Andrew Onozuka A16760043
ECE 111 HW4 | 10/30/2024

carry_lookahead_adder.sv

ucsd > ece111 > HW > Homework5 > Lab5 > carry_lookahead_adder > carry_lookahead_adder.sv

```
//`include "fulladder.sv"
module carry_lookahead_adder#(parameter N=4)(
  input logic[N-1:0] A, B,
  input logic CIN,
  output logic[N:0] result
);

 // Internal signals for generate and propagate terms
  logic [N-1:0] G;  // Generate terms
  logic [N-1:0] P;  // Propagate terms
  logic [N:0] C;    // Carry terms

  // Assign initial carry-in
  assign C[0] = CIN;

  // Generate and Propagate signals for each bit
  genvar i;
  generate
    for (i = 0; i < N; i = i + 1) begin : gen_generate_propagate
      assign G[i] = A[i] & B[i];     // Generate term
      assign P[i] = A[i] ^ B[i];     // Propagate term
    end
  endgenerate

  // Carry Lookahead Logic
  generate
    for (i = 1; i <= N; i = i + 1) begin : gen_carry
      assign C[i] = G[i-1] | (P[i-1] & C[i-1]);
    end
  endgenerate

  // Sum Calculation
  generate
    for (i = 0; i < N; i = i + 1) begin : gen_sum
      assign result[i] = P[i] ^ C[i];
    end
  endgenerate

  // Assign final carry-out as the MSB of result
  assign result[N] = C[N];

endmodule : carry_lookahead_adder
```

```
Type   ID     Message
  i  332140 No Minimum Pulse Width paths to report
  i   21076 High junction temperature operating condition is not set. Assuming a default value of '100'.
  i   21076 Low junction temperature operating condition is not set. Assuming a default value of '-40'.
  i  332142 No user constrained base clocks found in the design. Calling "derive_clocks -period 1.0"
  i  332096 The command derive_clocks did not find any clocks to derive.  No clocks were created or changed.
  !  332068 No clocks defined in design.
  i  332154 The derive_clock_uncertainty command did not apply clock uncertainty to any clock-to-clock transfers.
  i  332102 Design is not fully constrained for setup requirements
  i  332102 Design is not fully constrained for hold requirements
  i         Quartus Prime Timing Analyzer was successful. 0 errors, 6 warnings
  i         ************************************************************
  i         Running Quartus Prime EDA Netlist Writer
  i         Command: quartus_eda --read_settings_files=off --write_settings_files=off carry_lookahead_adder -c carry_lookahead_adder
  !  18236 Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PAI
  i  204019 Generated file carry_lookahead_adder.svo in folder "C:/Users/Ryo Andrew Onozuka/Documents/GitHub/notes/ucsd/ece111/HW/Homewor
  i         Quartus Prime EDA Netlist Writer was successful. 0 errors, 1 warning
  i  293000 Quartus Prime Full Compilation was successful. 0 errors, 16 warnings
```

Andrew Onozuka A16760043
ECE 111 HW4 | 10/30/2024

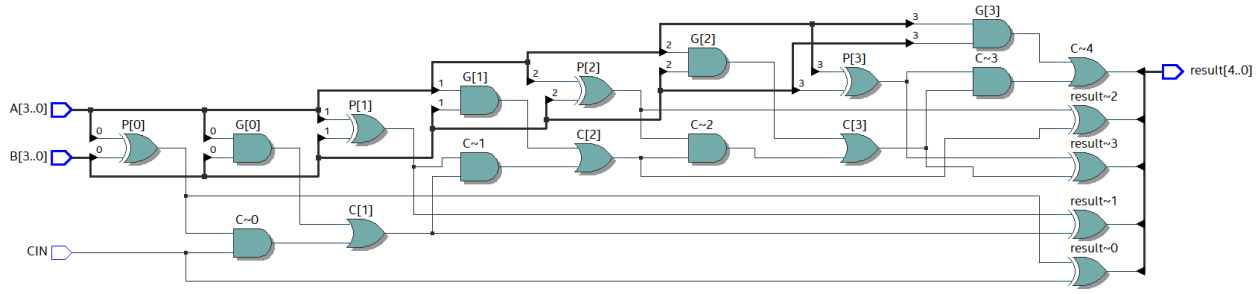
carry_lookahead_adder.sv Resource Usage Summary

ucsd > ece111 > HW > Homework5 > Lab5 > carry_lookahead_adder > carry_lookahead_adder-Resource Usage Summary.rpt

