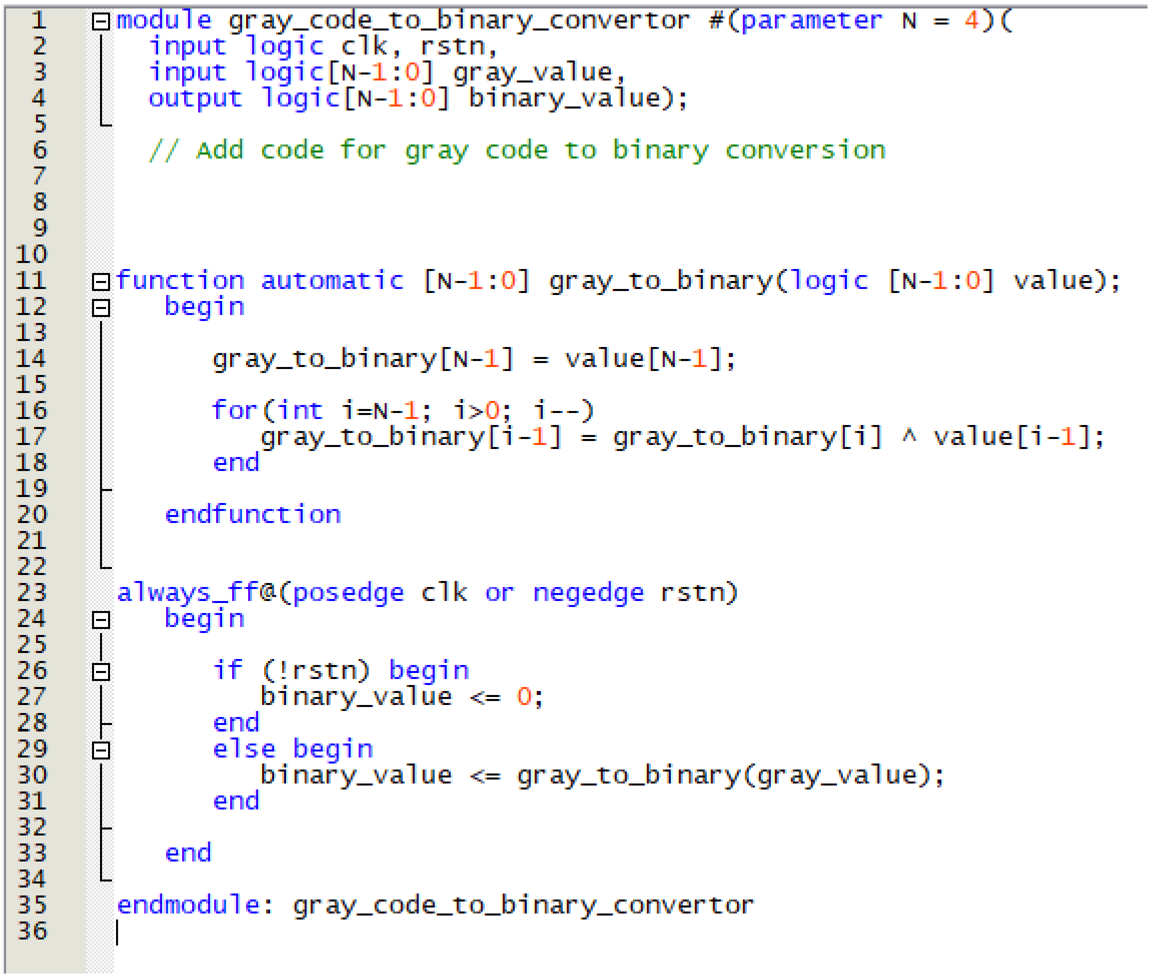
ECE 111 Winter 2022
HW5
Hao Le A15547504

