Carry Look-ahead Adder

A carry look-ahead adder reduces the propagation delay by introducing more complex hardware. In this design, the ripple carry design is suitably transformed such that the carry logic over fixed groups of bits of the adder is reduced to two-level logic.

The addition of two 1-digit inputs A and B is said to generate if the addition will always carry, regardless of whether there is an input-carry (equivalently, regardless of whether any less significant digits in the sum carry). In the case of binary addition, A+B generates if and only if both A and B are 1. If we write G(A, B) to represent the binary predicate that is true if and only if A+B generates, we have

$$G(A,B)=A.B$$

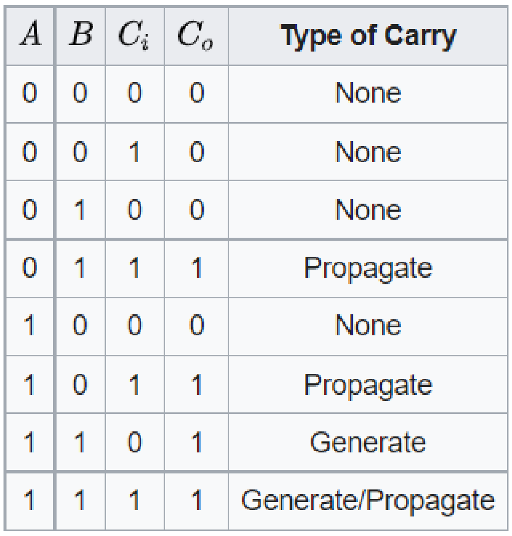
The addition of two 1-digit inputs A and B is said to propagate if the addition will carry whenever there is an input carry (equivalently, when the next less significant digit in the sum carries). In the case of binary addition, A+B propagates if and only if at least one of A or B is 1. If P(A,B) is written to represent the binary predicate that is true if and only if A+B propagates, one has

$$P(A,B)=A+B$$

Sometimes a slightly different definition of propagate is used. By this definition A + B is said to propagate if the addition will carry whenever there is an input carry, but will not carry if there is no input carry. Due to the way generate and propagate bits are used by the carry-lookahead logic, it doesn't matter which definition is used. In the case of binary addition, this definition is expressed by

$$P'(A,B) = A \oplus B$$

where $A \oplus B$ is an exclusive or (i.e., an *xor*).



A	B	C_i	C_o	Type of Carry
0	0	0	0	None
0	0	1	0	None
0	1	0	0	None
0	1	1	1	Propagate
1	0	0	0	None
1	0	1	1	Propagate
1	1	0	1	Generate
1	1	1	1	Generate/Propagate

Table showing *when* carries are propagated or generated.

<u>IMPLEMENTATION</u>

<u>Carry Dependency:</u>

$$C_{i+1} = G_i + (P_i \,.\, C_i)$$

$C_1 = G_0 + P_0.C_0$, $C_5 = G_4 + P_4.C_4$,

$C_2 = G_1 + P_1.C_1$, $C_6 = G_5 + P_5.C_5$,

$C_3 = G_2 + P_2.C_2$, $C_7 = G_6 + P_6.C_6$,

$C_4 = G_3 + P_3.C_3$, $C_8 = G_7 + P_7.C_7$,

$C_1 = G_0 + P_0.C_0$

$C_2 = G_1 + G_0.P_1 + C_0.P_0.P_1$

$C_3 = G_2 + G_1.P_2 + G_0.P_1.P_2 + P_0.C_0.P_1.P_2$

$C_4 = G_3 + G_2.P_3 + G_1.P_2.P_3 + G_0.P_1.P_2.P_3 + C_0.P_0.P_1.P_2.P_3$

$C_5 = G_4 + G_3.P_4 + G_2.P_3.P_4 + G_1.P_2.P_3.P_4 + G_0.P_1.P_2.P_3.P_4 + C_0.P_0.P_1.P_2.P_3.P_4$

$C_6 = G_5 + G_4.P_5 + G_3.P_4.P_5 + G_2.P_3.P_4.P_5 + G_1.P_2.P_3.P_4.P_5 + G_0.P_1.P_2.P_3.P_4.P_5 + C_0.P_0.P_1.P_2.P_3.P_4.P_5$