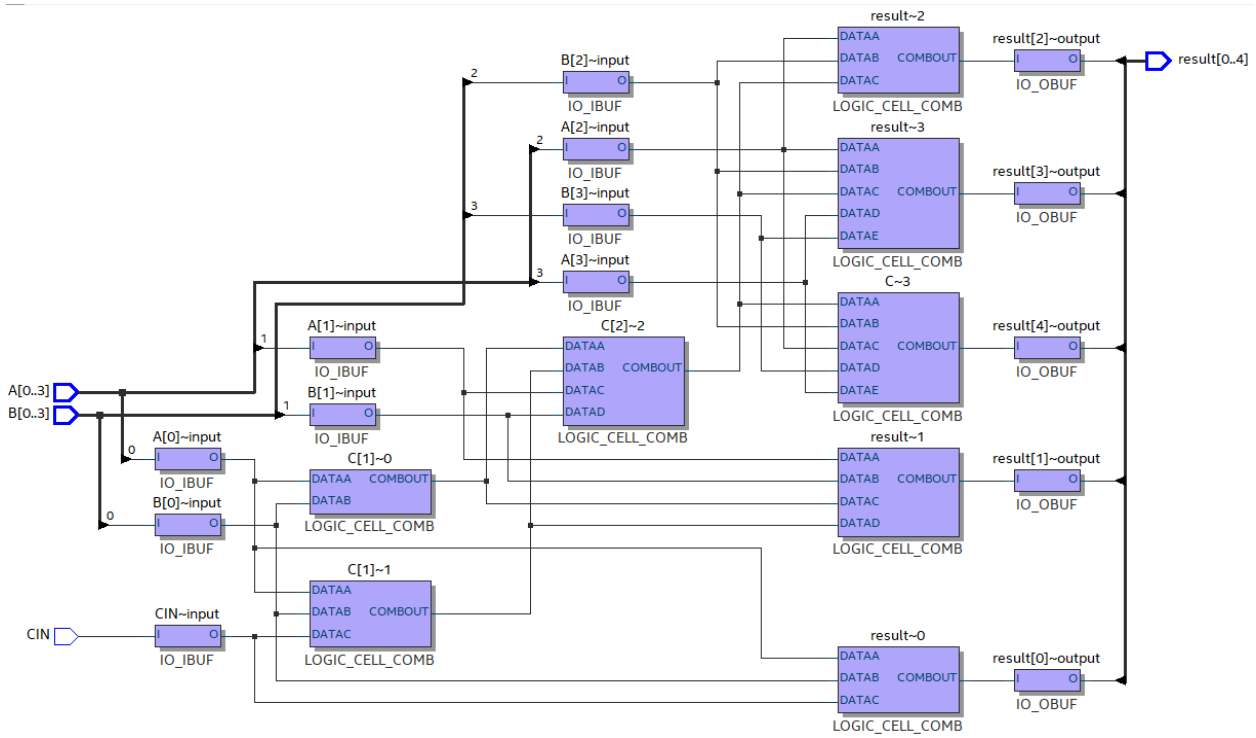
```
+-----------------------------------------------------+
; Analysis & Synthesis Resource Usage Summary         ;
+---------------------------------------------+-------+
; Resource                                    ; Usage ;
+---------------------------------------------+-------+
; Estimated ALUTs Used                        ; 8     ;
;     -- Combinational ALUTs                  ; 8     ;
;     -- Memory ALUTs                        ; 0     ;
;     -- LUT_REGs                            ; 0     ;
; Dedicated logic registers                   ; 0     ;
;                                             ;       ;
; Estimated ALUTs Unavailable                 ; 0     ;
;     -- Due to unpartnered combinational logic ; 0     ;
;     -- Due to Memory ALUTs                 ; 0     ;
;                                             ;       ;
; Total combinational functions               ; 8     ;
; Combinational ALUT usage by number of inputs ;       ;
;     -- 7 input functions                    ; 0     ;
;     -- 6 input functions                    ; 0     ;
;     -- 5 input functions                    ; 2     ;
;     -- 4 input functions                    ; 2     ;
;     -- <=3 input functions                  ; 4     ;
;                                             ;       ;
; Combinational ALUTs by mode                 ;       ;
;     -- normal mode                          ; 8     ;
;     -- extended LUT mode                    ; 0     ;
;     -- arithmetic mode                      ; 0     ;
;     -- shared arithmetic mode               ; 0     ;
;                                             ;       ;
; Estimated ALUT/register pairs used          ; 8     ;
;                                             ;       ;
; Total registers                             ; 0     ;
;     -- Dedicated logic registers            ; 0     ;
;     -- I/O registers                        ; 0     ;
;     -- LUT_REGs                            ; 0     ;
;                                             ;       ;
;                                             ;       ;
; I/O pins                                    ; 14    ;
;                                             ;       ;
; DSP block 18-bit elements                   ; 0     ;
;                                             ;       ;
; Maximum fan-out node                        ; C[2]~2 ;
; Maximum fan-out                             ; 3     ;
; Total fan-out                               ; 48    ;
; Average fan-out                             ; 1.33  ;
+---------------------------------------------+-------+
```