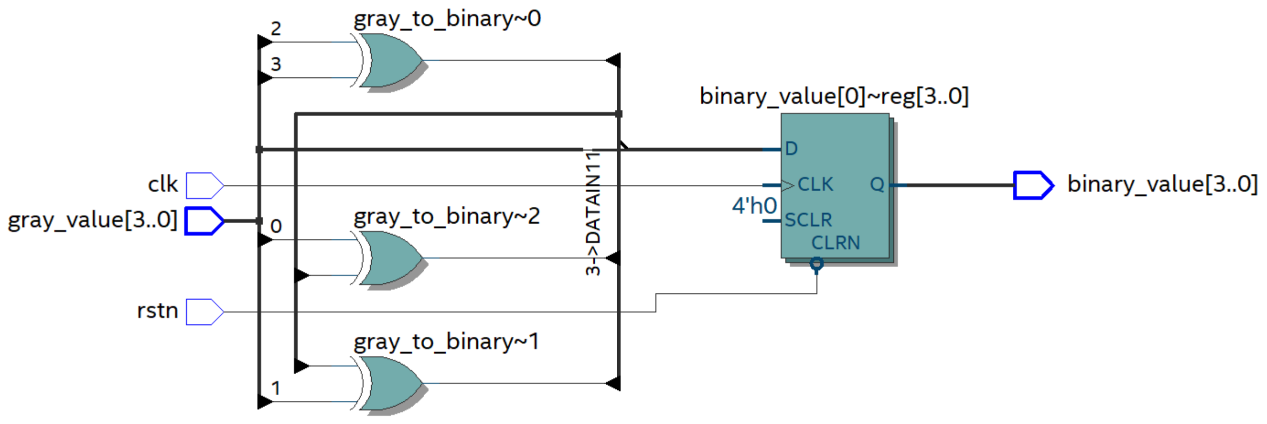
Gray to Binary Converter

Code

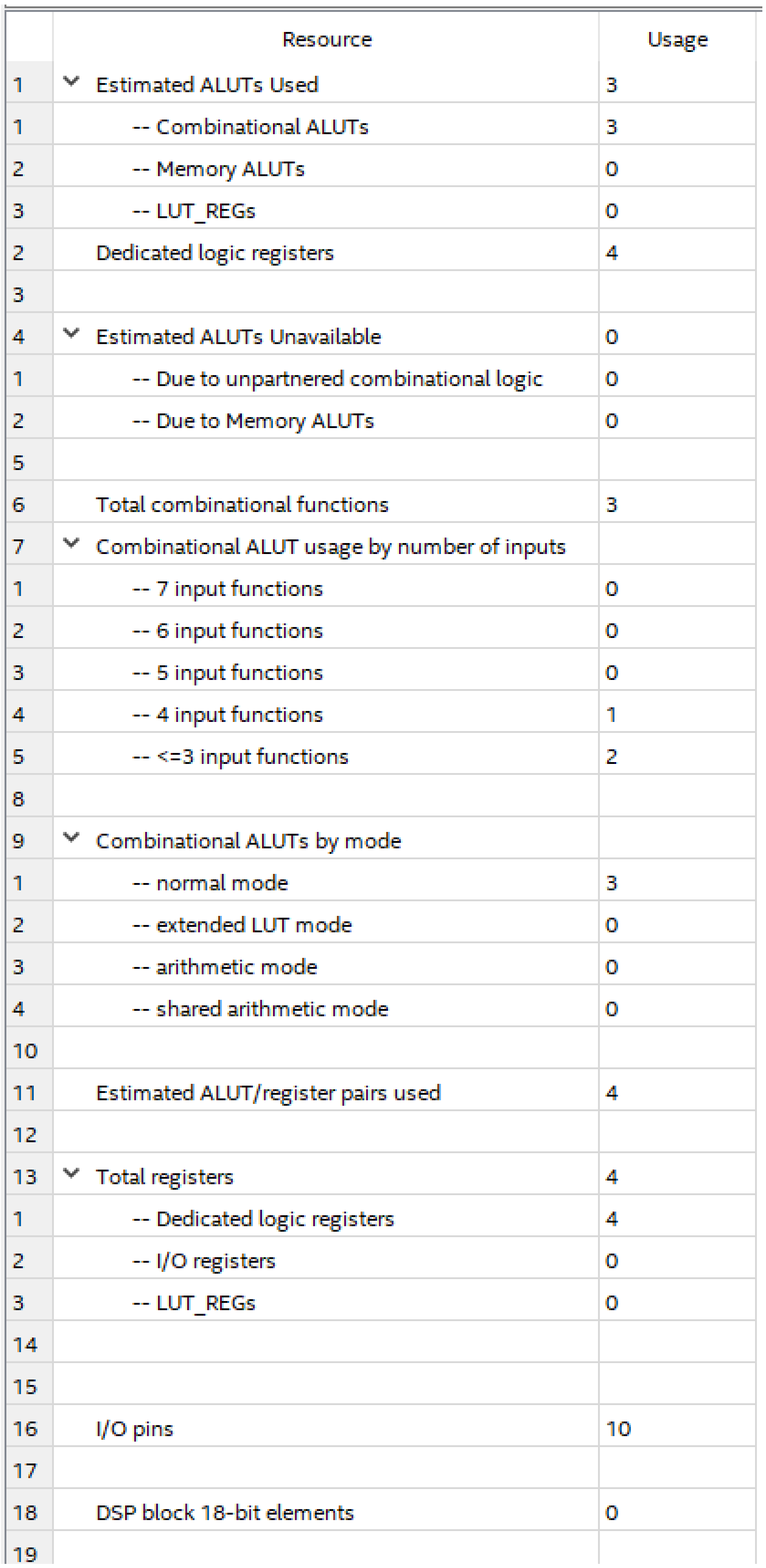


```
module gray_code_to_binary_convertor #(parameter N = 4)(
  input logic clk, rstn,
  input logic[N-1:0] gray_value,
  output logic[N-1:0] binary_value);

  // Add code for gray code to binary conversion



function automatic [N-1:0] gray_to_binary(logic [N-1:0] value);
   begin

      gray_to_binary[N-1] = value[N-1];

      for(int i=N-1; i>0; i--)
         gray_to_binary[i-1] = gray_to_binary[i] ^ value[i-1];
      end

   endfunction

always_ff@(posedge clk or negedge rstn)
   begin

      if (!rstn) begin
         binary_value <= 0;
      end
      else begin
         binary_value <= gray_to_binary(gray_value);
      end

   end

endmodule: gray_code_to_binary_convertor
```

