컴퓨터 공학 기초 실험2 보고서

실험제목: Ripple-Carry Adder (RCA)

실험일자: 2021년 09월 20일 (화)

제출일자: 2021년 09월 26일 (월)

학 과: 컴퓨터공학과

담당교수: 공영호 교수님

실습분반: 화요일 0, 1, 2

학 번: 2020202040

성 명: 박민형

1. 제목 및 목적

A. 제목

Ripple-Carry Counter

B. 목적

흔히들 컴퓨터는 0과 1로 이루어져 있다고 말한다. 실제로 컴퓨터는 전압이 높으면 1 전압이 낮으면 0으로 인식하여 0과 1로 이루어진 연산을 수행한다. 그런데 일정 기준보다 큰 숫자 간의 계산에서는 자릿수가 바뀌는 과정이 존재한다. 그렇다면 컴퓨터는 이러한 과정은 어떻게 수행하는 것일까? 또한 컴퓨터는 사람의 펜과 노트처럼 부호를 표시하라 수 없고 오로지 0과 1로 표현해야 한다. 이러한 표현은 어떻게 하는 것일까? 2의 보수 법과, Adder는 이러한 질문에 답을 해줄 수 있다. 이번 실험에서는 Adder의 원리와 이를 활용한 Ripple-Carry Adder에 대해서 탐구해 볼 것이다.

2. 원리(배경지식)

A. HA

반가산기(半加算器, half adder)는 이진수의 한 자리 수를 연산하고, 자리 올림 수는 자리 올림 수 출력(carry out)에 따라 출력한다. AND, OR NOT의 세 가지 종류의 논리회로만으로 구성할 수 있다. 최종 값은 2C+S와 같다.

B. FA

전가산기(全加算器, full adder)는 이진수의 한 자릿수를 연산하고, 하위의 자리 올림 수 입력을 포함하여 출력한다. 하위의 자리 올림 수 출력을 상위의 자리 올림 수 입력에 연결함으로써 임의의 자리수의 이진수 덧셈이 가능해진다. 하나의 전가산기는 두개의 반가산기와 하나의 OR로 구성된다.

C. RCA

복수의 전가산기를 이용하여 임의의 비트 수를 더하는 논리회로를 만들 수 있다. 각각의 전가산기가 자리 올림 수 입력 C in으로 직전의 자리 올림 수 출력 C out을 받는 형식으로, 자리올림수가 물결(ripple)치듯 다음 가산기로 옮겨 간다고 하여 리플 캐리 가산기라 한다. 첫 번째 전가산기에 한하여 반가산기로 대체될 수 있다.

D. Verilog – 벡터: 전체 bit에서 할당하기.

벡터란 n-bit 폭을 가진 하나의 원소를 말한다. <range1>에서 <range2> 범위의 선 폭을 갖는 벡터 신호 <identifier>가 선언되는데 일반적으로 [높은 수:낮은 수]로 표현한다. 이 경우 가장 왼쪽이 MSB(most significant bit)이고, 가장 오른쪽(range2) LSB(least significant bit)가 된다. 이렇게 선언된 벡터는 C언어의 배열처럼 A[n] (range1≤n≤range2)으로 할당해서 쓸 수 있다. 이번에 수행할 과제에서 input이 9개나 되기 때문에 공부했다. 이 방법을 쓰면 input을 번거롭게 여러 번 선언하지 않아도 된다.

Appendix) Two's Complement: 2의 보수 법

컴퓨터에는 정수의 연산을 할 때, 2의 보수법이라는 특별한 방법을 사용한다. 동일한 저장공간을 이용해서 일반적인 방법(1의 보수)보다 더 많은 수의 표현을 할 수 있기 때문이다.

2의 보수는 대부분의 산술연산에서 원래 숫자의 음수처럼 취급된다. 주어진 이진수보다 한 자리 높고 가장 높은 자리가 1이며 나머지가 0인 수에서 주어진 수를 빼서 얻은 수가 2의 보수이다. 혹은 주어진 이진수의 모든 자리의 숫자를 반전(0을 1로, 1을 0으로)시킨 뒤 여기에 1을 더하면 2의 보수를 얻을 수 있다.

