컴퓨터 공학 기초 실험2 보고서

실험제목 : Ripple Carry Adder

실험일자 : 2016년 09월 01일 (목)

제출일자 : 2016년 09월 07일 (수)

학 과: 컴퓨터공학과

담당교수: 이혁준 교수님

실습분반: 목요일 0, 1, 2

학 번: 2013722095

성 명: 최재은

1. 제목 및 목적
   1. 제목

Ripple Carry Adder

* 1. 목적

Quartus 프로그램을 이용하여 Verilog 언어의 문법 및 사용법을 배우고, 이를 응용하여 논리 연산을 가능하게 하는 gate를 직접 module로 만들어본다. 또한 이를 Instance하여 Half Adder, Full Adder, Ripple Carry Adder를 만들어 본다. 이 과정을 통해 각 가산기의 결과 값과 동작원리를 이해하고 숙지하여 응용함에 목적을 둔다.

1. 원리(배경지식)

Half adder와 Full Adder를 구현할 때 이에 대한 진리표 및 동작원리를 이해하기 위해 ‘논리 회로 실험’ 교재를 참고하였습니다. 교재를 통해 두 이진수의 합과 이때 발생하는 올림 수를 carry out으로 출력해준다는 것을 알게 되었고, 진리표를 보았을 때 올림수는 두 입력값이 1/1인 경우에만 출력되므로 이 부분을 and게이트로 두 입력의 합은 1/1일 때 올림수를 발생시키고 0이 되어야 하므로 xor게이트로 구현하였습니다. Full Adder는 Half Adder와 다르게 초기 입력되는 carry in이 있는데 이 carry in은 비트 계산에 포함되어 추가적인 올림값의 발생과 비트의 합에 영향을 줍니다. 두 개의 half adder와 한 개의 or게이트로 구현을 하였으며, 이때 하나의 half adder에서는 입력된 두 비트의 합과 올림수를, 다른 하나의 half adder에서는 입력된 두 비트의 합과 이전의 자리에서 올라온 올림수와의 합과 올림수를 계산하는데 이 때의 올림수와 첫번째 half adder에서의 올림수는 올림수가 아예 없는 경우에만 0을 carry out에 반영하므로 or 게이트로 연산하여 최종적인 올림수값을 산출하게 됩니다. 이렇게 만들어진 하나의 full adder는 한 비트를 연산하고 합과 올림수를 출력하므로, 이를 통해 n개의 비트연산자를 구현할 수 있게 합니다. (이 때 첫번째 비트를 계산하는 full adder는 이전자리의 비트가 없기 때문에 올라오는 올림수가 없어서 carry in을 0으로 설정합니다.) 위와 같은 내용을 수행하기 위해 기존의 베릴로그 언어의 디테일한 사용법이 기억나지 않아, 군휴학전에 들었던 디지털 논리회로 때의 베릴로그 수업교재 및 자료를 참고하여 특정 모듈을 다른 모듈에서 사용하는 방식이나, Qurtus 프로그램의 사용법을 다시 익혔습니다. Ripple Carry Adder에서는 기본적인 논리게이트를 모두 사용하는 듯싶어서, 이 게이트들의 동작 원리와 진리표를 공부했습니다.

2의 보수란 2진수의 연산에서 음의 수를 표현하기 위한 것으로 0을 1로, 1을 0으로 바꾸고 1을 더해서 만든다.(ex/ 1001 -> 0110 -> 0111)

1. 설계 세부사항

**\* Nand gate**

- n개의 입력이 모두 1인 경우에만 0을 출력하는 논리 게이트

|  |  |  |
| --- | --- | --- |
| A | B | Y(출력) |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

* 기본 구조가 and게이트 출력의 반대이기 때문에 module로 만든 and gate의

출력값이 inverter의 입력값이 되도록 설계

**\* Half Adder**

- 두 개의 1bit 입력값을 받아 합(s)과 올림수(co)를 출력해주는 가산기

**<Truth Table>**

|  |  |  |  |
| --- | --- | --- | --- |
| 입력(INPUT) | | 출력(OUTPUT) | |
| A | B | CO(올림수) | S(합) |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |

**<Karnaugh Map & Boolean Equation>**

1. About Carry out

|  |  |  |
| --- | --- | --- |
| a b | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 0 | 1 |

**Carry out co = a · b**

1. About Sum

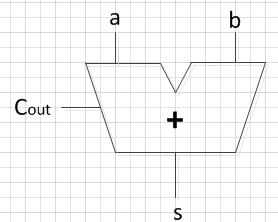
|  |  |  |
| --- | --- | --- |
| a b | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 0 |

