컴퓨터 구조 & 실험

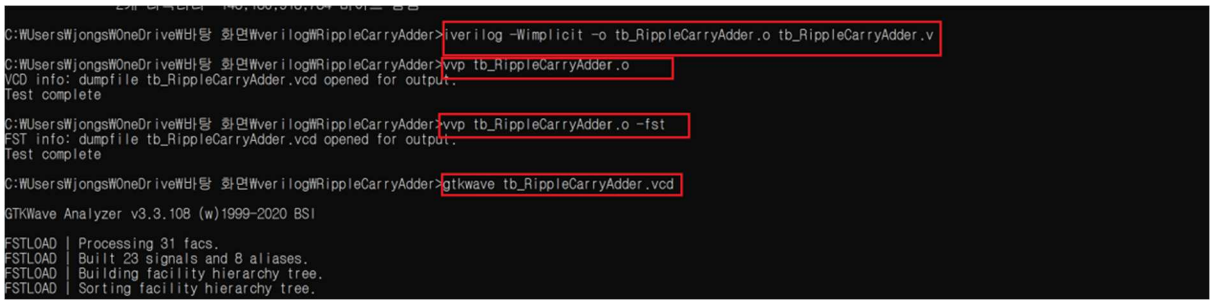
Verilog1 과제

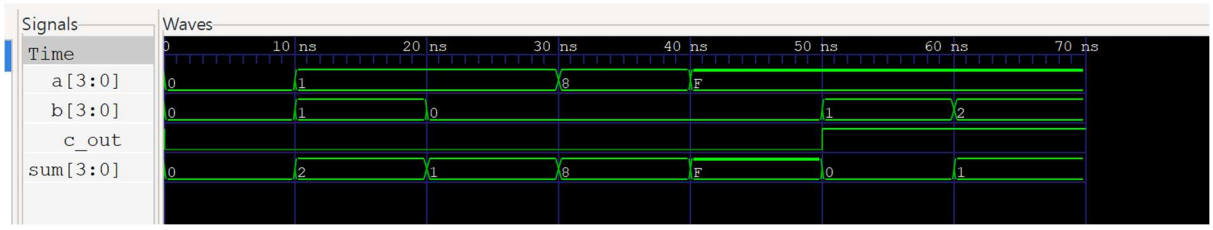
수업 명 : 컴퓨터구조실험
과제 이름 : Verilog1
담당 교수님 : 이성원 교수님
학 번 : 2019202005
이 름 : 남종식

과제 소개

이번 Verilog과제는 1bit Full Adder 4개를 이어서 만든 4bit Ripple Carry Adder를 디자인하는 과제입니다. 4bit Ripple Carry Adder는 Full Adder을 사용하여 각 비트를 더하고 자리올림이 있을 시 이를 처리하여 입력된 4비트 숫자의 합 sum과 올림 수 c_out을 출력합니다. 1bit Full Adder 4개를 이어서 만들 때 처음 c_in의 값으로 0으로 받습니다. 그리고 첫번째 모듈에서 나온 c_out의 output값을 그 다음 c_in의 input값으로 넣어줍니다. 세번째 네번째 모듈에서도 똑같이 이전에 모듈에서의 c_out값을 c_in으로 넣어줍니다. 이때 c_out값을 c_in으로 연결하는 과정에서 wire를 사용합니다. Wire의 역할은 모듈 내에서 신호를 전달하거나 출력하는 데 사용됩니다. 이를 사용하여 모듈 내에서 모듈 내부 또는 모듈 간의 연결을 나타낼 수 있습니다.

