Homework 6 Solutions

1. **Construct a 4-bit ripple-carry adder with four full-adder blocks using Aldec ActiveHDL. First construct - out of basic gates from the lib370 library - a single-bit full-adder block to reuse. Verify your design using simulation, turn in the schematic and timing waveforms showing what happens when you have "1111" and "0000" as the numbers to be added and you change the "0000" to "0001". How long does it take the sum to get to the right value? Repeat this experiment starting with "1010" and "0000" and changing the "0000" to "0101". Explain the differences between the two cases.**

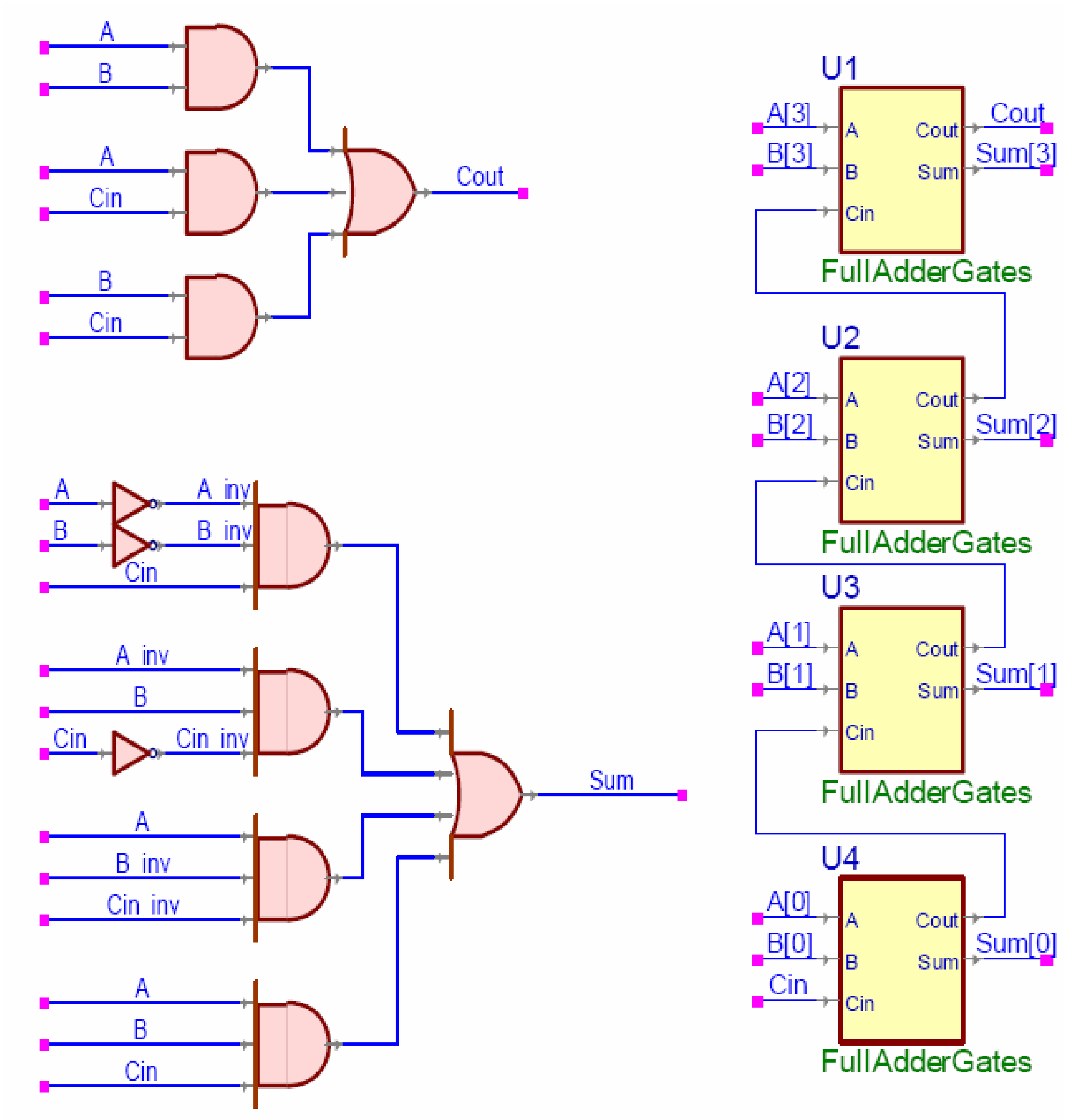
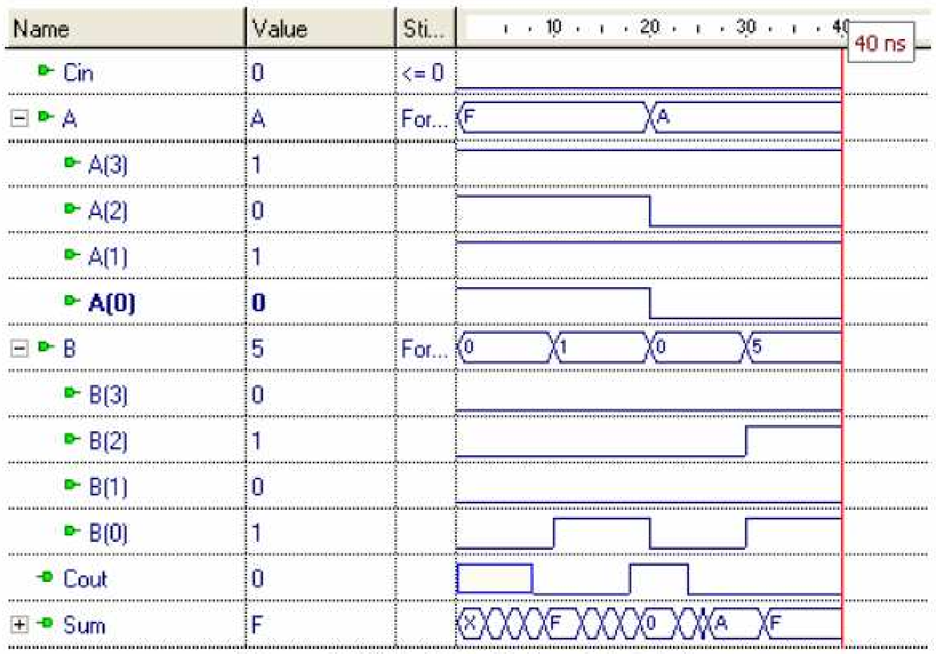


Figure 1 (left) Full Adder (right) 4-bit Ripple-Carry Adder.