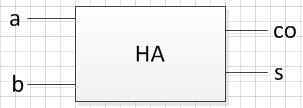
**Sum s = (a · b) · (a + b)**

* 올림수의 출력값은 AND GATE의 것과 같고, 합은 입력이 1/1일 경우 올림수가 발생하면서 0이 되어야 하기 때문에 XOR GATE로 설계

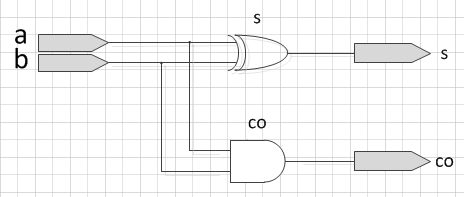
**<Graphical Symbol & Circuit>**

**1) Graphical Symbol**

 - 다음과 같은 형태의 Symbol이 많이 사용된다.



1. **Circuit**



**\*Full Adder**

- 두 개의 1bit 입력값과 Carry in(이전 모듈에서의 올림수) 값을 받아 합(s)과

올림수(co)를 출력해주는 가산기

**<Truth Table>**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 입력(INPUT) | | | 출력(OUTPUT) | |
| Carry In | A | B | CO(올림수) | S(합) |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 |

**<Karnaugh Map & Boolean Equation>**

\*\* Ci를 임의로 x라고 지칭함 / (+)를 exclusive or라 임의로 칭함(기호를 찾아도 없습니다.)

1. About Carry out

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ci ab | 00 | 01 | 10 | 11 |
| 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |

**Carry out co = abx’ + a’bx + ab’x+abx = ab(x+x’) + x(a’b+ab’)**

**= ab + x(a(+)b)**

1. About Sum

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ci ab | 00 | 01 | 10 | 11 |
| 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |

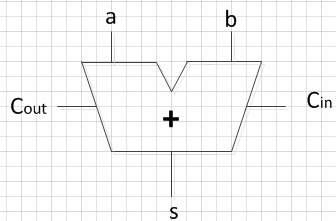
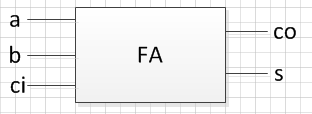
**Sum s = a’bx’ + ab’x’ + a’b’x + abx = x’(a’b+ab’) + x(a’b’+ab) = x’(a(+)b)+x(a(+)b)’**

**= x(+)a(+)b**

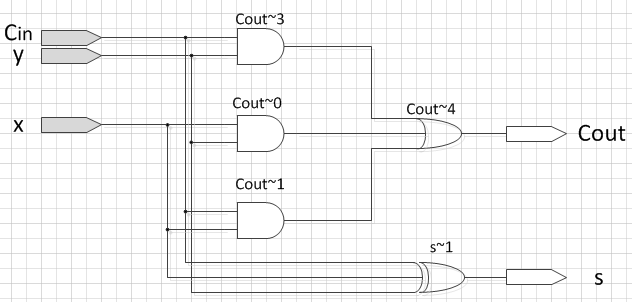
**<Graphical Symbol & Circuit>**

1. **Graphical Symbol**

- 다음과 같은 형태의 Symbol이 주로 사용된다.

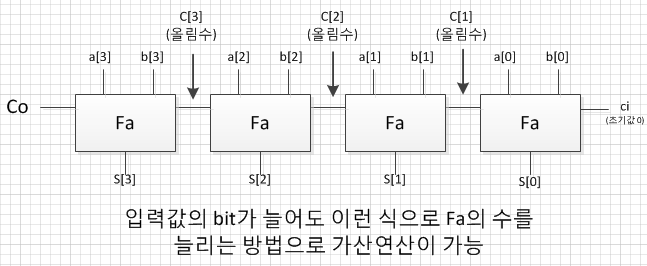


1. **Circuit**



**\* Ripple Carry Adder**

- n’bit의 두 수를 더해주는 가산기로써 n개의 1’bit Full Adder로 구성이 되어있다.

- Full Adder의 특성상 Carry in값을 받아주기 때문에 초기 1의자리 비트 FA는 ci값을 0으로 설정해주어야 한다.

-위와 같은 형태로 RCA module을 설계하였습니다.