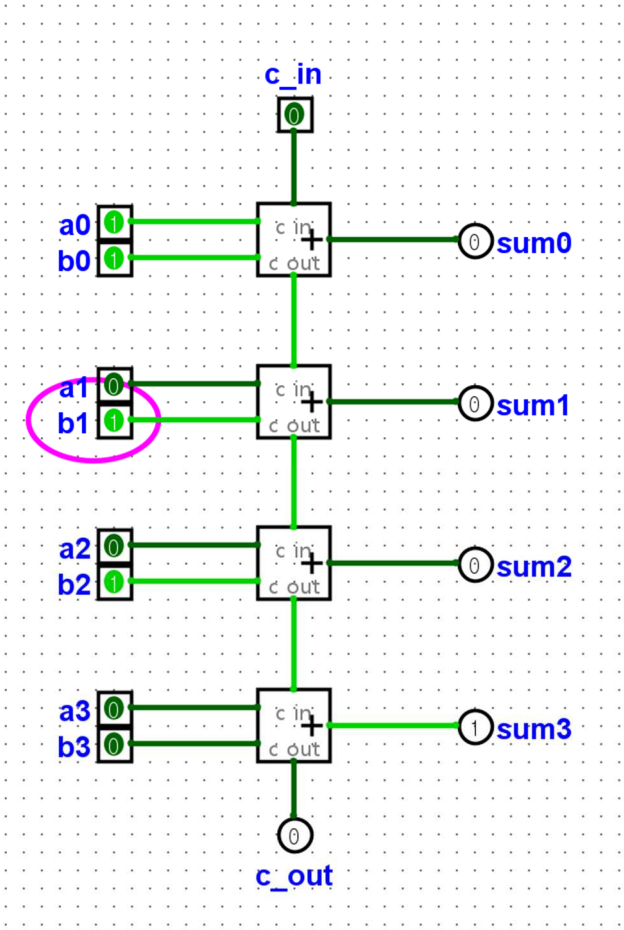
다음은 Icarus Verilog로 컴파일 후에 gtkwave로 파형을 확인한 화면입니다.



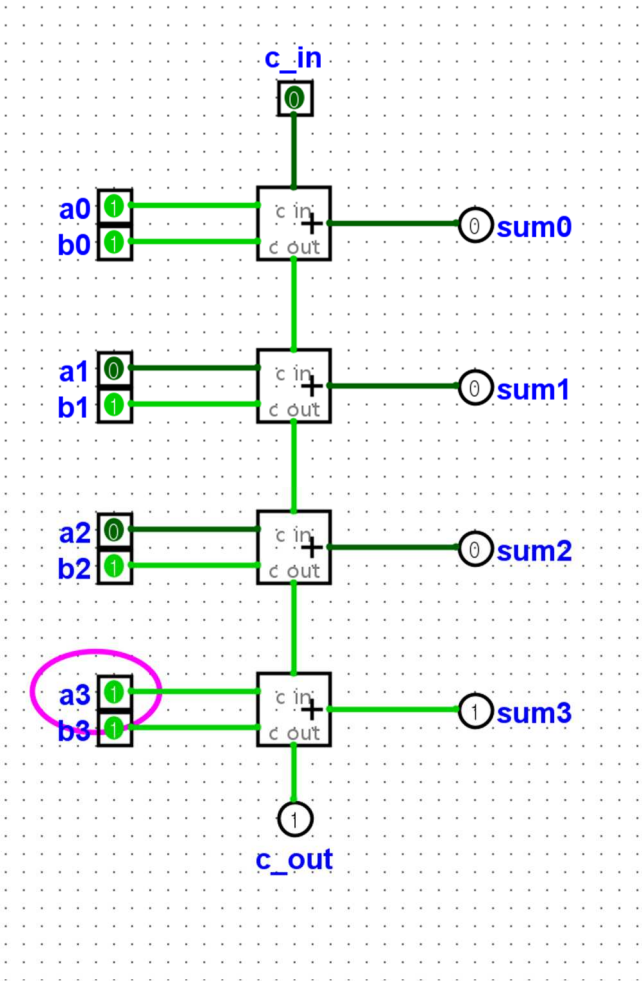


RippleCarryAdder.v파일과 tb_RippleCarryAdder.v파일을 저장한 폴더 경로에서 cmd창을 실행해줍니다. Iverilog -Wimplict -o tb_RippleCarryAdder.o tb_RippleCarryAdder.v을 입력해 tb_RippleCarryAdder.o파일을 생성해줍니다. 다음으로 vvp tb_RippleCarryAdder.o 입력을 통해 tb_RippleCarryAdder.o파일을 실행하고 tb_RippleCarryAdder.o -fst를 통해 vcd파일과 시뮬레이션 파일을 생성합니다. 마지막으로 gtkwave tb_RippleCarryAdder.vcd를 통해 .vcd파일을 열어 시뮬레이션 결과를 확인할 수 있습니다.