RTL netlist

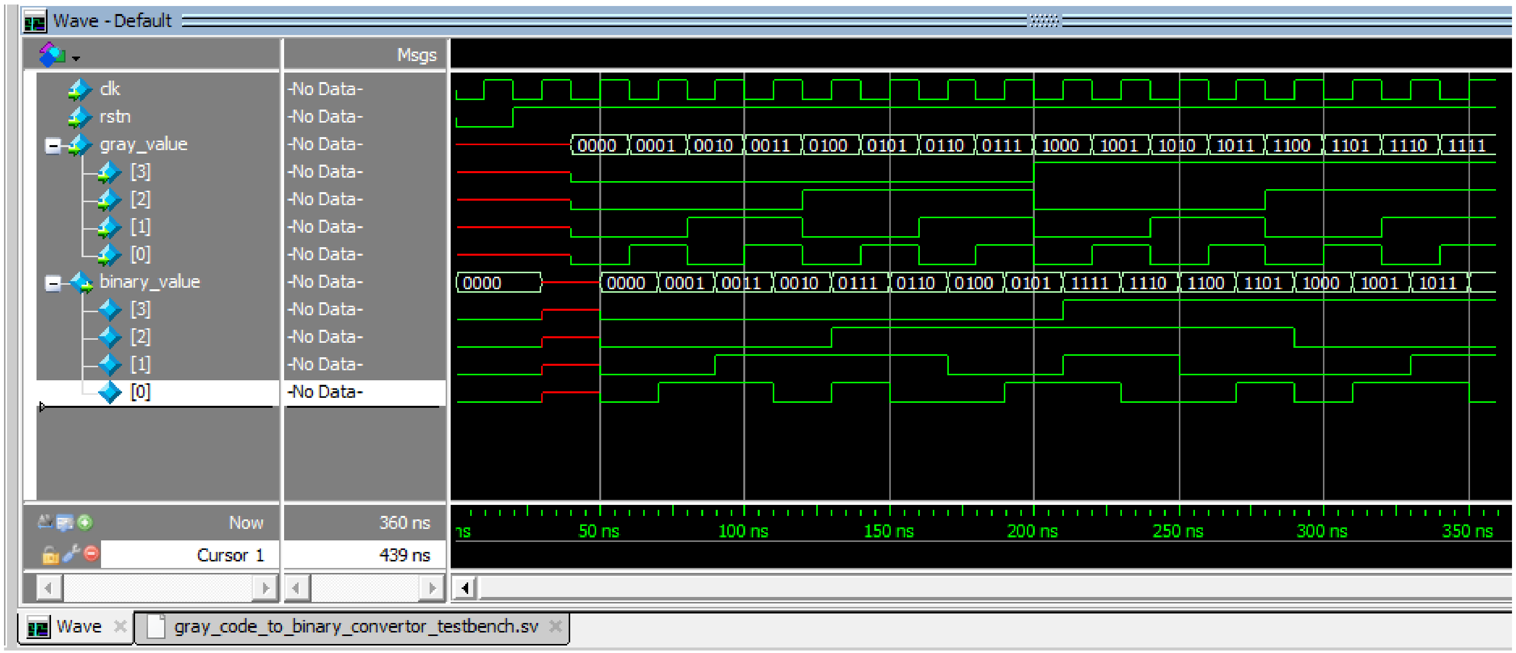

gray_to_binary~0
2
3
binary_value[0]~reg[3..0]
D
clk
CLK
Q
binary_value[3..0]
gray_value[3..0]
0
4'h0
gray_to_binary~2
SCLR
CLRN
3->DATAIN11
rstn
gray_to_binary~1
1

Resource usage



	Resource	Usage
1	Estimated ALUTs Used	3
1	-- Combinational ALUTs	3
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	4
3		
4	Estimated ALUTs Unavailable	0
1	-- Due to unpartnered combinational logic	0
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	3
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	0
3	-- 5 input functions	0
4	-- 4 input functions	1
5	-- <=3 input functions	2
8		
9	Combinational ALUTs by mode	
1	-- normal mode	3
2	-- extended LUT mode	0
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	4
12		
13	Total registers	4
1	-- Dedicated logic registers	4
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
15		
16	I/O pins	10
17		
18	DSP block 18-bit elements	0
19		

