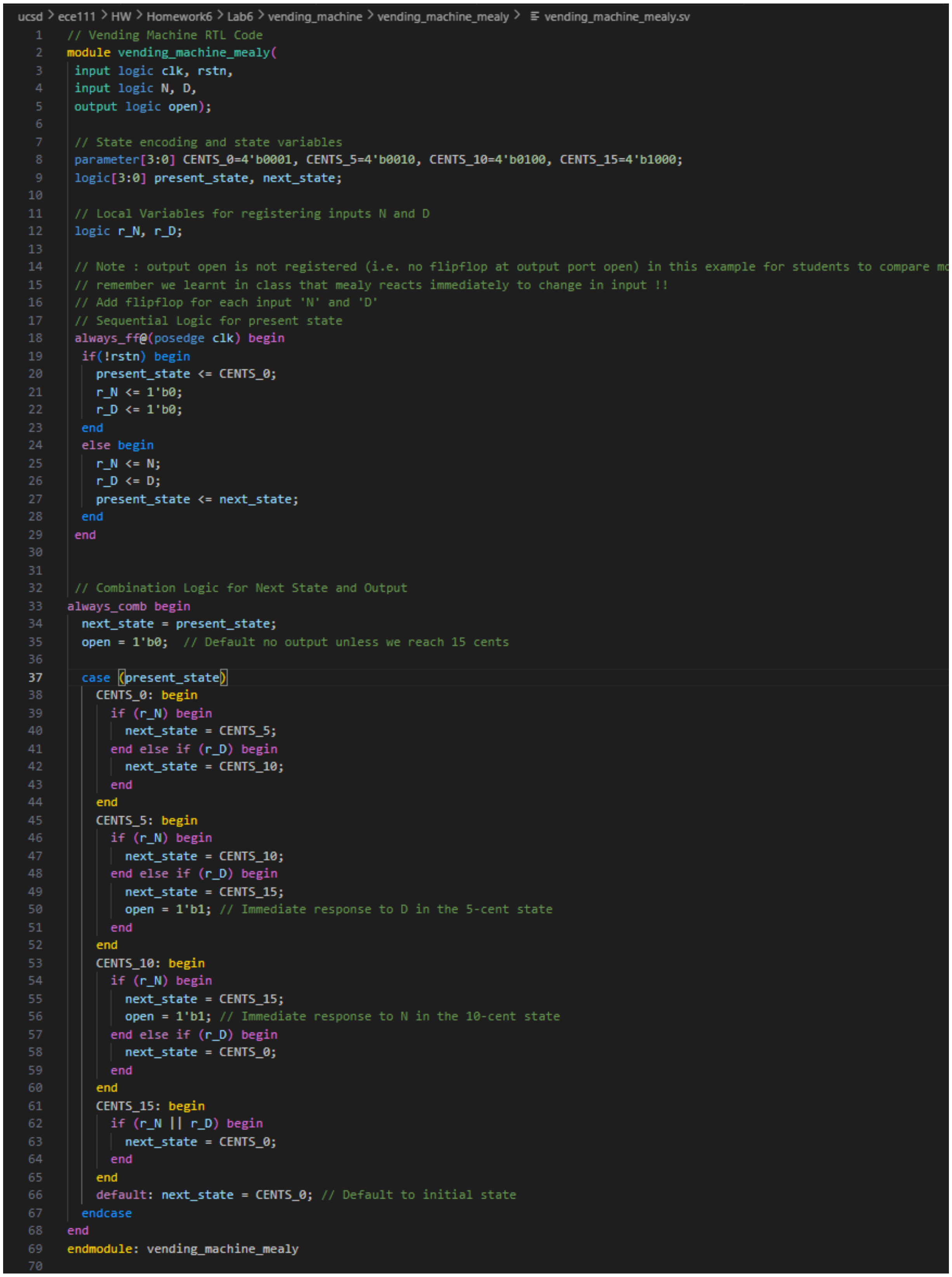
Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

PART A

vending_machine_mealy.sv Design Code

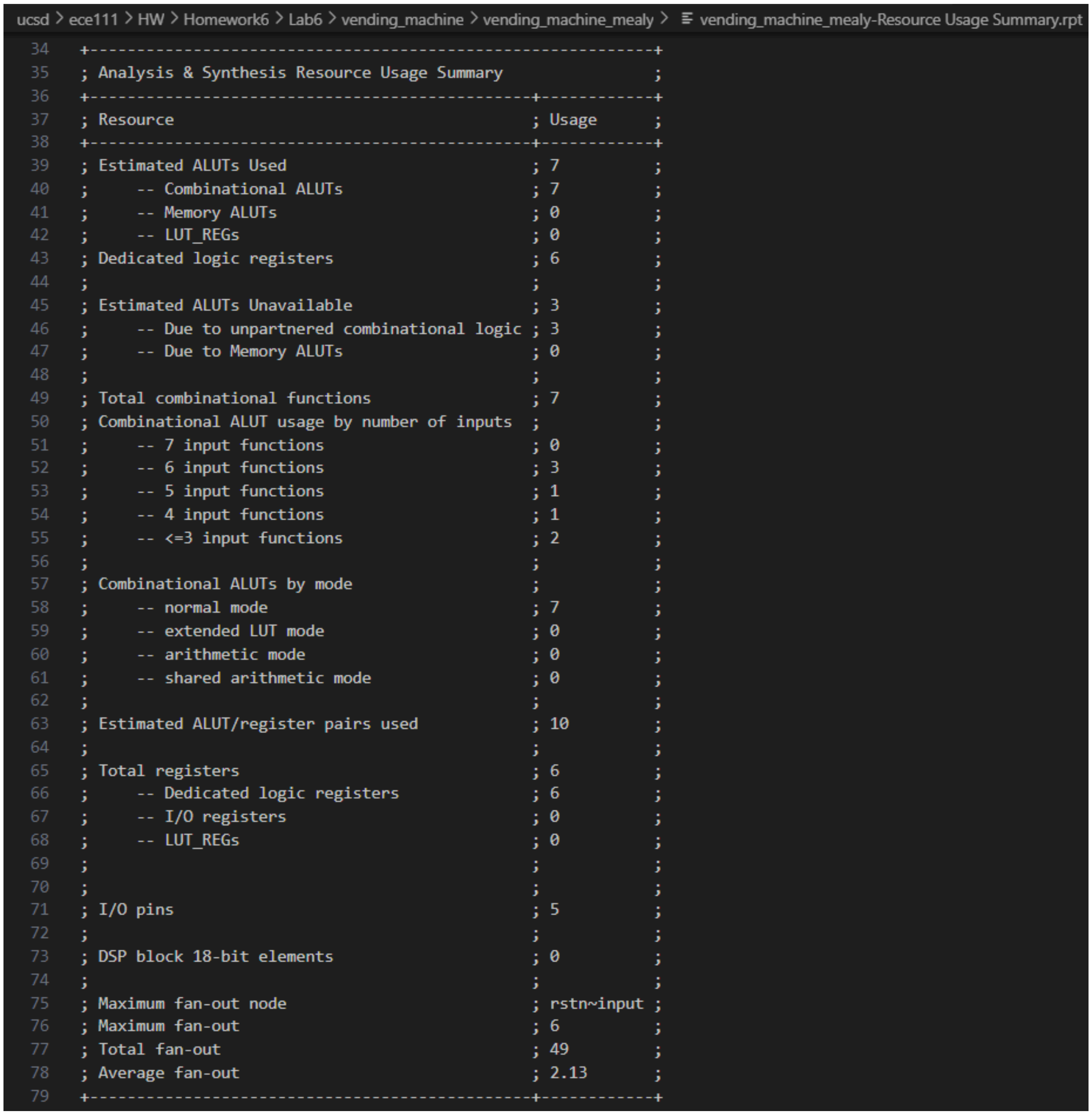


ucsd > ece111 > HW > Homework6 > Lab6 > vending_machine > vending_machine_mealy > vending_machine_mealy.sv

```
// Vending Machine RTL Code
module vending_machine_mealy(
 input logic clk, rstn,
 input logic N, D,
 output logic open);

 // State encoding and state variables
 parameter[3:0] CENTS_0=4'b0001, CENTS_5=4'b0010, CENTS_10=4'b0100, CENTS_15=4'b1000;
 logic[3:0] present_state, next_state;

 // Local Variables for registering inputs N and D
 logic r_N, r_D;

 // Note : output open is not registered (i.e. no flipflop at output port open) in this example for students to compare mo
 // remember we learnt in class that mealy reacts immediately to change in input !!
 // Add flipflop for each input 'N' and 'D'
 // Sequential Logic for present state
 always_ff@(posedge clk) begin
  if(!rstn) begin
    present_state <= CENTS_0;
    r_N <= 1'b0;
    r_D <= 1'b0;
  end
  else begin
    r_N <= N;
    r_D <= D;
    present_state <= next_state;
  end
 end


 // Combination Logic for Next State and Output
always_comb begin
  next_state = present_state;
  open = 1'b0;  // Default no output unless we reach 15 cents

  case (present_state)
    CENTS_0: begin
      if (r_N) begin
        next_state = CENTS_5;
      end else if (r_D) begin
        next_state = CENTS_10;
      end
    end
    CENTS_5: begin
      if (r_N) begin
        next_state = CENTS_10;
      end else if (r_D) begin
        next_state = CENTS_15;
        open = 1'b1; // Immediate response to D in the 5-cent state
      end
    end
    CENTS_10: begin
      if (r_N) begin
        next_state = CENTS_15;
        open = 1'b1; // Immediate response to N in the 10-cent state
      end else if (r_D) begin
        next_state = CENTS_0;
      end
    end
    CENTS_15: begin
      if (r_N || r_D) begin
        next_state = CENTS_0;
      end
    end
    default: next_state = CENTS_0; // Default to initial state
  endcase
end
endmodule: vending_machine_mealy
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

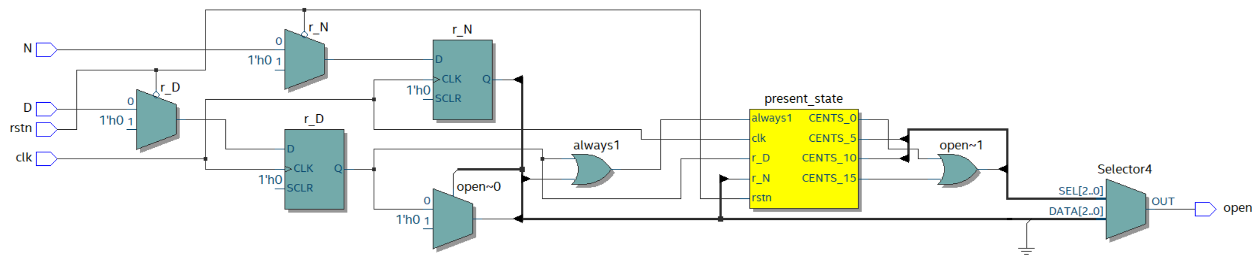
vending_machine_mealy.sv Resource Usage Summary


ucsd > ece111 > HW > Homework6 > Lab6 > vending_machine > vending_machine_mealy > vending_machine_mealy-Resource Usage Summary.rpt

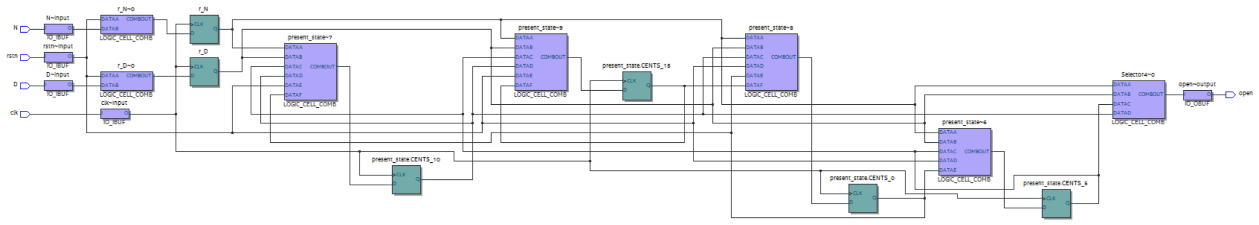
```
+-----------------------------------------------------------+
; Analysis & Synthesis Resource Usage Summary               ;
+---------------------------------------------+-------------+
; Resource                                    ; Usage       ;
+---------------------------------------------+-------------+
; Estimated ALUTs Used                        ; 7           ;
;     -- Combinational ALUTs                  ; 7           ;
;     -- Memory ALUTs                         ; 0           ;
;     -- LUT_REGs                             ; 0           ;
; Dedicated logic registers                   ; 6           ;
;                                             ;             ;
; Estimated ALUTs Unavailable                 ; 3           ;
;     -- Due to unpartnered combinational logic ; 3         ;
;     -- Due to Memory ALUTs                  ; 0           ;
;                                             ;             ;
; Total combinational functions               ; 7           ;
; Combinational ALUT usage by number of inputs ;            ;
;     -- 7 input functions                    ; 0           ;
;     -- 6 input functions                    ; 3           ;
;     -- 5 input functions                    ; 1           ;
;     -- 4 input functions                    ; 1           ;
;     -- <=3 input functions                  ; 2           ;
;                                             ;             ;
; Combinational ALUTs by mode                 ;             ;
;     -- normal mode                          ; 7           ;
;     -- extended LUT mode                    ; 0           ;
;     -- arithmetic mode                      ; 0           ;
;     -- shared arithmetic mode               ; 0           ;
;                                             ;             ;
; Estimated ALUT/register pairs used          ; 10          ;
;                                             ;             ;
; Total registers                             ; 6           ;
;     -- Dedicated logic registers            ; 6           ;
;     -- I/O registers                        ; 0           ;
;     -- LUT_REGs                             ; 0           ;
;                                             ;             ;
;                                             ;             ;
; I/O pins                                    ; 5           ;
;                                             ;             ;
; DSP block 18-bit elements                   ; 0           ;
;                                             ;             ;
; Maximum fan-out node                        ; rstn~input  ;
; Maximum fan-out                             ; 6           ;
; Total fan-out                               ; 49          ;
; Average fan-out                             ; 2.13        ;
+---------------------------------------------+-------------+
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

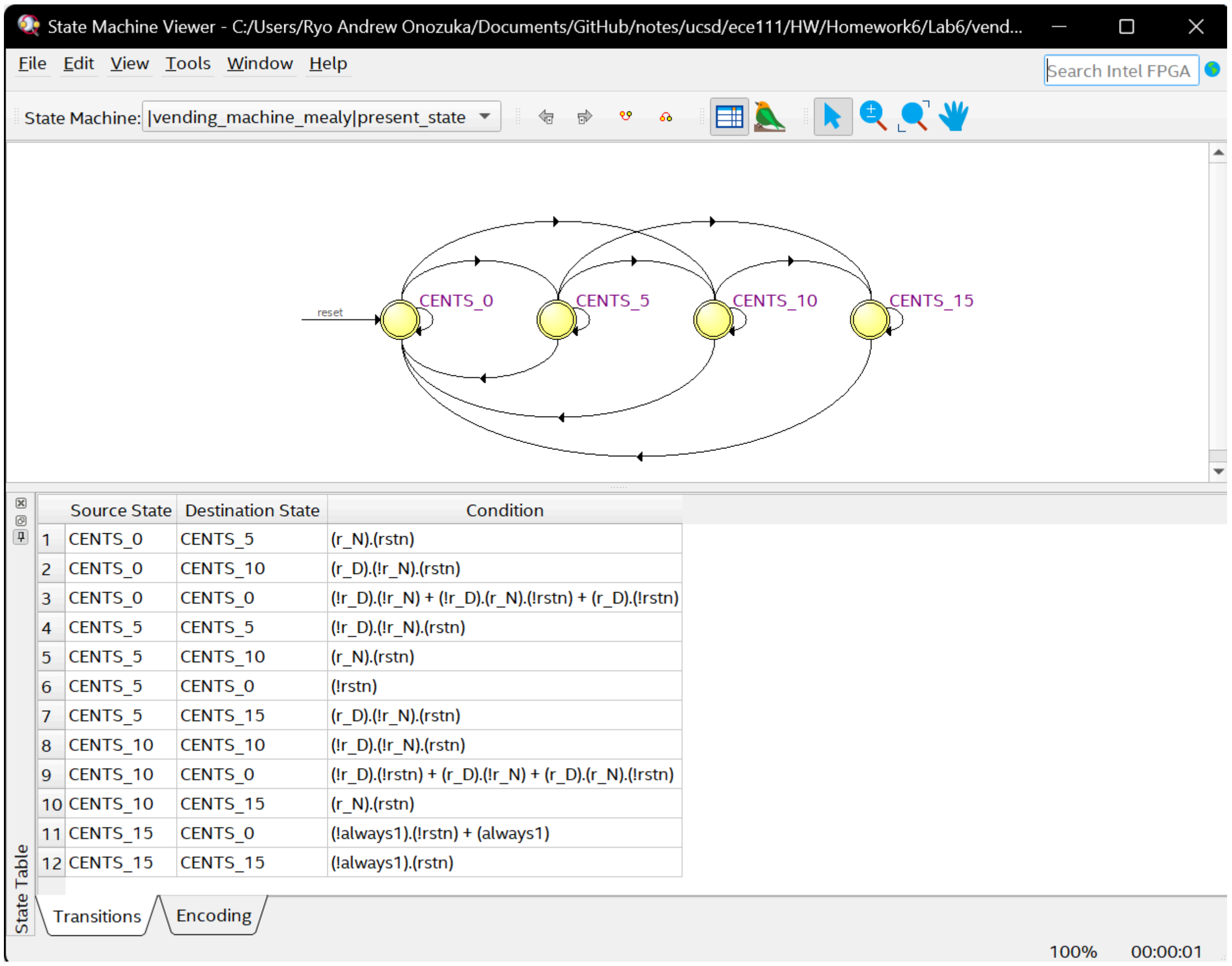
vending_machine_mealy.sv RTL viewer



vending_machine_mealy.sv post mapping viewer: not required



vending_machine_mealy.sv State Machine Viewer + State Transition Table

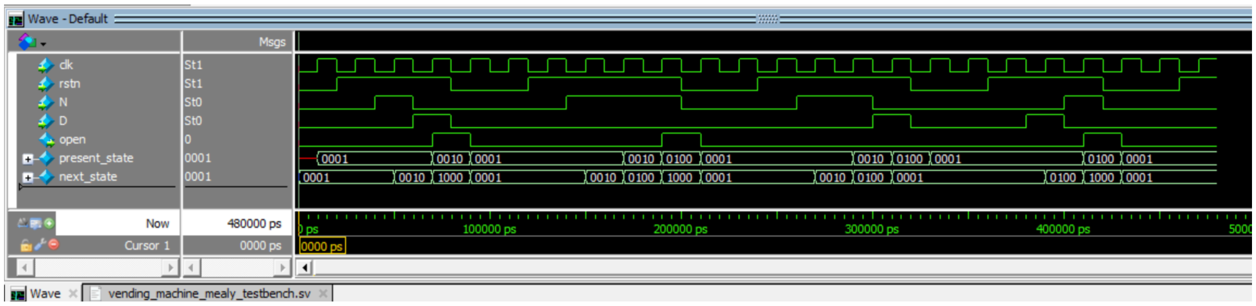

State Machine Viewer - C:/Users/Ryo Andrew Onozuka/Documents/GitHub/notes/ucsd/ece111/HW/Homework6/Lab6/vend...
File Edit View Tools Window Help
Search Intel FPGA
State Machine: |vending_machine_mealy|present_state
reset
CENTS_0
CENTS_5
CENTS_10
CENTS_15
State Table
Transitions
Encoding
100%
00:00:01

	Source State	Destination State	Condition
1	CENTS_0	CENTS_5	(r_N).(rstn)
2	CENTS_0	CENTS_10	(r_D).(!r_N).(rstn)
3	CENTS_0	CENTS_0	(!r_D).(!r_N) + (!r_D).(r_N).(!rstn) + (r_D).(!rstn)
4	CENTS_5	CENTS_5	(!r_D).(!r_N).(rstn)
5	CENTS_5	CENTS_10	(r_N).(rstn)
6	CENTS_5	CENTS_0	(!rstn)
7	CENTS_5	CENTS_15	(r_D).(!r_N).(rstn)
8	CENTS_10	CENTS_10	(!r_D).(!r_N).(rstn)
9	CENTS_10	CENTS_0	(!r_D).(!rstn) + (r_D).(!r_N) + (r_D).(r_N).(!rstn)
10	CENTS_10	CENTS_15	(r_N).(rstn)
11	CENTS_15	CENTS_0	(!always1).(!rstn) + (always1)
12	CENTS_15	CENTS_15	(!always1).(rstn)

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

simulation transcript: not required (no print statements)

simulation wavelength results explanation:



The Mealy FSM determines its output based on both the current state and the immediate inputs. In our vending machine example, this means that the output signal open (which indicates candy is dispensed) can be asserted in the same clock cycle that an input change occurs if the right conditions are met. Specifically, in the Mealy FSM code, open is set to 1 as soon as the inputs (Nickel N or Dime D) add up to 15 cents, regardless of whether the FSM completes a full state transition.