**<Design Specification>**

- In/Out Description

|  |  |  |  |
| --- | --- | --- | --- |
| Port | Name | Bandwidth | Description |
| Input | a | 4bit | Input Data |
| b | Input Data |
| ci | 1bit | Carry In |
| Output | co | 1bit | Carry Out |
| s | 4bit | Sum |
| Wire | c | 3 | Internal Carry |

- Module Description

|  |  |  |
| --- | --- | --- |
| Classification | Name | Description |
| Module | rca | 4’bit Ripple Carry Adder |
| Instance | U0\_fa | Full Adder |
| U1\_fa |
| U2\_fa |
| U3\_fa |

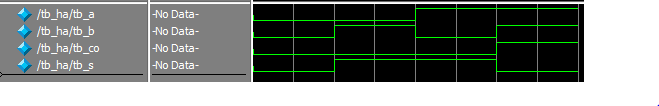
1. 설계 검증 및 실험 결과
   1. 시뮬레이션 결과

**\* Half Adder**

- 입력된 두 값의 합과 올림수를 출력해주는 가산기로 4가지의 경우밖에 없음.

- 0/0, 0/1, 1/0, 1/1의 경우의 수 중 올림수을 발생시키는 경우는 1/1뿐임.

- 결과는 예상과 다르지 않게 정상적으로 나왔습니다.

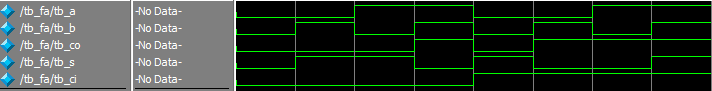


**\* Full Adder**

- Carry in이 있는 경우(1)와 없는 경우(0)로 나누어 Test Bench를 작성하였음.

- 경우의 수는 Carry in을 제외하고 ha와 동일하게 진행하였음.

- 이 또한 예상과 다르지 않은 결과를 확인할 수 있었습니다.



**\* Ripple Carry Adder**

-n개 중 첫 번째 Full Adder의 Carry in 값을 0으로 설정.

-경우의 수를 다음과 같이 진행하였음.

1) 올림값이 발생하지 않는 경우

2) 1의 자리에서 발생한 올림값으로 인해 2의 자리 비트에서 올림값이 발생한경우

3) 2의 자리에서 올림값이 발생한 경우

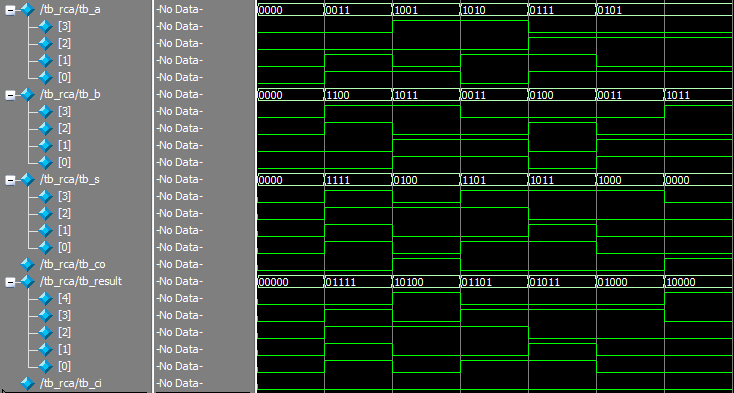
4) 4의 자리에서 올림값이 발생한 경우

5) 1의 자리에서 발생한 올림값으로 인해 8의 자리까지 올림값이 발생한 경우

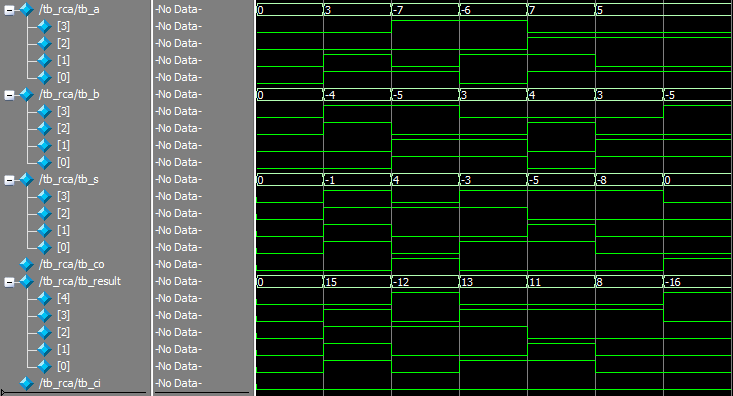
6) 1의 자리에서 발생한 올림값으로 인해 Carry out 이 발생한 경우

-Wave Form은 다음과 같이 도출되었으며 예상과 다르지 않았습니다.