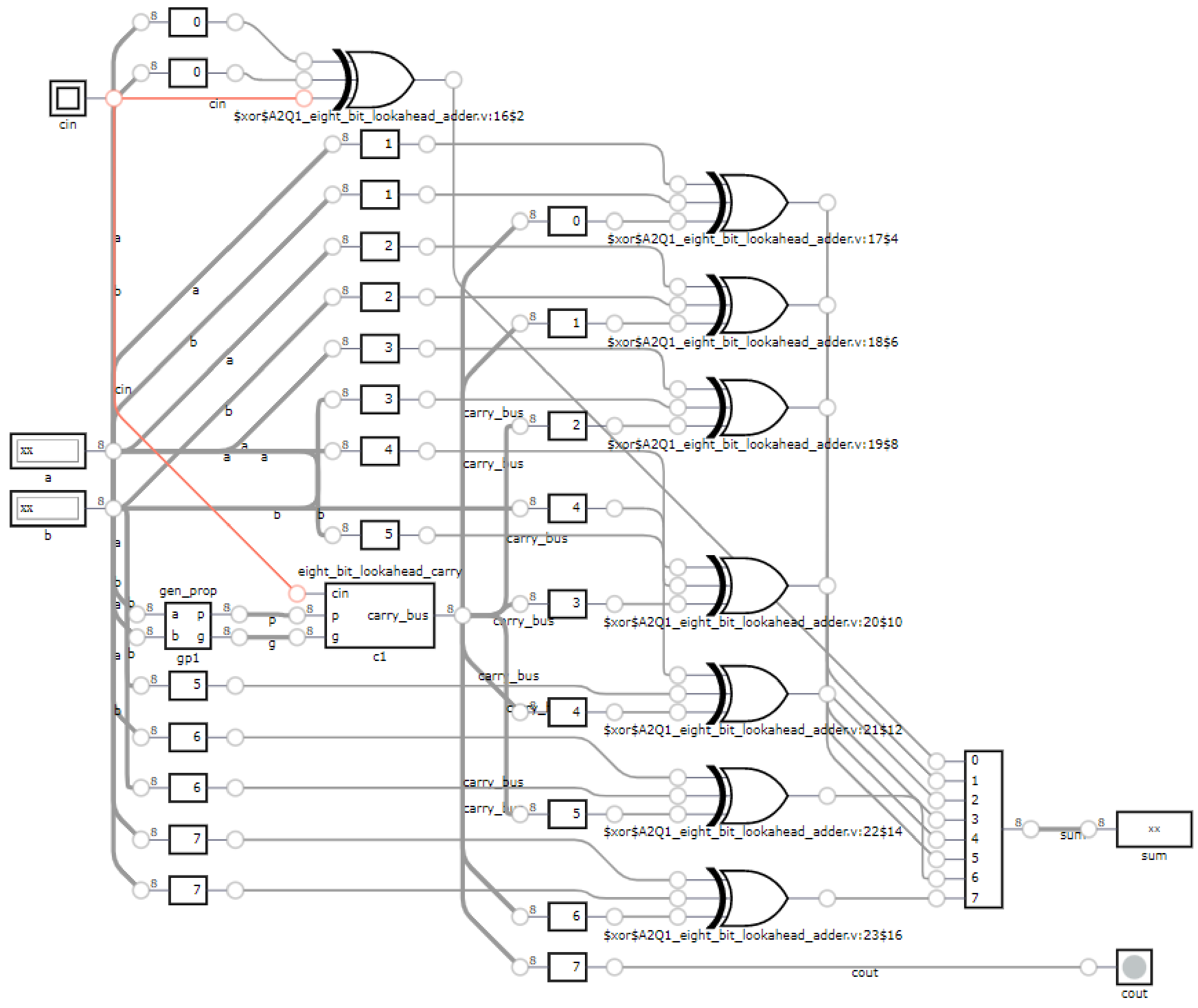
$C_7 = G_6 + G_5.P_6 + G_4.P_5.P_6 + G_3.P_4.P_5.P_6 + G_2.P_3.P_4.P_5.P_6 + G_1.P_2.P_3.P_4.P_5.P_6 + G_0.P_1.P_2.P_3.P_4.P_5.P_6 + C_0.P_0.P_1.P_2.P_3.P_4.P_5.P_6$