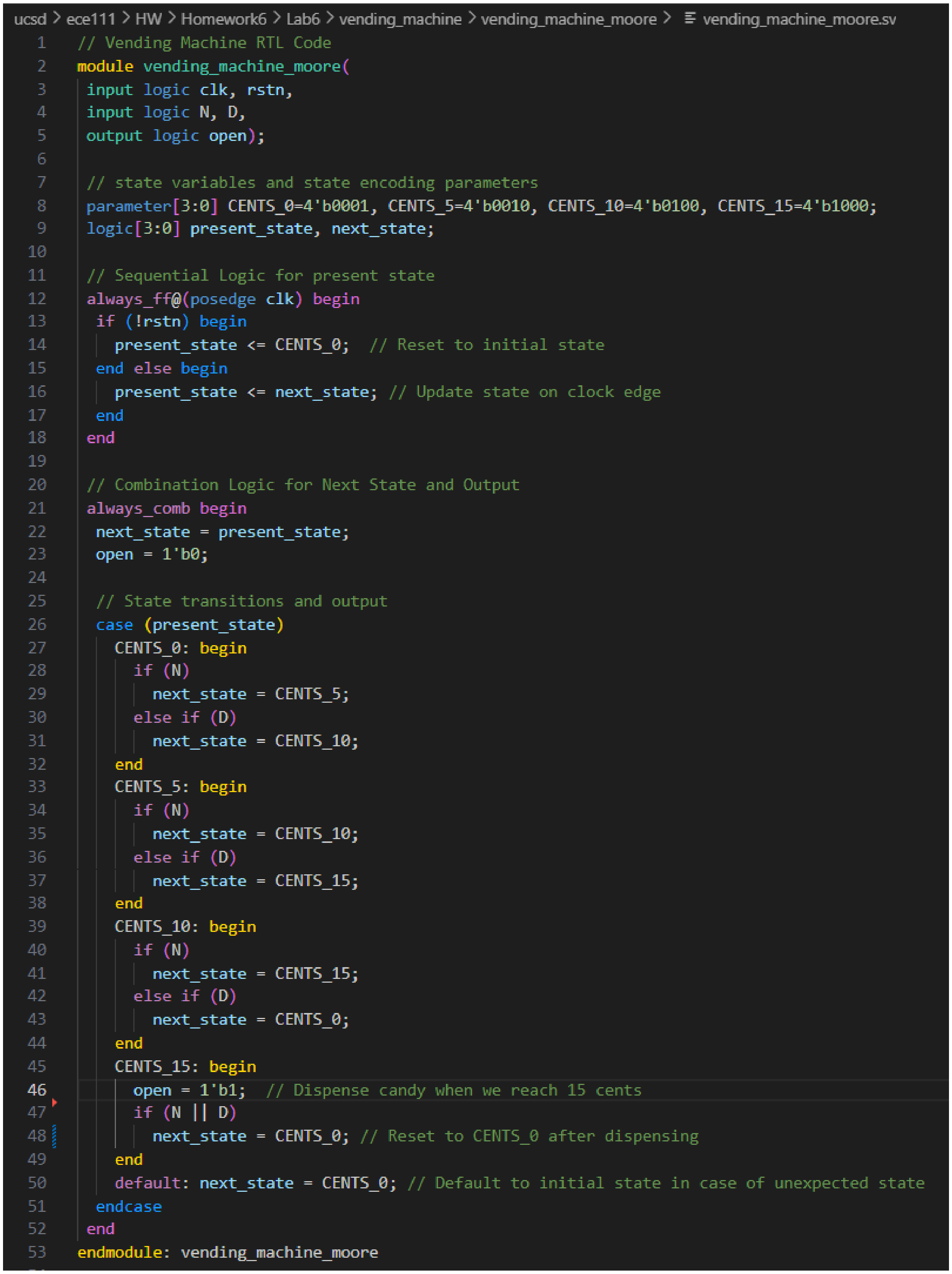
For example:

- If the FSM is in the CENTS_5 state and a Dime (D) is deposited, open is asserted immediately, as the combinational logic recognizes that the total amount now meets the requirement.
- Similarly, in the CENTS_10 state, a Nickel (N) input would cause open to go high in the same cycle, as the total is now sufficient.

This characteristic allows the Mealy FSM to react instantly to input changes, reflecting its structure where outputs are directly tied to both states and inputs. In a waveform, open will toggle as soon as the cumulative inputs reach the threshold, even if the state is in transition. This immediate responsiveness is what makes the Mealy FSM have a longer design but faster in reacting to input conditions, which can be seen by the fewer changes in next state as compared to the Moore waveforms a few pages later.

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

vending_machine_moore.sv Design Code

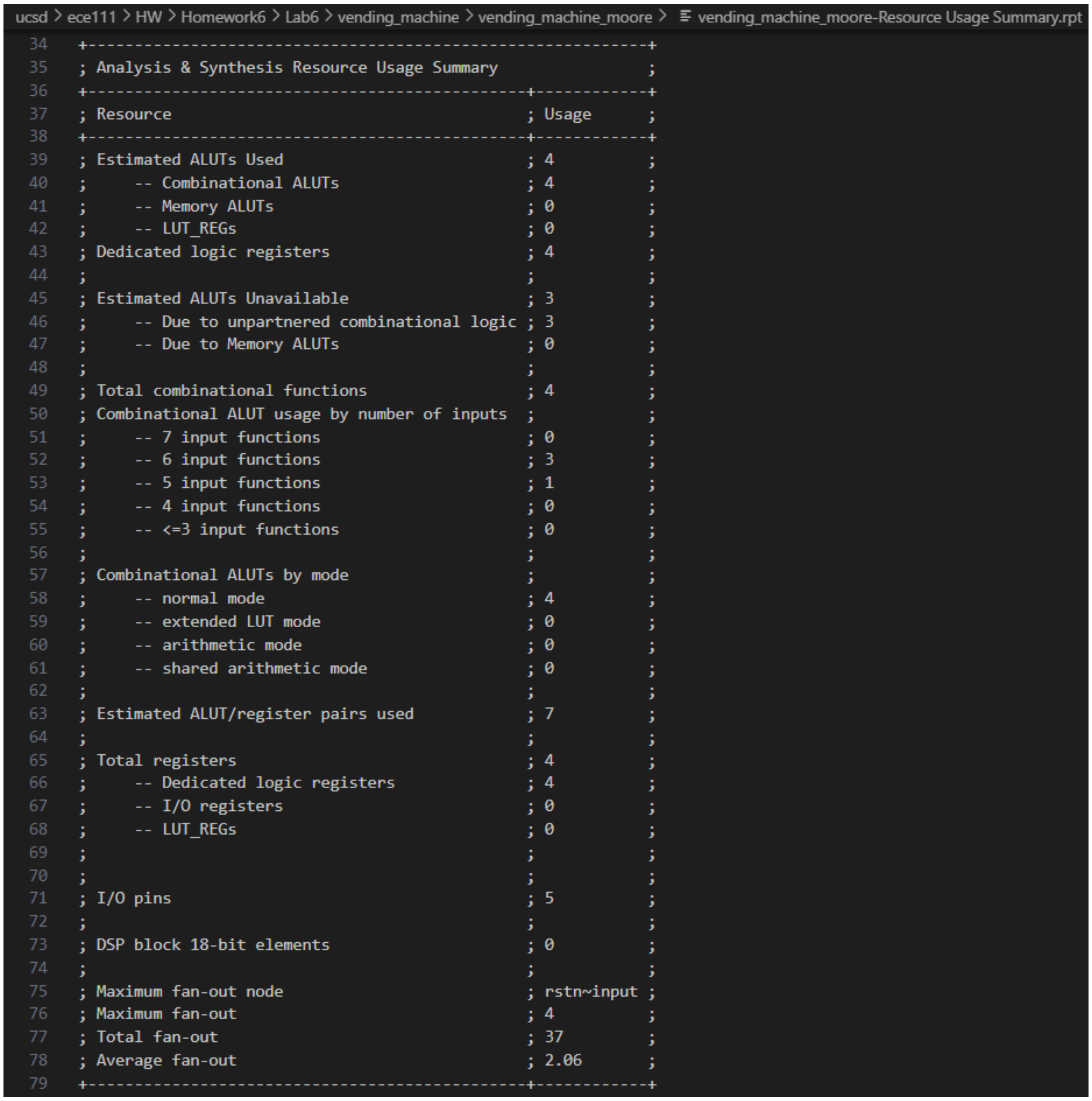


ucsd > ece111 > HW > Homework6 > Lab6 > vending_machine > vending_machine_moore > vending_machine_moore.sv

```
// Vending Machine RTL Code
module vending_machine_moore(
 input logic clk, rstn,
 input logic N, D,
 output logic open);

 // state variables and state encoding parameters
 parameter[3:0] CENTS_0=4'b0001, CENTS_5=4'b0010, CENTS_10=4'b0100, CENTS_15=4'b1000;
 logic[3:0] present_state, next_state;

 // Sequential Logic for present state
 always_ff@(posedge clk) begin
  if (!rstn) begin
    present_state <= CENTS_0;  // Reset to initial state
  end else begin
    present_state <= next_state; // Update state on clock edge
  end
 end

 // Combination Logic for Next State and Output
 always_comb begin
  next_state = present_state;
  open = 1'b0;

  // State transitions and output
  case (present_state)
    CENTS_0: begin
      if (N)
        next_state = CENTS_5;
      else if (D)
        next_state = CENTS_10;
    end
    CENTS_5: begin
      if (N)
        next_state = CENTS_10;
      else if (D)
        next_state = CENTS_15;
    end
    CENTS_10: begin
      if (N)
        next_state = CENTS_15;
      else if (D)
        next_state = CENTS_0;
    end
    CENTS_15: begin
      open = 1'b1;  // Dispense candy when we reach 15 cents
      if (N || D)
        next_state = CENTS_0; // Reset to CENTS_0 after dispensing
    end
    default: next_state = CENTS_0; // Default to initial state in case of unexpected state
  endcase
 end
endmodule: vending_machine_moore
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

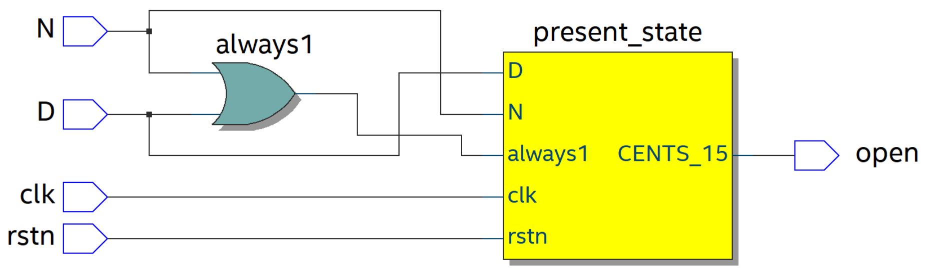
vending_machine_moore.sv Resource Usage Summary


ucsd > ece111 > HW > Homework6 > Lab6 > vending_machine > vending_machine_moore > vending_machine_moore-Resource Usage Summary.rpt

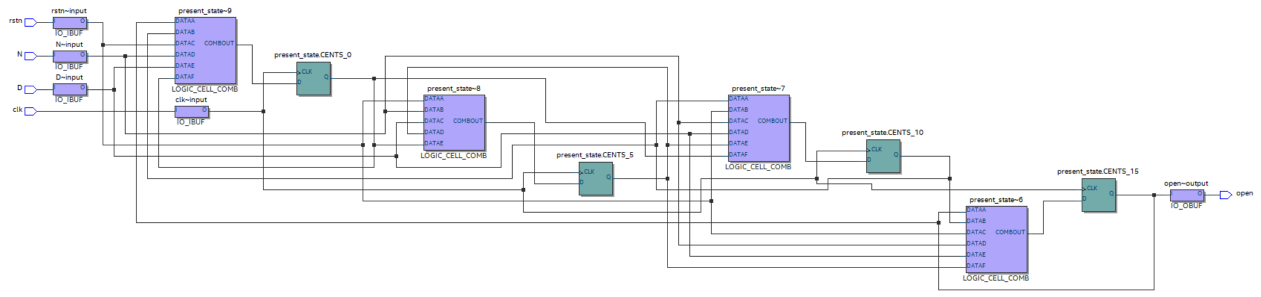
```
+-----------------------------------------------------------+
; Analysis & Synthesis Resource Usage Summary               ;
+-----------------------------------------------+-----------+
; Resource                                      ; Usage     ;
+-----------------------------------------------+-----------+
; Estimated ALUTs Used                          ; 4         ;
;     -- Combinational ALUTs                    ; 4         ;
;     -- Memory ALUTs                           ; 0         ;
;     -- LUT_REGs                               ; 0         ;
; Dedicated logic registers                     ; 4         ;
;                                               ;           ;
; Estimated ALUTs Unavailable                   ; 3         ;
;     -- Due to unpartnered combinational logic ; 3         ;
;     -- Due to Memory ALUTs                    ; 0         ;
;                                               ;           ;
; Total combinational functions                 ; 4         ;
; Combinational ALUT usage by number of inputs  ;           ;
;     -- 7 input functions                      ; 0         ;
;     -- 6 input functions                      ; 3         ;
;     -- 5 input functions                      ; 1         ;
;     -- 4 input functions                      ; 0         ;
;     -- <=3 input functions                    ; 0         ;
;                                               ;           ;
; Combinational ALUTs by mode                   ;           ;
;     -- normal mode                            ; 4         ;
;     -- extended LUT mode                      ; 0         ;
;     -- arithmetic mode                        ; 0         ;
;     -- shared arithmetic mode                 ; 0         ;
;                                               ;           ;
; Estimated ALUT/register pairs used            ; 7         ;
;                                               ;           ;
; Total registers                               ; 4         ;
;     -- Dedicated logic registers              ; 4         ;
;     -- I/O registers                          ; 0         ;
;     -- LUT_REGs                               ; 0         ;
;                                               ;           ;
;                                               ;           ;
; I/O pins                                      ; 5         ;
;                                               ;           ;
; DSP block 18-bit elements                     ; 0         ;
;                                               ;           ;
; Maximum fan-out node                          ; rstn~input ;
; Maximum fan-out                               ; 4         ;
; Total fan-out                                 ; 37        ;
; Average fan-out                               ; 2.06      ;
+-----------------------------------------------+-----------+
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

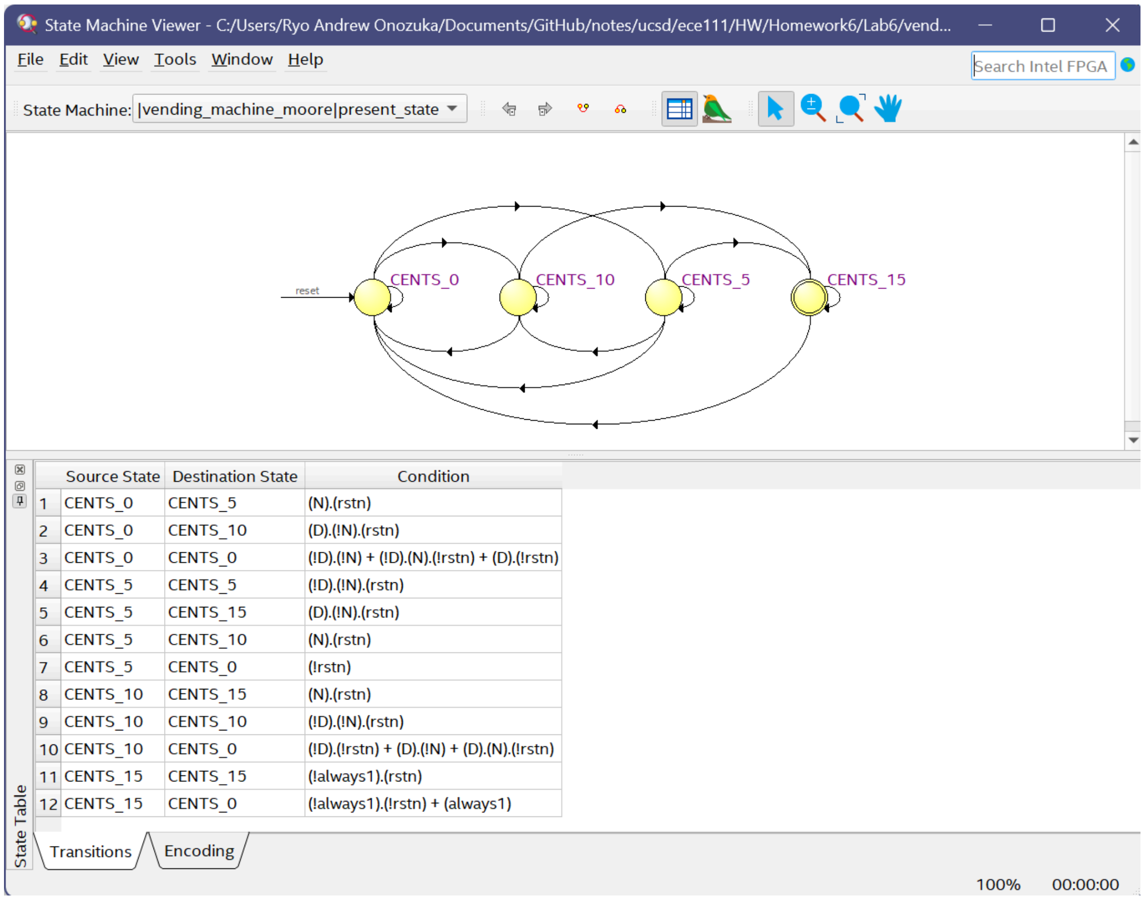
vending_machine_moore.sv RTL viewer



vending_machine_moore.sv post mapping viewer: not required



vending_machine_moore.sv State Machine Viewer + State Transition Table

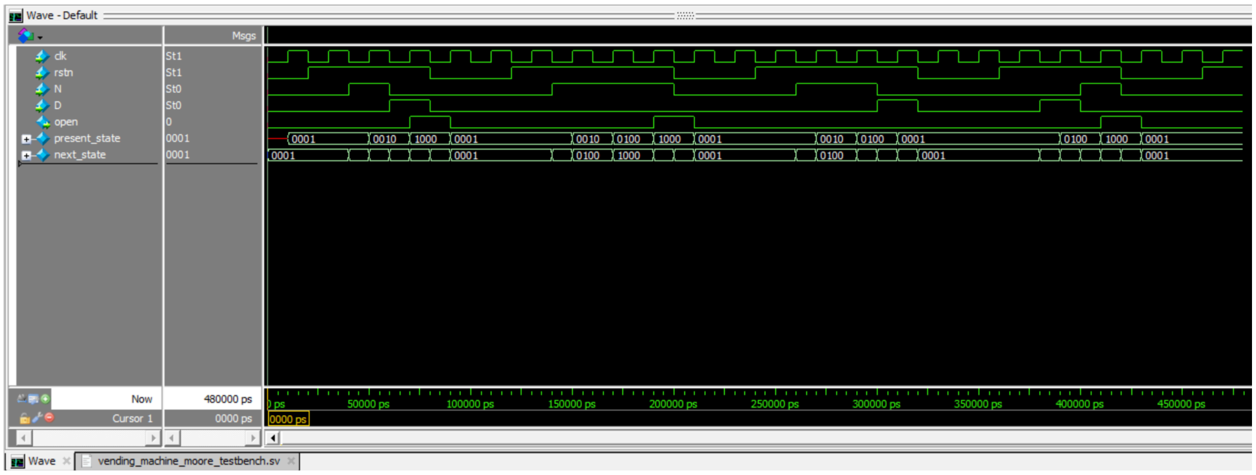

State Machine Viewer - C:/Users/Ryo Andrew Onozuka/Documents/GitHub/notes/ucsd/ece111/HW/Homework6/Lab6/vend...
File
Edit
View
Tools
Window
Help
Search Intel FPGA
State Machine: |vending_machine_moore|present_state
reset
CENTS_0
CENTS_10
CENTS_5
CENTS_15
State Table
Transitions
Encoding
100%
00:00:00

	Source State	Destination State	Condition
1	CENTS_0	CENTS_5	(N).(rstn)
2	CENTS_0	CENTS_10	(D).(!N).(rstn)
3	CENTS_0	CENTS_0	(!D).(!N) + (!D).(N).(!rstn) + (D).(!rstn)
4	CENTS_5	CENTS_5	(!D).(!N).(rstn)
5	CENTS_5	CENTS_15	(D).(!N).(rstn)
6	CENTS_5	CENTS_10	(N).(rstn)
7	CENTS_5	CENTS_0	(!rstn)
8	CENTS_10	CENTS_15	(N).(rstn)
9	CENTS_10	CENTS_10	(!D).(!N).(rstn)
10	CENTS_10	CENTS_0	(!D).(!rstn) + (D).(!N) + (D).(N).(!rstn)
11	CENTS_15	CENTS_15	(!always1).(rstn)
12	CENTS_15	CENTS_0	(!always1).(!rstn) + (always1)

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

simulation transcript: not required (no print statements)

simulation wavelength results explanation:



The Moore FSM, in contrast to the Mealy, produces outputs based solely on the current state. In this vending machine design, open is asserted only when the FSM reaches a specific state, CENTS_15. The output does not directly depend on the inputs but is tied strictly to the state machine's progression.

For example:

- If the FSM is in CENTS_10 and a Nickel (N) is deposited, the FSM transitions to CENTS_15 on the next clock edge. Only upon reaching this state does open get asserted, indicating that candy is dispensed.
- Regardless of the exact timing of the N or D inputs, the Moore FSM will only set open to 1 when it is fully in the CENTS_15 state.

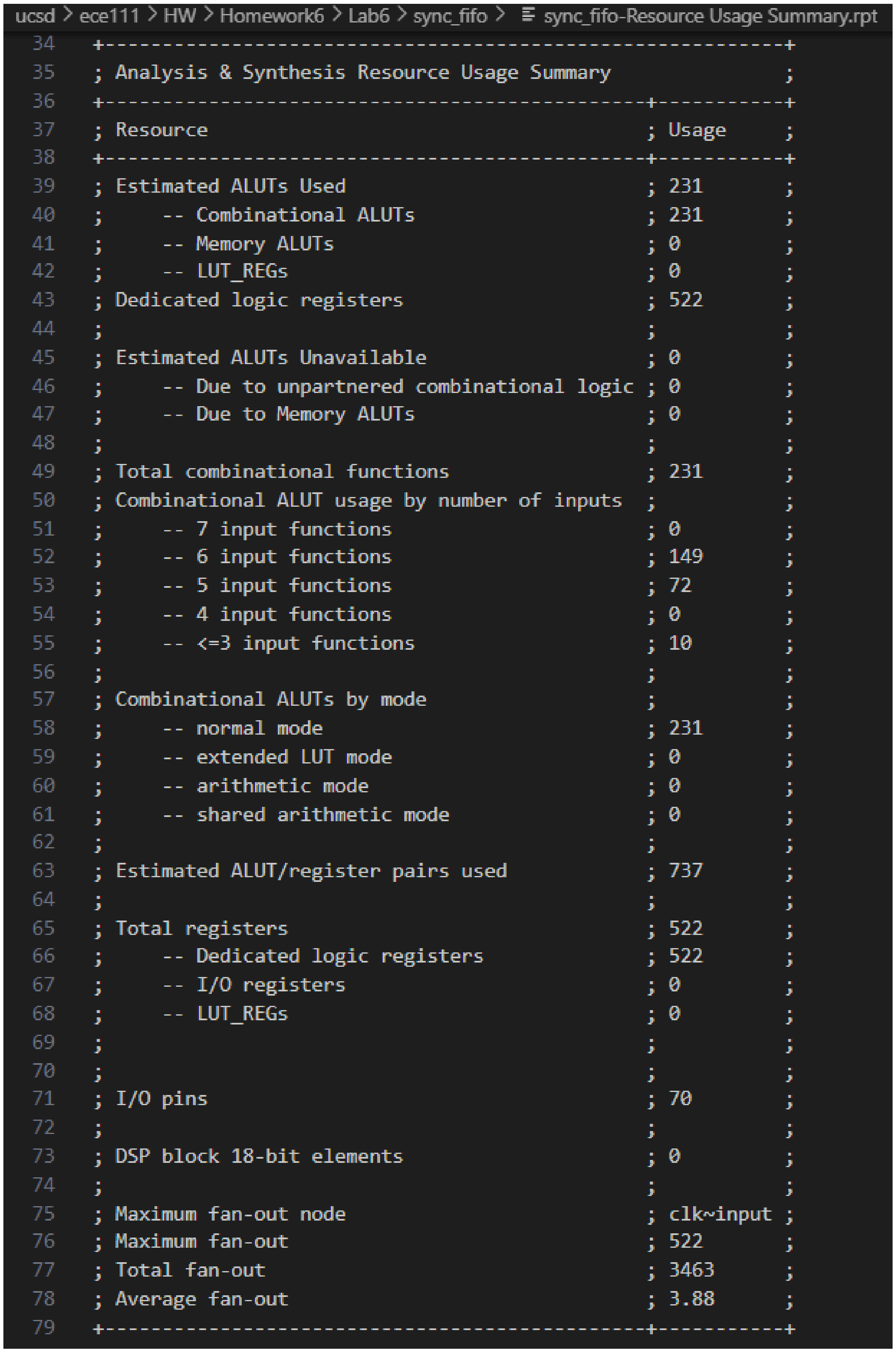
This state-based approach ensures that outputs are stable and change only when the FSM transitions to a new state. The Moore FSM structure is simpler in terms of length of code and less sensitive to rapid input fluctuations as outputs don't directly respond to inputs, but we can see how it requires more state transitions compared to the mealy waveforms.