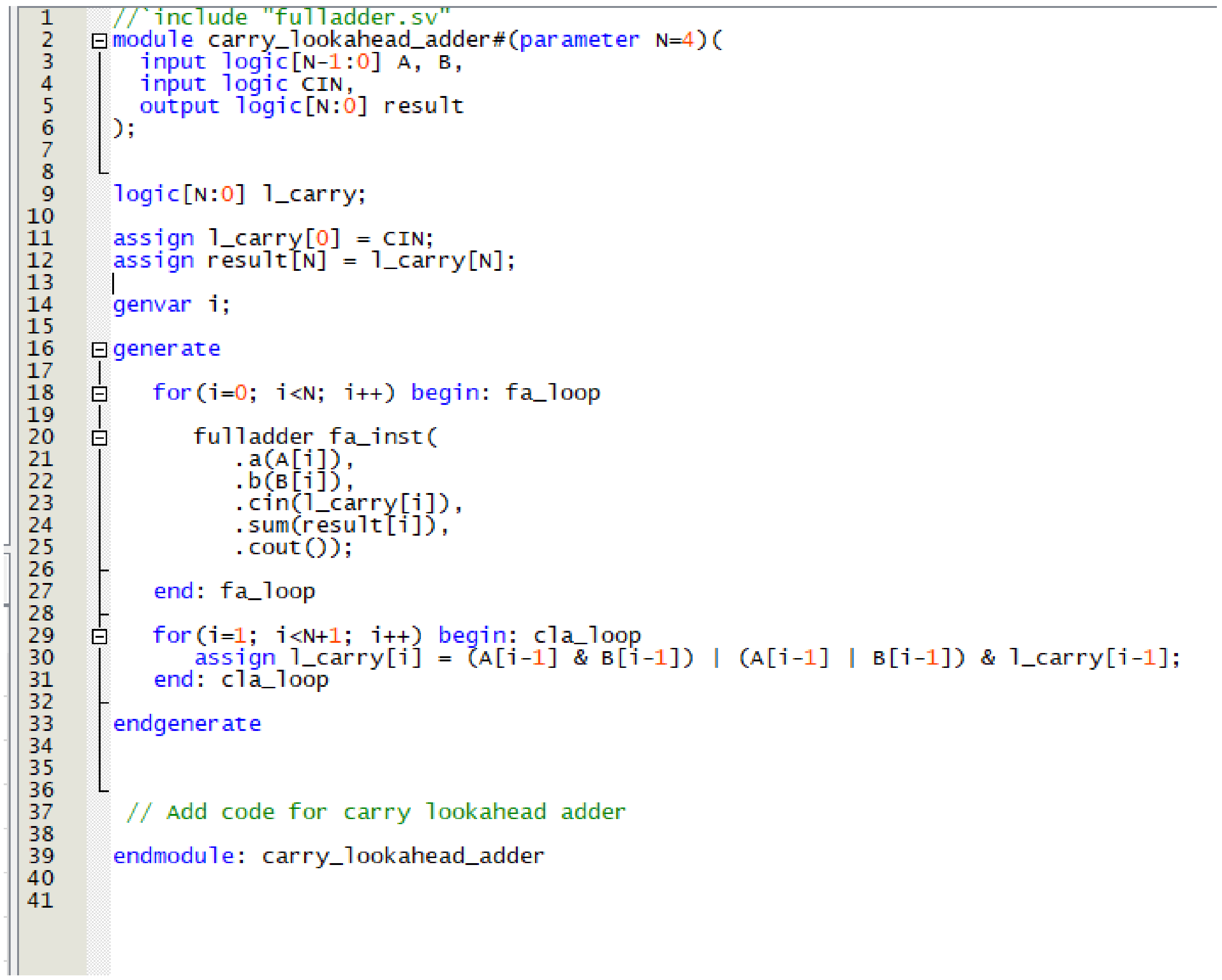
Testbench simulation waveform



- The testbench in this case was for a 4-bit Gray to Binary converter. From the waveform, we see initially rstn going low, thus binary_value goes to 0 because of the negative edge which is expected. The converter will convert gray_value upon every positive edge of clk. For example, when 0100 is loaded into gray_value, binary_value turns into 0111 which is the correct conversion according to the truth table. This correctness holds for all 15 other cases of gray_value which has a one-to-one mapping to 15 binary_value's.
- Testbench was modified to instantiate the converter at different bit values by modifying the N parameter. The converter works for cases 2-8.