입력 값과 동작 결과 확인



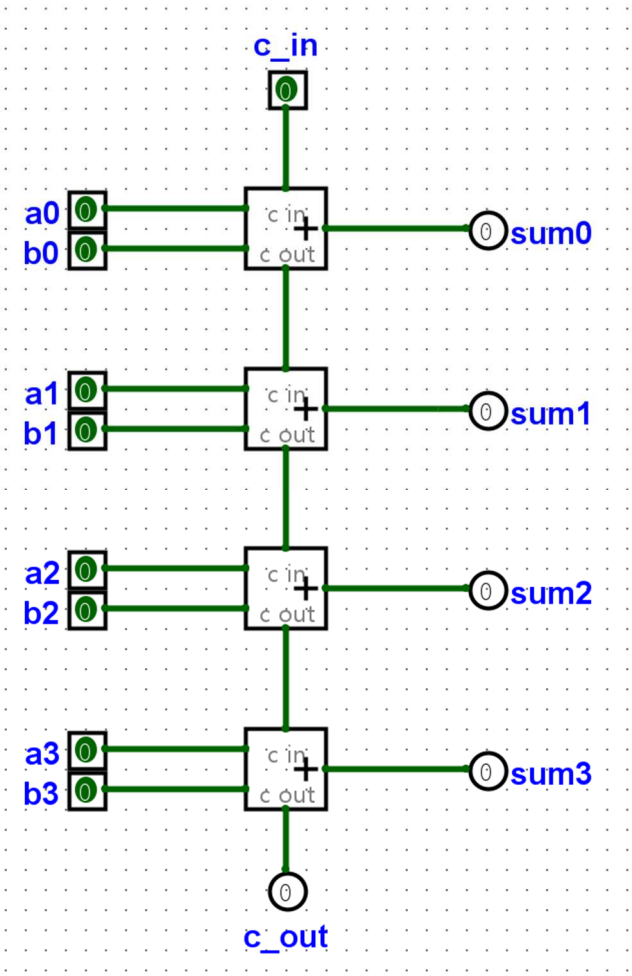
예를 들어 a의 입력 값이 4'b0001이고 b의 입력 값이 4'b0111일 때 두 수의 합은 4'b1000입니다. 이때 sum을 확인해보면 4'b1000이 나온 것을 확인할 수 있습니다. 그럼 올림수 c_out이 있는 상황을 살펴보겠습니다.



위 사진에서 a의 입력 값으로 4'b1001 b의 입력 값으로 4'b1111일 때 두 수의 합은 올림 수가 1이 발생하고 4'b1000입니다. 이때 sum의 값을 확인해보면 4'b1000이 나오고 c_out의 값 1이 출력된 모습을 확인할 수 있습니다.

결과 화면 및 gtkwave와 Logisim_evolution비교

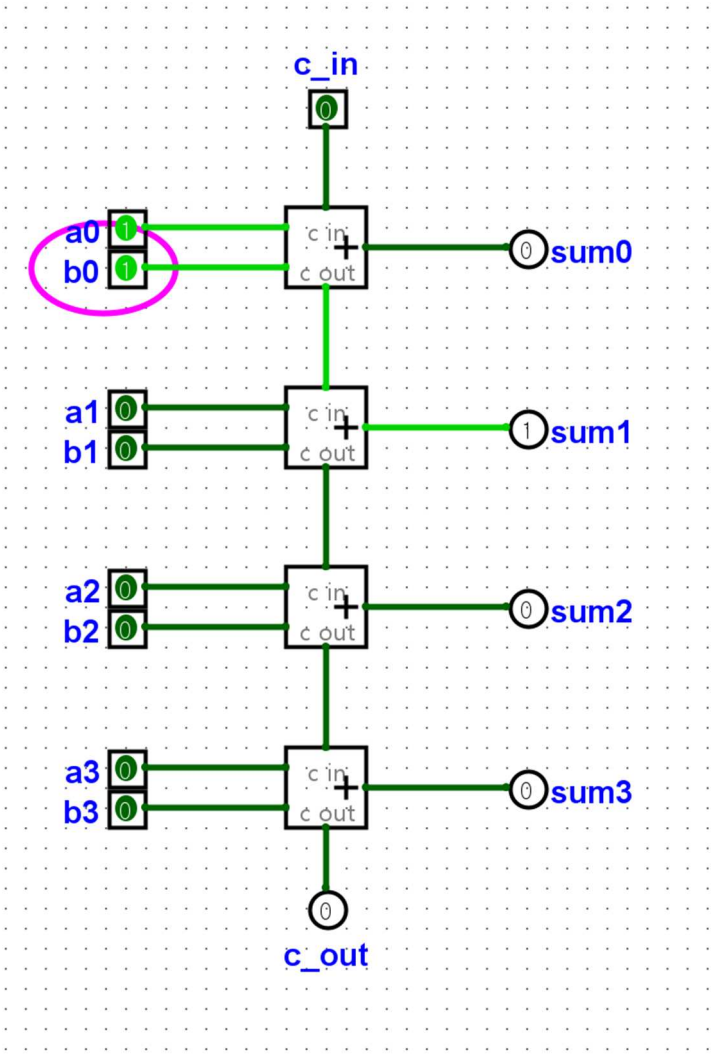
1) a=0 b=0 (a=4'b0000 b=4'b0000)