Andrew Onozuka A16760043
ECE 111 HW4 | 10/30/2024

carry_lookahead_adder.sv RTL viewer

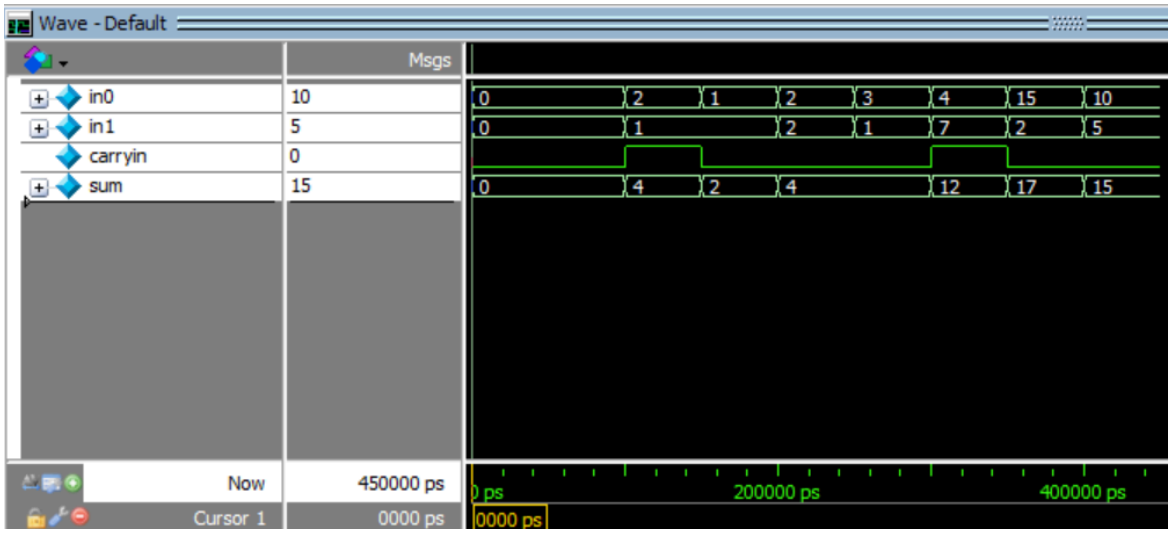


carry_lookahead_adder.sv post mapping viewer



Andrew Onozuka A16760043
ECE 111 HW4 | 10/30/2024

simulation wavelength results explanation:




```
Transcript
# Errors: 0, Warnings: 0
ModelSim> vsim work.carry_lookahead_adder_testbench
# vsim work.carry_lookahead_adder_testbench
# Start time: 11:33:08 on Nov 06,2024
# Loading sv_std.std
# Loading work.carry_lookahead_adder_testbench
# Loading work.carry_lookahead_adder
add wave sim:/carry_lookahead_adder_testbench/*
VSIM 7> run -all
#  time=0   A= 0   B= 0   CIN=0   result= 0
#
#  time=100   A= 2   B= 1   CIN=1   result= 4
#
#  time=150   A= 1   B= 1   CIN=0   result= 2
#
#  time=200   A= 2   B= 2   CIN=0   result= 4
#
#  time=250   A= 3   B= 1   CIN=0   result= 4
#
#  time=300   A= 4   B= 7   CIN=1   result=12
#
#  time=350   A=15   B= 2   CIN=0   result=17
#
#  time=400   A=10   B= 5   CIN=0   result=15
#

VSIM 8>
```

The testbench for the carry lookahead adder initializes inputs A, B, and CIN to various values at different times to verify the functionality of the adder. At each step, it sets specific values for A, B, and CIN and observes the output result, which represents the sum of A, B, and CIN. The testbench uses delays (#50 or #100) between each input change to allow enough time for the computation and for observing results. The simulation output confirms the correct operation of the adder: for instance, when A=2, B=1, and CIN=1, the expected result of 2+1+1=4 is correctly output. Each subsequent test case follows this pattern, where the computed sum aligns with the expected result, such as 3+1=4, 4+7+1=12, and 15+2=17, validating the our carry lookahead adder model's ability to perform addition accurately across various input combinations.