$C_8 = G_7 + G_6.P_7 + G_5.P_6.P_7 + G_4.P_5.P_6.P_7 + G_3.P_4.P_5.P_6.P_7 + G_2.P_3.P_4.P_5.P_6.P_7 + G_1.P_2.P_3.P_4.P_5.P_6.P_7 + G_0.P_1.P_2.P_3.P_4.P_5.P_6.P_7 + C_0.P_0.P_1.P_2.P_3.P_4.P_5.P_6.P_7$

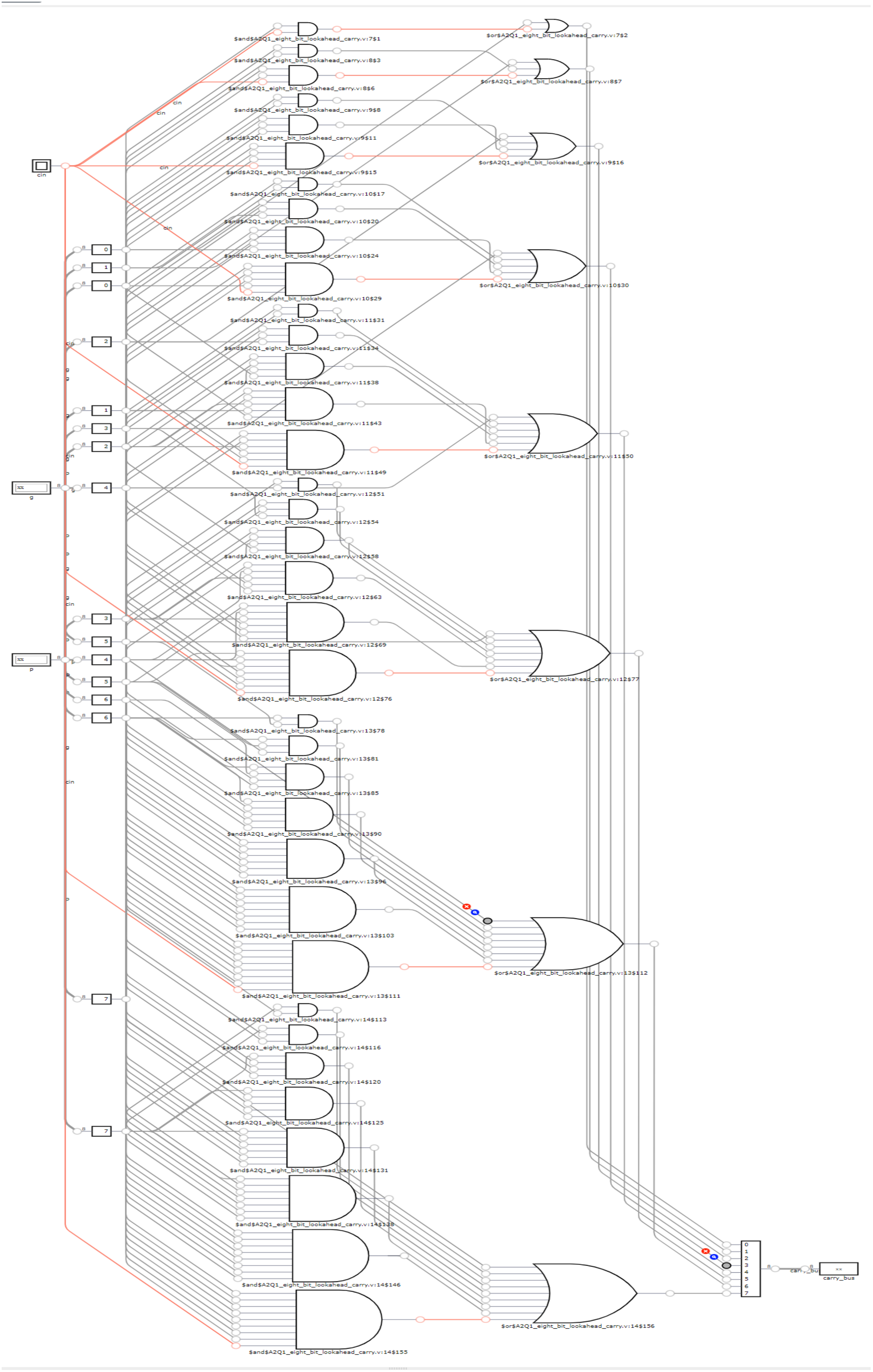
Summation Function:

$$S_i = XOR(A_i , B_i , C_i)$$

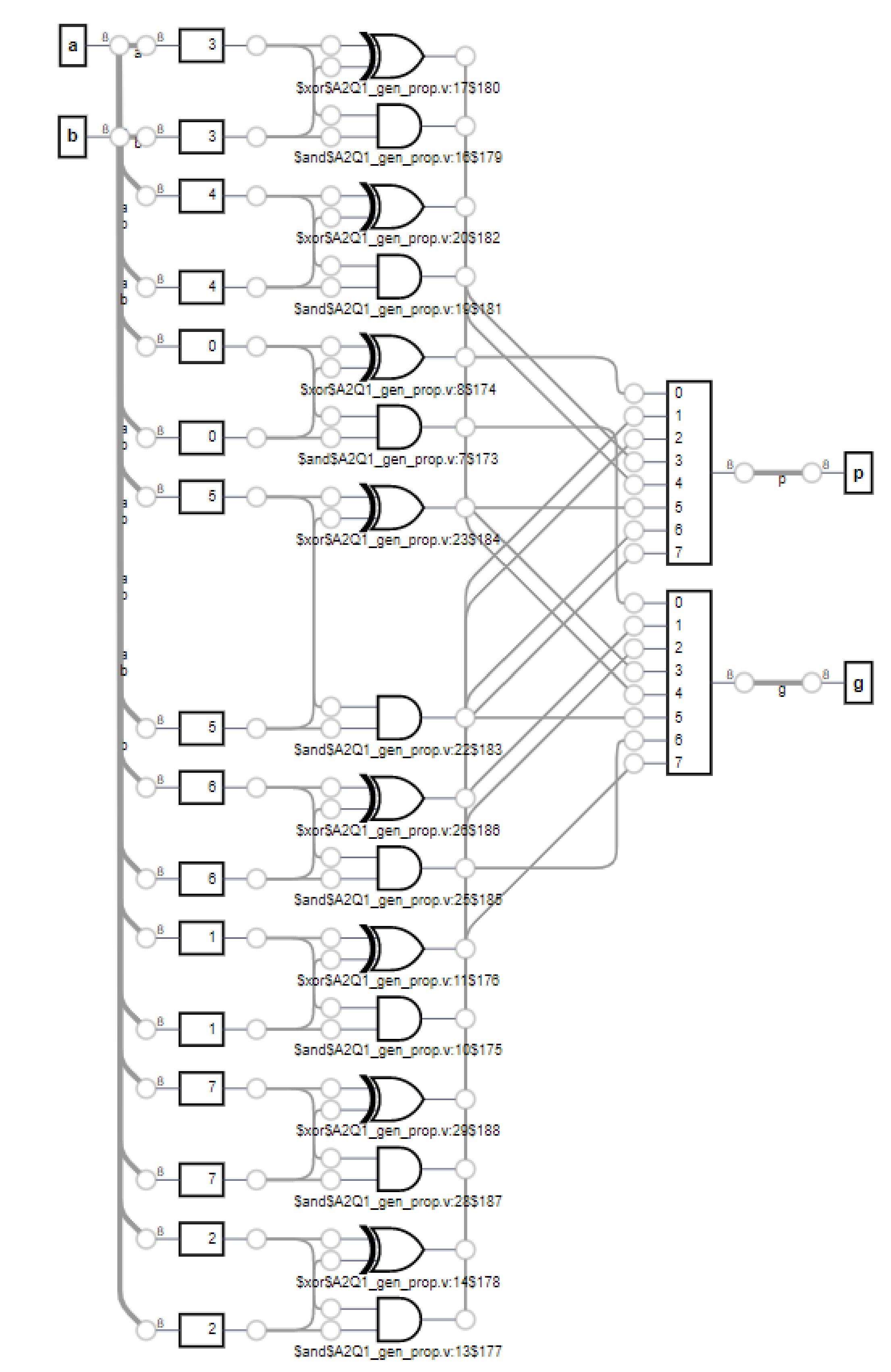
Citcuit Diagrams:



cin
a
b
xorA2Q1_eight_bit_lookahead_adder.v:16$2
xorA2Q1_eight_bit_lookahead_adder.v:17$4
xorA2Q1_eight_bit_lookahead_adder.v:18$6
xorA2Q1_eight_bit_lookahead_adder.v:19$8
xorA2Q1_eight_bit_lookahead_adder.v:20$10
xorA2Q1_eight_bit_lookahead_adder.v:21$12
xorA2Q1_eight_bit_lookahead_adder.v:22$14
xorA2Q1_eight_bit_lookahead_adder.v:23$16
gen_prop
gp1
eight_bit_lookahead_carry
c1
carry_bus
p
g
sum
cout