3. 설계 세부사항

A. Half Adder

Half Adder(HA)의 I/O는 변수 a, b와 두 수의 합인 s(sum), 자리 올림수인 co(carry out)으로 구성 되어있다. HA의 Truth Table은 다음과 같다.

a	b	co	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Karnaugh Map은 다음과 같다.

co의 K-Map

a \ b	0	1
0	0	0
1	0	1

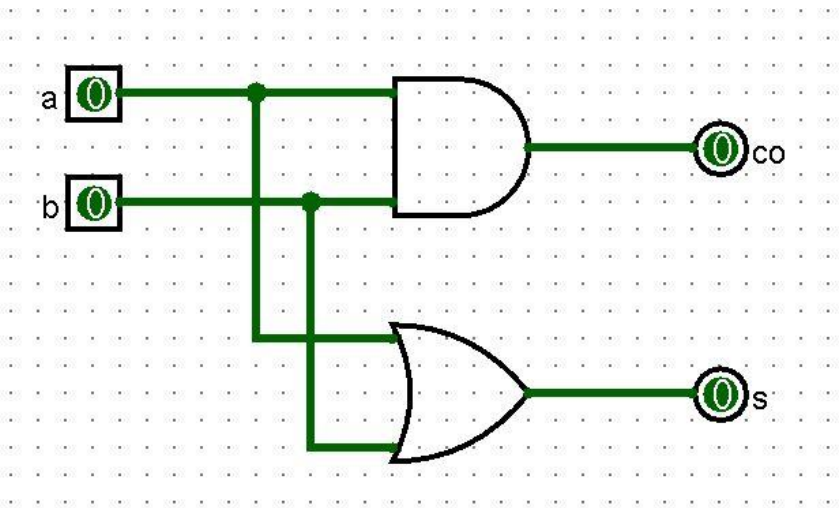
s의 K-Map

a \ b	0	1
0	0	1
1	1	1

따라서 HA의 Equation은 다음과 같다.

$$co = ab$$
$$s = a + b$$

이를 바탕으로 구성한 HA의 회로도는 다음과 같다.



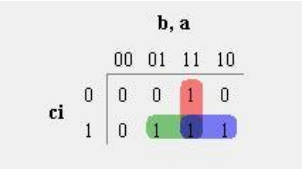
B. Full Adder

Full Adder(FA)를 단순하게 설명하자면 input이 3개인 Half Adder라고 할 수 있겠다. I/O는 변수 a, b와 두 수의 합인 s(sum), 자리 올림수인 co(carry out)으로 구성 되어있다. HA의 Truth Table은 다음과 같다.

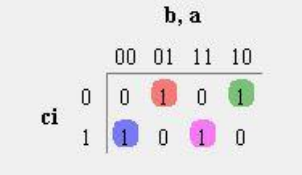
ci	b	a	co	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Karnaugh Map은 다음과 같다.

co의 K-Map



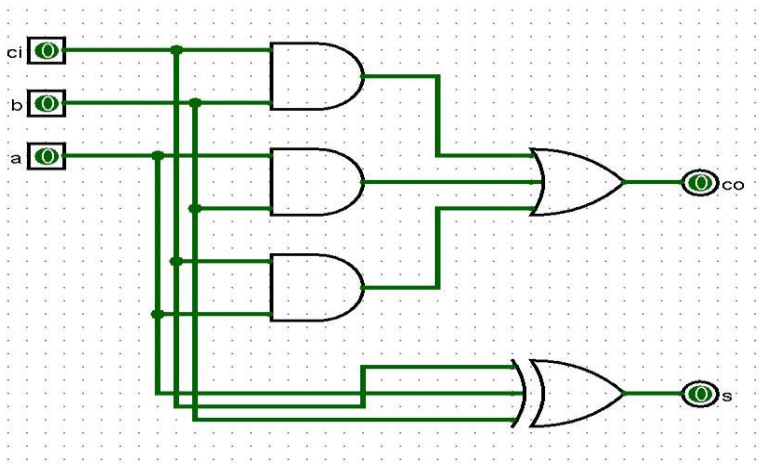
s의 K-Map



따라서 FA 의 Equation 은 다음과 같다.