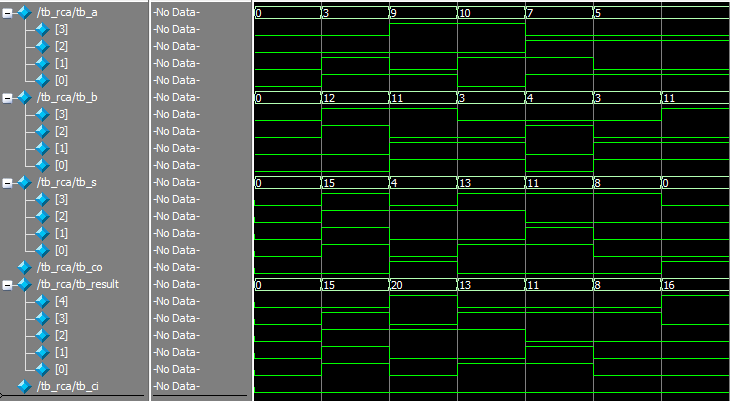
1) Binary



2) Decimal



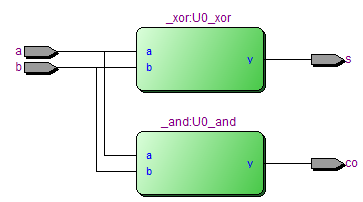
3) Unsigned



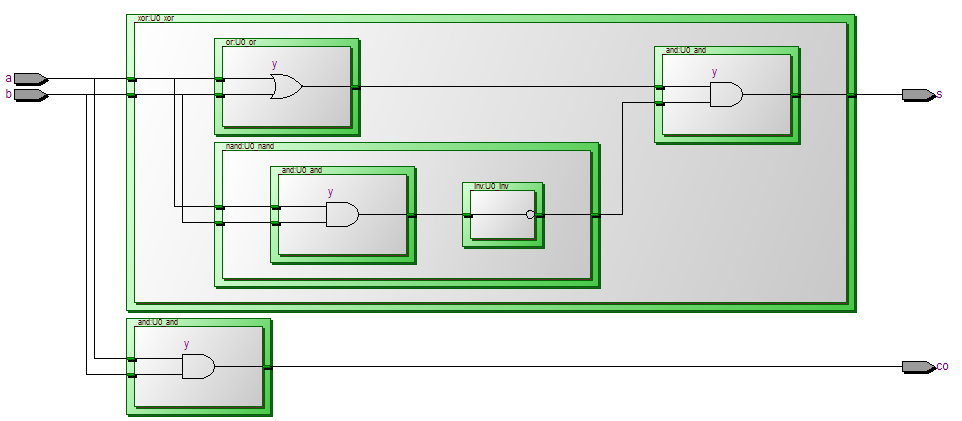
* 1. 합성(synthesis) 결과

**\* Half Adder**

**<RTL Viewer>**



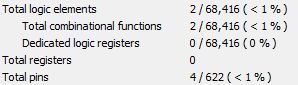
**<Technology Map Viewer>**



- 두개의 입력을 받아 합을 계산하기 위해 XOR Gate를 사용, 더했을 때의 올림값을

출력하기 위해 and 게이트를 사용하였음.