Andrew Onozuka A16760043
ECE 111 HW4 | 10/30/2024

booth_multiplier.sv

ucsd > ece111 > HW > Homework5 > Lab5 > booth_multiplier > booth_multiplier.sv

```
`timescale 1ns/1ps
//`include "carry_lookahead_adder.sv"

module booth_multiplier // Module start declaration
#(parameter N=4) // Parameter declaration
(
        input clock, reset, start,
        input logic signed [N-1:0] multiplicand, multiplier,
        output logic signed [(2*N)-1:0] product,
        output logic done
);

//Variable to store 2's complement of Multiplicand
logic [N:0] multiplicand_neg;

// Count variable for ADD/SHIFT stages
logic [$clog2(N)-1:0] count;

// Register to store Adder sum and multipiler
logic signed [(2*N)+1:0] shift_reg;

// Register to load multiplicand value
logic signed [N:0] load_reg_pos;
logic signed [N:0] load_reg_neg;

// wires to connect with carry lookahead adder
logic[N:0] add_operand1, add_operand2;
logic[N:0] sum;
logic cla_carry;

// next_state encoding and next_state variable
enum logic[2:0]{
        IDLE            = 3'b000,
        INITIALIZE      = 3'b001,
        TEST            = 3'b010,
        ADD             = 3'b011,
        SHIFT_AND_COUNT = 3'b100,
        DONE            = 3'b101
} next_state;

// Instantiate (N+1)-bit carry lookahead adder
//Use add_operand1, add_operand2, sum to connect carry lookahead adder
//Hint: Carry out from the adder is ignored in our calculations and output sum
//has same length as add_operand1 and add_operand2
// Tie CIN to '0'
carry_lookahead_adder #(.N(N+1)) adder_inst(
        .A(add_operand1),
        .B(add_operand2),
        .CIN(1'b0),
        .result(sum)
);

// Create negative multiplicand value
assign multiplicand_neg = -multiplicand;


// Control FSM for Signed Multiplier
// Use Single always block FSM approach
// Use *only* non-blocking assignment statements within always block
always_ff@(posedge clock, posedge reset) begin
        if(reset) begin
                count <= 0;
                next_state <= IDLE;
                load_reg_pos <= 0;
                load_reg_neg <= 0;
                shift_reg <= 0;
        end
        else begin
        case(next_state)
        // Wait for start signal
        IDLE: begin
                if (start) begin
                        next_state <= INITIALIZE;
                end
        end
```