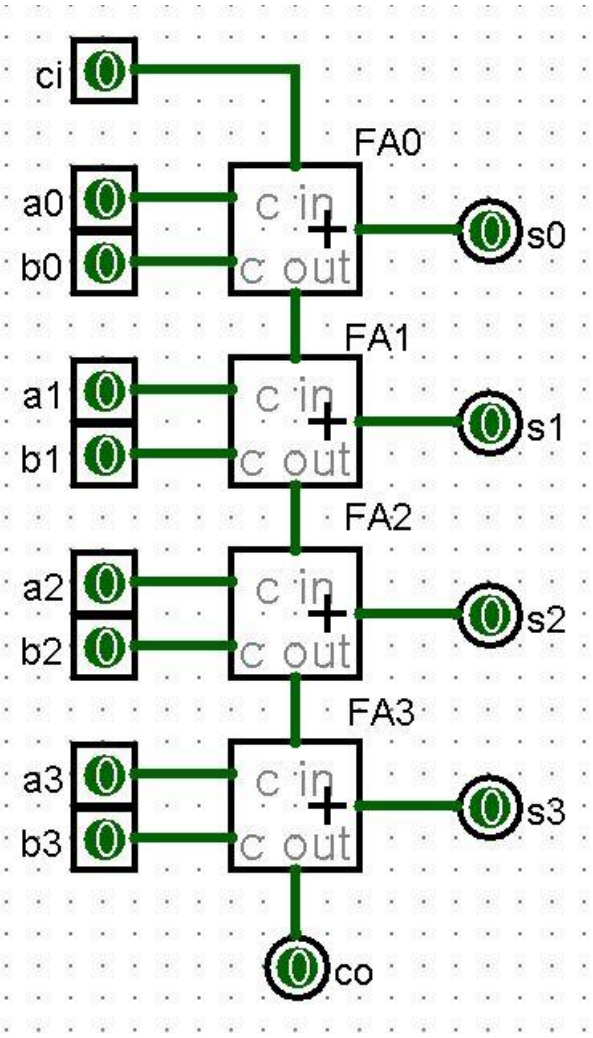
$$co = ab + b\ ci + ci\ a$$
$$s = a \sim b \sim ci + \sim a\ b \sim ci + \sim a \sim b\ ci + a\ b\ ci$$

이를 바탕으로 구성한 FA 의 회로도는 다음과 같다.



C. Ripple-Carry Adder (RCA)

지금까지 1 bit 단위에서 가산을 가능하게 하는 HA와 FA에 대해서 알아보았다. 그렇다면 1보다 큰 n bit 단위에서의 연산은 어떻게 할 수 있을까? 여기서 HA보다 복잡한 FA를 굳이 설계한 이유가 나온다. RCA는 FA를 원하는 자릿수만큼 이어 붙여서 1bit 보다 많은 자리수의 연산을 수행하는 것을 가능하게 한다.



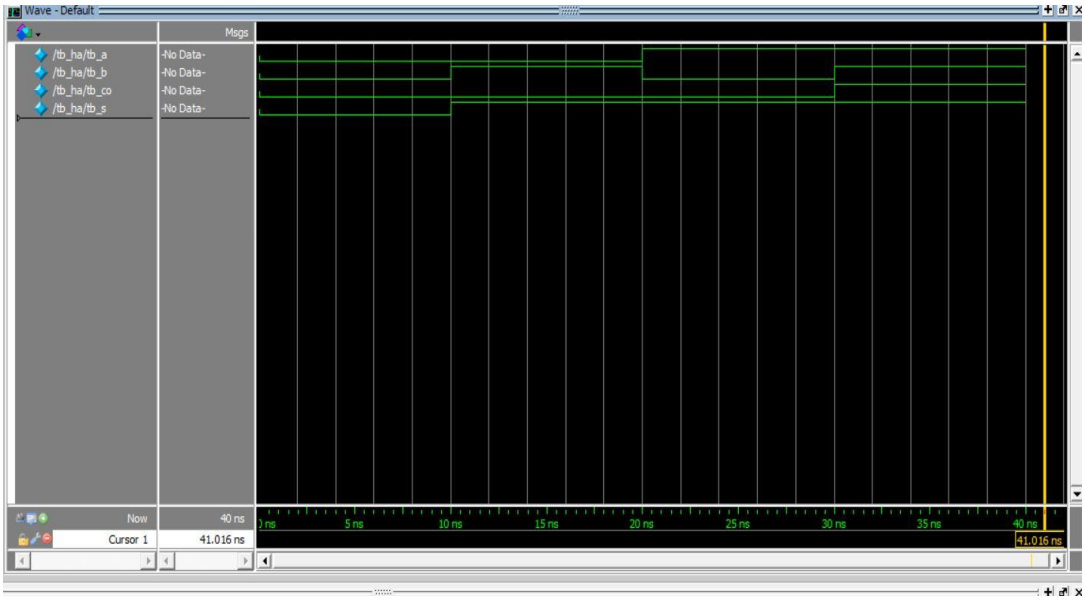
다음 그림에서, FA0의 c out은 FA1의 c in이 되고, FA1의 c out은 F2의 c in이 된다. 결국 c in이 존재 이유는 더했을 때 발생할 수 있는 자리올림수가 필요했기 때문이었고 이런 식으로 FA를 연결하면 FA의 개수 n개만큼 n개 자리수의 binary 숫자를 연산할 수 있다.