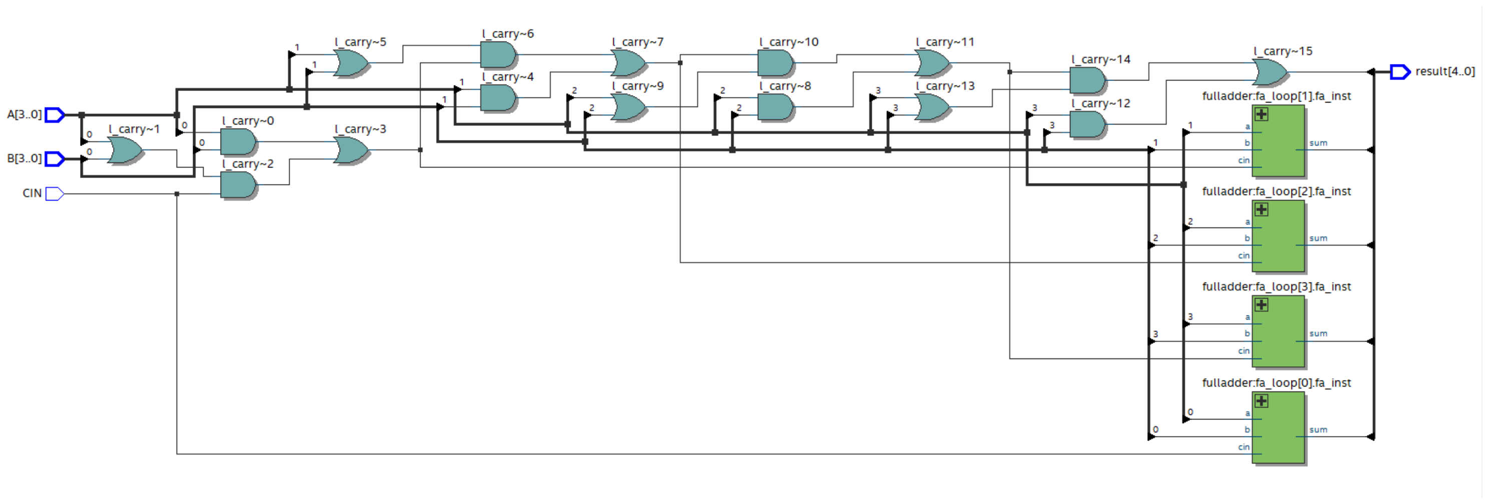
Carry Lookahead Adder

Code



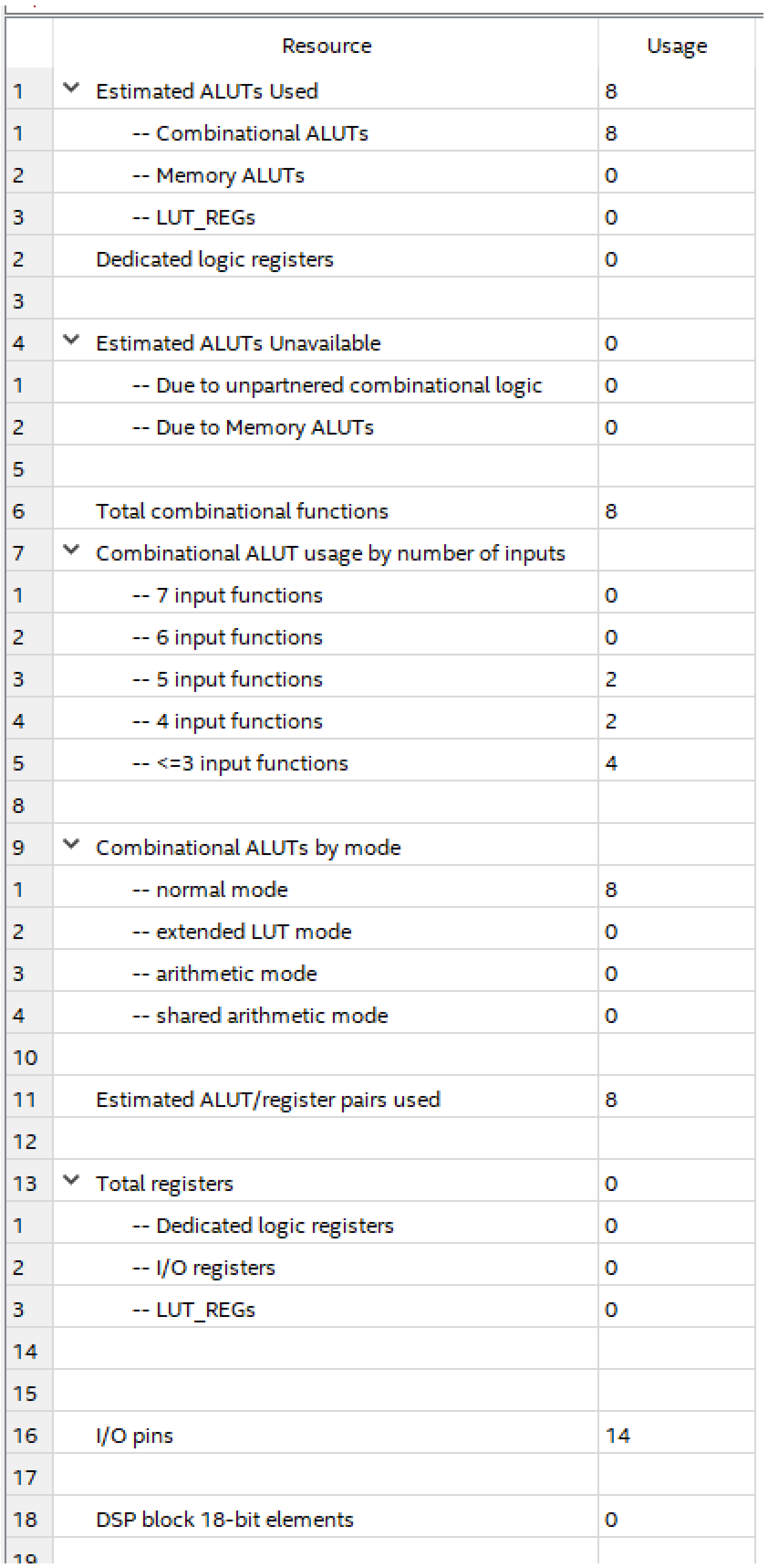
```
//`include "fulladder.sv"
module carry_lookahead_adder#(parameter N=4)(
  input logic[N-1:0] A, B,
  input logic CIN,
  output logic[N:0] result
);


logic[N:0] l_carry;

assign l_carry[0] = CIN;
assign result[N] = l_carry[N];

genvar i;

generate

   for(i=0; i<N; i++) begin: fa_loop

      fulladder fa_inst(
         .a(A[i]),
         .b(B[i]),
         .cin(l_carry[i]),
         .sum(result[i]),
         .cout());

   end: fa_loop

   for(i=1; i<N+1; i++) begin: cla_loop
      assign l_carry[i] = (A[i-1] & B[i-1]) | (A[i-1] | B[i-1]) & l_carry[i-1];
   end: cla_loop

endgenerate


 // Add code for carry lookahead adder

endmodule: carry_lookahead_adder
```