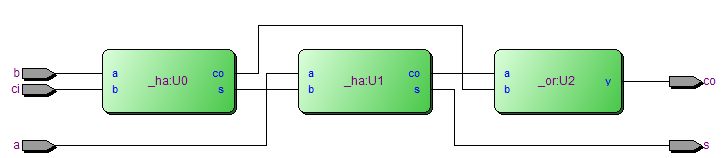
**<Flow Summary>**

 - Input 변수로 a, b, output 변수로 co, s,

총 4개의 pin 사용

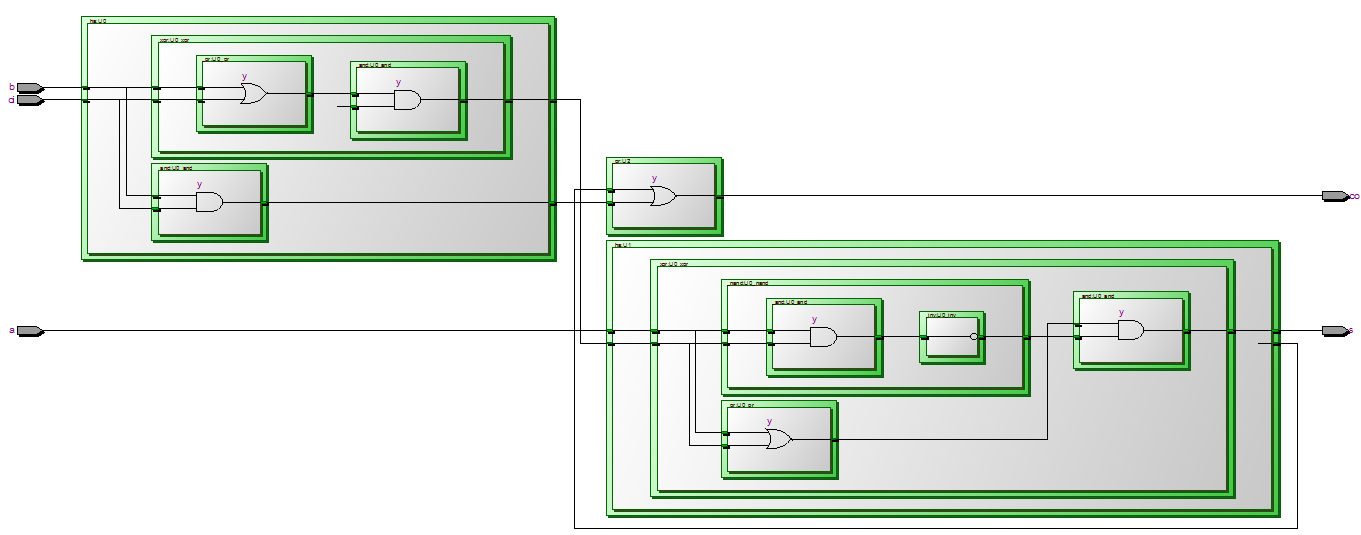
- 사용된 logic은 2종료이며 기본 소자이기 때문에 combinational function이라 표기됨.

- 따로 기억소자를 사용하지 않아 register는 0개라고 나옴.

**\* Full Adder**

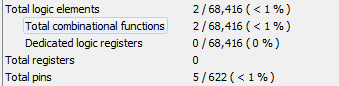
**<RTL viewer>**

**<Technology Map Viewer>**



- 이전 모듈에서 받은 올림값과 두 개의 비트를 입력인자로 받는 형태의 전가산기로써

받은 올림값과 한 개 비트의 합을 계산하고 다른 하나의 비트와 더해 최종적인 합값(s)을 계산함. 두 개의 Half Adder에서 나온 올림값을 or게이트를 통해 최종적인 Carry out 값으로 함.