Ripple Carry Adder의 전체적인 틀은 이렇습니다. input값인 a, b, c_in이 모두 0일 때 ouput값인 c_out, sum의 값이 모두 0이 출력된 것을 확인할 수 있습니다.

2) a=1, b=1 (a=4'b0001 b=4'b0001)

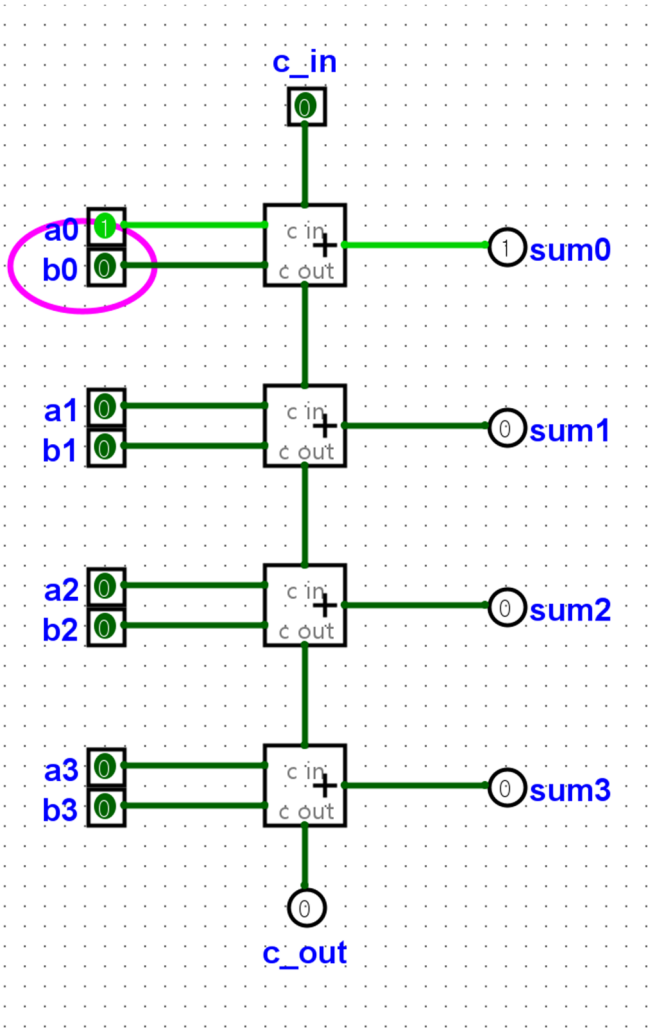
Logisim-evolution과 비교하기 쉽게 data format을 binary로 변경하여 확인했습니다.
c_in값은 1'b0이므로 첫번째 모듈에서의 input값은 a, b만 확인해주면 됩니다.
a, b에 input값으로 4'b0001이 들어왔을 때 sum값이 4'b0010 즉 2가 나오므로 덧셈이 잘 이루어져 출력된 것을 확인할 수 있습니다.



Logisim-evolution에서도 a와 b에 각각 4'b0001을 input값으로 넣어줬을 때 sum값이 4'b0010으로 잘 출력된 것을 확인할 수 있습니다.