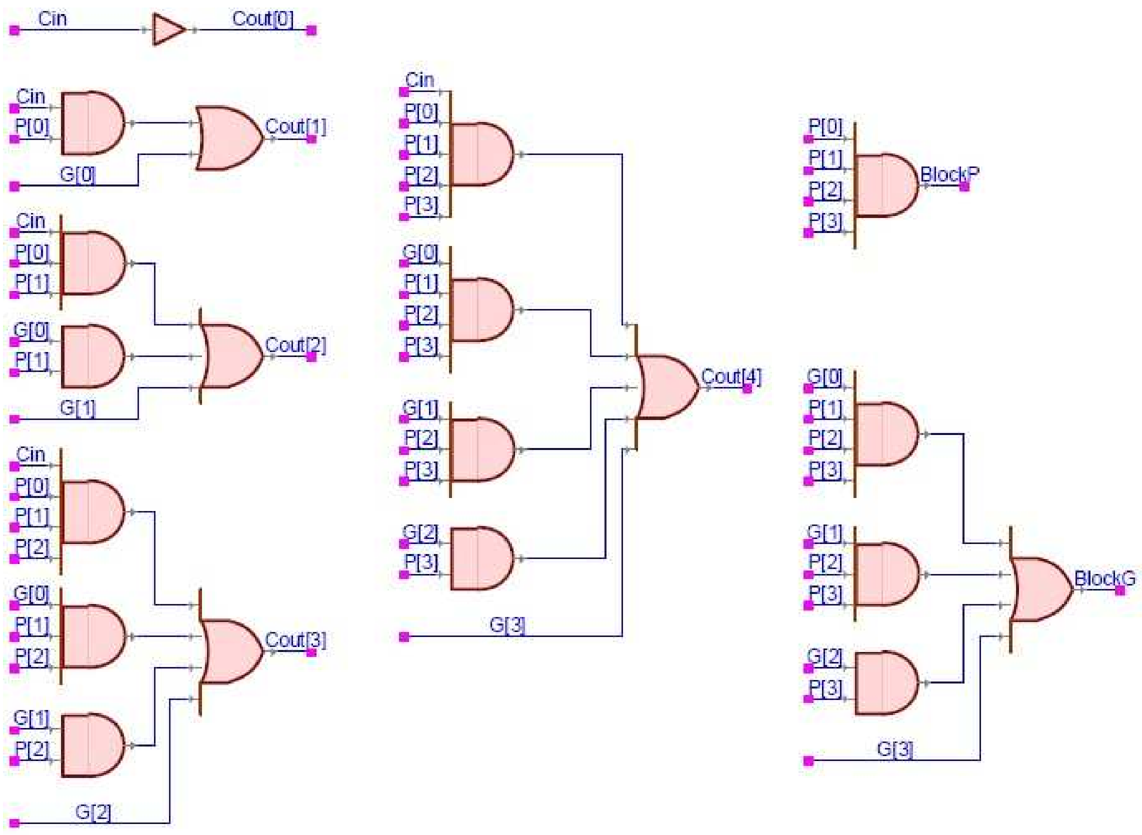
Name	Value	Sti...
Cin	0	<= 0
A	A	For...
A(3)	1	
A(2)	0	
A(1)	1	
A(0)	0	
B	5	For...
B(3)	0	
B(2)	1	
B(1)	0	
B(0)	1	
Cout	0	
Sum	F	

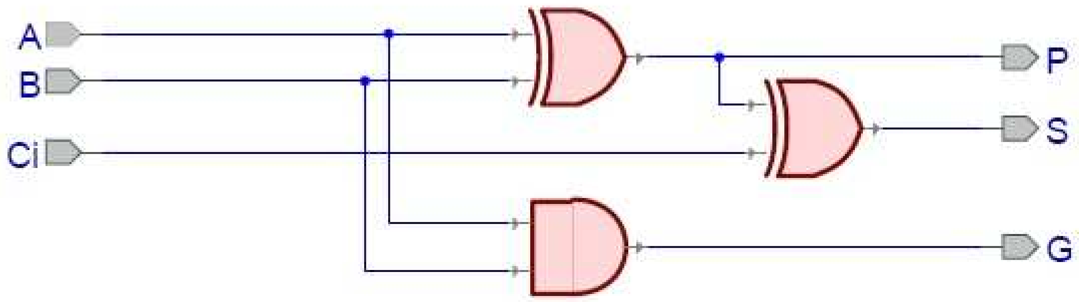
Figure 3 With Gate Delays Enabled.

In the first case (for F + 0 -> F + 1), the Sum is calculated in 9 ns and the Cout is calculated in 7 ns. In the second case (for A + 0 -> A + 5), the Sum is calculated in 2 ns The first case suffers from a much worse delay because of the carry-ripple effect; notice how the carry has to be propagated up the chain of full adders. In contrast, the second case requires no carry's and therefore computes much faster.

In the first case, from F + 0 → F + 1, the Sum is calculated in 9 ns and the Cout is calculated in 7 ns. For the second case, from A + 0 → A + 5, the Sum is calculated in 2 ns and Cout in 0 ns (doesn't change). The difference is that the first case suffers from a ripple-carry effect, where the first carry has to propagate all the way to the last bit. The second case does not have a carry and finishes much faster.