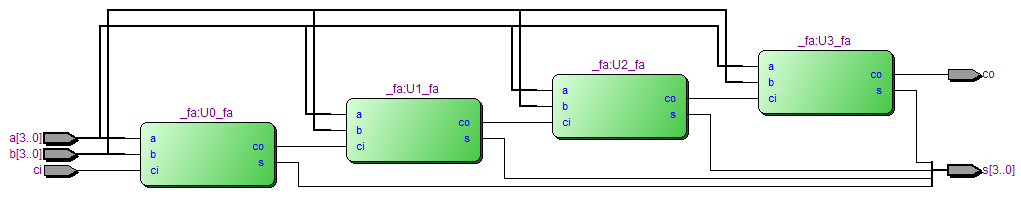
**<Flow Summary>**

- Input 변수로 a, b, ci, Output변수로 co, s

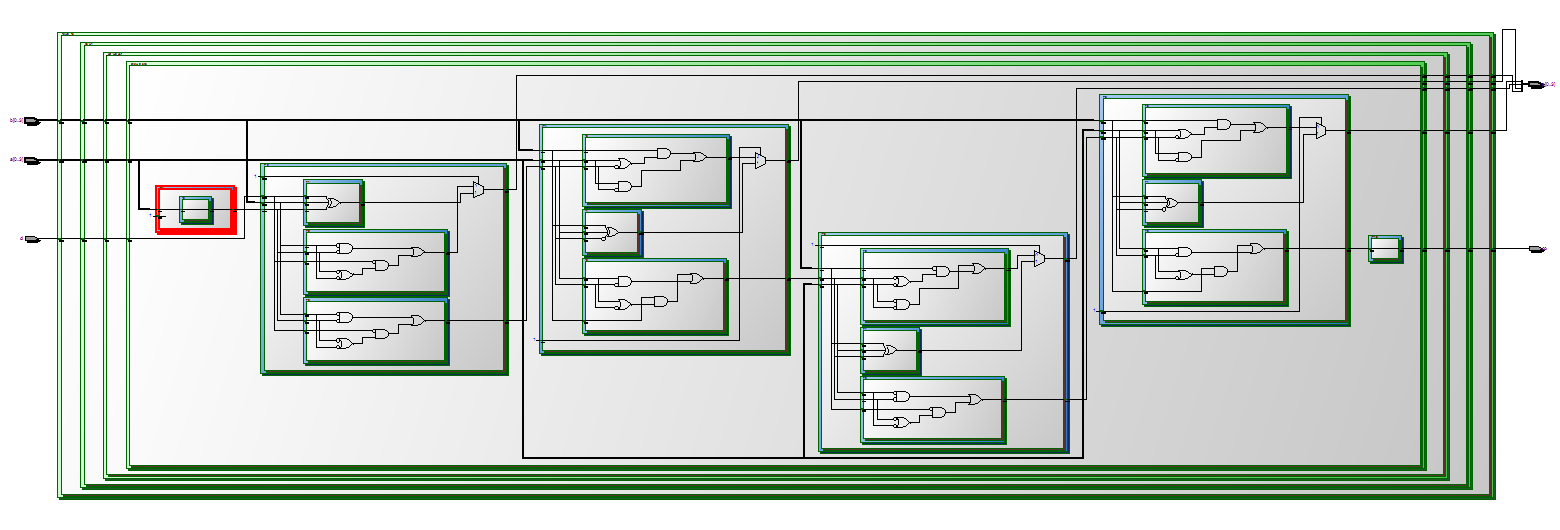
총 5개의 pin 사용

- half adder 때와 같은 결과이나 사용된 logic은 Half Adder와 OR 게이트가 사용되었음.

**\* Ripple Carry Adder**

**<RTL Viewer>**

**<Technology Map Viewer>**



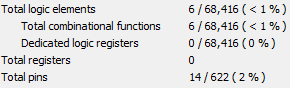
- n’bit의 수를 더하는 가산기로써 n개의 Full Adder로 구성이 되어있다.

- Full Adder의 특성상 첫 번째 Fa에도 Carry in 값을 주어야 하지만 이전 모듈이 없어 초기값을 0으로 준다.

- 각 모듈에서 출력된 합은 n’bit짜리 합(s)에 저장이 되고, 계산중에 발생한 올림값은

다음 bit를 계산하는 모듈에 Carry in 으로 들어간다.

- n’bit짜리 합(s)의 표현범위를 넘는 올림값이 발생하면 Co인자가 1이 된다.

**<Flow Summary>**

- logic은 Full Adder 4개와 그 이외의 2개(무엇인지 잘 모르겠습니다.)가 사용되었음.

- 4’bit짜리 input a,b 와 output s, 그리고

Ci과 co가 사용되어 pin이 총 14개가

사용되었음.

1. 고찰 및 결론
   1. 고찰

Ripple Carry Adder의 Testbench를 작성하고 Wave Form을 보기 위해서 Modelsim을

돌렸는데, s인자와 co인자에 ‘xxx1’라고 표기가 되었었습니다. 무슨 말인지 모르겠어서

주변에도 물어보고 인터넷에서 찾아보고 했더니 ‘High Impedance’ 상태라고 나오더군요.

유심히 보아도 틀린 부분을 찾지 못해 심혈을 기울여 조심조심 Testbench를 다시 짰습니다. 아마 Test Bench에서 인자간에 연결을 잘못한 것이 아닌가 합니다.

* 1. 결론

이번 실험을 진행하면서 예전에 디지털논리1 수업을 들을 때에는 half adder와 full adder의 차이를 이해하지 못했습니다. ‘둘 다 입력된 값의 합이랑 올림값을 계산하는 건데…’라는 생각과 함께 Verilog 수업을 그냥 따라서만 하는 식으로 듣다 보니, 코드를 봐도 이게 어떻게 연결이 되고 뭘 넘겨주고 이 문장이 의미하는 것이 도대체 무엇인지에 대한 생각뿐이었지만, 제대로 공부를 하고, 궁금한 것을 바로 바로 물어보면서 진행을 하다 보니 눈에 확 들어오는 것을 느꼈습니다.

32’bit Ripple Carry Adder를 만드는 방법으론 간단하게 4bit rca를 만들 때처럼

Full Adder를 32개 연결시켜서 연산을 하게 하면 될 것 같습니다.

1. 참고문헌

이원석, 정길수 共著 / 실용적인 IC를 익히는 논리회로 실험 / 생능출판사 / 2013년