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

PART B

sync_fifo.sv Design Code: not required

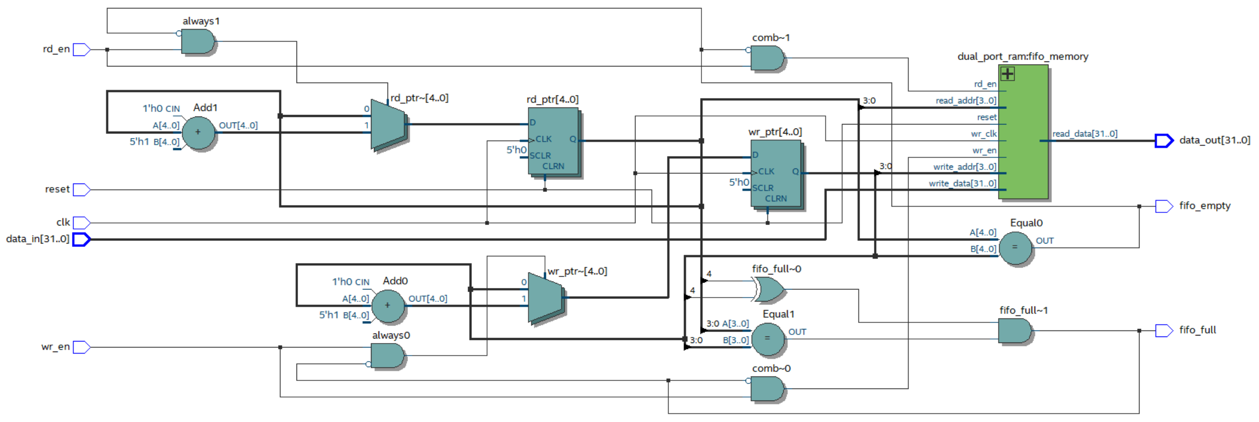
sync_fifo.sv Resource Usage Summary


ucsd > ece111 > HW > Homework6 > Lab6 > sync_fifo > sync_fifo-Resource Usage Summary.rpt

```
+-------------------------------------------------------------+
; Analysis & Synthesis Resource Usage Summary                ;
+-------------------------------------------------+-----------+
; Resource                                        ; Usage     ;
+-------------------------------------------------+-----------+
; Estimated ALUTs Used                            ; 231       ;
;     -- Combinational ALUTs                      ; 231       ;
;     -- Memory ALUTs                             ; 0         ;
;     -- LUT_REGs                                 ; 0         ;
; Dedicated logic registers                       ; 522       ;
;                                                 ;           ;
; Estimated ALUTs Unavailable                     ; 0         ;
;     -- Due to unpartnered combinational logic   ; 0         ;
;     -- Due to Memory ALUTs                      ; 0         ;
;                                                 ;           ;
; Total combinational functions                   ; 231       ;
; Combinational ALUT usage by number of inputs    ;           ;
;     -- 7 input functions                        ; 0         ;
;     -- 6 input functions                        ; 149       ;
;     -- 5 input functions                        ; 72        ;
;     -- 4 input functions                        ; 0         ;
;     -- <=3 input functions                      ; 10        ;
;                                                 ;           ;
; Combinational ALUTs by mode                     ;           ;
;     -- normal mode                              ; 231       ;
;     -- extended LUT mode                        ; 0         ;
;     -- arithmetic mode                          ; 0         ;
;     -- shared arithmetic mode                   ; 0         ;
;                                                 ;           ;
; Estimated ALUT/register pairs used              ; 737       ;
;                                                 ;           ;
; Total registers                                 ; 522       ;
;     -- Dedicated logic registers                ; 522       ;
;     -- I/O registers                            ; 0         ;
;     -- LUT_REGs                                 ; 0         ;
;                                                 ;           ;
;                                                 ;           ;
; I/O pins                                        ; 70        ;
;                                                 ;           ;
; DSP block 18-bit elements                       ; 0         ;
;                                                 ;           ;
; Maximum fan-out node                            ; clk~input ;
; Maximum fan-out                                 ; 522       ;
; Total fan-out                                   ; 3463      ;
; Average fan-out                                 ; 3.88      ;
+-------------------------------------------------+-----------+
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

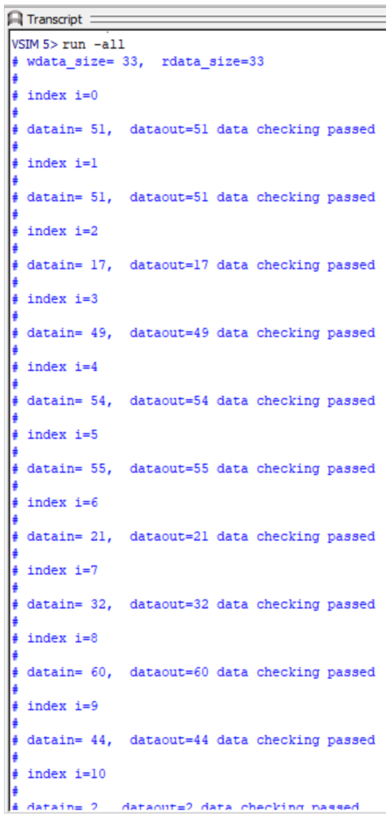
sync_fifo.sv RTL viewer



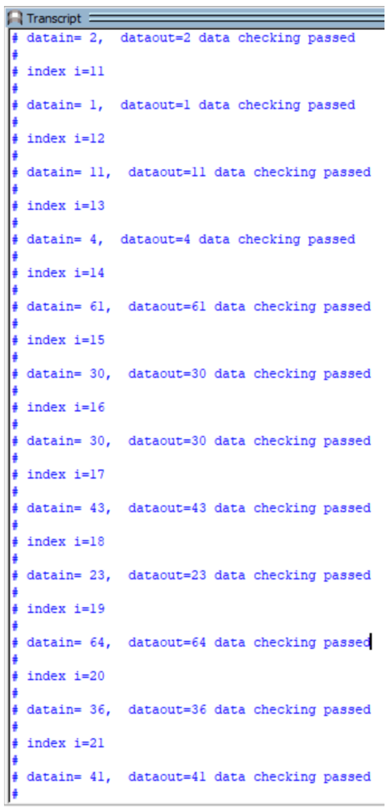
sync_fifo.sv post mapping viewer: not required

sync_fifo.sv State Machine Viewer + State Transition Table: not required

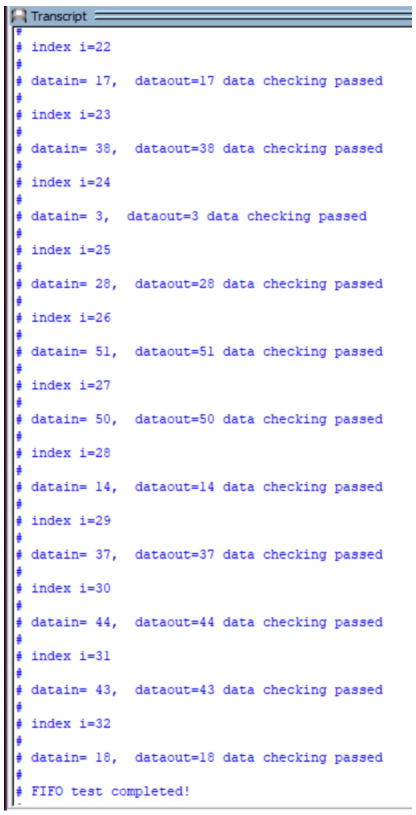
full simulation transcript:



```
VSIM 5> run -all
# wdata_size= 33,  rdata_size=33
#
# index i=0
#
# datain= 51,  dataout=51 data checking passed
#
# index i=1
#
# datain= 51,  dataout=51 data checking passed
#
# index i=2
#
# datain= 17,  dataout=17 data checking passed
#
# index i=3
#
# datain= 49,  dataout=49 data checking passed
#
# index i=4
#
# datain= 54,  dataout=54 data checking passed
#
# index i=5
#
# datain= 55,  dataout=55 data checking passed
#
# index i=6
#
# datain= 21,  dataout=21 data checking passed
#
# index i=7
#
# datain= 32,  dataout=32 data checking passed
#
# index i=8
#
# datain= 60,  dataout=60 data checking passed
#
# index i=9
#
# datain= 44,  dataout=44 data checking passed
#
# index i=10
#
# datain= 2,  dataout=2 data checking passed
```



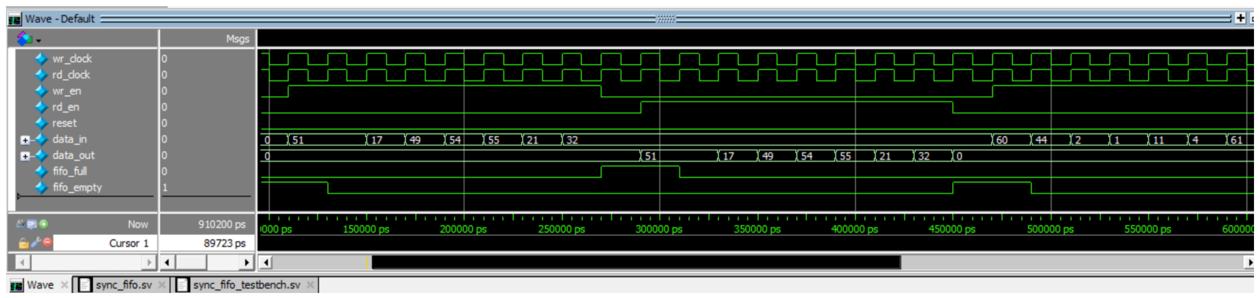
```
# datain= 2,  dataout=2 data checking passed
#
# index i=11
#
# datain= 1,  dataout=1 data checking passed
#
# index i=12
#
# datain= 11,  dataout=11 data checking passed
#
# index i=13
#
# datain= 4,  dataout=4 data checking passed
#
# index i=14
#
# datain= 61,  dataout=61 data checking passed
#
# index i=15
#
# datain= 30,  dataout=30 data checking passed
#
# index i=16
#
# datain= 30,  dataout=30 data checking passed
#
# index i=17
#
# datain= 43,  dataout=43 data checking passed
#
# index i=18
#
# datain= 23,  dataout=23 data checking passed
#
# index i=19
#
# datain= 64,  dataout=64 data checking passed
#
# index i=20
#
# datain= 36,  dataout=36 data checking passed
#
# index i=21
#
# datain= 41,  dataout=41 data checking passed
#
```

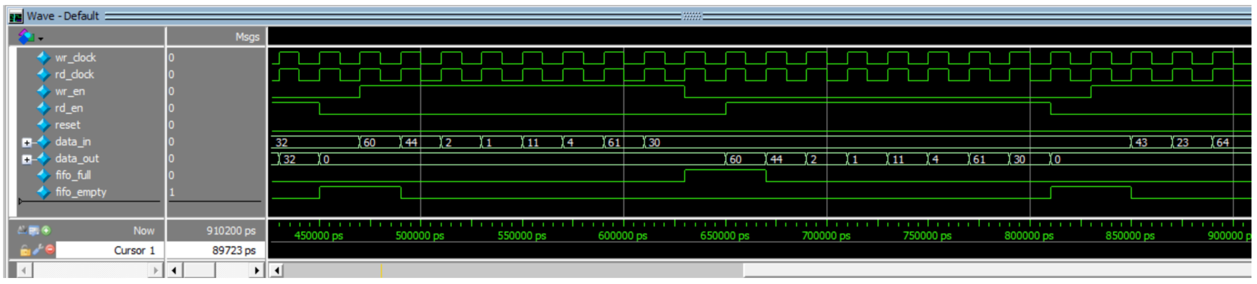


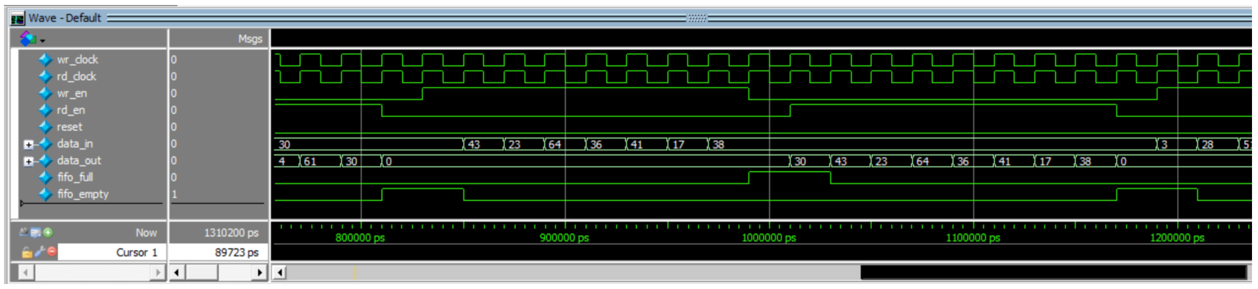
```
#
# index i=22
#
# datain= 17,  dataout=17 data checking passed
#
# index i=23
#
# datain= 38,  dataout=38 data checking passed
#
# index i=24
#
# datain= 3,  dataout=3 data checking passed
#
# index i=25
#
# datain= 28,  dataout=28 data checking passed
#
# index i=26
#
# datain= 51,  dataout=51 data checking passed
#
# index i=27
#
# datain= 50,  dataout=50 data checking passed
#
# index i=28
#
# datain= 14,  dataout=14 data checking passed
#
# index i=29
#
# datain= 37,  dataout=37 data checking passed
#
# index i=30
#
# datain= 44,  dataout=44 data checking passed
#
# index i=31
#
# datain= 43,  dataout=43 data checking passed
#
# index i=32
#
# datain= 18,  dataout=18 data checking passed
#
# FIFO test completed!
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

simulation wavelength results explanation:







In these waveforms, wr_clock and rd_clock are used to drive the write and read operations. At the beginning of the test, the reset signal is asserted to initialize the FIFO, setting both the write (wr_ptr) and read (rd_ptr) pointers to their starting positions and activating the fifo_empty flag, indicating that the FIFO has no data.

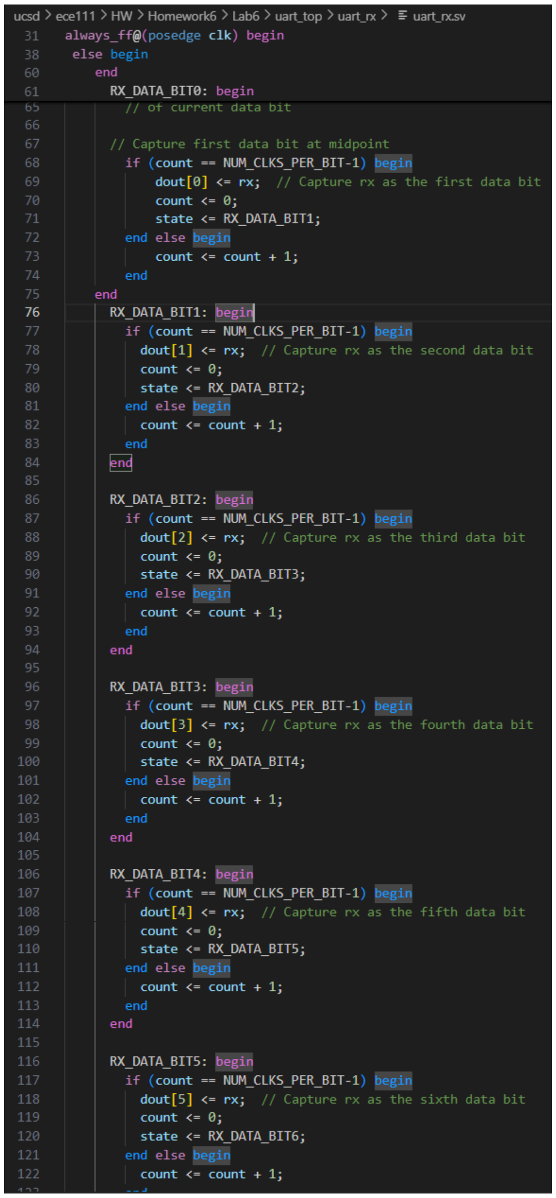
Once reset is de-asserted, wr_en is asserted on wr_clock edges to enable data to be written into the FIFO. Each rising edge of wr_clock with wr_en asserted results in a new value on data_in being stored in the FIFO memory. As data is written, the fifo_empty flag deactivates, showing that data is present in the FIFO. The fifo_full flag remains low until the FIFO reaches its maximum capacity, at which point fifo_full activates to prevent further writes.

In the read phase, rd_en is asserted on rd_clock edges, allowing data to be read from the FIFO in the order it was written (first-in, first-out). Each read operation updates data_out with the next value stored in the FIFO, demonstrating proper sequential data retrieval. As data is read out, the fifo_empty flag eventually reactivates, indicating that all data has been retrieved. Throughout this process, the FIFO handles data storage and retrieval, with fifo_full and fifo_empty flags functioning to manage read and write operations. The above snippets of the testbench confirms that the FIFO design works as intended, ensuring proper data ordering and flow control.

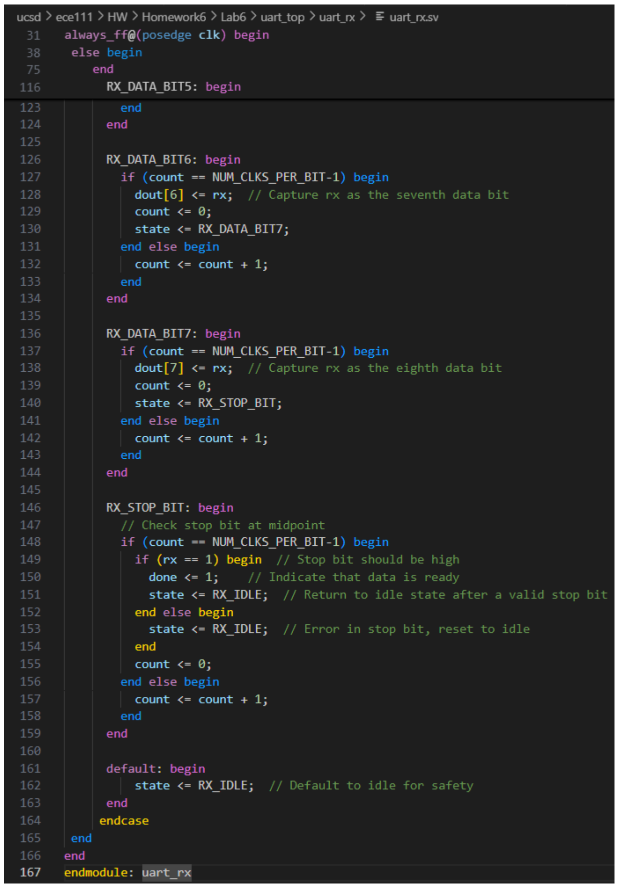
Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

PART C

uart_rx.sv Design Code: only included modified code as it would've been too long.