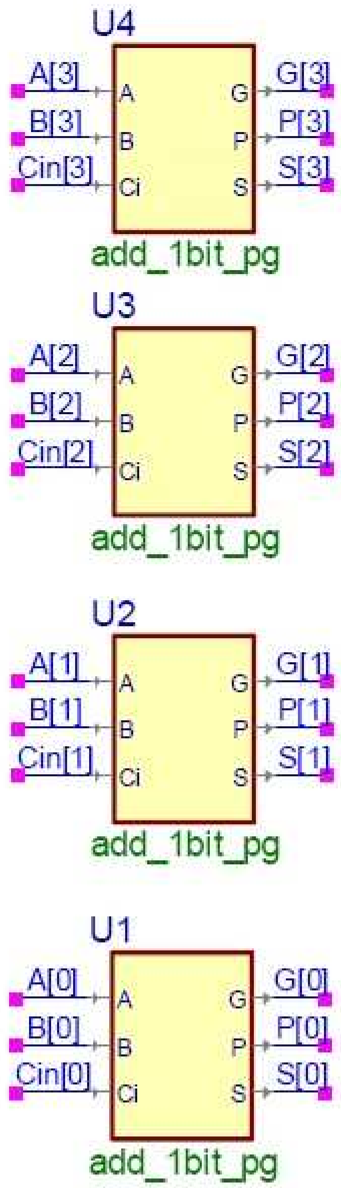
2. **Repeat the previous problem but now construct a 4-bit carry-lookahead instead. Use the same full-adder module as the previous problem. Repeat the two simulations. How much faster is the carry-lookahead adder in both cases? Explain the differences with the result of the previous problem. How do your circuits from this problem and the previous one compare in the total number of gates they use (remember to consider gates in all sub-blocks)?**



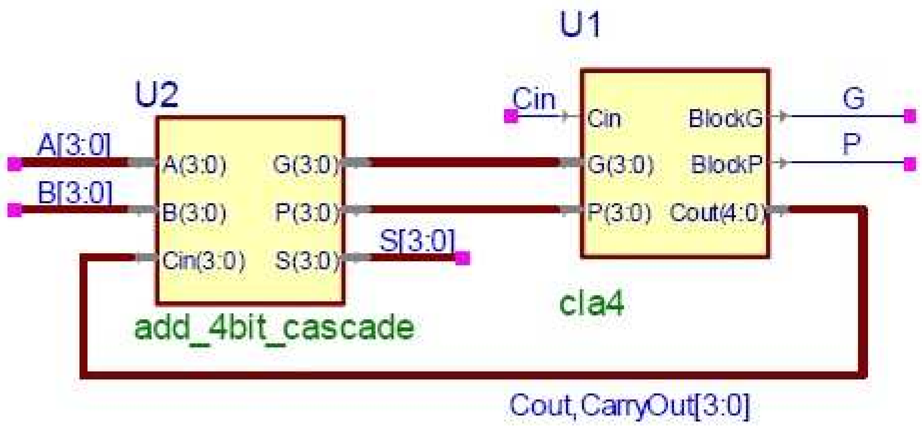


NOTE: This solution uses a 5-input OR-gate. Since this is not present in the lib370 package, we accepted cascading gates to make this 5-input work.

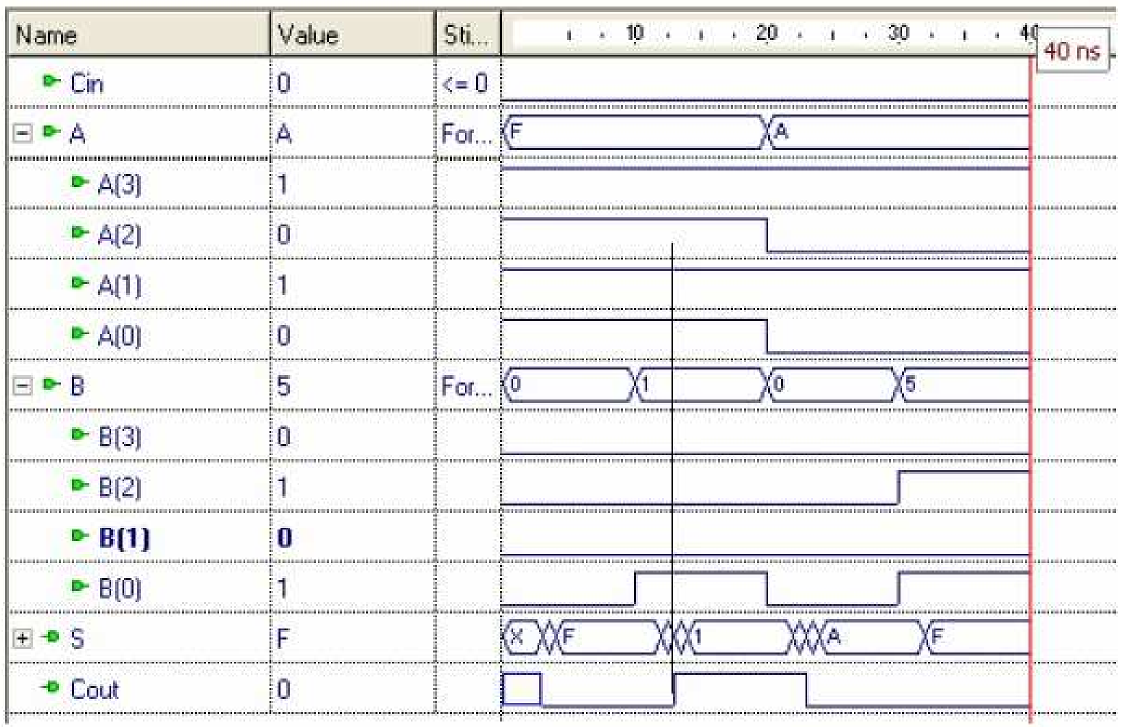
Now we take the one-bit full adders and cascade them:



Finally we hook the circuit together:



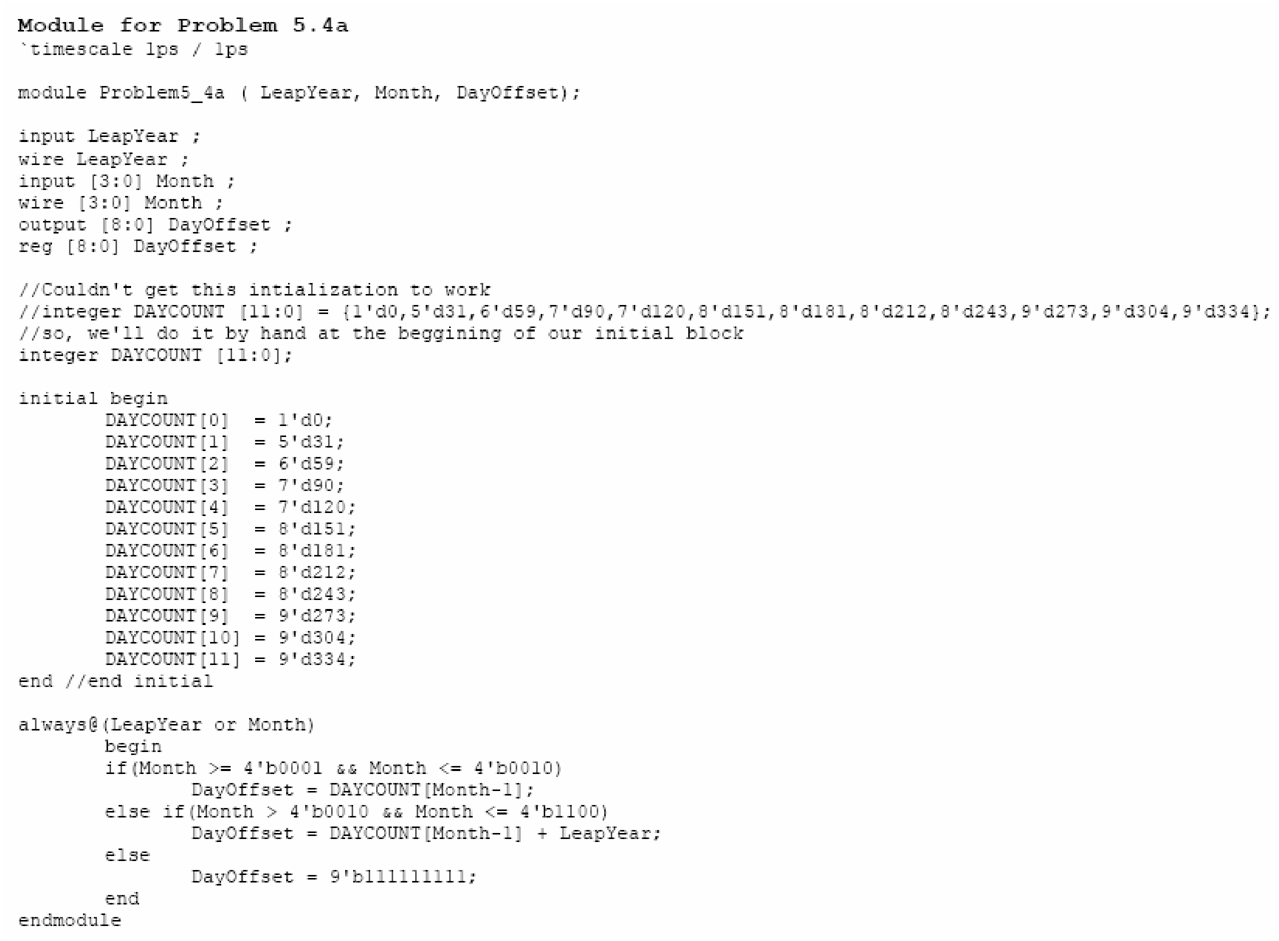
Timing waveforms:



Name	Value	Sti...
Cin	0	<= 0
A	A	For...
A(3)	1	
A(2)	0	
A(1)	1	
A(0)	0	
B	5	For...
B(3)	0	
B(2)	1	
B(1)	**0**	
B(0)	1	
S	F	
Cout	0	

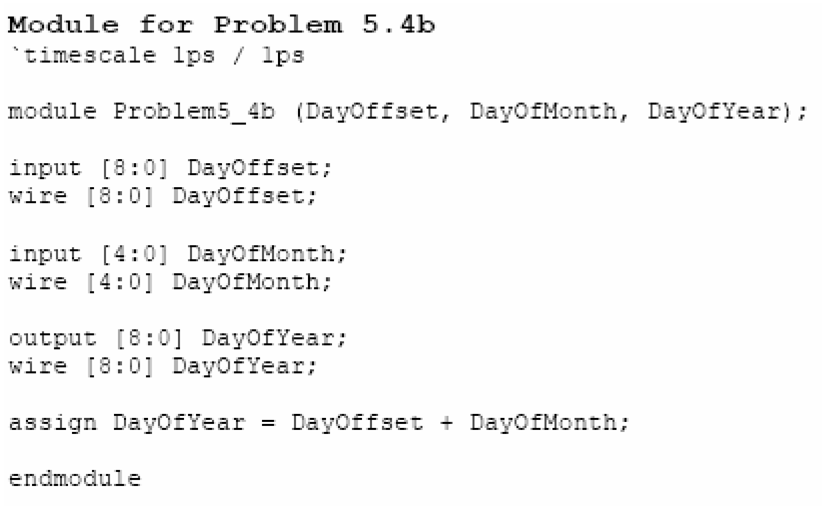
In the first case, where (F + 0 → F + 1), the sum is calculated in 4 ns and Cout is calculated in 3 ns. The second case (A +0 → A + 5) calculates sum in 2 ns. Again, there is no carry in the second case. The carry-lookahead adder is much faster in the first case because we do not have to wait for the carry-propagation delay. Instead we can use the P and G functions to calculate carries in parallel. Note that the performance time is exactly the same in the second case because this does not require carries! Problem one required 48 gates, while problem two requires approximately 64 gates.