Andrew Onozuka A16760043
ECE 111 HW4 | 10/30/2024

```
        // Load Multiplicand and Multiplier in a load register and a shift register
        INITIALIZE: begin
                // load multiplicand to load_reg_pos
                load_reg_pos <= {multiplicand[N-1], multiplicand};
                // load multiplicand_neg[N:0] to load_reg_neg
                load_reg_neg <= {multiplicand_neg[N-1], multiplicand_neg};

                shift_reg       <= {1'b0, {N{1'b0}}, multiplier, 1'b0};
                next_state      <= TEST;
                count           <= 0;
        end

        // Check shift register LSB and based on that perform ADD/Shift operation
        // if last 2 LSB='01' then perform ADD with positive multiplicand followed by Right Shift by 1
        // if last 2 LSB='10' then perform ADD with negative multiplicand followed by Right Shift by 1
        // if last 2 LSB='00' then perform Right Shift by 1
        // if last 2 LSB='11' then perform Right Shift by 1
        TEST: begin
                if(shift_reg[1:0] == 2'b01) begin
                    // Pass positive Multiplicand to carry lookadahead adder input
                    // Pass previous adder output value after shift to add with Multiplicand
                    // move to add state
                      add_operand1 <= load_reg_pos;
                      add_operand2 <= shift_reg[(2*N):N+1];
                      next_state <= ADD;
                end
                else if(shift_reg[1:0] == 2'b10) begin
                    // Pass negative Multiplicand to carry lookadahead adder input
                    // Pass previous adder output value after shift to add with Multiplicand
                    // move to add state
                      add_operand1 <= load_reg_neg;
                      add_operand2 <= shift_reg[(2*N):N+1];
                      next_state <= ADD;

                end
                else begin
                     // assign add_operand1 to 0, Since no add operation to be perform pass 0 to carry lookadder input
                     // Pass previous adder output value after shift to add with Multiplicand
                     // move to shift and increment count state
                      add_operand1 <= {(N+1){1'b0}};
                      add_operand2 <= shift_reg[(2*N):N+1];
                      next_state <= SHIFT_AND_COUNT;

                end
        end

        ADD: begin
                shift_reg <= {sum, shift_reg[N:0]}; // Load shift register : Output sum from Adder which includes carry and retain previous lower bit of shift register
                // Move to shift and increment count state
                next_state <= SHIFT_AND_COUNT;
        end

        SHIFT_AND_COUNT: begin
                shift_reg <= (shift_reg >>> 1); // Right Arithmetic shift entire shift register by 1 position

                // Increment count
                count <= count + 1;

                if(count == N-1) begin // If 'N' times SHIFT operation performed then move to Done state else go back to Test state
                        next_state <= DONE;
                end
                else begin
                        next_state <= TEST;
                end
        end

        DONE: begin
                next_state <= IDLE; // Wait for right shift value to be available. This is the final product value.
        end
        endcase
        end
end

// Generate done=1 when FSM reaches DONE state
assign done = (next_state == DONE) ? 1 : 0;

// Generate Product in DONE state by loading shift_reg value to it
assign product = (next_state == DONE) ? {shift_reg[(2*N)], shift_reg[(2*N):1]} : 0;

endmodule: booth_multiplier
```

```
Type   ID     Message
   0 332140 No Recovery paths to report
   0 332140 No Removal paths to report
   0 332146 Worst-case minimum pulse width slack is -2.846
   0        Analyzing Fast 900mV -40C Model
   0 332123 Deriving Clock Uncertainty. Please refer to report_sdc in the Timing Analyzer to see clock uncertainties.
   0 332146 Worst-case setup slack is 0.296
   0 332146 Worst-case hold slack is 0.113
   0 332140 No Recovery paths to report
   0 332140 No Removal paths to report
   1 332148 Timing requirements not met
   0 332146 Worst-case minimum pulse width slack is -2.846
   0  21076 High junction temperature operating condition is not set. Assuming a default value of '100'.
   0  21076 Low junction temperature operating condition is not set. Assuming a default value of '-40'.
   0 332123 Deriving Clock Uncertainty. Please refer to report_sdc in the Timing Analyzer to see clock uncertainties.
   0 332102 Design is not fully constrained for setup requirements
   0 332102 Design is not fully constrained for hold requirements
   0        Quartus Prime Timing Analyzer was successful. 0 errors, 5 warnings
   0        ******************************************************************
   0        Running Quartus Prime EDA Netlist Writer
   0        Command: quartus_eda --read_settings_files=off --write_settings_files=off booth_multiplier -c booth_multiplier
   1  18236 Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PAI
   0 204019 Generated file booth_multiplier.svo in folder "C:/Users/Ryo Andrew Onozuka/Documents/GitHub/notes/ucsd/ece111/HW/Homework5/Lal
   0        Quartus Prime EDA Netlist Writer was successful. 0 errors, 1 warning
   0 293000 Quartus Prime Full Compilation was successful. 0 errors, 19 warnings
```