RTL netlist



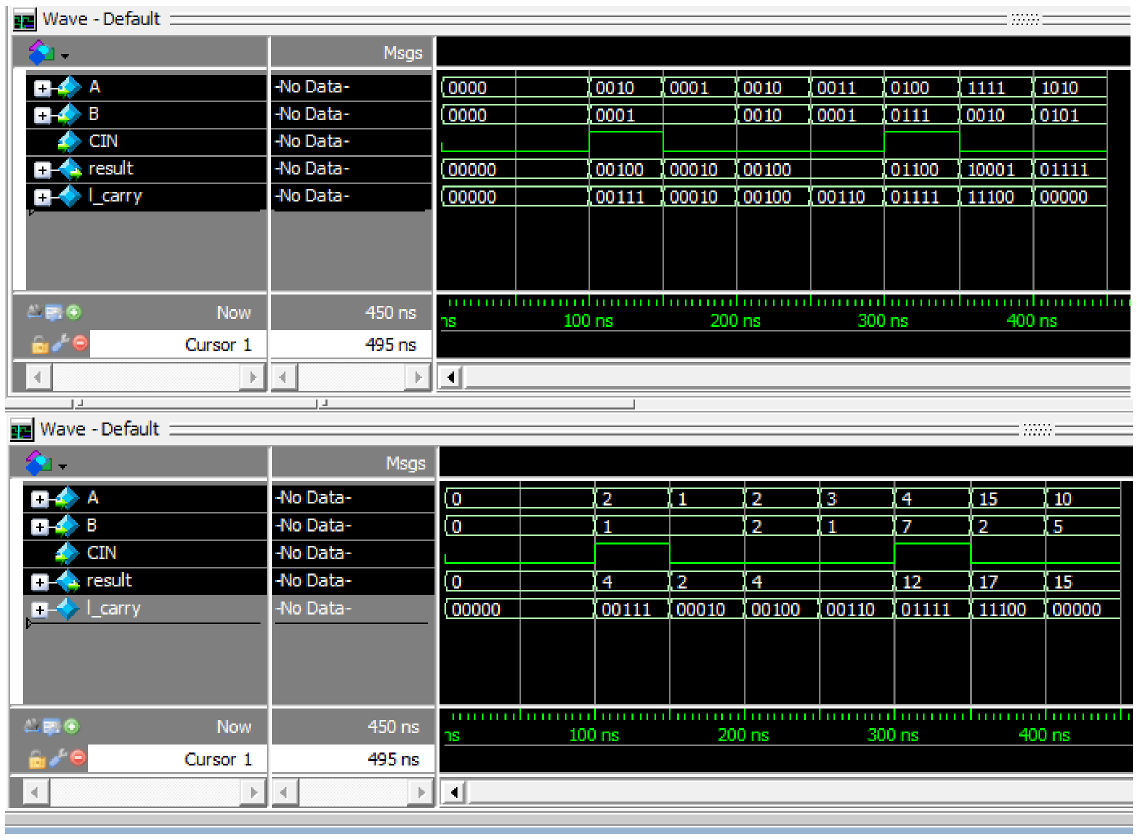
- The netlist shows that all 4 full adders output to the result, and inputs do not depend on the outputs of any other full adder. Hence, the addition stage is all parallel which is the advantage of CLA.

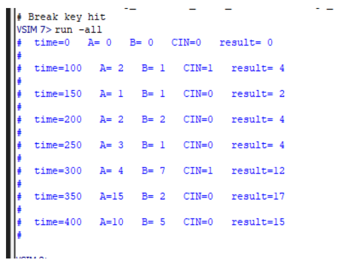
Resource usage



	Resource	Usage
1	Estimated ALUTs Used	8
1	-- Combinational ALUTs	8
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	0
3		
4	Estimated ALUTs Unavailable	0
1	-- Due to unpartnered combinational logic	0
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	8
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	0
2	-- 6 input functions	0
3	-- 5 input functions	2
4	-- 4 input functions	2
5	-- <=3 input functions	4
8		
9	Combinational ALUTs by mode	
1	-- normal mode	8
2	-- extended LUT mode	0
3	-- arithmetic mode	0
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	8
12		
13	Total registers	0
1	-- Dedicated logic registers	0
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
15		
16	I/O pins	14
17		
18	DSP block 18-bit elements	0

Testbench simulation waveform





```
# Break key hit
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
#
```

- The testbench waveform shows that the CLA functions correctly as indicated by the transcript which converts the binary values to decimal for intuitive validation. For example, at 300 ns A=4, B=7, and CIN=1 is fed in, and the result is 12 which is correct since 4 + 7 + 1 = 12. This holds true for the other test cases. Also, as soon as A or B changes, the result changes to correct summation, indicating that the CLA is purely combinational which is expected.