3) a=1, b=0 (a=4'b0001 b=4'b0000)

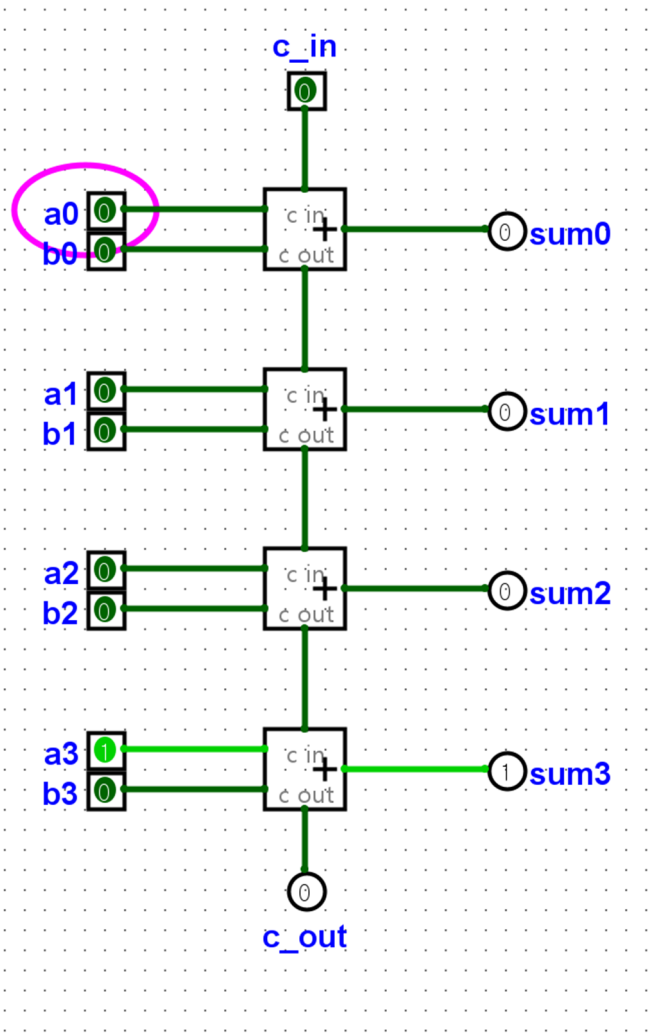
a와 b의 input값으로 각각 4'b0001, 4'b0000을 넣어줬을 때 c_out 올림 수 없이 sum값이 4'b0001로 잘 출력된 모습을 확인할 수 있습니다.



a=4'b0001 b=4'b0000을 input으로 받았을 때 sum값이 4'b0001로 잘 출력된 모습입니다.

4) a=8, b=0 (a=4'b1000 b=4'b0000)

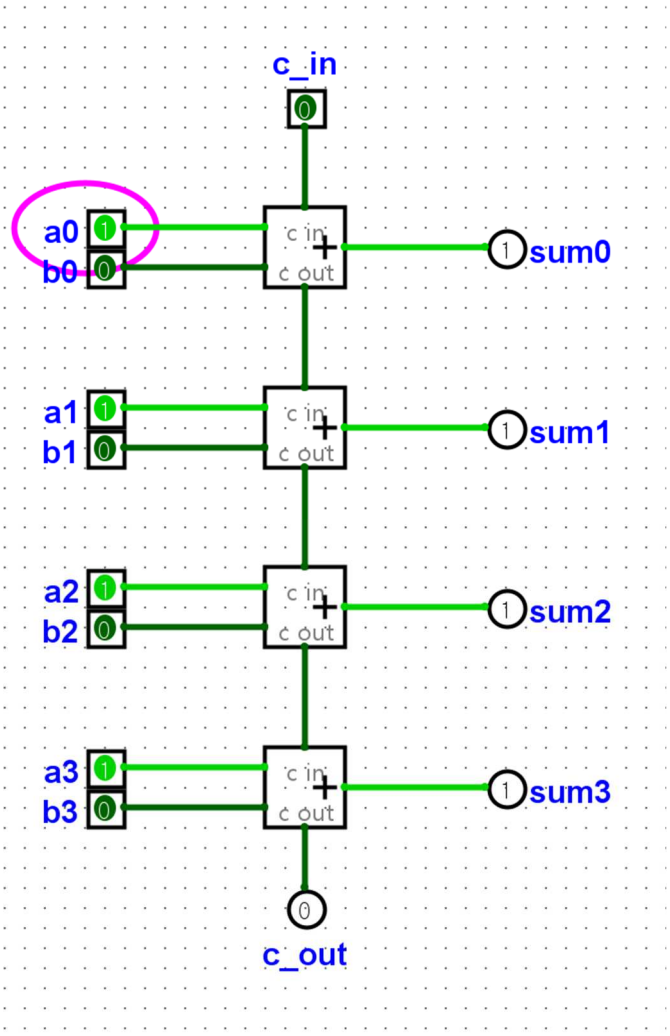
a와 b의 input 값으로 각각 8과 0을 넣어줬을 때 두 수의 합의 결과인 sum값이 4'b1000 즉 10진수 8이 나온 것을 확인할 수 있습니다.



a=4'b1000 b=4'b0000을 input값으로 받았을 때 sum=4'b1000으로 나온 것을 확인할 수 있습니다.