Andrew Onozuka A16760043
ECE 111 HW4 | 10/30/2024

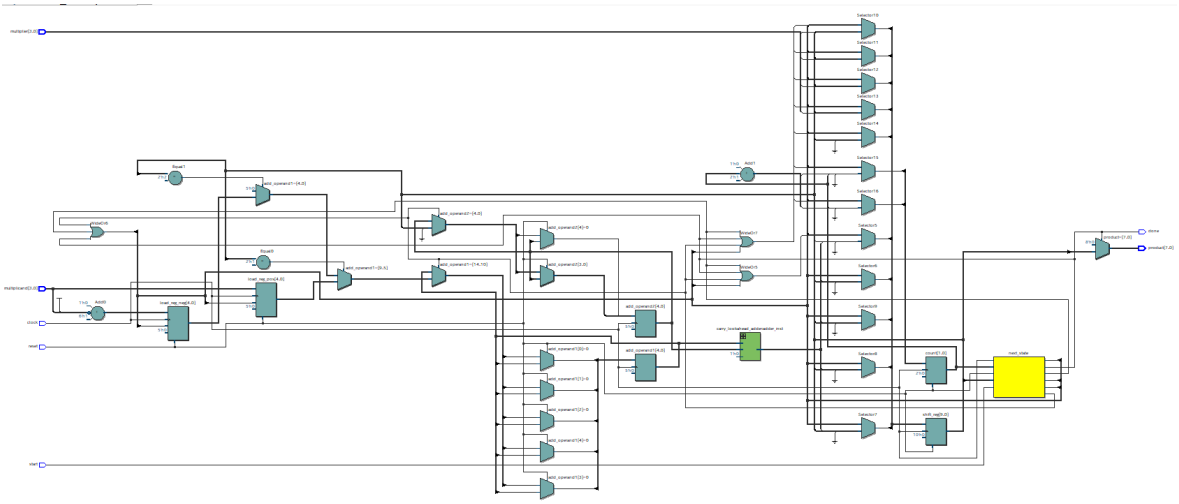
booth_multiplier.sv Resource Usage Summary

ucsd > ece111 > HW > Homework5 > Lab5 > booth_multiplier > booth_multiplier-Resource Usage Summary.rpt

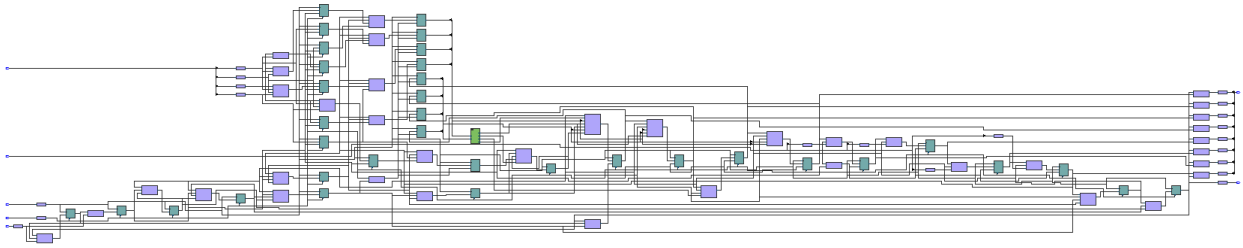
```
+-----------------------------------------------------------------+
; Analysis & Synthesis Resource Usage Summary                     ;
+-------------------------------------------------+---------------+
; Resource                                        ; Usage         ;
+-------------------------------------------------+---------------+
; Estimated ALUTs Used                            ; 43            ;
;     -- Combinational ALUTs                      ; 43            ;
;     -- Memory ALUTs                            ; 0             ;
;     -- LUT_REGs                                ; 0             ;
; Dedicated logic registers                       ; 35            ;
;                                                 ;               ;
; Estimated ALUTs Unavailable                     ; 1             ;
;     -- Due to unpartnered combinational logic   ; 1             ;
;     -- Due to Memory ALUTs                      ; 0             ;
;                                                 ;               ;
; Total combinational functions                   ; 43            ;
; Combinational ALUT usage by number of inputs    ;               ;
;     -- 7 input functions                        ; 1             ;
;     -- 6 input functions                        ; 2             ;
;     -- 5 input functions                        ; 3             ;
;     -- 4 input functions                        ; 13            ;
;     -- <=3 input functions                      ; 24            ;
;                                                 ;               ;
; Combinational ALUTs by mode                     ;               ;
;     -- normal mode                              ; 42            ;
;     -- extended LUT mode                        ; 1             ;
;     -- arithmetic mode                          ; 0             ;
;     -- shared arithmetic mode                   ; 0             ;
;                                                 ;               ;
; Estimated ALUT/register pairs used              ; 48            ;
;                                                 ;               ;
; Total registers                                 ; 35            ;
;     -- Dedicated logic registers                ; 35            ;
;     -- I/O registers                            ; 0             ;
;     -- LUT_REGs                                ; 0             ;
;                                                 ;               ;
;                                                 ;               ;
; I/O pins                                        ; 20            ;
;                                                 ;               ;
; DSP block 18-bit elements                       ; 0             ;
;                                                 ;               ;
; Maximum fan-out node                            ; clock~input   ;
; Maximum fan-out                                 ; 35            ;
; Total fan-out                                   ; 295           ;
; Average fan-out                                 ; 2.50          ;
+-------------------------------------------------+---------------+