4. 설계 검증 및 실험 결과

A. 시뮬레이션 결과

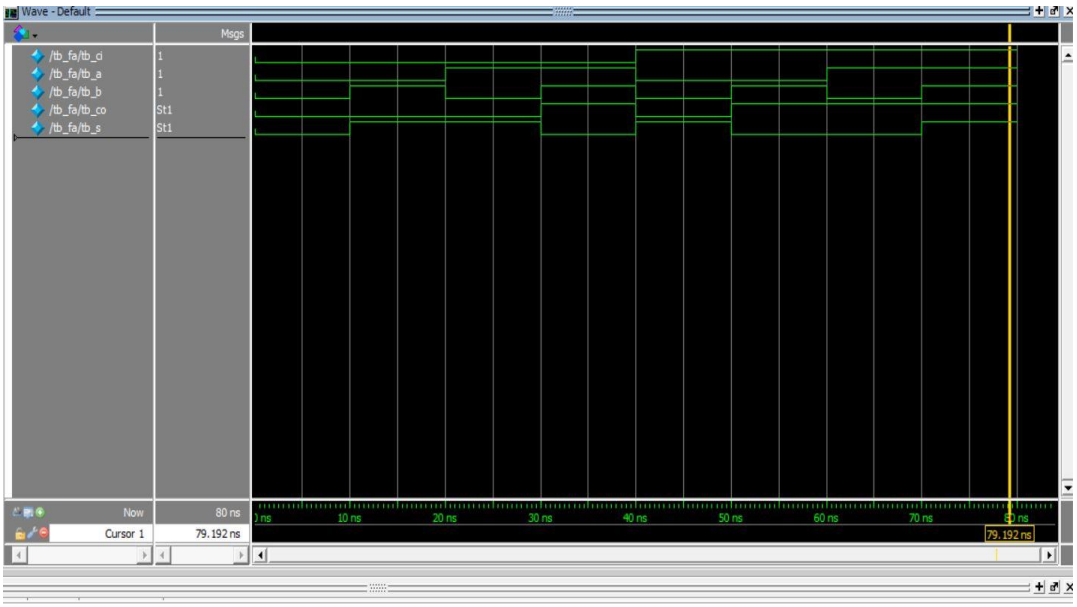
Half Adder의 Waveform

Input은 a와 b, output은 co와 s이고, exhaustive verification을 실시하였다.



Full Adder의 Waveform

Input은 ci와 a, b이다. output은 co와 s이고, exhaustive verification을 실시하였다.



Ripple-Carry Adder의 Waveform

Input은 ci와 a[3:0], b[3:0]이다. output은 co와 s[3:0]이고, Input이 9개나 되기 때문에 몇 가지 경우를 선정하여 directed verification을 실시하였다. 선정 기준은 다음과 같다.