andA2Q1_eight_bit_lookahead_carry.v:7$1
orA2Q1_eight_bit_lookahead_carry.v:7$2
andA2Q1_eight_bit_lookahead_carry.v:8$3
orA2Q1_eight_bit_lookahead_carry.v:8$7
andA2Q1_eight_bit_lookahead_carry.v:8$6
andA2Q1_eight_bit_lookahead_carry.v:9$8
andA2Q1_eight_bit_lookahead_carry.v:9$11
orA2Q1_eight_bit_lookahead_carry.v:9$16
cin
andA2Q1_eight_bit_lookahead_carry.v:9$15
andA2Q1_eight_bit_lookahead_carry.v:10$17
andA2Q1_eight_bit_lookahead_carry.v:10$20
andA2Q1_eight_bit_lookahead_carry.v:10$24
orA2Q1_eight_bit_lookahead_carry.v:10$30
andA2Q1_eight_bit_lookahead_carry.v:10$29
andA2Q1_eight_bit_lookahead_carry.v:11$31
andA2Q1_eight_bit_lookahead_carry.v:11$34
andA2Q1_eight_bit_lookahead_carry.v:11$38
andA2Q1_eight_bit_lookahead_carry.v:11$43
orA2Q1_eight_bit_lookahead_carry.v:11$50
andA2Q1_eight_bit_lookahead_carry.v:11$49
g
andA2Q1_eight_bit_lookahead_carry.v:12$51
andA2Q1_eight_bit_lookahead_carry.v:12$54
andA2Q1_eight_bit_lookahead_carry.v:12$58
andA2Q1_eight_bit_lookahead_carry.v:12$63
andA2Q1_eight_bit_lookahead_carry.v:12$69
p
orA2Q1_eight_bit_lookahead_carry.v:12$77
andA2Q1_eight_bit_lookahead_carry.v:12$76
andA2Q1_eight_bit_lookahead_carry.v:13$78
andA2Q1_eight_bit_lookahead_carry.v:13$81
andA2Q1_eight_bit_lookahead_carry.v:13$85
andA2Q1_eight_bit_lookahead_carry.v:13$90
andA2Q1_eight_bit_lookahead_carry.v:13$96
andA2Q1_eight_bit_lookahead_carry.v:13$103
orA2Q1_eight_bit_lookahead_carry.v:13$112
andA2Q1_eight_bit_lookahead_carry.v:13$111
andA2Q1_eight_bit_lookahead_carry.v:14$113
andA2Q1_eight_bit_lookahead_carry.v:14$116
andA2Q1_eight_bit_lookahead_carry.v:14$120
andA2Q1_eight_bit_lookahead_carry.v:14$125
andA2Q1_eight_bit_lookahead_carry.v:14$131
andA2Q1_eight_bit_lookahead_carry.v:14$138
andA2Q1_eight_bit_lookahead_carry.v:14$146
orA2Q1_eight_bit_lookahead_carry.v:14$156
andA2Q1_eight_bit_lookahead_carry.v:14$155
carry_bus

gen_prop gp1



a
b
3
4
0
5
6
1
7
2
xorA2Q1_gen_prop.v:17$180
andA2Q1_gen_prop.v:16$179
xorA2Q1_gen_prop.v:20$182
andA2Q1_gen_prop.v:19$181
xorA2Q1_gen_prop.v:8$174
andA2Q1_gen_prop.v:7$173
xorA2Q1_gen_prop.v:23$184
andA2Q1_gen_prop.v:22$183
xorA2Q1_gen_prop.v:26$186
andA2Q1_gen_prop.v:25$185
xorA2Q1_gen_prop.v:11$176
andA2Q1_gen_prop.v:10$175
xorA2Q1_gen_prop.v:29$188
andA2Q1_gen_prop.v:28$187
xorA2Q1_gen_prop.v:14$178
andA2Q1_gen_prop.v:13$177
p
g

Advantages and Disadvantages of Carry Look-Ahead Adder :
Advantages –

- The propagation delay is reduced.
- It provides the fastest addition logic.

Disadvantages –

- The Carry Look-ahead adder circuit gets complicated as the number of variables increase.
- The circuit is costlier as it involves more number of hardware.