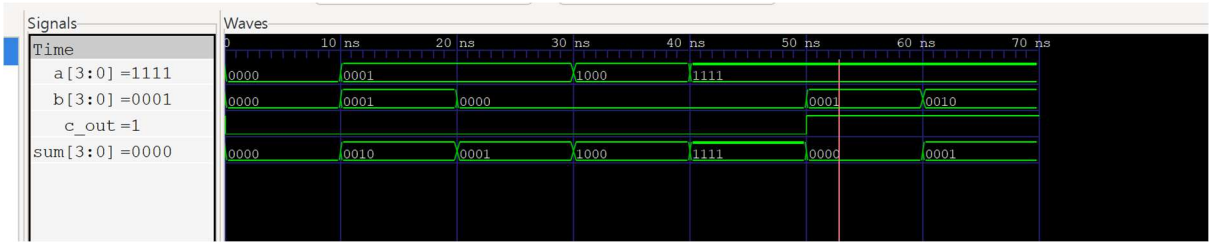
5) a=15 b=0 (a=4'b1111 b=4'b0000)

b의 input값이 4'b0000이므로 a와 b의 값을 더했을 때 sum의 값이 a값 그대로 나온 것을 확인할 수 있습니다.

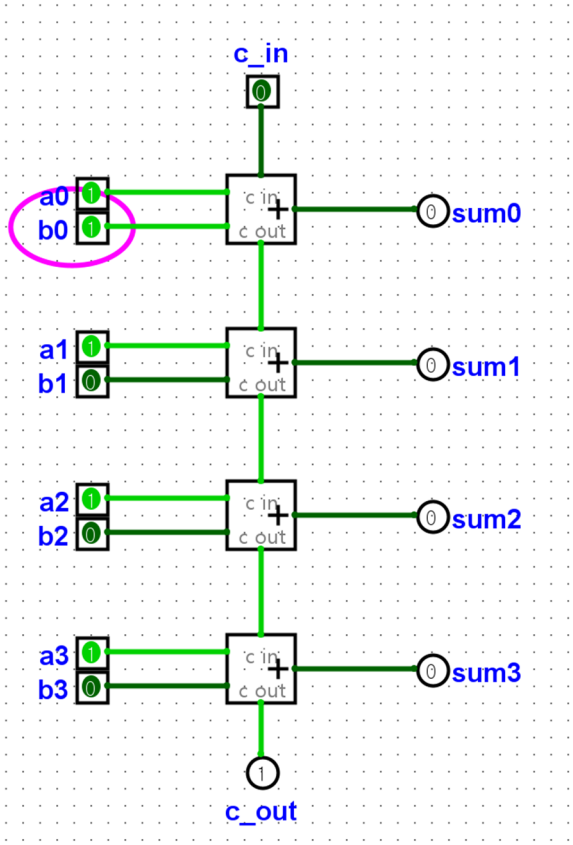


Logisim-evolution에서도 sum의 값이 a의 값 그대로 출력된 것을 확인할 수 있습니다.

6) a=15 b=1 (a=4'b1111 b=4'b0001)



a와 b의 input값으로 각각 15와 1을 넣어준 케이스입니다. 이때 a와 b를 2진수로 나타내어 bit연산을 하면 올림 수가 발생해 c_out의 값이 1이 나오고 sum의 값은 0이 나온 것을 확인할 수 있습니다.