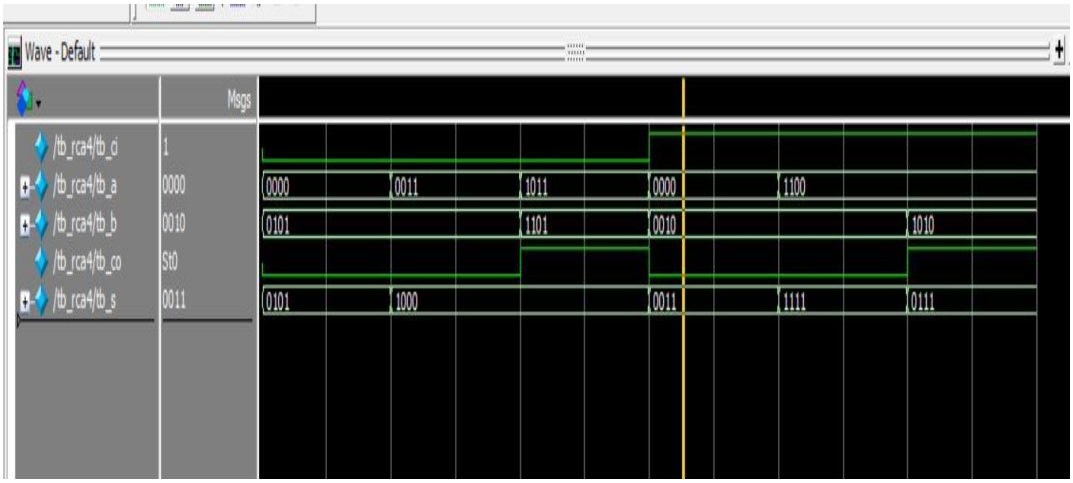
a. ci가 0인경우

- 0000을 포함: 0000 + 0101
- 4-bit 자리 수를 벗어나지 않는 경우: 0011 + 0101
- 4-bit 자리 수를 벗어나는 경우: 1011 + 1101

b. ci가 1인 경우:

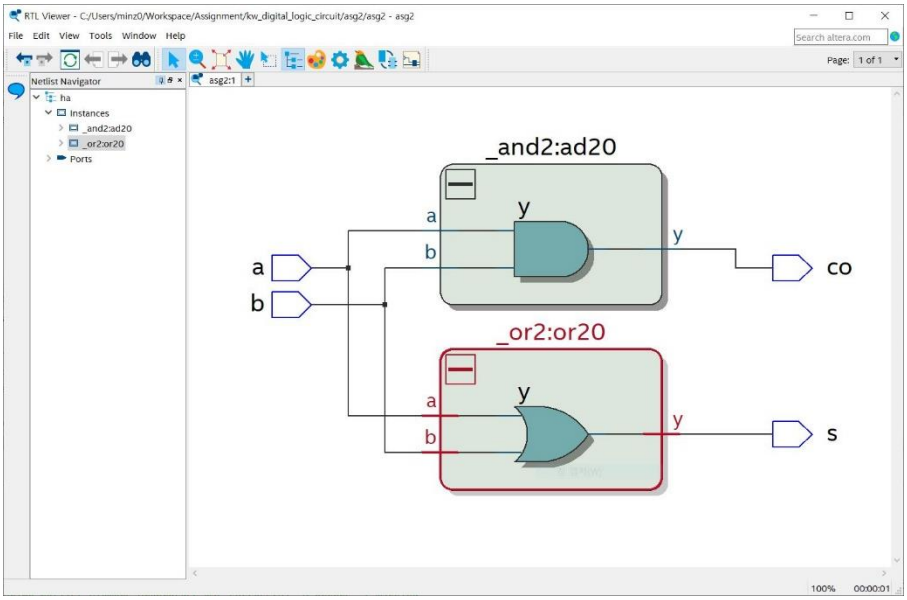
- 0000을 포함: 0000 + 0010
- 4-bit 자리 수를 벗어나지 않는 경우: 1100 + 0010
- 4-bit 자리 수를 벗어나는 경우: 1100 + 1010

Waveform은 다음과 같다.