```

Andrew Onozuka A16760043
ECE 111 HW4 | 10/30/2024

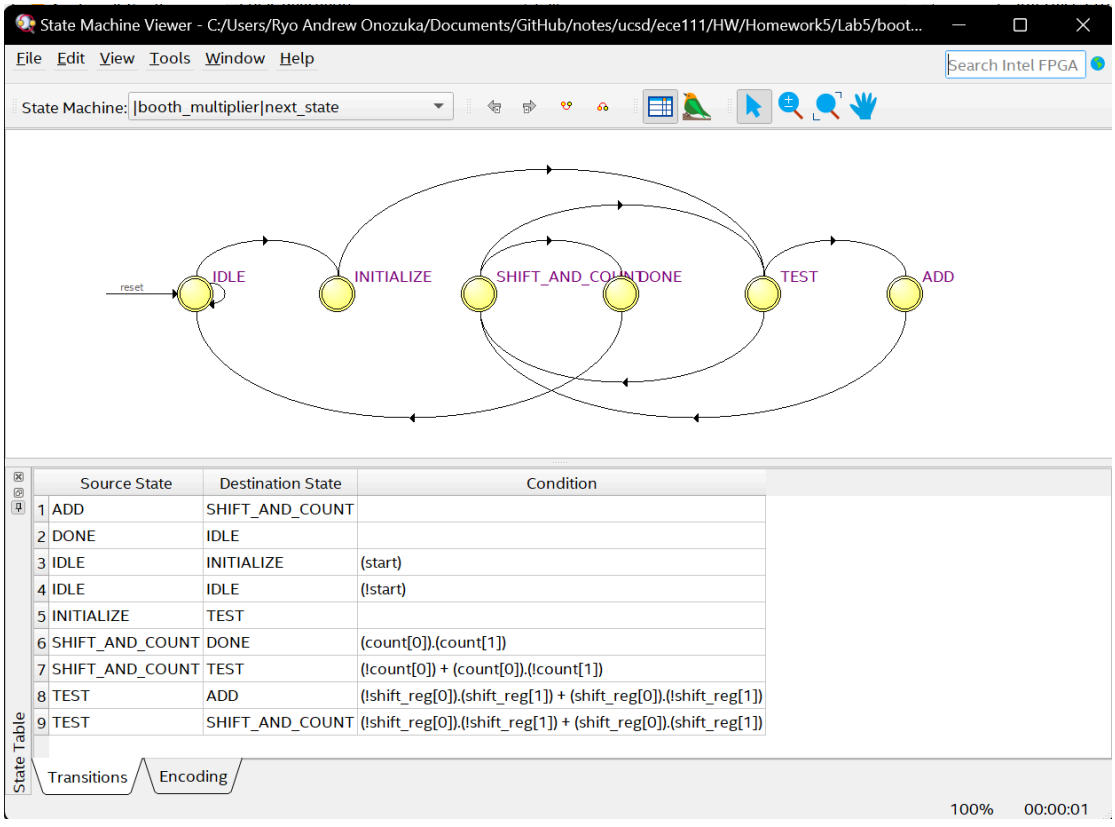
booth_multiplier.sv RTL viewer



booth_multiplier.sv post mapping viewer



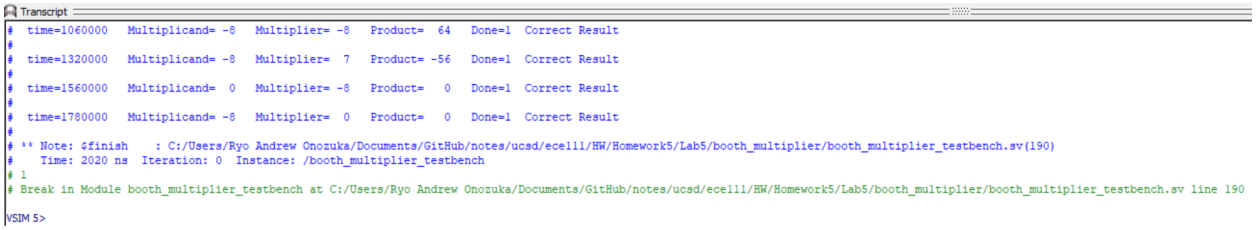
booth_multiplier.sv state machine viewer