3. CLD-II, Chapter 5, problem 5.4, parts a and b (use a 9-bit binary representation for the output).



Module for Problem 5.4a

```
`timescale 1ps / 1ps

module Problem5_4a ( LeapYear, Month, DayOffset);

input LeapYear ;
wire LeapYear ;
input [3:0] Month ;
wire [3:0] Month ;
output [8:0] DayOffset ;
reg [8:0] DayOffset ;

//Couldn't get this intialization to work
//integer DAYCOUNT [11:0] = {1'd0,5'd31,6'd59,7'd90,7'd120,8'd151,8'd181,8'd212,8'd243,9'd273,9'd304,9'd334};
//so, we'll do it by hand at the beggining of our initial block
integer DAYCOUNT [11:0];

initial begin
        DAYCOUNT[0]  = 1'd0;
        DAYCOUNT[1]  = 5'd31;
        DAYCOUNT[2]  = 6'd59;
        DAYCOUNT[3]  = 7'd90;
        DAYCOUNT[4]  = 7'd120;
        DAYCOUNT[5]  = 8'd151;
        DAYCOUNT[6]  = 8'd181;
        DAYCOUNT[7]  = 8'd212;
        DAYCOUNT[8]  = 8'd243;
        DAYCOUNT[9]  = 9'd273;
        DAYCOUNT[10] = 9'd304;
        DAYCOUNT[11] = 9'd334;
end //end initial

always@(LeapYear or Month)
        begin
        if(Month >= 4'b0001 && Month <= 4'b0010)
                DayOffset = DAYCOUNT[Month-1];
        else if(Month > 4'b0010 && Month <= 4'b1100)
                DayOffset = DAYCOUNT[Month-1] + LeapYear;
        else
                DayOffset = 9'b111111111;
        end
endmodule
```



Module for Problem 5.4b

```
`timescale 1ps / 1ps

module Problem5_4b (DayOffset, DayOfMonth, DayOfYear);

input [8:0] DayOffset;
wire [8:0] DayOffset;

input [4:0] DayOfMonth;
wire [4:0] DayOfMonth;

output [8:0] DayOfYear;
wire [8:0] DayOfYear;

assign DayOfYear = DayOffset + DayOfMonth;

endmodule
```

The LEAP_YEAR input is dealt with in part A of the problem, where we conditionally add

4. CLD-II, Chapter 5, problem 5.9, parts a and b.

(a) Draw block diagrams for the 32- and 64-bit adders, showing all interconnections.