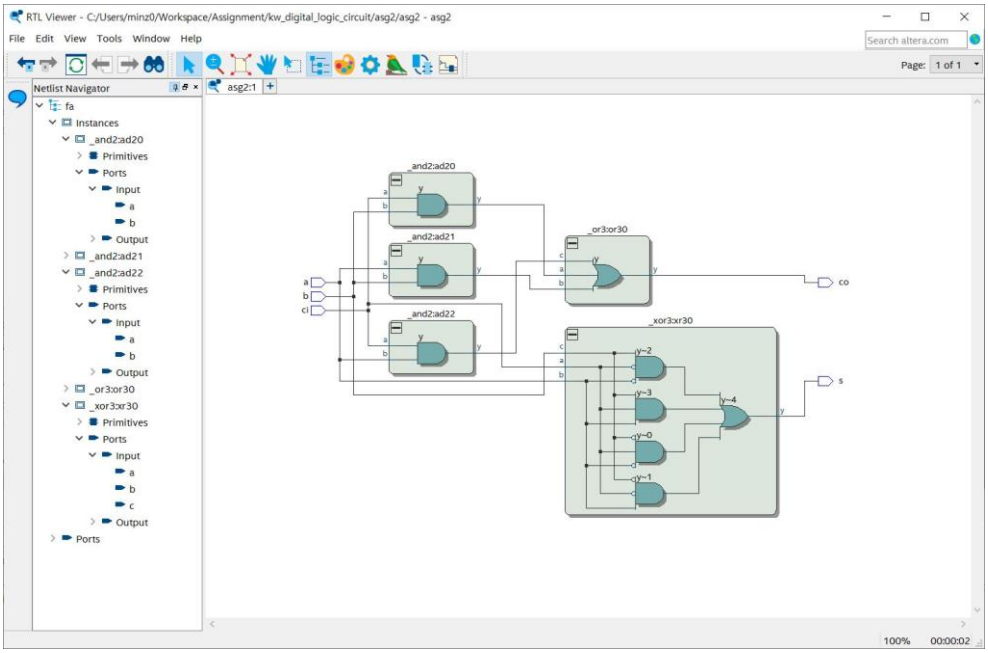
B. 합성(synthesis) 결과

Half Adder의 Map Viewer



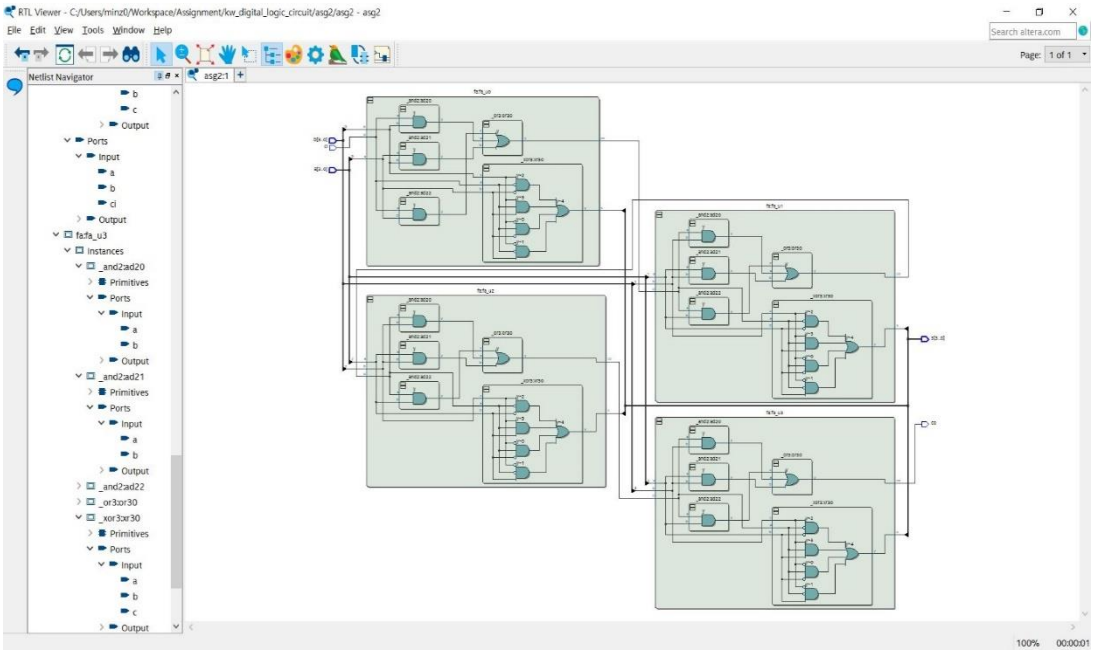
가장 단순한 Adder다.