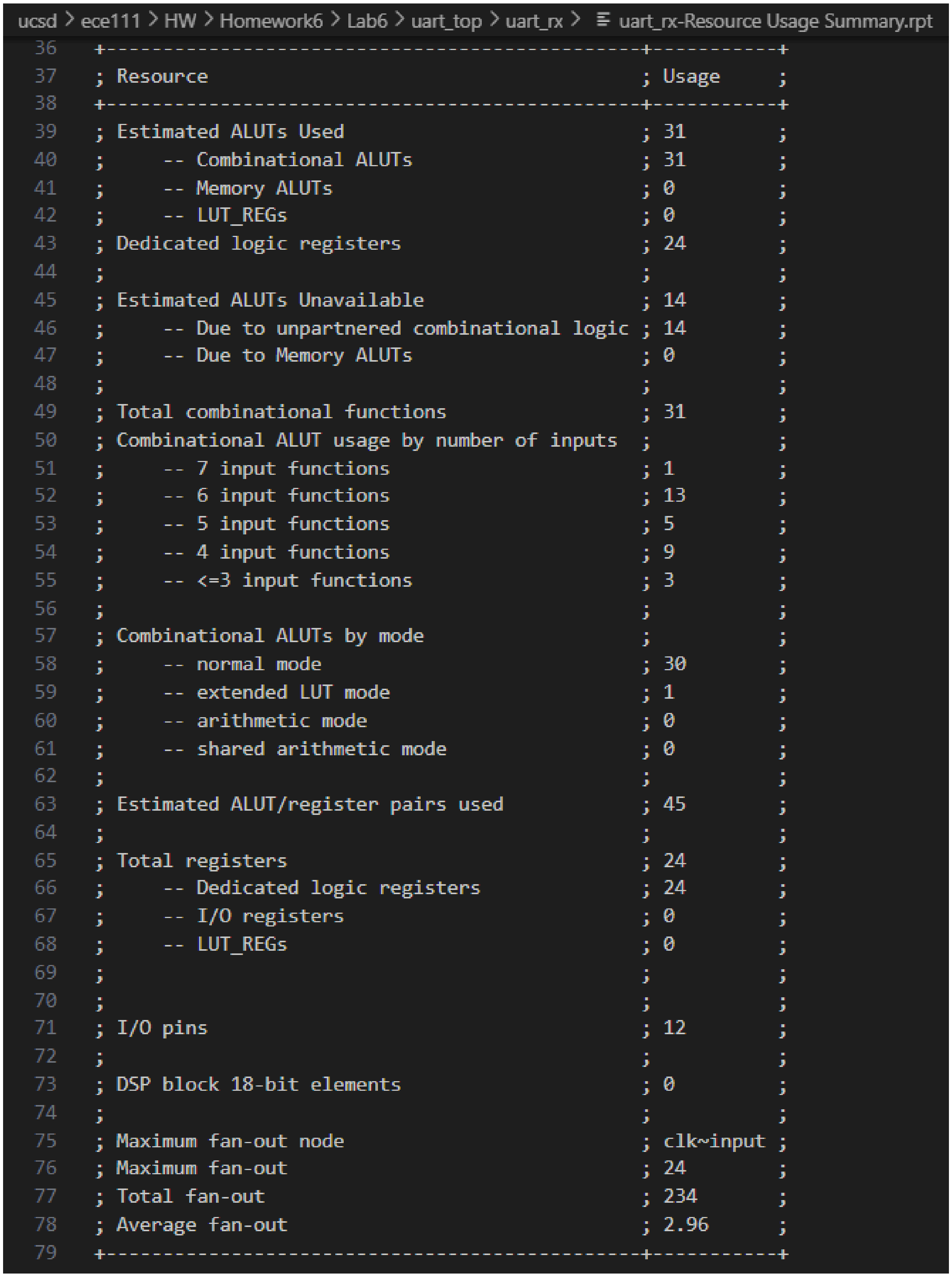
```
ucsd > ece111 > HW > Homework6 > Lab6 > uart_top > uart_rx > uart_rx.sv
 always_ff@(posedge clk) begin
  else begin
     end
       RX_DATA_BIT0: begin
         // of current data bit

       // Capture first data bit at midpoint
         if (count == NUM_CLKS_PER_BIT-1) begin
             dout[0] <= rx;  // Capture rx as the first data bit
             count <= 0;
             state <= RX_DATA_BIT1;
         end else begin
             count <= count + 1;
         end
     end
       RX_DATA_BIT1: begin
         if (count == NUM_CLKS_PER_BIT-1) begin
           dout[1] <= rx;  // Capture rx as the second data bit
           count <= 0;
           state <= RX_DATA_BIT2;
         end else begin
           count <= count + 1;
         end
       end

       RX_DATA_BIT2: begin
         if (count == NUM_CLKS_PER_BIT-1) begin
           dout[2] <= rx;  // Capture rx as the third data bit
           count <= 0;
           state <= RX_DATA_BIT3;
         end else begin
           count <= count + 1;
         end
       end

       RX_DATA_BIT3: begin
         if (count == NUM_CLKS_PER_BIT-1) begin
           dout[3] <= rx;  // Capture rx as the fourth data bit
           count <= 0;
           state <= RX_DATA_BIT4;
         end else begin
           count <= count + 1;
         end
       end

       RX_DATA_BIT4: begin
         if (count == NUM_CLKS_PER_BIT-1) begin
           dout[4] <= rx;  // Capture rx as the fifth data bit
           count <= 0;
           state <= RX_DATA_BIT5;
         end else begin
           count <= count + 1;
         end
       end

       RX_DATA_BIT5: begin
         if (count == NUM_CLKS_PER_BIT-1) begin
           dout[5] <= rx;  // Capture rx as the sixth data bit
           count <= 0;
           state <= RX_DATA_BIT6;
         end else begin
           count <= count + 1;
```



```
ucsd > ece111 > HW > Homework6 > Lab6 > uart_top > uart_rx > uart_rx.sv
 always_ff@(posedge clk) begin
  else begin
     end
       RX_DATA_BIT5: begin
         end
       end

       RX_DATA_BIT6: begin
         if (count == NUM_CLKS_PER_BIT-1) begin
           dout[6] <= rx;  // Capture rx as the seventh data bit
           count <= 0;
           state <= RX_DATA_BIT7;
         end else begin
           count <= count + 1;
         end
       end

       RX_DATA_BIT7: begin
         if (count == NUM_CLKS_PER_BIT-1) begin
           dout[7] <= rx;  // Capture rx as the eighth data bit
           count <= 0;
           state <= RX_STOP_BIT;
         end else begin
           count <= count + 1;
         end
       end

       RX_STOP_BIT: begin
         // Check stop bit at midpoint
         if (count == NUM_CLKS_PER_BIT-1) begin
           if (rx == 1) begin  // Stop bit should be high
             done <= 1;    // Indicate that data is ready
             state <= RX_IDLE;  // Return to idle state after a valid stop bit
           end else begin
             state <= RX_IDLE;  // Error in stop bit, reset to idle
           end
           count <= 0;
         end else begin
           count <= count + 1;
         end
       end

       default: begin
           state <= RX_IDLE;  // Default to idle for safety
       end
      endcase
  end
 end
 endmodule: uart_rx
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

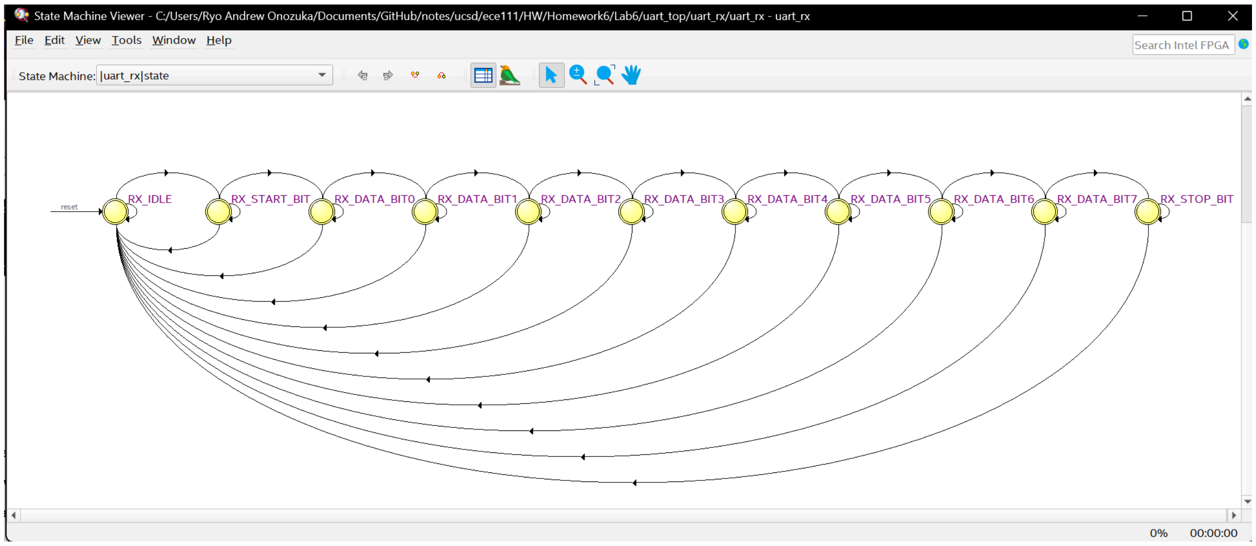
uart_rx.sv Resource Usage Summary

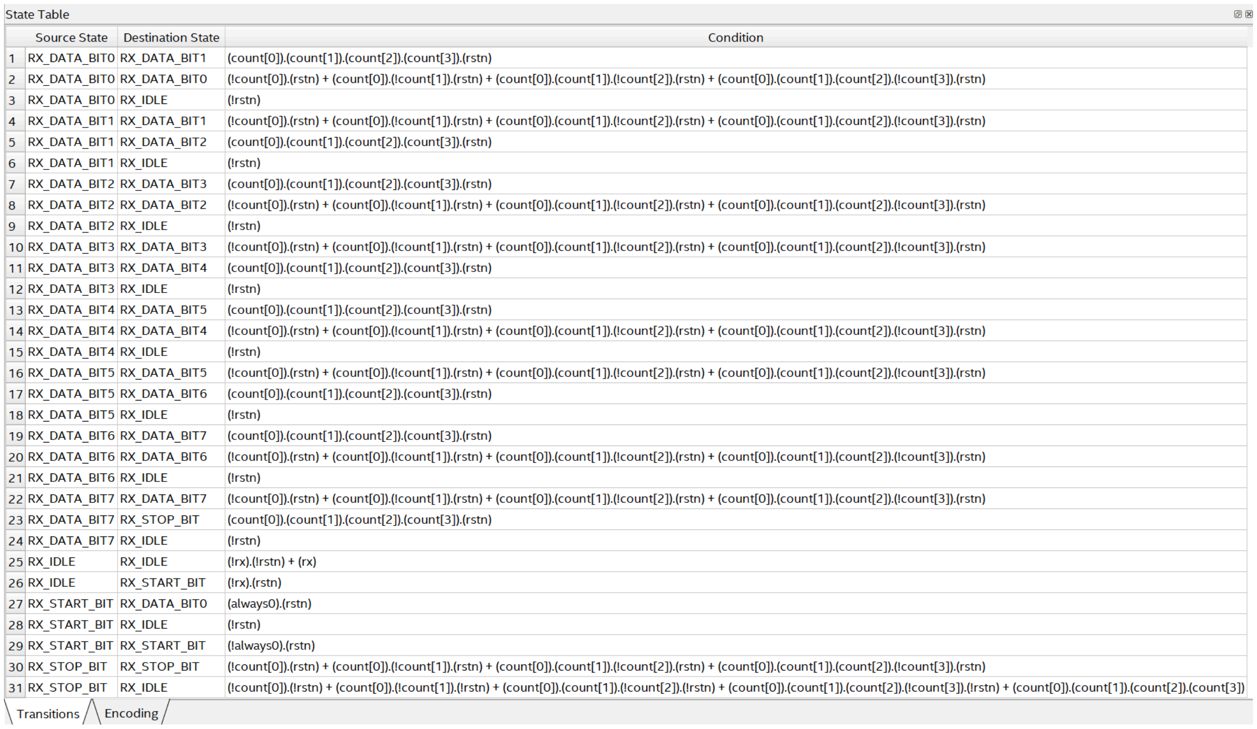

ucsd > ece111 > HW > Homework6 > Lab6 > uart_top > uart_rx > uart_rx-Resource Usage Summary.rpt

```
+---------------------------------------------+-----------+
; Resource                                    ; Usage     ;
+---------------------------------------------+-----------+
; Estimated ALUTs Used                        ; 31        ;
;     -- Combinational ALUTs                  ; 31        ;
;     -- Memory ALUTs                         ; 0         ;
;     -- LUT_REGs                             ; 0         ;
; Dedicated logic registers                   ; 24        ;
;                                             ;           ;
; Estimated ALUTs Unavailable                 ; 14        ;
;     -- Due to unpartnered combinational logic ; 14        ;
;     -- Due to Memory ALUTs                  ; 0         ;
;                                             ;           ;
; Total combinational functions               ; 31        ;
; Combinational ALUT usage by number of inputs ;           ;
;     -- 7 input functions                    ; 1         ;
;     -- 6 input functions                    ; 13        ;
;     -- 5 input functions                    ; 5         ;
;     -- 4 input functions                    ; 9         ;
;     -- <=3 input functions                  ; 3         ;
;                                             ;           ;
; Combinational ALUTs by mode                 ;           ;
;     -- normal mode                          ; 30        ;
;     -- extended LUT mode                    ; 1         ;
;     -- arithmetic mode                      ; 0         ;
;     -- shared arithmetic mode               ; 0         ;
;                                             ;           ;
; Estimated ALUT/register pairs used          ; 45        ;
;                                             ;           ;
; Total registers                             ; 24        ;
;     -- Dedicated logic registers            ; 24        ;
;     -- I/O registers                        ; 0         ;
;     -- LUT_REGs                             ; 0         ;
;                                             ;           ;
;                                             ;           ;
; I/O pins                                    ; 12        ;
;                                             ;           ;
; DSP block 18-bit elements                   ; 0         ;
;                                             ;           ;
; Maximum fan-out node                        ; clk~input ;
; Maximum fan-out                             ; 24        ;
; Total fan-out                               ; 234       ;
; Average fan-out                             ; 2.96      ;
+---------------------------------------------+-----------+
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

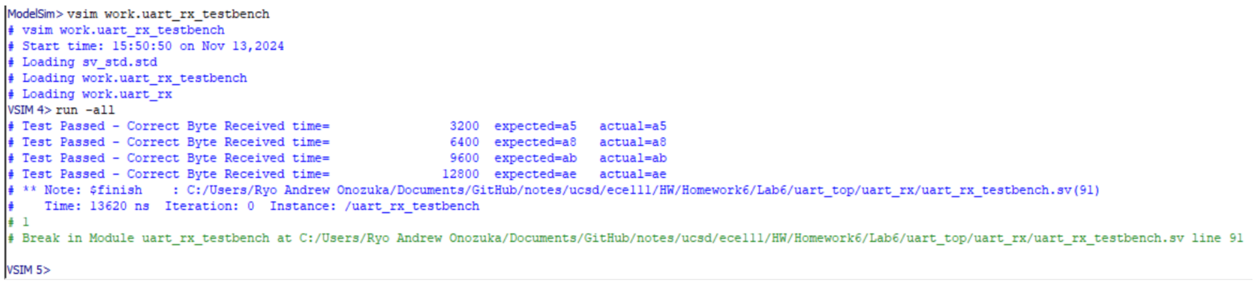
uart_rx.sv RTL viewer: not required
uart_rx.sv post mapping viewer: not required
uart_rx.sv State Machine Viewer + State Transition Table: (table too large for 1 photo)




State Table

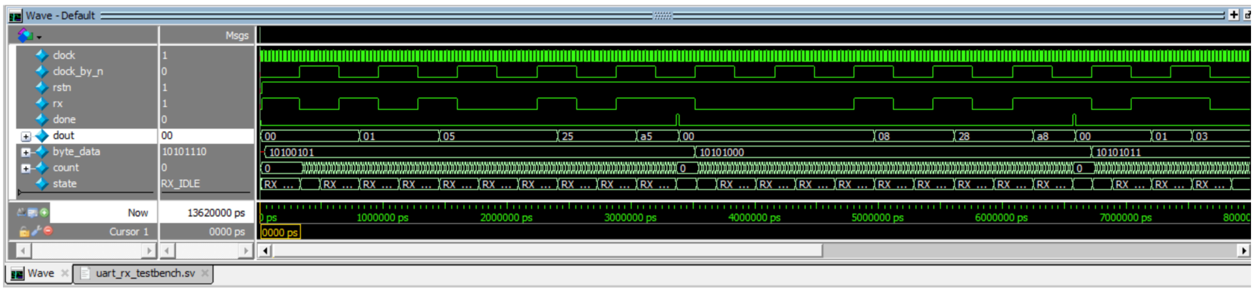
	Source State	Destination State	Condition
1	RX_DATA_BIT0	RX_DATA_BIT1	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
2	RX_DATA_BIT0	RX_DATA_BIT0	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
3	RX_DATA_BIT0	RX_IDLE	(!rstn)
4	RX_DATA_BIT1	RX_DATA_BIT1	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
5	RX_DATA_BIT1	RX_DATA_BIT2	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
6	RX_DATA_BIT1	RX_IDLE	(!rstn)
7	RX_DATA_BIT2	RX_DATA_BIT3	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
8	RX_DATA_BIT2	RX_DATA_BIT2	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
9	RX_DATA_BIT2	RX_IDLE	(!rstn)
10	RX_DATA_BIT3	RX_DATA_BIT3	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
11	RX_DATA_BIT3	RX_DATA_BIT4	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
12	RX_DATA_BIT3	RX_IDLE	(!rstn)
13	RX_DATA_BIT4	RX_DATA_BIT5	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
14	RX_DATA_BIT4	RX_DATA_BIT4	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
15	RX_DATA_BIT4	RX_IDLE	(!rstn)
16	RX_DATA_BIT5	RX_DATA_BIT5	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
17	RX_DATA_BIT5	RX_DATA_BIT6	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
18	RX_DATA_BIT5	RX_IDLE	(!rstn)
19	RX_DATA_BIT6	RX_DATA_BIT7	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
20	RX_DATA_BIT6	RX_DATA_BIT6	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
21	RX_DATA_BIT6	RX_IDLE	(!rstn)
22	RX_DATA_BIT7	RX_DATA_BIT7	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
23	RX_DATA_BIT7	RX_STOP_BIT	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
24	RX_DATA_BIT7	RX_IDLE	(!rstn)
25	RX_IDLE	RX_IDLE	(!rx).(!rstn) + (rx)
26	RX_IDLE	RX_START_BIT	(!rx).(rstn)
27	RX_START_BIT	RX_DATA_BIT0	(always0).(rstn)
28	RX_START_BIT	RX_IDLE	(!rstn)
29	RX_START_BIT	RX_START_BIT	(!always0).(rstn)
30	RX_STOP_BIT	RX_STOP_BIT	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
31	RX_STOP_BIT	RX_IDLE	(!count[0]).(!rstn) + (count[0]).(!count[1]).(!rstn) + (count[0]).(count[1]).(!count[2]).(!rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(!rstn) + (count[0]).(count[1]).(count[2]).(count[3])

full simulation transcript:

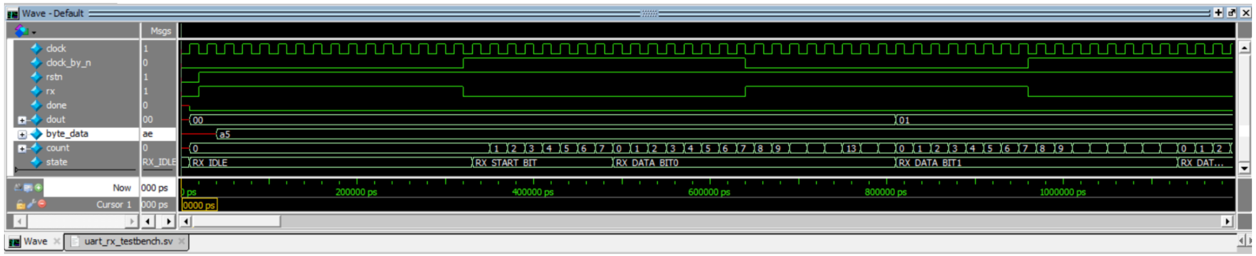

```
ModelSim> vsim work.uart_rx_testbench
# vsim work.uart_rx_testbench
# Start time: 15:50:50 on Nov 13,2024
# Loading sv_std.std
# Loading work.uart_rx_testbench
# Loading work.uart_rx
VSIM 4> run -all
# Test Passed - Correct Byte Received time=                 3200  expected=a5   actual=a5
# Test Passed - Correct Byte Received time=                 6400  expected=a8   actual=a8
# Test Passed - Correct Byte Received time=                 9600  expected=ab   actual=ab
# Test Passed - Correct Byte Received time=                12800  expected=ae   actual=ae
# ** Note: $finish    : C:/Users/Ryo Andrew Onozuka/Documents/GitHub/notes/ucsd/ece111/HW/Homework6/Lab6/uart_top/uart_rx/uart_rx_testbench.sv(91)
#    Time: 13620 ns  Iteration: 0  Instance: /uart_rx_testbench
# 1
# Break in Module uart_rx_testbench at C:/Users/Ryo Andrew Onozuka/Documents/GitHub/notes/ucsd/ece111/HW/Homework6/Lab6/uart_top/uart_rx/uart_rx_testbench.sv line 91

VSIM 5>
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

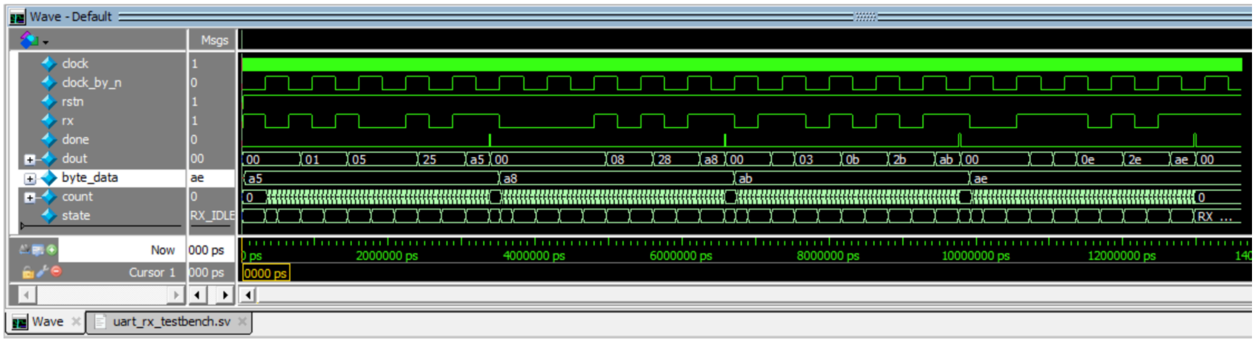
simulation snapshots + explanations:



zoomed to match waveform view 1 in the writeup. this view shows the changes in the start bit, displaying going through all of the states. it shows when the first byte is received by uart_rx and done is set to 1 once all of the start bit, 8 data bits, and stop bit are serially transmitted. it also shows the beginning of the second byte received. in this screenshot the byte_data is still in the wrong radix (not hexadecimal), but we see it stay constant as it is provided by the testbench to see if we are accurately receiving the information. see previous page of simulation transcripts confirming that we do indeed get the correct byte received.



zoomed to match waveform view 2 in the writeup. view 2 highlights the 16 clocks per rx serial bit for NUM_CLKS_PER_BIT=16. We also see that we sample rx=0 start bit at the midpoint when the count is 7, and when count is 15 we sample the first data bit at the midpoint.

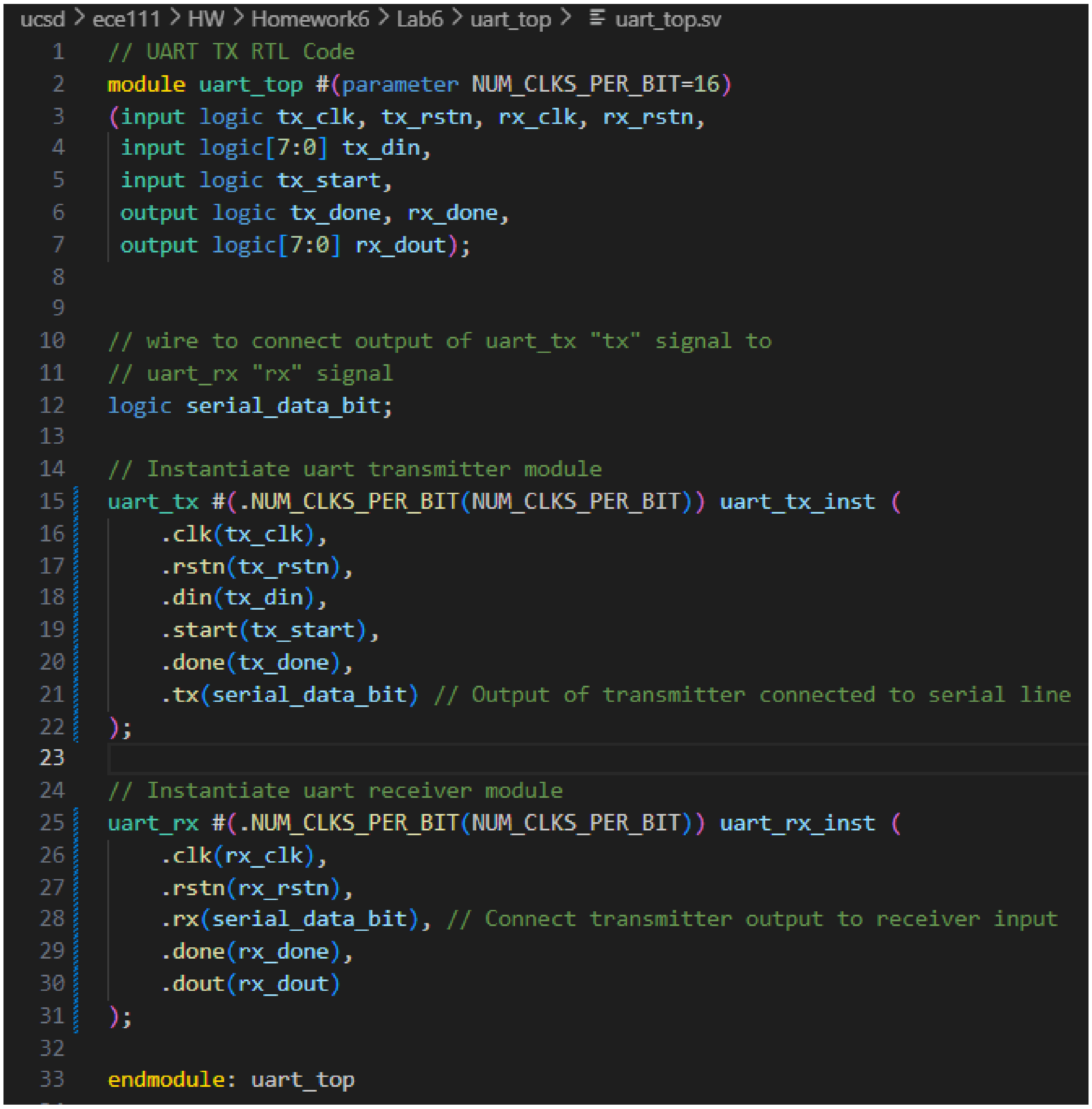


zoomed to match waveform view 3 in the writeup. view 3 highlights the four parallel 8-bit douts and confirming that they are correct and match with the byte_data provided by the testbench.

These 3 views of the waveforms as well as the correct output given by the transcript collectively demonstrate that the design of uart_rx operates as expected, with correct state transitions, bit sampling, and data reconstruction. The done signal accurately indicates when a byte is ready, and dout matches the expected values from byte_data, verifying the reliability of the UART receiver design.

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

uart_top.sv Design Code:

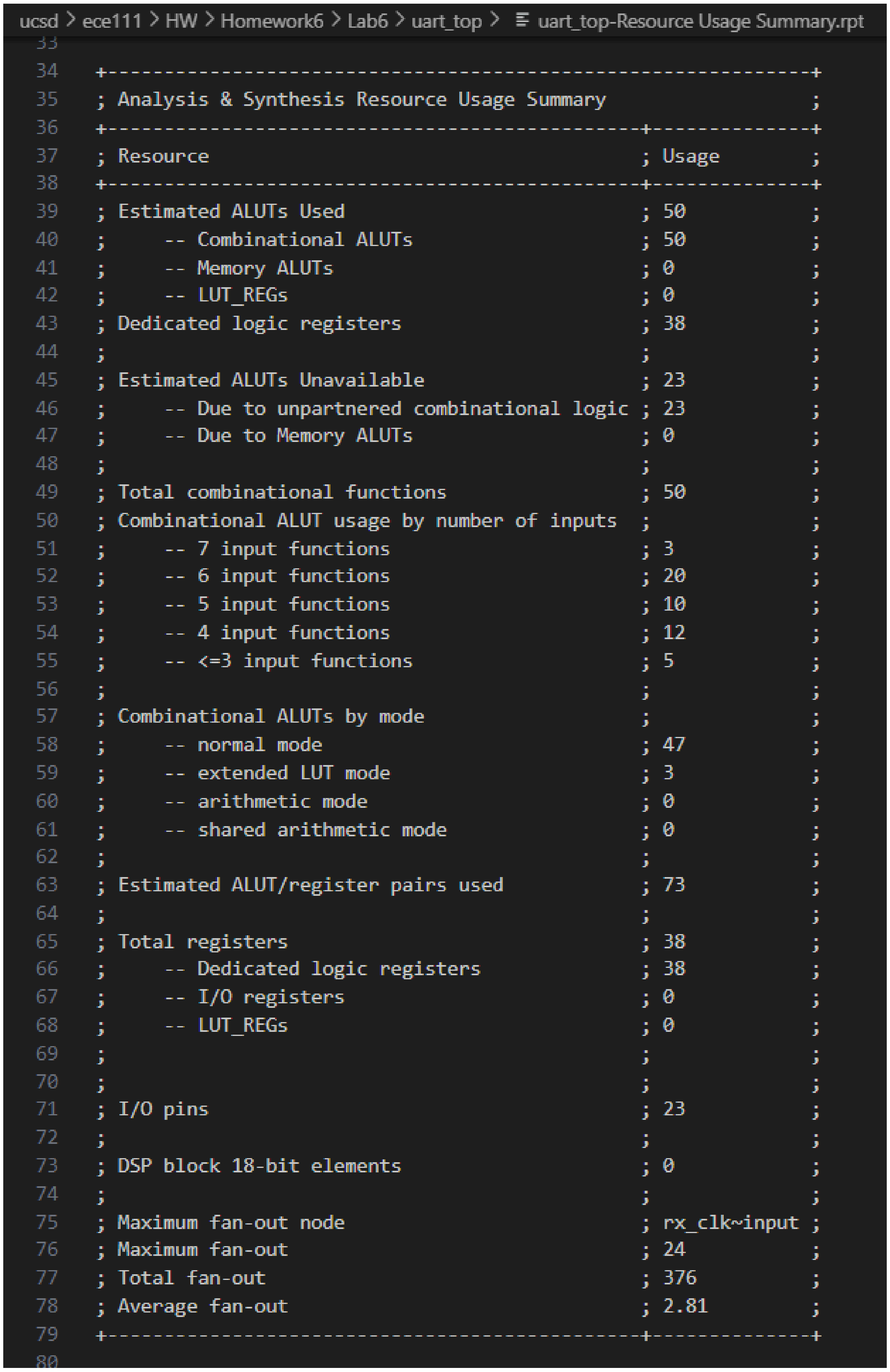


ucsd > ece111 > HW > Homework6 > Lab6 > uart_top > uart_top.sv

```
// UART TX RTL Code
module uart_top #(parameter NUM_CLKS_PER_BIT=16)
(input logic tx_clk, tx_rstn, rx_clk, rx_rstn,
 input logic[7:0] tx_din,
 input logic tx_start,
 output logic tx_done, rx_done,
 output logic[7:0] rx_dout);


// wire to connect output of uart_tx "tx" signal to
// uart_rx "rx" signal
logic serial_data_bit;

// Instantiate uart transmitter module
uart_tx #(.NUM_CLKS_PER_BIT(NUM_CLKS_PER_BIT)) uart_tx_inst (
    .clk(tx_clk),
    .rstn(tx_rstn),
    .din(tx_din),
    .start(tx_start),
    .done(tx_done),
    .tx(serial_data_bit) // Output of transmitter connected to serial line
);

// Instantiate uart receiver module
uart_rx #(.NUM_CLKS_PER_BIT(NUM_CLKS_PER_BIT)) uart_rx_inst (
    .clk(rx_clk),
    .rstn(rx_rstn),
    .rx(serial_data_bit), // Connect transmitter output to receiver input
    .done(rx_done),
    .dout(rx_dout)
);

endmodule: uart_top
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

uart_top.sv Resource Usage Summary


ucsd > ece111 > HW > Homework6 > Lab6 > uart_top > uart_top-Resource Usage Summary.rpt

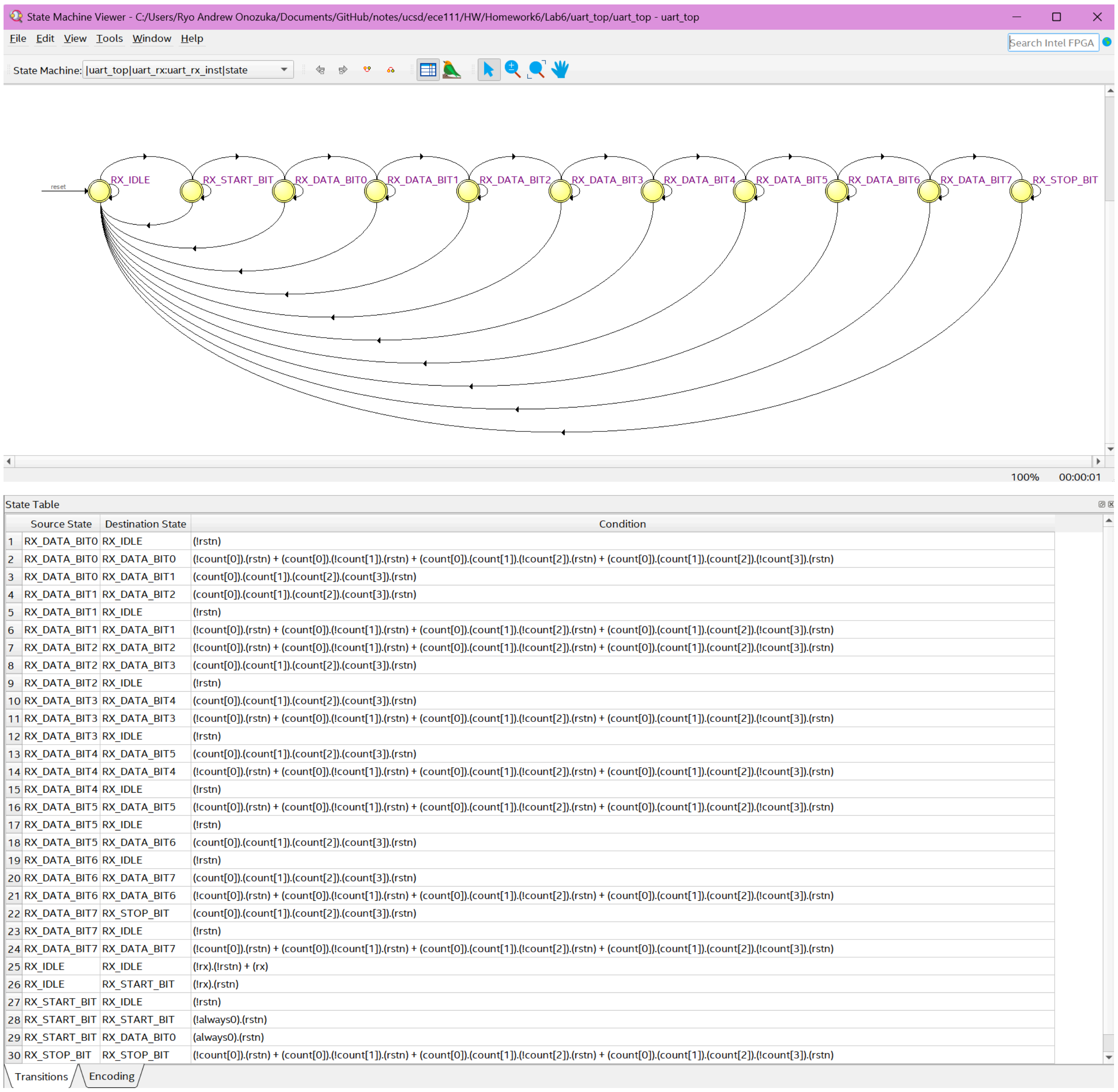
```
+--------------------------------------------------------------------+
; Analysis & Synthesis Resource Usage Summary                        ;
+------------------------------------------------+-------------------+
; Resource                                       ; Usage             ;
+------------------------------------------------+-------------------+
; Estimated ALUTs Used                           ; 50                ;
;     -- Combinational ALUTs                     ; 50                ;
;     -- Memory ALUTs                            ; 0                 ;
;     -- LUT_REGs                                ; 0                 ;
; Dedicated logic registers                      ; 38                ;
;                                                ;                   ;
; Estimated ALUTs Unavailable                    ; 23                ;
;     -- Due to unpartnered combinational logic  ; 23                ;
;     -- Due to Memory ALUTs                     ; 0                 ;
;                                                ;                   ;
; Total combinational functions                  ; 50                ;
; Combinational ALUT usage by number of inputs   ;                   ;
;     -- 7 input functions                       ; 3                 ;
;     -- 6 input functions                       ; 20                ;
;     -- 5 input functions                       ; 10                ;
;     -- 4 input functions                       ; 12                ;
;     -- <=3 input functions                     ; 5                 ;
;                                                ;                   ;
; Combinational ALUTs by mode                    ;                   ;
;     -- normal mode                             ; 47                ;
;     -- extended LUT mode                       ; 3                 ;
;     -- arithmetic mode                         ; 0                 ;
;     -- shared arithmetic mode                  ; 0                 ;
;                                                ;                   ;
; Estimated ALUT/register pairs used             ; 73                ;
;                                                ;                   ;
; Total registers                                ; 38                ;
;     -- Dedicated logic registers               ; 38                ;
;     -- I/O registers                           ; 0                 ;
;     -- LUT_REGs                                ; 0                 ;
;                                                ;                   ;
;                                                ;                   ;
; I/O pins                                       ; 23                ;
;                                                ;                   ;
; DSP block 18-bit elements                      ; 0                 ;
;                                                ;                   ;
; Maximum fan-out node                           ; rx_clk~input      ;
; Maximum fan-out                                ; 24                ;
; Total fan-out                                  ; 376               ;
; Average fan-out                                ; 2.81              ;
+------------------------------------------------+-------------------+
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

uart_top.sv RTL viewer: not required

uart_top.sv post mapping viewer: not required

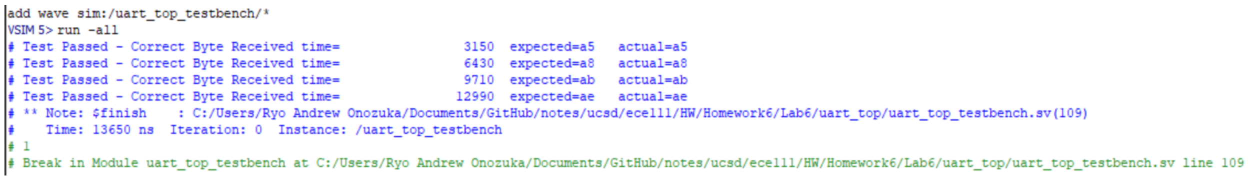
uart_top.sv State Machine Viewer + State Transition Table: (table too large for 1 photo)



State Table

	Source State	Destination State	Condition
1	RX_DATA_BIT0	RX_IDLE	(!rstn)
2	RX_DATA_BIT0	RX_DATA_BIT0	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
3	RX_DATA_BIT0	RX_DATA_BIT1	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
4	RX_DATA_BIT1	RX_DATA_BIT2	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
5	RX_DATA_BIT1	RX_IDLE	(!rstn)
6	RX_DATA_BIT1	RX_DATA_BIT1	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
7	RX_DATA_BIT2	RX_DATA_BIT2	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
8	RX_DATA_BIT2	RX_DATA_BIT3	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
9	RX_DATA_BIT2	RX_IDLE	(!rstn)
10	RX_DATA_BIT3	RX_DATA_BIT4	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
11	RX_DATA_BIT3	RX_DATA_BIT3	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
12	RX_DATA_BIT3	RX_IDLE	(!rstn)
13	RX_DATA_BIT4	RX_DATA_BIT5	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
14	RX_DATA_BIT4	RX_DATA_BIT4	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
15	RX_DATA_BIT4	RX_IDLE	(!rstn)
16	RX_DATA_BIT5	RX_DATA_BIT5	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
17	RX_DATA_BIT5	RX_IDLE	(!rstn)
18	RX_DATA_BIT5	RX_DATA_BIT6	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
19	RX_DATA_BIT6	RX_IDLE	(!rstn)
20	RX_DATA_BIT6	RX_DATA_BIT7	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
21	RX_DATA_BIT6	RX_DATA_BIT6	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
22	RX_DATA_BIT7	RX_STOP_BIT	(count[0]).(count[1]).(count[2]).(count[3]).(rstn)
23	RX_DATA_BIT7	RX_IDLE	(!rstn)
24	RX_DATA_BIT7	RX_DATA_BIT7	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)
25	RX_IDLE	RX_IDLE	(!rx).(!rstn) + (rx)
26	RX_IDLE	RX_START_BIT	(!rx).(rstn)
27	RX_START_BIT	RX_IDLE	(!rstn)
28	RX_START_BIT	RX_START_BIT	(!always0).(rstn)
29	RX_START_BIT	RX_DATA_BIT0	(always0).(rstn)
30	RX_STOP_BIT	RX_STOP_BIT	(!count[0]).(rstn) + (count[0]).(!count[1]).(rstn) + (count[0]).(count[1]).(!count[2]).(rstn) + (count[0]).(count[1]).(count[2]).(!count[3]).(rstn)

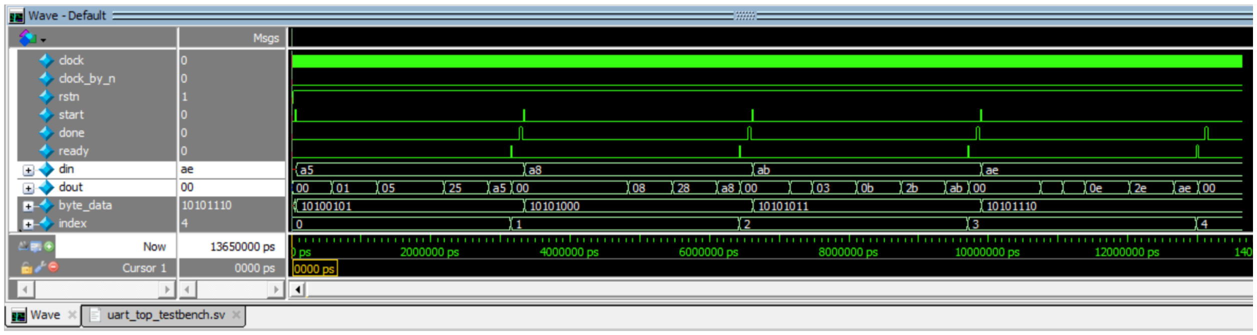
full simulation transcript:



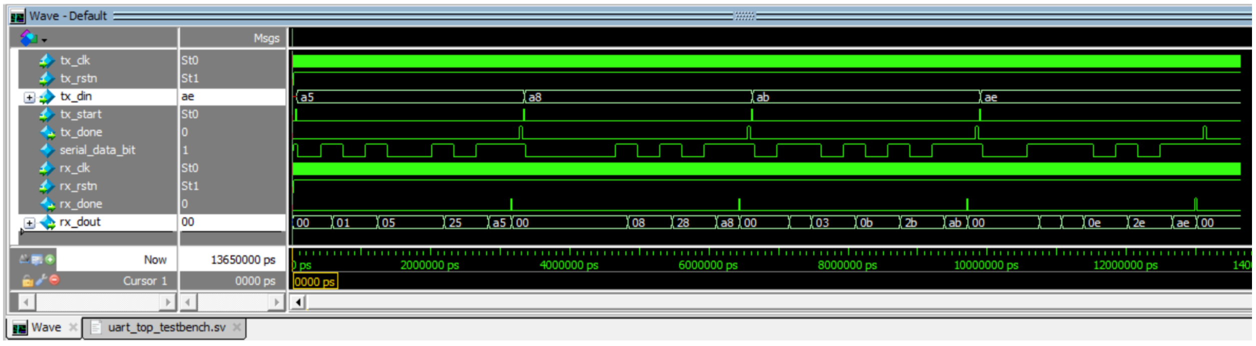
```
add wave sim:/uart_top_testbench/*
VSIM 5> run -all
# Test Passed - Correct Byte Received time=                 3150  expected=a5   actual=a5
# Test Passed - Correct Byte Received time=                 6430  expected=a8   actual=a8
# Test Passed - Correct Byte Received time=                 9710  expected=ab   actual=ab
# Test Passed - Correct Byte Received time=                12990  expected=ae   actual=ae
# ** Note: $finish    : C:/Users/Ryo Andrew Onozuka/Documents/GitHub/notes/ucsd/ece111/HW/Homework6/Lab6/uart_top/uart_top_testbench.sv(109)
#    Time: 13650 ns  Iteration: 0  Instance: /uart_top_testbench
# 1
# Break in Module uart_top_testbench at C:/Users/Ryo Andrew Onozuka/Documents/GitHub/notes/ucsd/ece111/HW/Homework6/Lab6/uart_top/uart_top_testbench.sv line 109
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

simulation snapshot + explanation:



initial snapshot



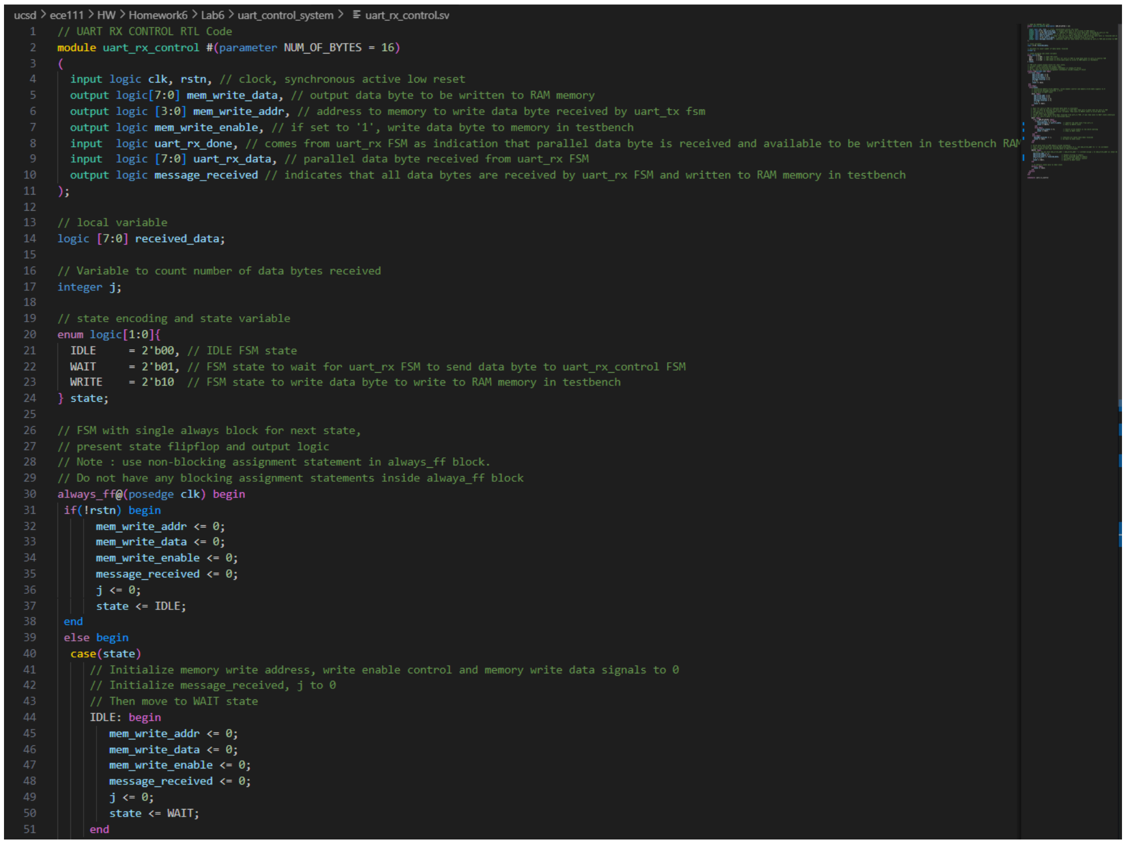
second snapshot taken to match the waveforms particularly focused on in the writeup. this view shows the tx and rx signals, which displays how the serial data is transmitted by uart_tx and received by uart_rx. uart_rx converts the serial data into parallel 8-bit data which can be seen at rx_dout.