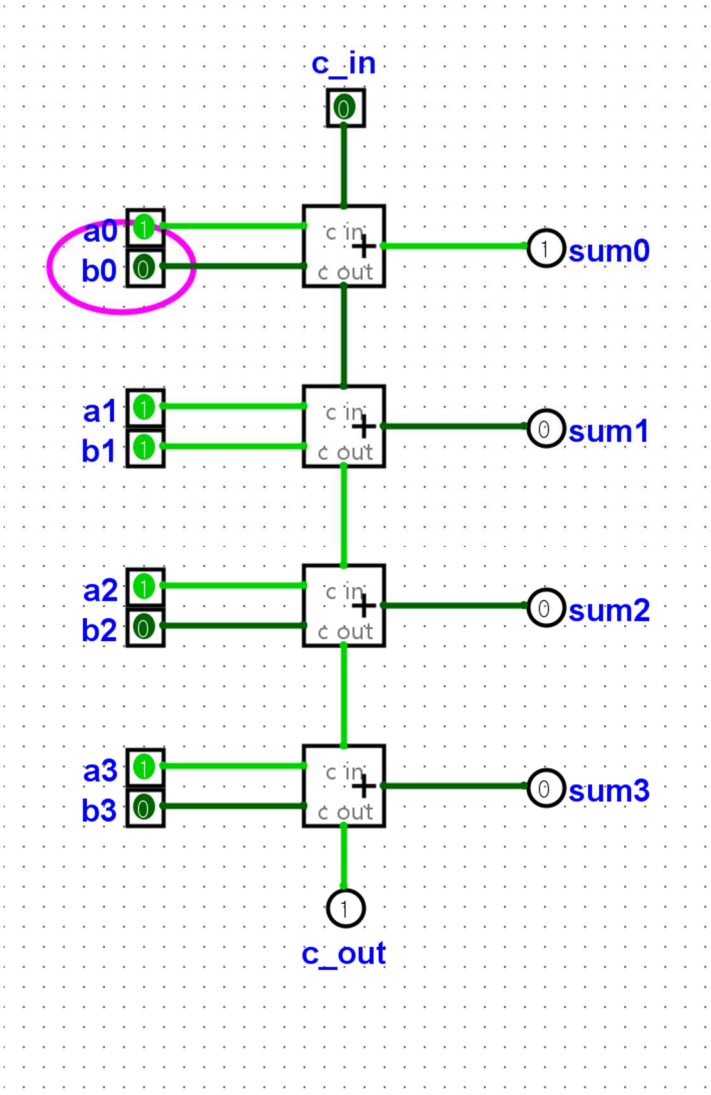


a와 b가 각각 4'b1111과 4'b0001을 input값으로 받았을 때 c_out이 1'b1이 출력되고 sum이 4'b0000으로 출력된 것을 확인할 수 있습니다.

7) a=15 b=2 (a=4'b1111 b=4'b0010)

마지막으로 a=15 b=2를 input으로 넣어줬을 때 c_out=1'b1 sum=4'b0001로 출력된 결과입니다.

a와 b를 2진수로 나타내어 각 비트끼리 덧셈을 진행했을 때 올림 수가 발생하여 c_out의 output값으로 1'b1 그리고 sum의 output값으로 4'b0001이 출력된 것을 확인할 수 있습니다.



Logisim-evolution에서도 a=4'b1111 b=4'b0001을 input으로 넣어준 결과 c_out=1'b1 sum=4'b0001이 출력된 모습을 확인할 수 있습니다.

고찰

컴퓨터구조실험 과목을 통하여 이번에 Logisim-evolution프로그램을 처음 사용해보았는데 이를 통해 제가 만들고자 하는 회로를 만들고 시뮬레이션을 실행하여 회로 동작을 확인할 수 있었습니다. 처음에는 wire도 잘 세심하게 연결하지 못하고 data bit수도 서로 맞춰주지 않아서 의도한대로 결과 값이 나오지 않고 오류도 많았습니다. 그래도 이번주 수업을 듣고 과제를 하면서 조금은 더 익숙해져서 사용하는 데 큰 어려움은 없었습니다. Ripple Carry Adder를 설계하는 것에 있어서도 생각보다 빨리 구현할 수 있었으며 이전에 모듈의 output값을 다음 모듈의 input값으로 전달해주는 부분이 중요했던 것 같습니다. Logisim evolution으로 설계한 회로와 작성한 코드를 Icarus verilog로 컴파일해 gtkwave로 나타낸 파형을 서로 비교하면서 확인하니까 왜 이 값이 나오고 어떻게 이 값이 나왔는지 좀 더 수월하게 이해할 수 있었습니다.

Reference

Verilog_1 제안서 및 Computer Architecture Lab 2주차 강의자료