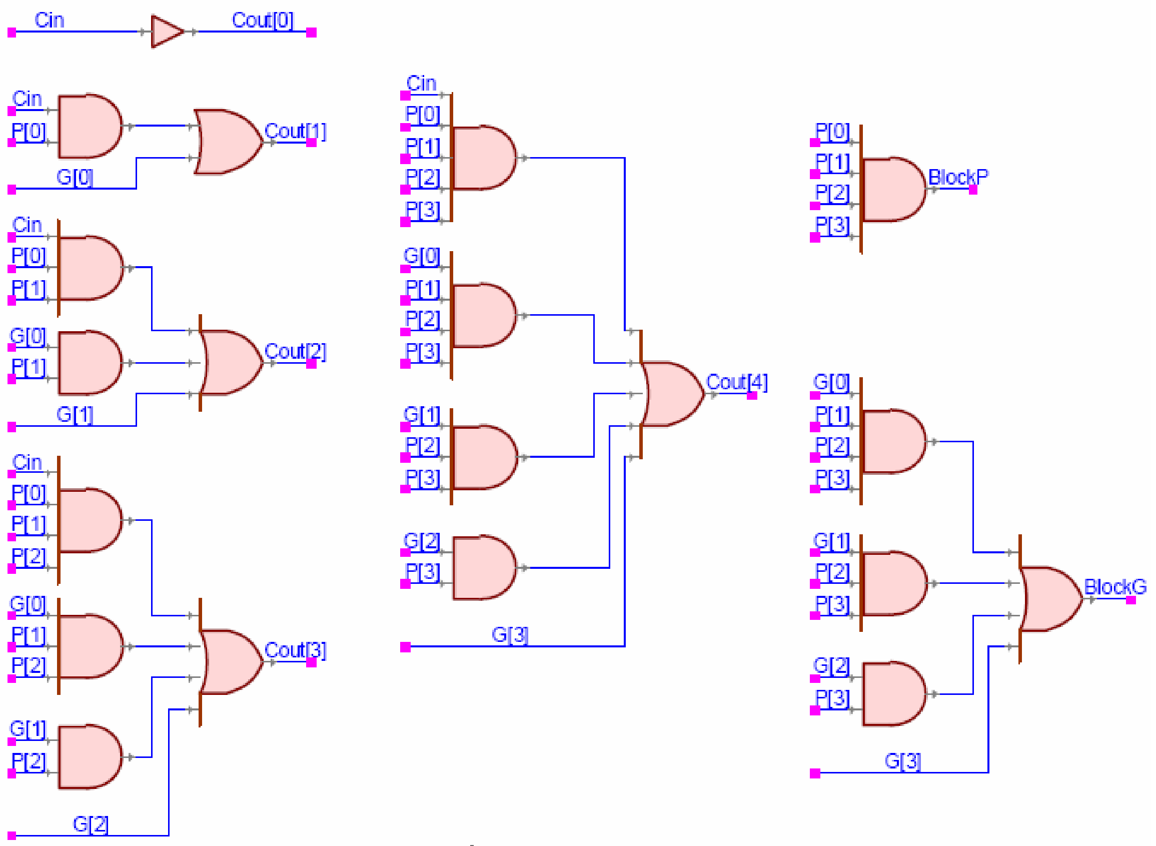


Figure 10 CLA4

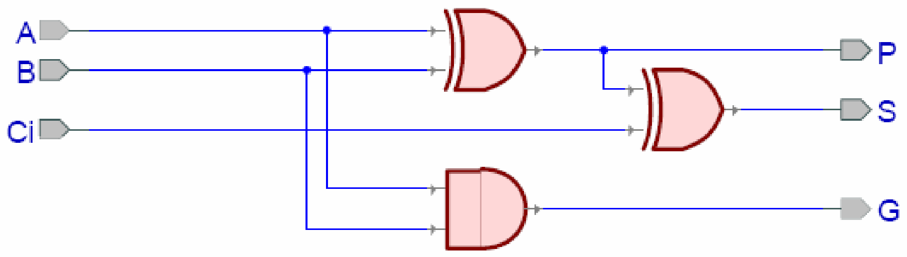


Figure 11 add_1bit_pg

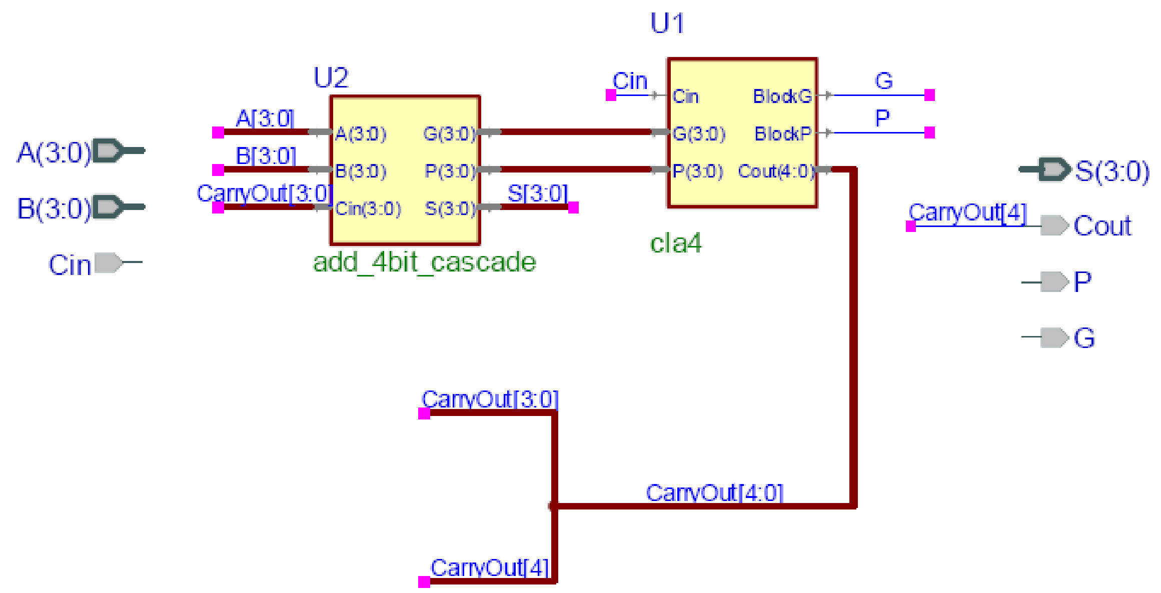


Figure 12 add4bit_cla

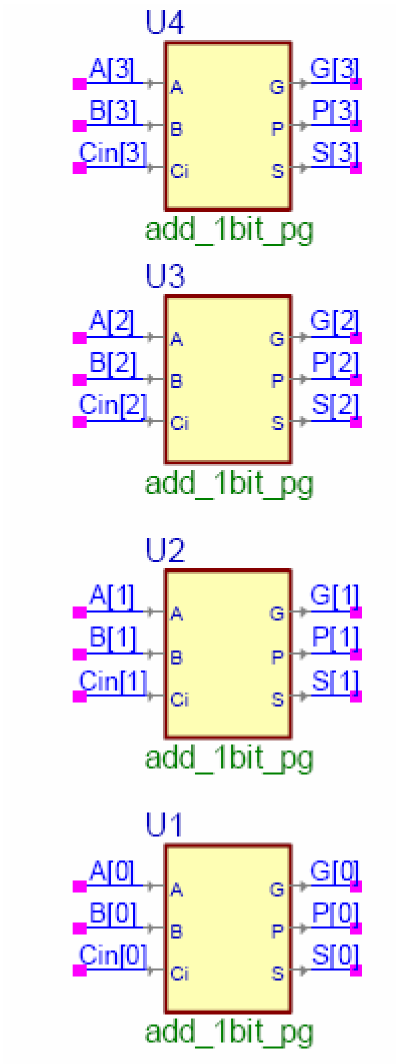


Figure 13 add_4bit_cascade

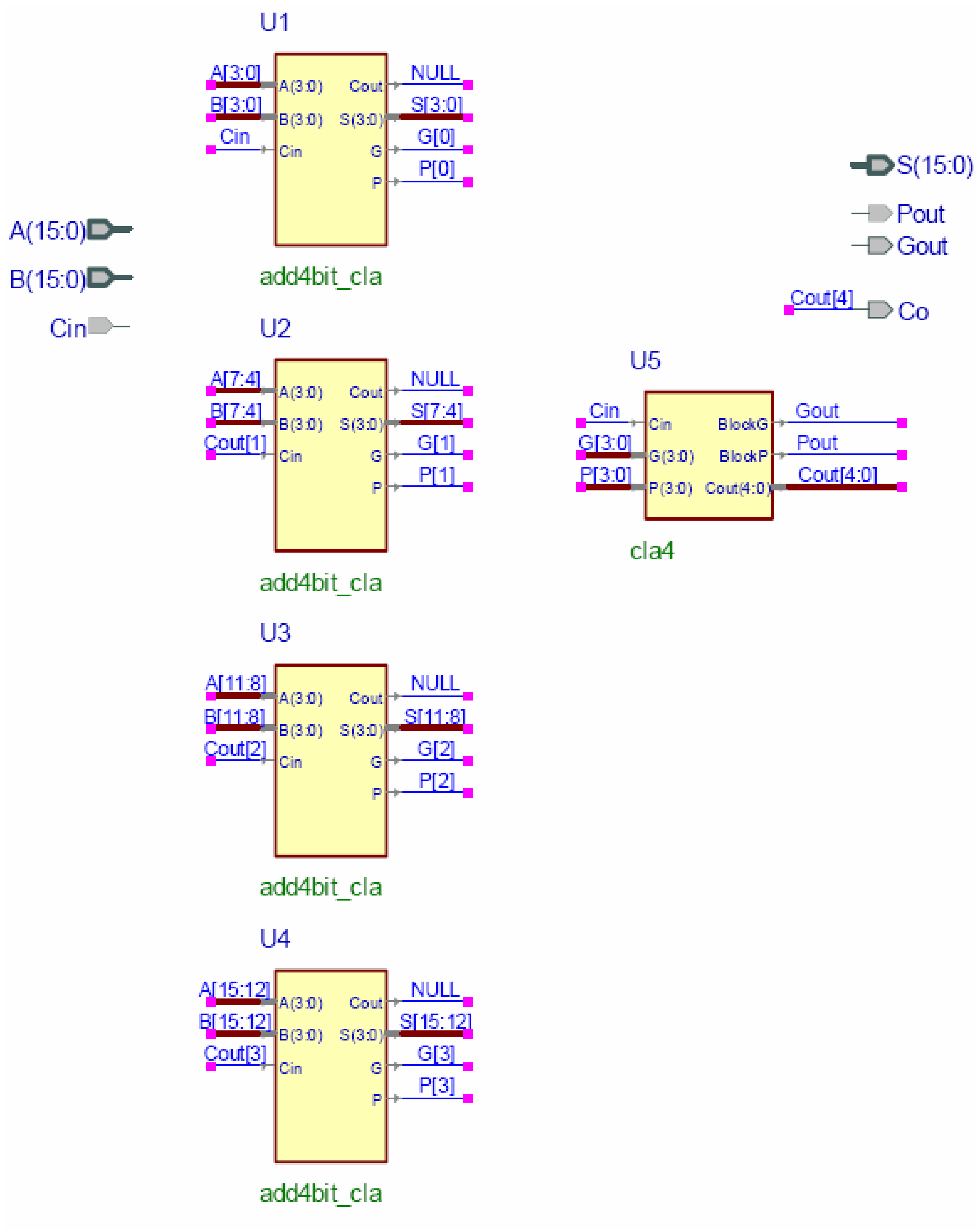


Figure 14 16-bit carry-lookahead adder

(b) Analyze the worst-case gate delays encountered in 32- and 64-bit addition. Use the simple delay models as in Section 5.6.

32-bit adder time analysis (see figure below):

- Worst case gate delay is 11
  - To understand how we ended up with 11, follow the gate delays as we use the 32-bit adder to add 0xFFFFFFFFF + 0x00000001. The first 16-bit adder block has a gate delay of 8 (as we found in previous analysis). We know that P0 and G0 have delays of 3 and 5 respectively and C1 is simply (C0 * P0 + G0), so C1 is dependent on the G0 (5 gate delays) plus an OR gate. This is a total of 6 gate delays for C1. Once we have C1, the second 16-bit adder module computes the sum in 5 more gate delays. This is a total of 11 gate delays. The second 16-bit adder module overlaps its propagate and generate computations with the carry calculations in the external carry-lookahead unit. (see pages 243 and 244 for more information).
  - The Cout value is valid at time 7

64-bit adder time analysis (see figures below):