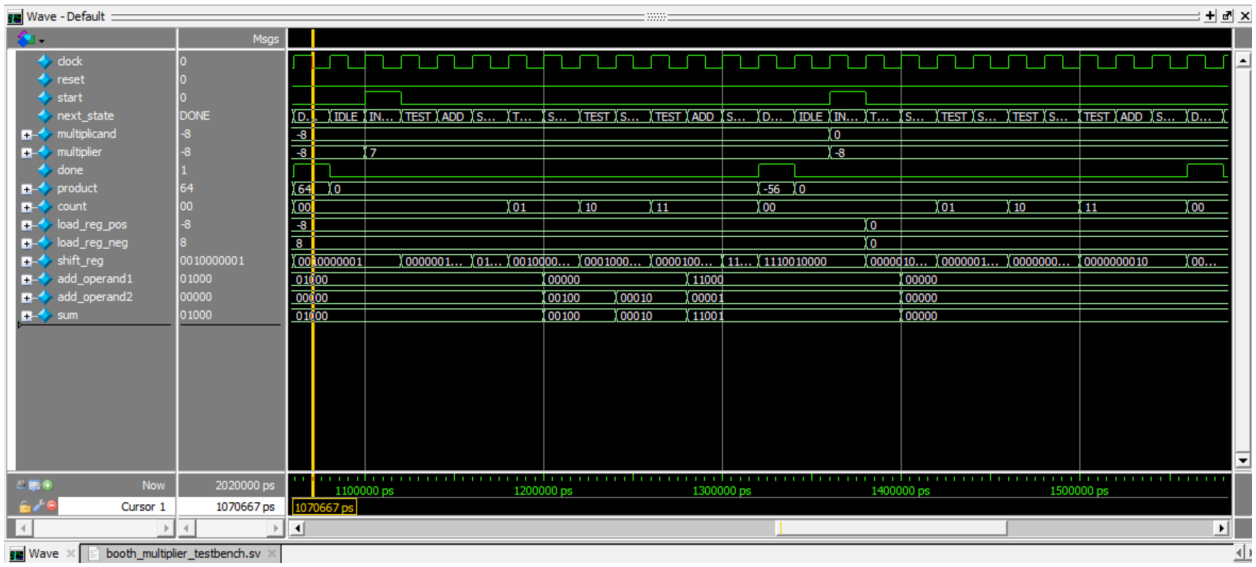
	Source State	Destination State	Condition
1	ADD	SHIFT_AND_COUNT	
2	DONE	IDLE	
3	IDLE	INITIALIZE	(start)
4	IDLE	IDLE	(!start)
5	INITIALIZE	TEST	
6	SHIFT_AND_COUNT	DONE	(count[0]).(count[1])
7	SHIFT_AND_COUNT	TEST	(!count[0]) + (count[0]).(!count[1])
8	TEST	ADD	(!shift_reg[0]).(shift_reg[1]) + (shift_reg[0]).(!shift_reg[1])
9	TEST	SHIFT_AND_COUNT	(!shift_reg[0]).(!shift_reg[1]) + (shift_reg[0]).(shift_reg[1])

Andrew Onozuka A16760043
ECE 111 HW4 | 10/30/2024

simulation transcript:



simulation wavelength results explanation:



This waveform demonstrates the behavior of a Booth multiplier as it processes a sequence of signed multiplication operations. Key signals such as multiplicand, multiplier, product, next_state, and various internal registers (shift_reg, add_operand1, add_operand2, and sum) are displayed, showing the step-by-step computation and state transitions. The next_state signal progresses through different stages (e.g., IDLE, INITIALIZE, TEST, ADD, SHIFT_AND_COUNT, and DONE), indicating the control flow of the Booth multiplication algorithm. Also see the state machine shown above.

For each multiplication operation, multiplicand and multiplier values are loaded, and the computation proceeds as the count variable increments and the shift_reg is updated. Notably, in the first computation (multiplicand = -8, multiplier = -8), the product correctly reaches 64 when next_state enters DONE, confirming the accuracy of the result. Similarly, subsequent operations with different combinations of positive, negative, and zero inputs (multiplicand = -8, multiplier = 7, and others) produce the expected results, as indicated by the correct product values and the final Done=1 assertion, verifying successful completion for each case.

This waveform effectively illustrates how the Booth multiplier module handles control flow and internal arithmetic operations, providing a visual confirmation of its correctness in computing signed products.