The waveform and testbench output demonstrate the correct operation of the uart_top module, which integrates UART transmission and reception. After initialization, the testbench provides 8-bit values to tx_din and asserts tx_start to begin each transmission. The transmitter sends each byte serially through the tx line, which is directly connected to the rx input of the receiver. This serial data includes a start bit, 8 data bits, and a stop bit. When the receiver completes each byte, it asserts rx_done, indicating that the received byte is available on rx_dout.

The testbench verifies that rx_dout matches the expected tx_din values, and each successful reception is confirmed by messages in the output, showing matching expected and actual byte values with accurate timestamps. This process is repeated for four bytes, demonstrating that the uart_top module reliably transmits and receives data with correct timing and data integrity. The waveforms confirm each step, from start to stop bit, ensuring that the design meets the requirements for UART communication.

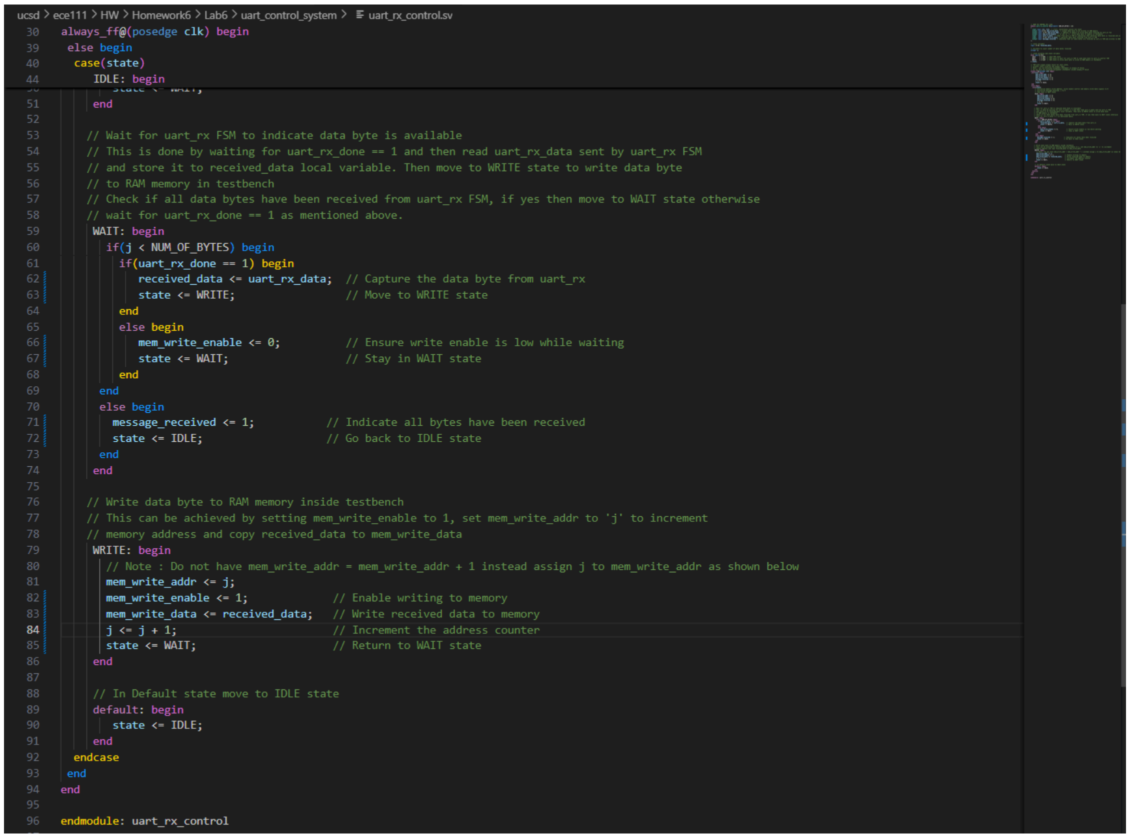
Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

uart_rx_control.sv Design Code:



ucsd > ece111 > HW > Homework6 > Lab6 > uart_control_system > uart_rx_control.sv

```
// UART RX CONTROL RTL Code
module uart_rx_control #(parameter NUM_OF_BYTES = 16)
(
  input logic clk, rstn, // clock, synchronous active low reset
  output logic[7:0] mem_write_data, // output data byte to be written to RAM memory
  output logic [3:0] mem_write_addr, // address to memory to write data byte received by uart_tx fsm
  output logic mem_write_enable, // if set to '1', write data byte to memory in testbench
  input  logic uart_rx_done, // comes from uart_rx FSM as indication that parallel data byte is received and available to be written in testbench RAM
  input  logic [7:0] uart_rx_data, // parallel data byte received from uart_rx FSM
  output logic message_received // indicates that all data bytes are received by uart_rx FSM and written to RAM memory in testbench
);

// local variable
logic [7:0] received_data;

// Variable to count number of data bytes received
integer j;

// state encoding and state variable
enum logic[1:0]{
  IDLE     = 2'b00, // IDLE FSM state
  WAIT     = 2'b01, // FSM state to wait for uart_rx FSM to send data byte to uart_rx_control FSM
  WRITE    = 2'b10  // FSM state to write data byte to write to RAM memory in testbench
} state;

// FSM with single always block for next state,
// present state flipflop and output logic
// Note : use non-blocking assignment statement in always_ff block.
// Do not have any blocking assignment statements inside alwaya_ff block
always_ff@(posedge clk) begin
 if(!rstn) begin
      mem_write_addr <= 0;
      mem_write_data <= 0;
      mem_write_enable <= 0;
      message_received <= 0;
      j <= 0;
      state <= IDLE;
 end
 else begin
  case(state)
    // Initialize memory write address, write enable control and memory write data signals to 0
    // Initialize message_received, j to 0
    // Then move to WAIT state
    IDLE: begin
      mem_write_addr <= 0;
      mem_write_data <= 0;
      mem_write_enable <= 0;
      message_received <= 0;
      j <= 0;
      state <= WAIT;
    end
```

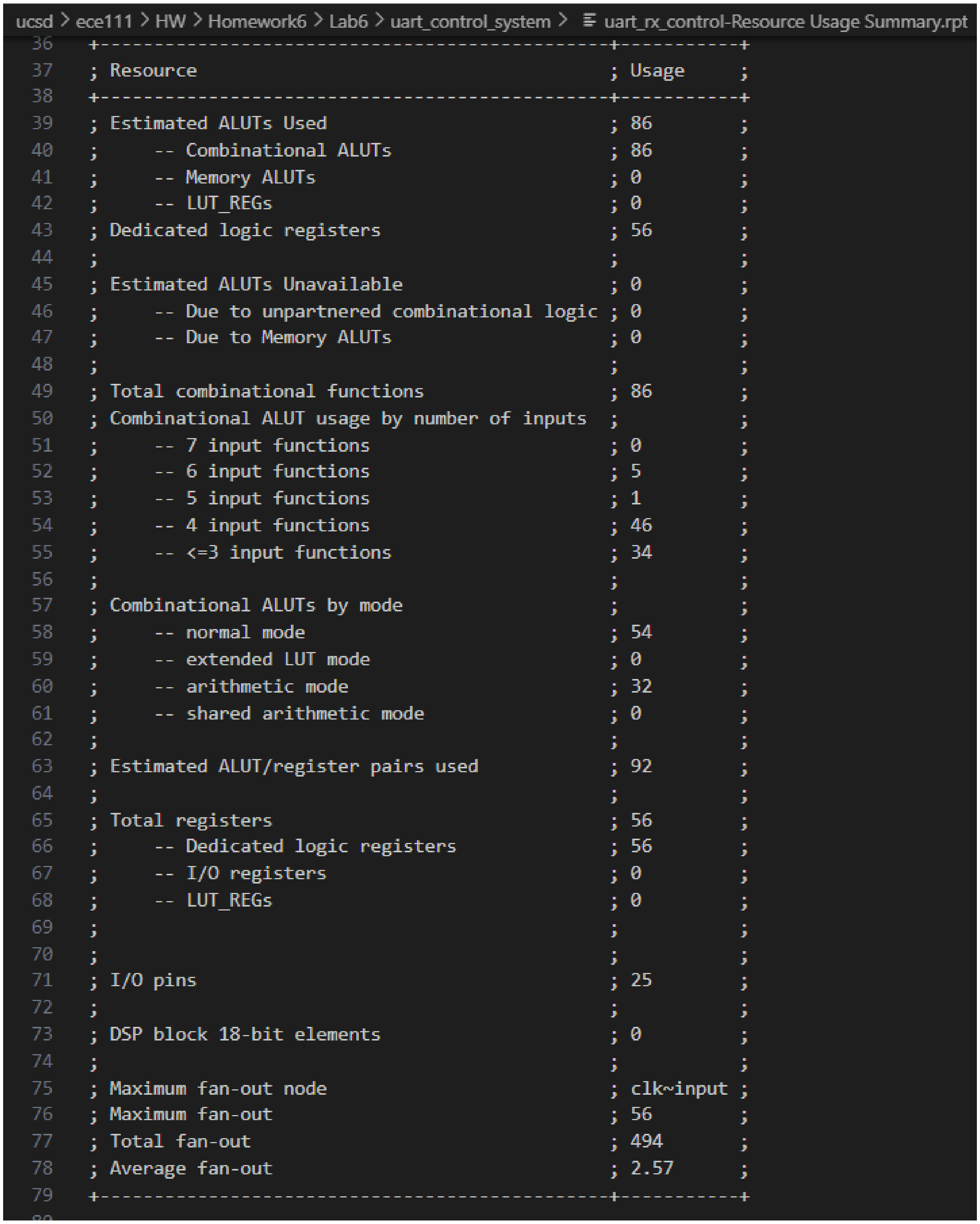


ucsd > ece111 > HW > Homework6 > Lab6 > uart_control_system > uart_rx_control.sv

```
always_ff@(posedge clk) begin
 else begin
  case(state)
    IDLE: begin
    end

    // Wait for uart_rx FSM to indicate data byte is available
    // This is done by waiting for uart_rx_done == 1 and then read uart_rx_data sent by uart_rx FSM
    // and store it to received_data local variable. Then move to WRITE state to write data byte
    // to RAM memory in testbench
    // Check if all data bytes have been received from uart_rx FSM, if yes then move to WAIT state otherwise
    // wait for uart_rx_done == 1 as mentioned above.
    WAIT: begin
      if(j < NUM_OF_BYTES) begin
        if(uart_rx_done == 1) begin
          received_data <= uart_rx_data;  // Capture the data byte from uart_rx
          state <= WRITE;                 // Move to WRITE state
        end
        else begin
          mem_write_enable <= 0;          // Ensure write enable is low while waiting
          state <= WAIT;                  // Stay in WAIT state
        end
      end
      else begin
        message_received <= 1;          // Indicate all bytes have been received
        state <= IDLE;                  // Go back to IDLE state
      end
    end

    // Write data byte to RAM memory inside testbench
    // This can be achieved by setting mem_write_enable to 1, set mem_write_addr to 'j' to increment
    // memory address and copy received_data to mem_write_data
    WRITE: begin
      // Note : Do not have mem_write_addr = mem_write_addr + 1 instead assign j to mem_write_addr as shown below
      mem_write_addr <= j;
      mem_write_enable <= 1;              // Enable writing to memory
      mem_write_data <= received_data;    // Write received data to memory
      j <= j + 1;                         // Increment the address counter
      state <= WAIT;                      // Return to WAIT state
    end

    // In Default state move to IDLE state
    default: begin
      state <= IDLE;
    end
  endcase
 end
end

endmodule: uart_rx_control
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

uart_rx_control.sv Resource Usage Summary

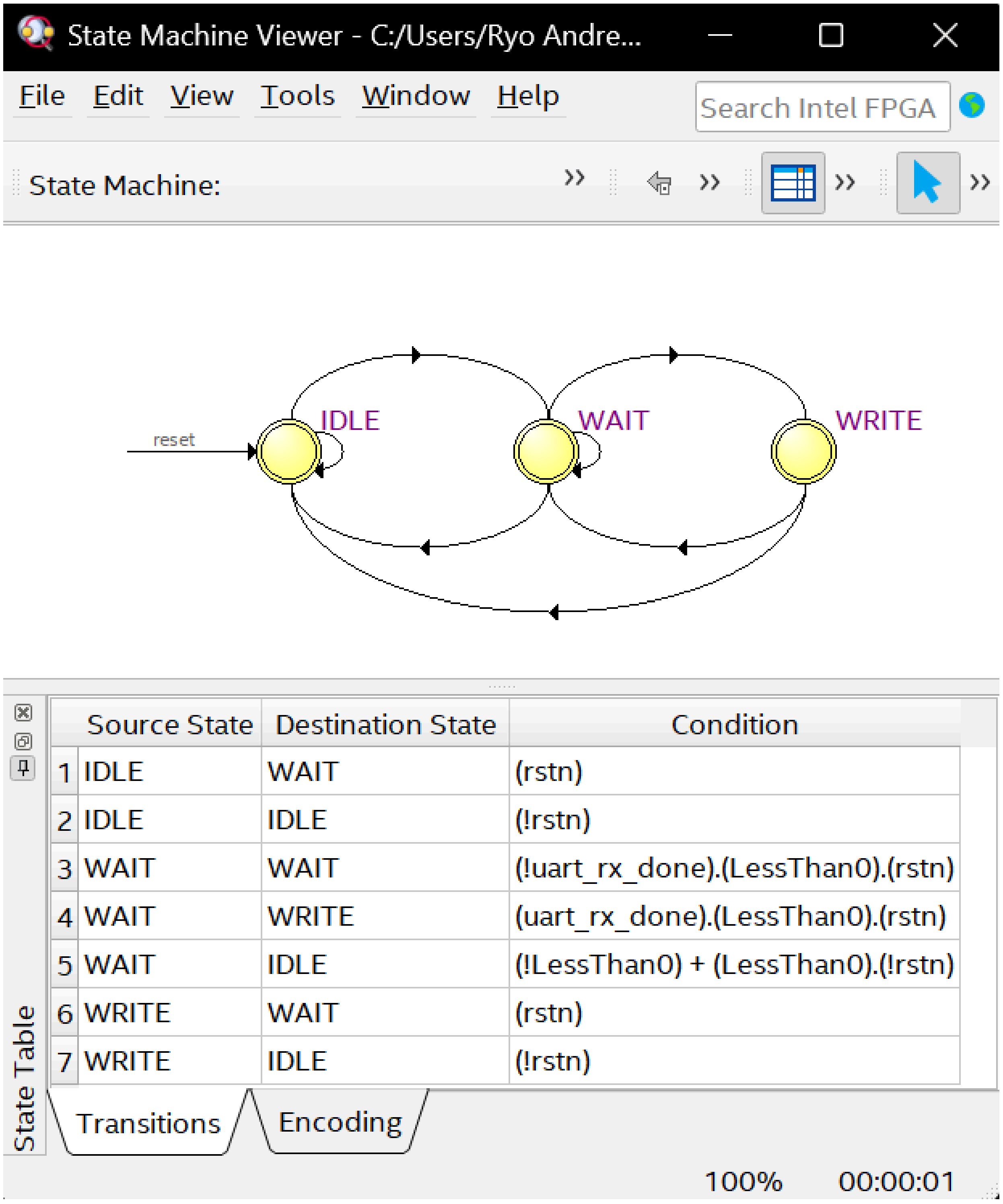


ucsd > ece111 > HW > Homework6 > Lab6 > uart_control_system > uart_rx_control-Resource Usage Summary.rpt

```
+---------------------------------------------+-----------+
; Resource                                    ; Usage     ;
+---------------------------------------------+-----------+
; Estimated ALUTs Used                        ; 86        ;
;     -- Combinational ALUTs                  ; 86        ;
;     -- Memory ALUTs                         ; 0         ;
;     -- LUT_REGs                             ; 0         ;
; Dedicated logic registers                   ; 56        ;
;                                             ;           ;
; Estimated ALUTs Unavailable                 ; 0         ;
;     -- Due to unpartnered combinational logic ; 0       ;
;     -- Due to Memory ALUTs                  ; 0         ;
;                                             ;           ;
; Total combinational functions               ; 86        ;
; Combinational ALUT usage by number of inputs ;          ;
;     -- 7 input functions                    ; 0         ;
;     -- 6 input functions                    ; 5         ;
;     -- 5 input functions                    ; 1         ;
;     -- 4 input functions                    ; 46        ;
;     -- <=3 input functions                  ; 34        ;
;                                             ;           ;
; Combinational ALUTs by mode                 ;           ;
;     -- normal mode                          ; 54        ;
;     -- extended LUT mode                    ; 0         ;
;     -- arithmetic mode                      ; 32        ;
;     -- shared arithmetic mode               ; 0         ;
;                                             ;           ;
; Estimated ALUT/register pairs used          ; 92        ;
;                                             ;           ;
; Total registers                             ; 56        ;
;     -- Dedicated logic registers            ; 56        ;
;     -- I/O registers                        ; 0         ;
;     -- LUT_REGs                             ; 0         ;
;                                             ;           ;
;                                             ;           ;
; I/O pins                                    ; 25        ;
;                                             ;           ;
; DSP block 18-bit elements                   ; 0         ;
;                                             ;           ;
; Maximum fan-out node                        ; clk~input ;
; Maximum fan-out                             ; 56        ;
; Total fan-out                               ; 494       ;
; Average fan-out                             ; 2.57      ;
+---------------------------------------------+-----------+
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

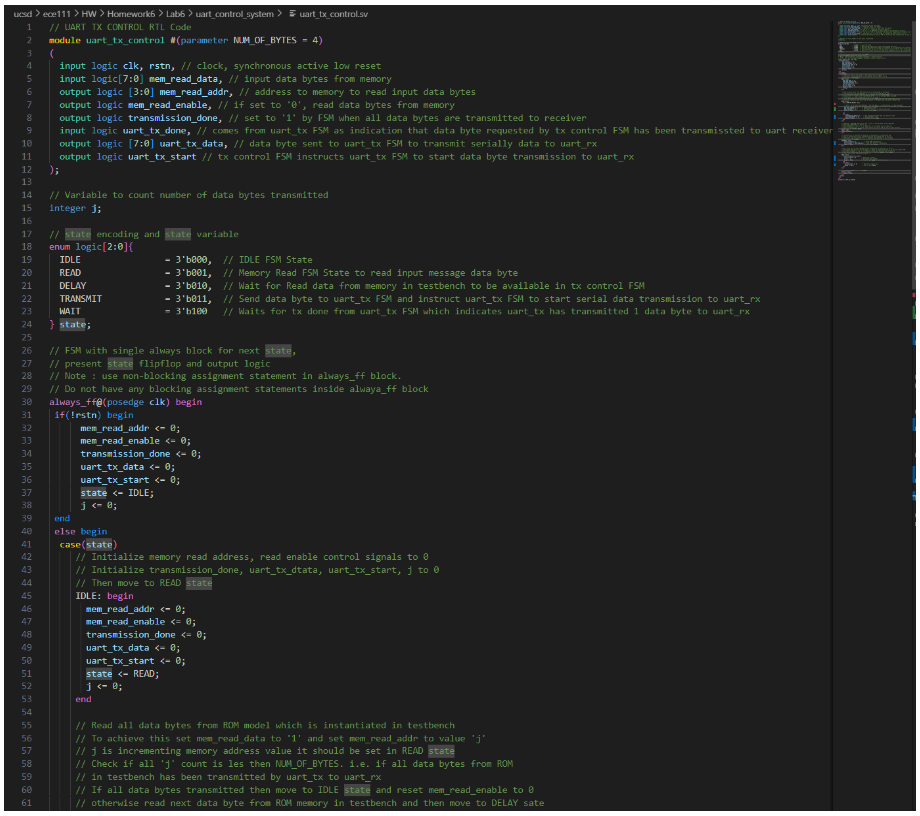
uart_rx_control.sv RTL viewer: not required
uart_rx_control.sv post mapping viewer: not required
uart_rx_control.sv State Machine Viewer + State Transition Table:



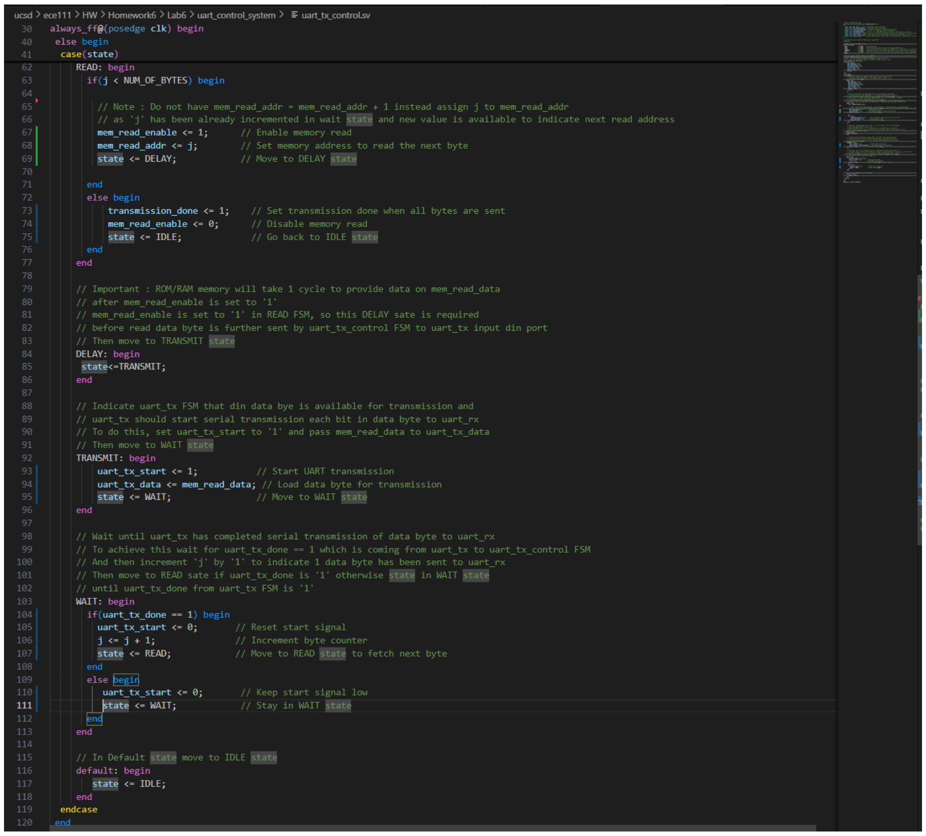
	Source State	Destination State	Condition
1	IDLE	WAIT	(rstn)
2	IDLE	IDLE	(!rstn)
3	WAIT	WAIT	(!uart_rx_done).(LessThan0).(rstn)
4	WAIT	WRITE	(uart_rx_done).(LessThan0).(rstn)
5	WAIT	IDLE	(!LessThan0) + (LessThan0).(!rstn)
6	WRITE	WAIT	(rstn)
7	WRITE	IDLE	(!rstn)

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

uart_tx_control.sv Design Code:


ucsd > ece111 > HW > Homework6 > Lab6 > uart_control_system > uart_tx_control.sv

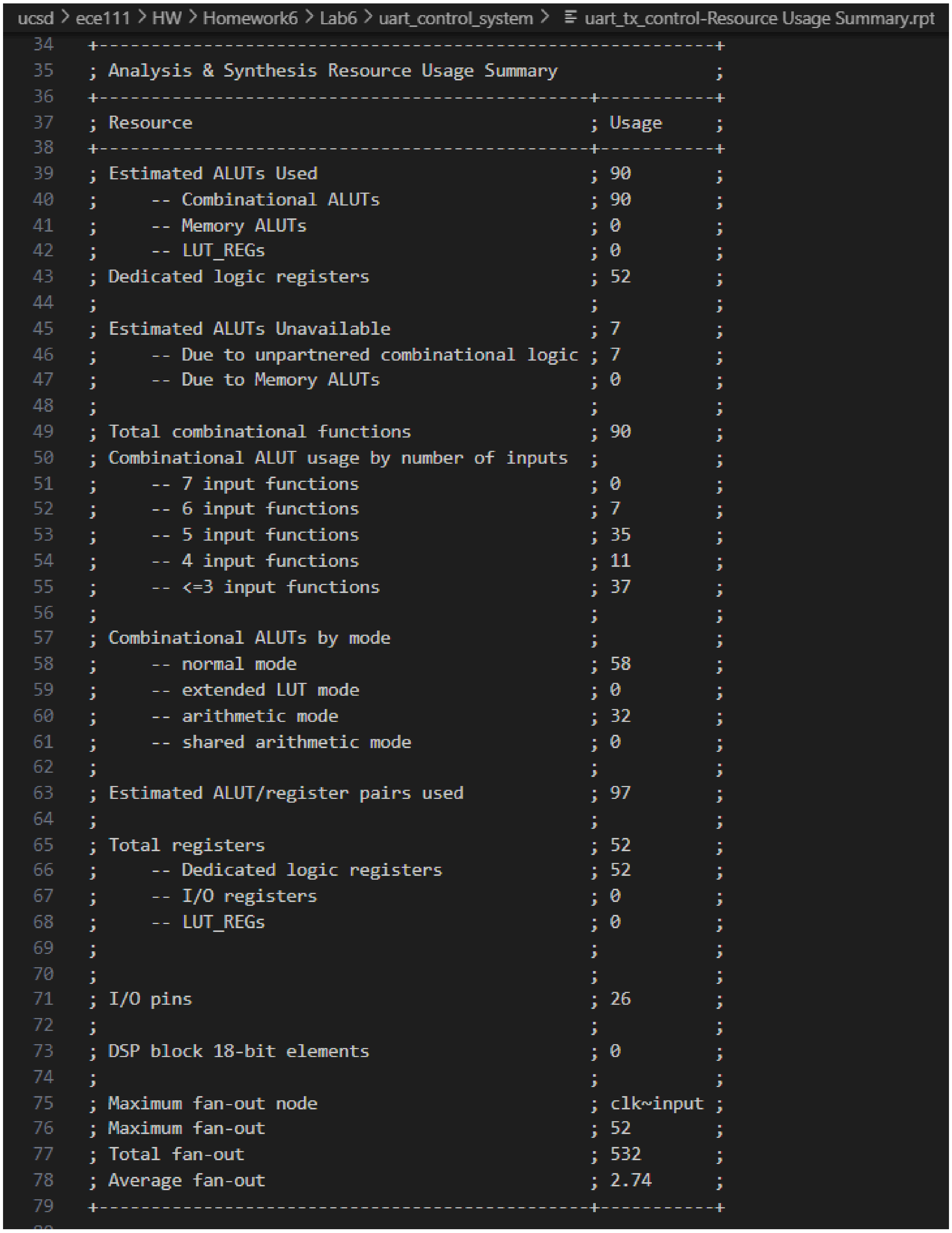
```
// UART TX CONTROL RTL Code
module uart_tx_control #(parameter NUM_OF_BYTES = 4)
(
  input logic clk, rstn, // clock, synchronous active low reset
  input logic[7:0] mem_read_data, // input data bytes from memory
  output logic [3:0] mem_read_addr, // address to memory to read input data bytes
  output logic mem_read_enable, // if set to '0', read data bytes from memory
  output logic transmission_done, // set to '1' by FSM when all data bytes are transmitted to receiver
  input logic uart_tx_done, // comes from uart_tx FSM as indication that data byte requested by tx control FSM has been transmissted to uart receiver
  output logic [7:0] uart_tx_data, // data byte sent to uart_tx FSM to transmit serially data to uart_rx
  output logic uart_tx_start // tx control FSM instructs uart_tx FSM to start data byte transmission to uart_rx
);

// Variable to count number of data bytes transmitted
integer j;

// state encoding and state variable
enum logic[2:0]{
  IDLE                 = 3'b000,  // IDLE FSM State
  READ                 = 3'b001,  // Memory Read FSM State to read input message data byte
  DELAY                = 3'b010,  // Wait for Read data from memory in testbench to be available in tx control FSM
  TRANSMIT             = 3'b011,  // Send data byte to uart_tx FSM and instruct uart_tx FSM to start serial data transmission to uart_rx
  WAIT                 = 3'b100   // Waits for tx done from uart_tx FSM which indicates uart_tx has transmitted 1 data byte to uart_rx
} state;

// FSM with single always block for next state,
// present state flipflop and output logic
// Note : use non-blocking assignment statement in always_ff block.
// Do not have any blocking assignment statements inside alwaya_ff block
always_ff@(posedge clk) begin
 if(!rstn) begin
      mem_read_addr <= 0;
      mem_read_enable <= 0;
      transmission_done <= 0;
      uart_tx_data <= 0;
      uart_tx_start <= 0;
      state <= IDLE;
      j <= 0;
 end
 else begin
  case(state)
     // Initialize memory read address, read enable control signals to 0
     // Initialize transmission_done, uart_tx_dtata, uart_tx_start, j to 0
     // Then move to READ state
     IDLE: begin
       mem_read_addr <= 0;
       mem_read_enable <= 0;
       transmission_done <= 0;
       uart_tx_data <= 0;
       uart_tx_start <= 0;
       state <= READ;
       j <= 0;
     end

     // Read all data bytes from ROM model which is instantiated in testbench
     // To achieve this set mem_read_data to '1' and set mem_read_addr to value 'j'
     // j is incrementing memory address value it should be set in READ state
     // Check if all 'j' count is les then NUM_OF_BYTES. i.e. if all data bytes from ROM
     // in testbench has been transmitted by uart_tx to uart_rx
     // If all data bytes transmitted then move to IDLE state and reset mem_read_enable to 0
     // otherwise read next data byte from ROM memory in testbench and then move to DELAY sate
```


ucsd > ece111 > HW > Homework6 > Lab6 > uart_control_system > uart_tx_control.sv

```
always_ff@(posedge clk) begin
 else begin
  case(state)
     READ: begin
       if(j < NUM_OF_BYTES) begin

         // Note : Do not have mem_read_addr = mem_read_addr + 1 instead assign j to mem_read_addr
         // as 'j' has been already incremented in wait state and new value is available to indicate next read address
         mem_read_enable <= 1;      // Enable memory read
         mem_read_addr <= j;        // Set memory address to read the next byte
         state <= DELAY;            // Move to DELAY state

       end
       else begin
           transmission_done <= 1;    // Set transmission done when all bytes are sent
           mem_read_enable <= 0;      // Disable memory read
           state <= IDLE;             // Go back to IDLE state
       end
     end

     // Important : ROM/RAM memory will take 1 cycle to provide data on mem_read_data
     // after mem_read_enable is set to '1'
     // mem_read_enable is set to '1' in READ FSM, so this DELAY sate is required
     // before read data byte is further sent by uart_tx_control FSM to uart_tx input din port
     // Then move to TRANSMIT state
     DELAY: begin
      state<=TRANSMIT;
     end

     // Indicate uart_tx FSM that din data bye is available for transmission and
     // uart_tx should start serial transmission each bit in data byte to uart_rx
     // To do this, set uart_tx_start to '1' and pass mem_read_data to uart_tx_data
     // Then move to WAIT state
     TRANSMIT: begin
         uart_tx_start <= 1;           // Start UART transmission
         uart_tx_data <= mem_read_data; // Load data byte for transmission
         state <= WAIT;                // Move to WAIT state
     end

     // Wait until uart_tx has completed serial transmission of data byte to uart_rx
     // To achieve this wait for uart_tx_done == 1 which is coming from uart_tx to uart_tx_control FSM
     // And then increment 'j' by '1' to indicate 1 data byte has been sent to uart_rx
     // Then move to READ sate if uart_tx_done is '1' otherwise state in WAIT state
     // until uart_tx_done from uart_tx FSM is '1'
     WAIT: begin
       if(uart_tx_done == 1) begin
         uart_tx_start <= 0;       // Reset start signal
         j <= j + 1;               // Increment byte counter
         state <= READ;            // Move to READ state to fetch next byte
       end
       else begin
          uart_tx_start <= 0;       // Keep start signal low
          state <= WAIT;            // Stay in WAIT state
       end
     end

     // In Default state move to IDLE state
     default: begin
        state <= IDLE;
     end
  endcase
 end
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

uart_tx_control.sv Resource Usage Summary

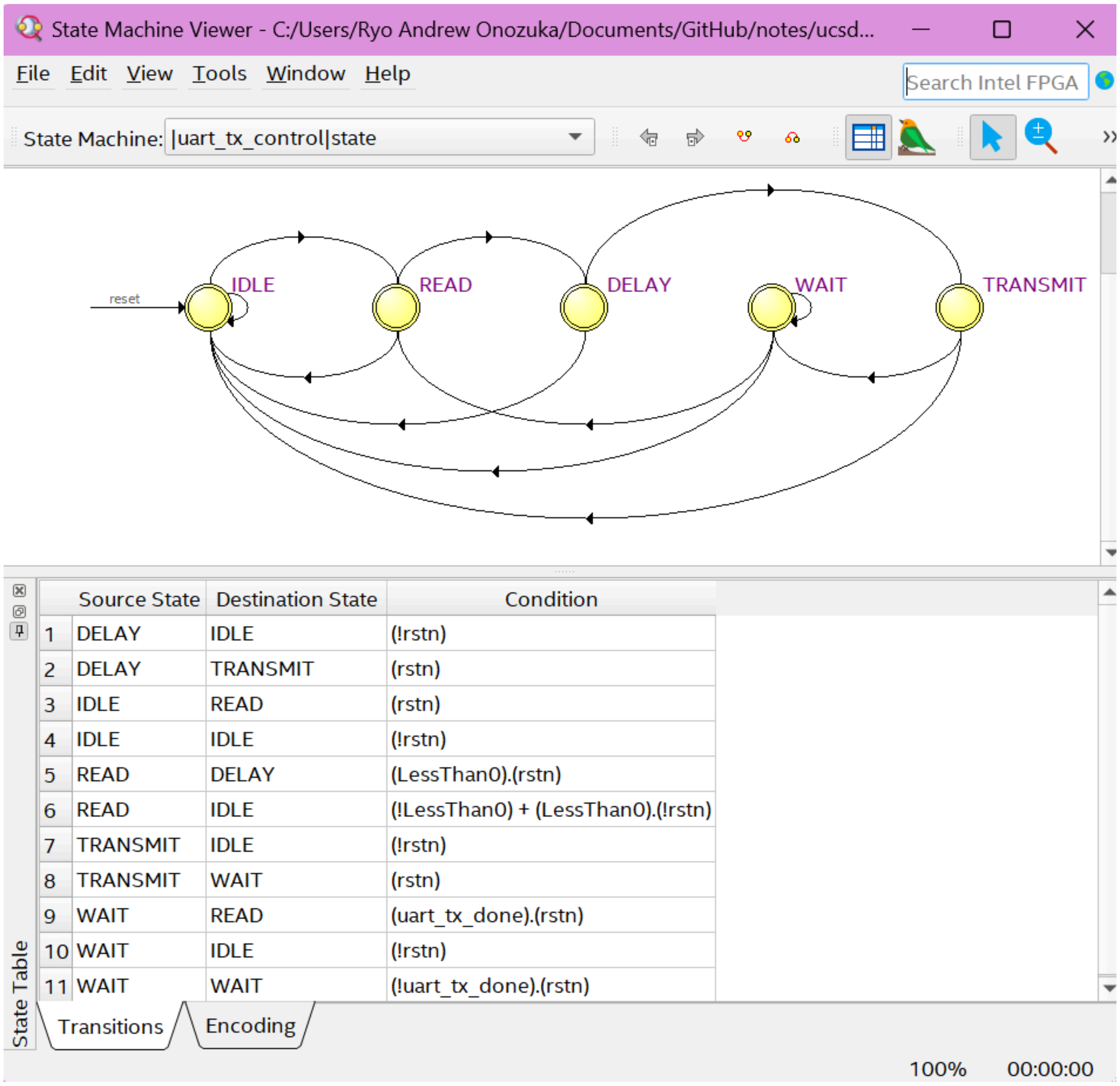


ucsd > ece111 > HW > Homework6 > Lab6 > uart_control_system > uart_tx_control-Resource Usage Summary.rpt

```
+--------------------------------------------------------------+
; Analysis & Synthesis Resource Usage Summary                    ;
+------------------------------------------------+-------------+
; Resource                                       ; Usage       ;
+------------------------------------------------+-------------+
; Estimated ALUTs Used                           ; 90          ;
;     -- Combinational ALUTs                     ; 90          ;
;     -- Memory ALUTs                            ; 0           ;
;     -- LUT_REGs                                ; 0           ;
; Dedicated logic registers                      ; 52          ;
;                                                ;             ;
; Estimated ALUTs Unavailable                    ; 7           ;
;     -- Due to unpartnered combinational logic  ; 7           ;
;     -- Due to Memory ALUTs                     ; 0           ;
;                                                ;             ;
; Total combinational functions                  ; 90          ;
; Combinational ALUT usage by number of inputs   ;             ;
;     -- 7 input functions                       ; 0           ;
;     -- 6 input functions                       ; 7           ;
;     -- 5 input functions                       ; 35          ;
;     -- 4 input functions                       ; 11          ;
;     -- <=3 input functions                     ; 37          ;
;                                                ;             ;
; Combinational ALUTs by mode                    ;             ;
;     -- normal mode                             ; 58          ;
;     -- extended LUT mode                       ; 0           ;
;     -- arithmetic mode                         ; 32          ;
;     -- shared arithmetic mode                  ; 0           ;
;                                                ;             ;
; Estimated ALUT/register pairs used             ; 97          ;
;                                                ;             ;
; Total registers                                ; 52          ;
;     -- Dedicated logic registers               ; 52          ;
;     -- I/O registers                           ; 0           ;
;     -- LUT_REGs                                ; 0           ;
;                                                ;             ;
;                                                ;             ;
; I/O pins                                       ; 26          ;
;                                                ;             ;
; DSP block 18-bit elements                      ; 0           ;
;                                                ;             ;
; Maximum fan-out node                           ; clk~input   ;
; Maximum fan-out                                ; 52          ;
; Total fan-out                                  ; 532         ;
; Average fan-out                                ; 2.74        ;
+------------------------------------------------+-------------+
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

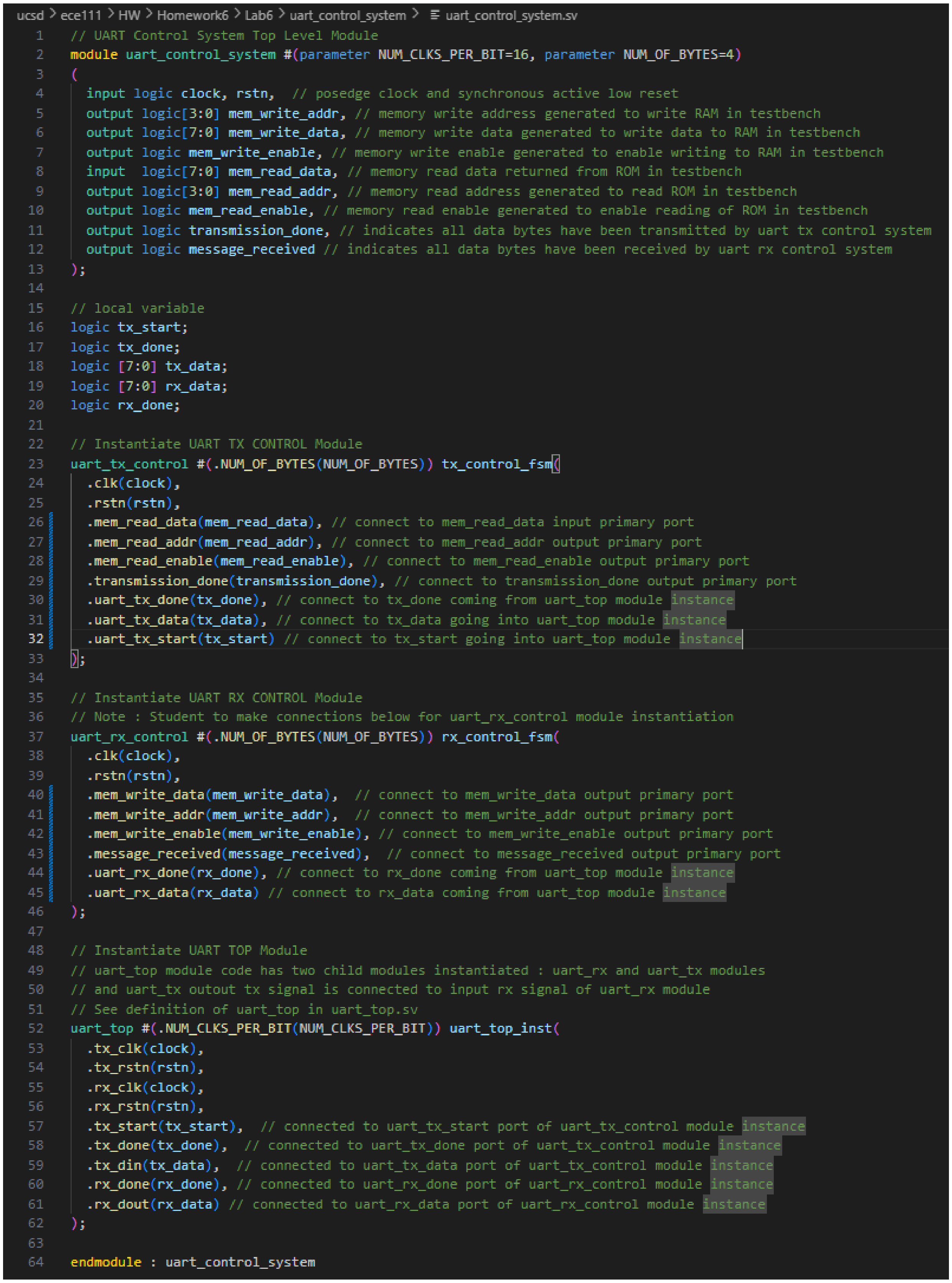
uart_tx_control.sv RTL viewer: not required
uart_tx_control.sv post mapping viewer: not required
uart_tx_control.sv State Machine Viewer + State Transition Table:


State Machine Viewer - C:/Users/Ryo Andrew Onozuka/Documents/GitHub/notes/ucsd...
File
Edit
View
Tools
Window
Help
Search Intel FPGA
State Machine: |uart_tx_control|state
reset
IDLE
READ
DELAY
WAIT
TRANSMIT
State Table
Transitions
Encoding
100%
00:00:00

	Source State	Destination State	Condition
1	DELAY	IDLE	(!rstn)
2	DELAY	TRANSMIT	(rstn)
3	IDLE	READ	(rstn)
4	IDLE	IDLE	(!rstn)
5	READ	DELAY	(LessThan0).(rstn)
6	READ	IDLE	(!LessThan0) + (LessThan0).(!rstn)
7	TRANSMIT	IDLE	(!rstn)
8	TRANSMIT	WAIT	(rstn)
9	WAIT	READ	(uart_tx_done).(rstn)
10	WAIT	IDLE	(!rstn)
11	WAIT	WAIT	(!uart_tx_done).(rstn)

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

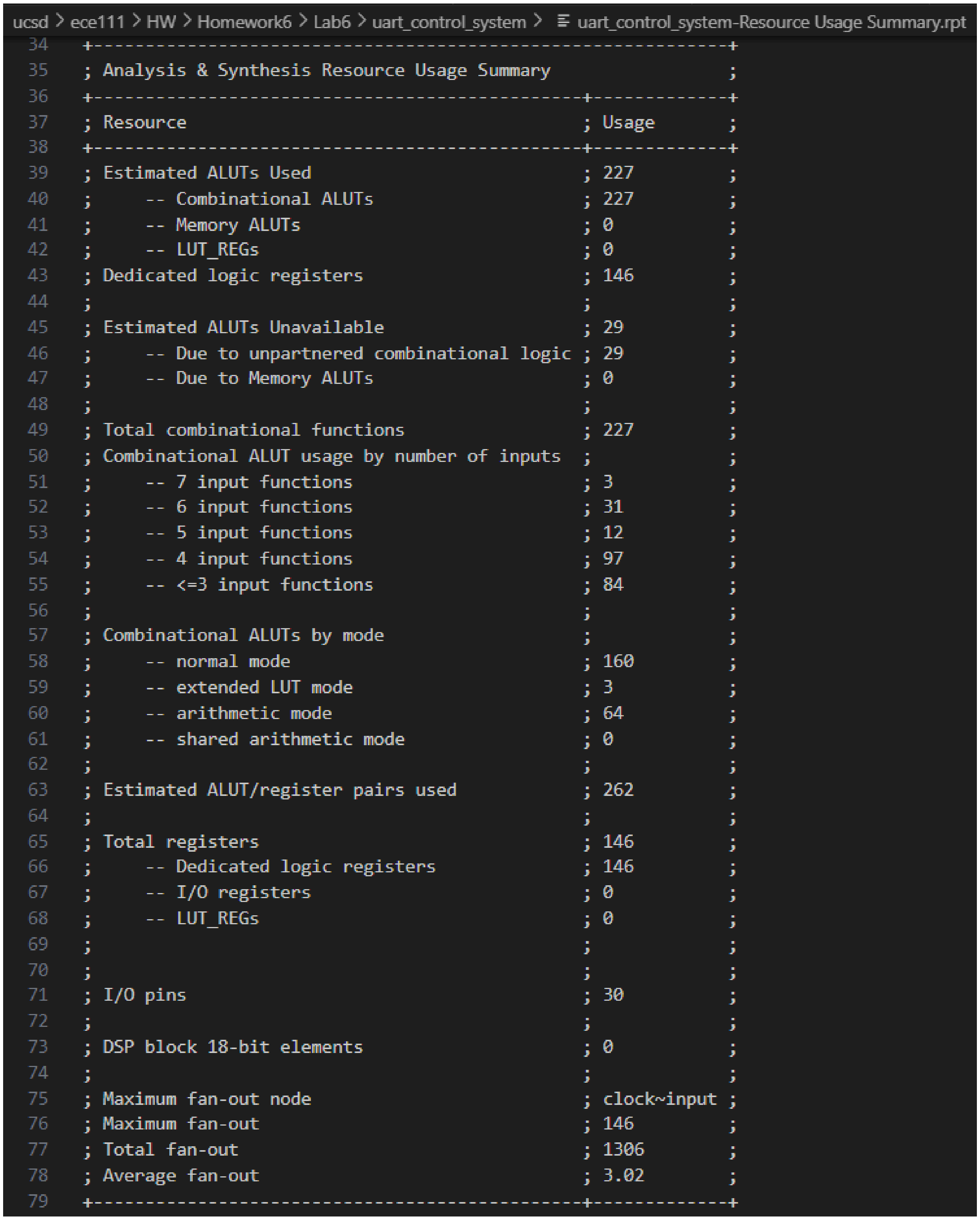
uart_control_system.sv Design Code:


ucsd > ece111 > HW > Homework6 > Lab6 > uart_control_system > uart_control_system.sv

```
// UART Control System Top Level Module
module uart_control_system #(parameter NUM_CLKS_PER_BIT=16, parameter NUM_OF_BYTES=4)
(
  input logic clock, rstn,  // posedge clock and synchronous active low reset
  output logic[3:0] mem_write_addr, // memory write address generated to write RAM in testbench
  output logic[7:0] mem_write_data, // memory write data generated to write data to RAM in testbench
  output logic mem_write_enable, // memory write enable generated to enable writing to RAM in testbench
  input  logic[7:0] mem_read_data, // memory read data returned from ROM in testbench
  output logic[3:0] mem_read_addr, // memory read address generated to read ROM in testbench
  output logic mem_read_enable, // memory read enable generated to enable reading of ROM in testbench
  output logic transmission_done, // indicates all data bytes have been transmitted by uart tx control system
  output logic message_received // indicates all data bytes have been received by uart rx control system
);

// local variable
logic tx_start;
logic tx_done;
logic [7:0] tx_data;
logic [7:0] rx_data;
logic rx_done;

// Instantiate UART TX CONTROL Module
uart_tx_control #(.NUM_OF_BYTES(NUM_OF_BYTES)) tx_control_fsm(
  .clk(clock),
  .rstn(rstn),
  .mem_read_data(mem_read_data), // connect to mem_read_data input primary port
  .mem_read_addr(mem_read_addr), // connect to mem_read_addr output primary port
  .mem_read_enable(mem_read_enable), // connect to mem_read_enable output primary port
  .transmission_done(transmission_done), // connect to transmission_done output primary port
  .uart_tx_done(tx_done), // connect to tx_done coming from uart_top module instance
  .uart_tx_data(tx_data), // connect to tx_data going into uart_top module instance
  .uart_tx_start(tx_start) // connect to tx_start going into uart_top module instance
);

// Instantiate UART RX CONTROL Module
// Note : Student to make connections below for uart_rx_control module instantiation
uart_rx_control #(.NUM_OF_BYTES(NUM_OF_BYTES)) rx_control_fsm(
  .clk(clock),
  .rstn(rstn),
  .mem_write_data(mem_write_data),  // connect to mem_write_data output primary port
  .mem_write_addr(mem_write_addr),  // connect to mem_write_addr output primary port
  .mem_write_enable(mem_write_enable), // connect to mem_write_enable output primary port
  .message_received(message_received),  // connect to message_received output primary port
  .uart_rx_done(rx_done), // connect to rx_done coming from uart_top module instance
  .uart_rx_data(rx_data) // connect to rx_data coming from uart_top module instance
);

// Instantiate UART TOP Module
// uart_top module code has two child modules instantiated : uart_rx and uart_tx modules
// and uart_tx outout tx signal is connected to input rx signal of uart_rx module
// See definition of uart_top in uart_top.sv
uart_top #(.NUM_CLKS_PER_BIT(NUM_CLKS_PER_BIT)) uart_top_inst(
  .tx_clk(clock),
  .tx_rstn(rstn),
  .rx_clk(clock),
  .rx_rstn(rstn),
  .tx_start(tx_start),  // connected to uart_tx_start port of uart_tx_control module instance
  .tx_done(tx_done),  // connected to uart_tx_done port of uart_tx_control module instance
  .tx_din(tx_data),  // connected to uart_tx_data port of uart_tx_control module instance
  .rx_done(rx_done), // connected to uart_rx_done port of uart_rx_control module instance
  .rx_dout(rx_data) // connected to uart_rx_data port of uart_rx_control module instance
);

endmodule : uart_control_system
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

uart_control_system.sv Resource Usage Summary



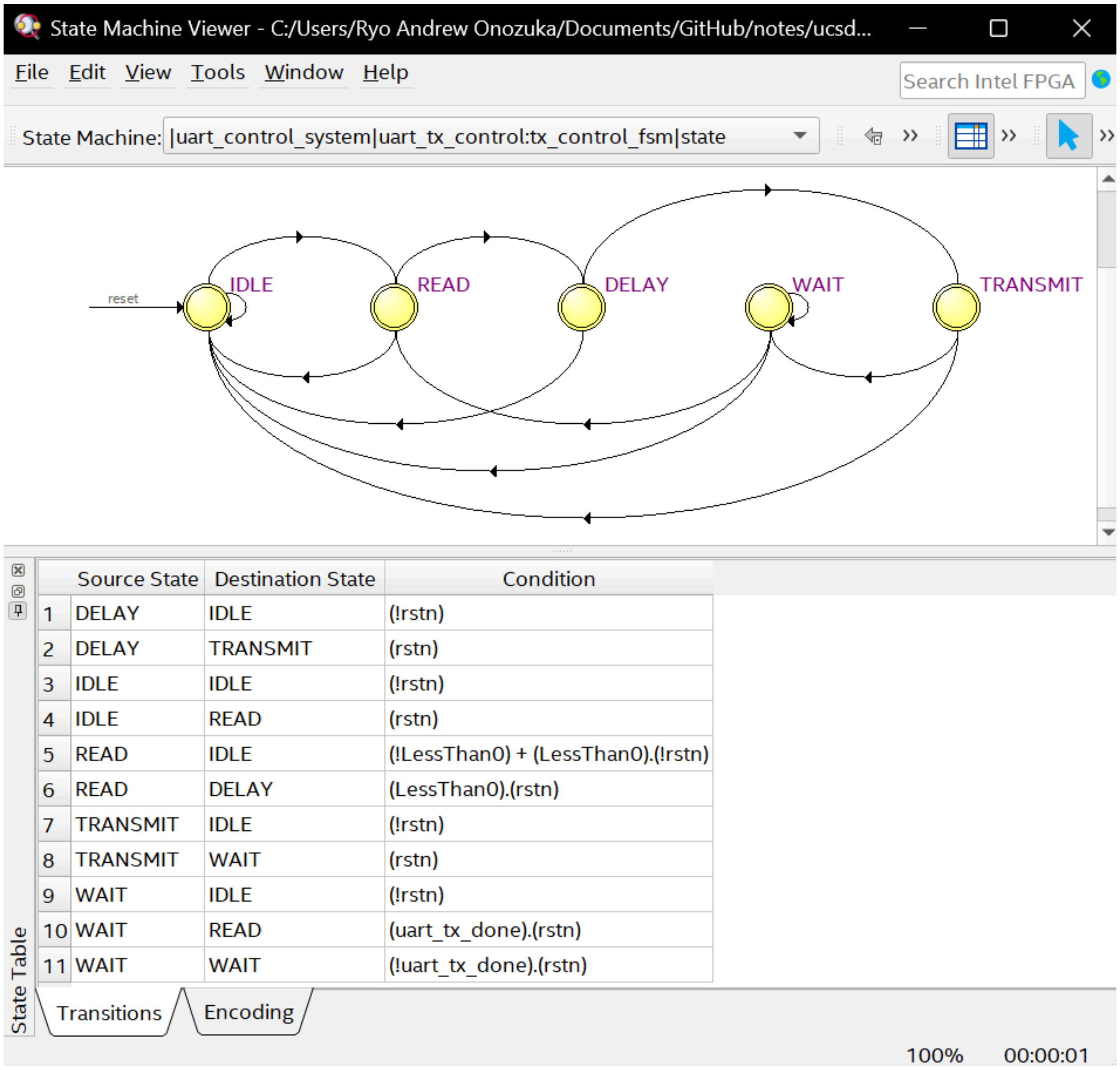
ucsd > ece111 > HW > Homework6 > Lab6 > uart_control_system > uart_control_system-Resource Usage Summary.rpt

```
+--------------------------------------------------------------+
; Analysis & Synthesis Resource Usage Summary                   ;
+-------------------------------------------------+------------+
; Resource                                        ; Usage      ;
+-------------------------------------------------+------------+
; Estimated ALUTs Used                            ; 227        ;
;     -- Combinational ALUTs                      ; 227        ;
;     -- Memory ALUTs                             ; 0          ;
;     -- LUT_REGs                                 ; 0          ;
; Dedicated logic registers                       ; 146        ;
;                                                 ;            ;
; Estimated ALUTs Unavailable                     ; 29         ;
;     -- Due to unpartnered combinational logic   ; 29         ;
;     -- Due to Memory ALUTs                      ; 0          ;
;                                                 ;            ;
; Total combinational functions                   ; 227        ;
; Combinational ALUT usage by number of inputs    ;            ;
;     -- 7 input functions                        ; 3          ;
;     -- 6 input functions                        ; 31         ;
;     -- 5 input functions                        ; 12         ;
;     -- 4 input functions                        ; 97         ;
;     -- <=3 input functions                      ; 84         ;
;                                                 ;            ;
; Combinational ALUTs by mode                     ;            ;
;     -- normal mode                              ; 160        ;
;     -- extended LUT mode                        ; 3          ;
;     -- arithmetic mode                          ; 64         ;
;     -- shared arithmetic mode                   ; 0          ;
;                                                 ;            ;
; Estimated ALUT/register pairs used              ; 262        ;
;                                                 ;            ;
; Total registers                                 ; 146        ;
;     -- Dedicated logic registers                ; 146        ;
;     -- I/O registers                            ; 0          ;
;     -- LUT_REGs                                 ; 0          ;
;                                                 ;            ;
;                                                 ;            ;
; I/O pins                                        ; 30         ;
;                                                 ;            ;
; DSP block 18-bit elements                       ; 0          ;
;                                                 ;            ;
; Maximum fan-out node                            ; clock~input ;
; Maximum fan-out                                 ; 146        ;
; Total fan-out                                   ; 1306       ;
; Average fan-out                                 ; 3.02       ;
+-------------------------------------------------+------------+
```

uart_control_system.sv RTL viewer: not required
uart_control_system.sv post mapping viewer: not required

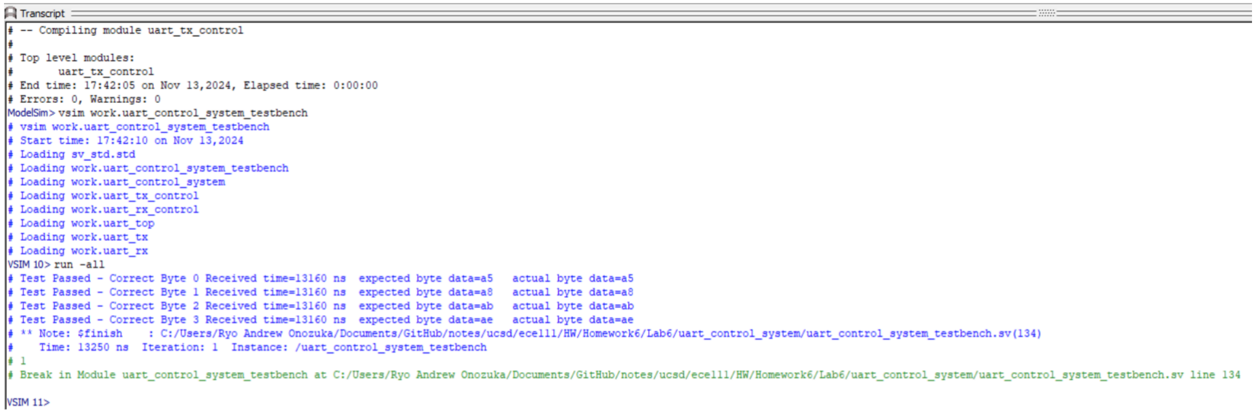
Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

uart_control_system.sv State Machine Viewer + State Transition Table:


State Machine Viewer - C:/Users/Ryo Andrew Onozuka/Documents/GitHub/notes/ucsd...
File Edit View Tools Window Help
Search Intel FPGA
State Machine: |uart_control_system|uart_tx_control:tx_control_fsm|state
reset
IDLE
READ
DELAY
WAIT
TRANSMIT
State Table
Transitions
Encoding
100%
00:00:01

	Source State	Destination State	Condition
1	DELAY	IDLE	(!rstn)
2	DELAY	TRANSMIT	(rstn)
3	IDLE	IDLE	(!rstn)
4	IDLE	READ	(rstn)
5	READ	IDLE	(!LessThan0) + (LessThan0).(!rstn)
6	READ	DELAY	(LessThan0).(rstn)
7	TRANSMIT	IDLE	(!rstn)
8	TRANSMIT	WAIT	(rstn)
9	WAIT	IDLE	(!rstn)
10	WAIT	READ	(uart_tx_done).(rstn)
11	WAIT	WAIT	(!uart_tx_done).(rstn)

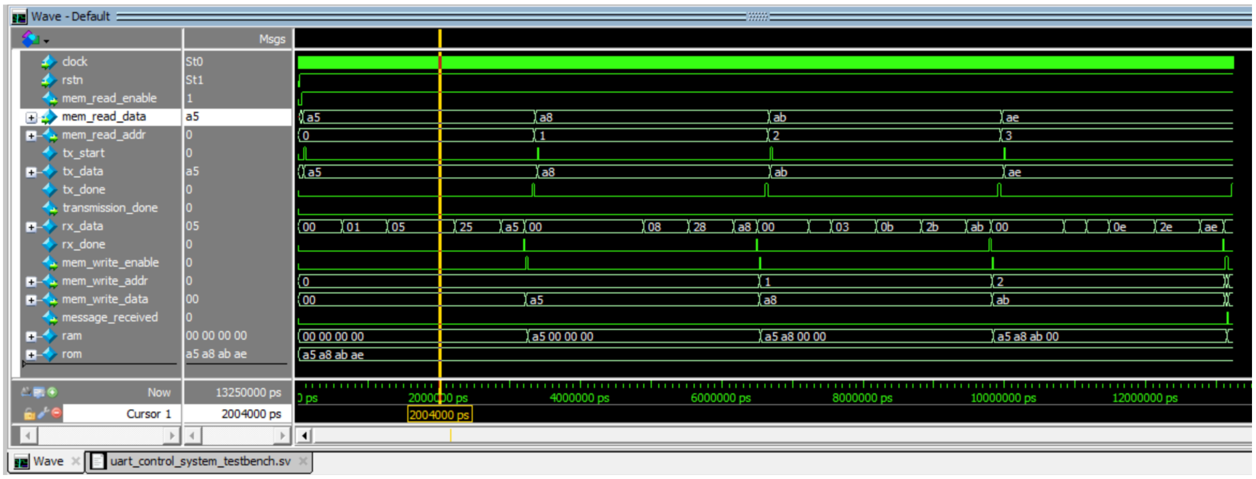
full simulation transcript:



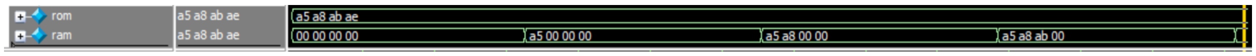
```
# -- Compiling module uart_tx_control
#
# Top level modules:
#       uart_tx_control
# End time: 17:42:05 on Nov 13,2024, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
ModelSim> vsim work.uart_control_system_testbench
# vsim work.uart_control_system_testbench
# Start time: 17:42:10 on Nov 13,2024
# Loading sv_std.std
# Loading work.uart_control_system_testbench
# Loading work.uart_control_system
# Loading work.uart_tx_control
# Loading work.uart_rx_control
# Loading work.uart_top
# Loading work.uart_tx
# Loading work.uart_rx
VSIM 10> run -all
# Test Passed - Correct Byte 0 Received time=13160 ns  expected byte data=a5   actual byte data=a5
# Test Passed - Correct Byte 1 Received time=13160 ns  expected byte data=a8   actual byte data=a8
# Test Passed - Correct Byte 2 Received time=13160 ns  expected byte data=ab   actual byte data=ab
# Test Passed - Correct Byte 3 Received time=13160 ns  expected byte data=ae   actual byte data=ae
# ** Note: $finish    : C:/Users/Ryo Andrew Onozuka/Documents/GitHub/notes/ucsd/ece111/HW/Homework6/Lab6/uart_control_system/uart_control_system_testbench.sv(134)
#    Time: 13250 ns  Iteration: 1  Instance: /uart_control_system_testbench
# 1
# Break in Module uart_control_system_testbench at C:/Users/Ryo Andrew Onozuka/Documents/GitHub/notes/ucsd/ece111/HW/Homework6/Lab6/uart_control_system/uart_control_system_testbench.sv line 134

VSIM 11>
```

Andrew Onozuka A16760043
ECE 111 HW6 | 11/13/2024

simulation snapshot + explanation:



waves are matched and ordered to be consistent with the provided snapshot in the writeup. the rom and ram show that we originally start with 00 00 00 00 and as data bytes are received from uart_rx they are written in ram by rx_control. if we zoom in at the end, we can see that ram fully matches the rom



To summarize all of part c, the modules in the UART Control System handle data flow from ROM to RAM via UART transmission. The uart_tx_control module reads bytes from the ROM, manages the read address, and initiates UART transmission through the uart_top module, which handles bit-by-bit serial transmission of each byte. Meanwhile, uart_rx_control awaits the serial data from uart_top, assembles the bits back into bytes, and writes each byte sequentially into the RAM.

This process is synchronized, ensuring that each byte is only read from ROM and transmitted once the previous byte has been sent and acknowledged by the UART system. uart_tx_control and uart_rx_control work with clear handshakes, where tx_start signals transmission start and tx_done and rx_done verify that each byte is fully transmitted and received. The system ensures each byte received by uart_rx_control is written correctly to RAM with proper addresses, maintaining data integrity and flow control.

The test output confirms that all received bytes match the expected data, proving that the whole system is able to work together accurately.