- Worse case gate delay is 12
- Cout is valid at time 7

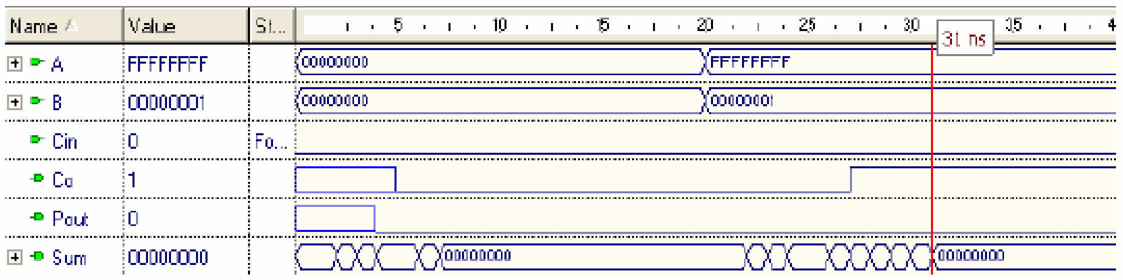


Figure 17 Worst Case Delay in 32-bit CLA

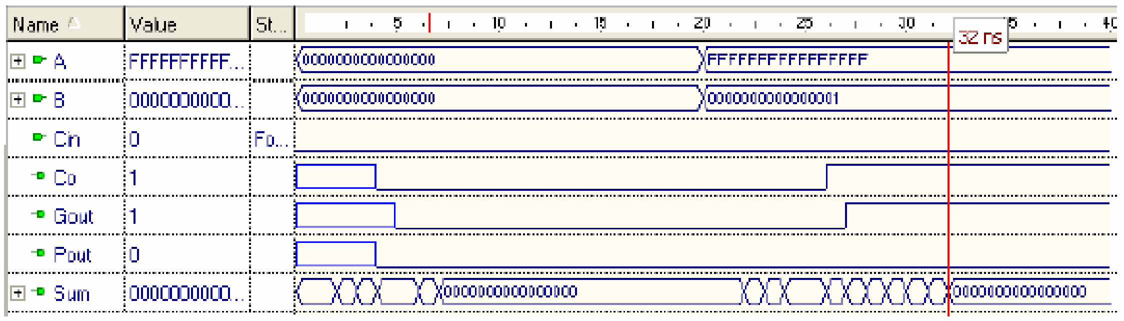


Figure 18 Worst Case Delay in 64-bit CLA

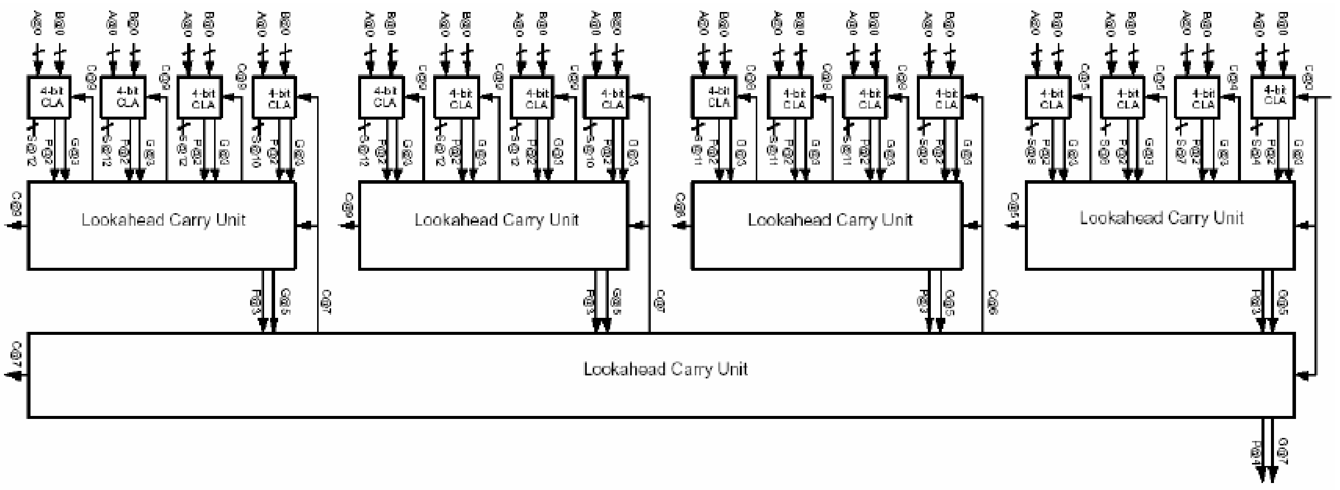


Figure 19 64-bit CLA with Time Analysis*

*diagram from http://www-inst.eecs.berkeley.edu/~cs150/sp00/homeworks/hw9-soln.pdf

5. CLD-II, Chapter 5, problem 5.10.

Consider a 16-bit adder implemented with the carry-select technique described in Section 5.6. The adder is implemented with three 8-bit carry-lookahead adders and eight 2:1 multiplexers. Estimate the gate delay and compare it against a conventional 16-bit ripple adder and a 16-bit carry-lookahead adder.

Adders	Gate Delays	Desc.	Page #
16-bit carry-select (unrealistic implementation)	6	Assume the 8-bit adders are 8-bit carry-lookahead adders then the critical path of our carry-select structure is 4 gate delays plus 2 gate delays for the multiplexer for a total of 6 gate delays. Note that this would require an incredibly high fan-in for the 8-bit cla and, therefore, is impractical.	Pg 243, 245
16-bit carry-select	9	Assume the 8-bit adders are made up of 2 4-bit carry-lookahead adders then the critical path of our carry-select structure is 7 gate delays plus 2 gate delays for the multiplexer for a total of 9 gate delays.	
16-bit ripple-carry	32	See previous problems in this homework.	Pg 240-241
16-bit carry-lookahead	8	See previous problems in this homework.	Pg 243-244

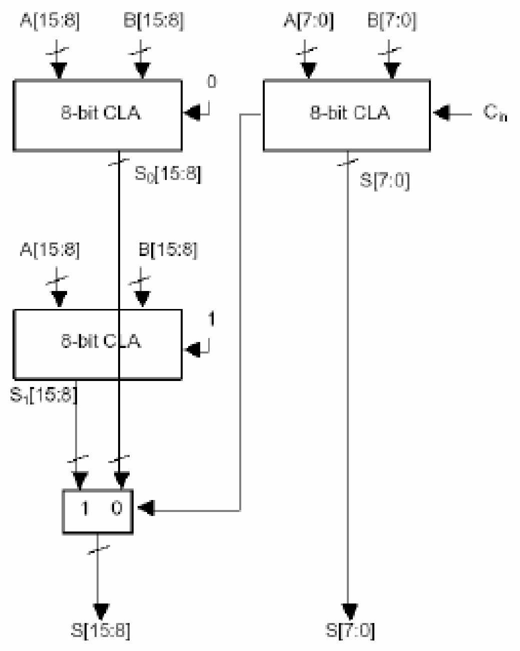


Figure 20 16-bit Carry-Select Adder*

*diagram from http://www-inst.eecs.berkeley.edu/~cs150/sp00/homeworks/hw9-soln.pdf