Full Adder의 Map Viewer



input이 3개인 OR게이트와 XOR 게이트 "gates.v"에만들어서 썼다.

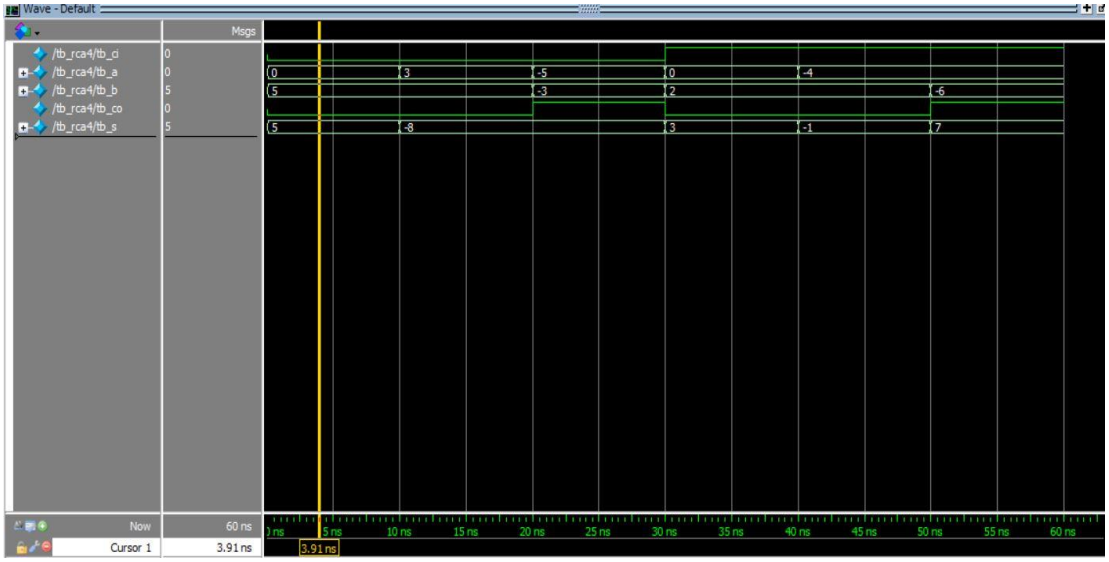
Ripple-Carry Adder의 Map Viewer



위에서 만든 Full Adder를 4개 이어 붙였다.

5. 고찰 및 결론

A. 고찰



Radix를 decimal로 나타냈을 때는 두 번째 연산마다 s에 -부호가 생기는 현상이 발생했다. 5번째 숫자를 부호를 나타내는 숫자로 인식한 것은 확실한데 구체적인 원인은 모르겠다.

Radix를 unsigned로 나타냈을 때는 이상한 점 없이 잘 출력되었다.

B. 결론

이번 실험을 통해 컴퓨터가 가산하는 과정에 대해서 알게 되었다. 그렇다면 여기서 궁금해지는 것은 4bit보다더 큰 bit의 연산은 어떻게 할까 궁금해졌다. 우리는 4bit RCA는 1bit 연산이 가능한 Full Adder를 4개 연속으로 붙여서 4bit의 숫자의 연산을 할 수 있다는 것을 이번 실험을 통해서 알았다. 따라서 32-bit RCA는 1-bit 연산이 가능한 FA가 32개 필요하다는 뜻이므로, 이것은 4-bit RCA가 8개 필요함을 알 수 있다.

6. 참고문헌

가산기/https://ko.wikipedia.org/wiki/%EA%B0%80%EC%82%B0%EA%B8%B0

2의 보수/https://ko.wikipedia.org/wiki/2%EC%9D%98_%EB%B3%B4%EC%88%98

Verilog기본문법

/https://hizino.tistory.com/entry/Verilog%EC%9D%98-%EA%B8%B0%EB%B3%B8-%EC%88%AB%EC%9E%90%ED%91%9C%ED%98%84-%EC%9E%90%EB%A3%8C%ED%98%95-%EB%B2%A1%ED%84%B0-%EB%B0%B0%EC%97%B4-%EB%AC%B8%EC%9E%90%EC%97%B4