**디지털회로실험 보고서**

-7주차-

전자공학과

2015104027

박정진

**실험 결과**

쿼터스 시뮬레이션 결과

회로도 (schematic.pdf 에서 더 자세히 볼 수 있음)

빵판 시뮬레이터를 하다 빵판 최대 6개의 공간이 부족하여 쿼터스 시뮬레이션으로 진행하였다.

A picture containing text

Description automatically generated

동작 방법 :

초기에 아무것도 누르지 않았을 때에는 A 와 B 모두 10'b1111111111 의 상태로 push button에서 pull up 저항을 달아 설정한다. 따라서 버튼을 클릭하면 해당되는 부분은 (1->0) 으로 전압이 떨어진다. 버스 순서는 (MSB)9, 8, 7, 6, ... , 1, 0(LSB)이다. 이 상태에서 start(push button, pull down)을 누른 상태(0->1)에서 계산할 두 수를 클릭하고 Equal(push button, pull down)을 누르면 (0->1) 계산 결과가 나온다. 해당 상태에서 Start 버튼을 다시 누르면(1->0) 이 되고 계산 결과는 0 으로 리셋 된다.

Equal 버튼은 Clock 과 and로 연결하여 CLK\_Enable 신호로 74174의 클락으로 사용하였다. 따라서 Equal 버튼이 눌려야만 74147에 클락이 들어간다.

Start 버튼은 74147 즉 DFF의 CLRb 로 들어가며 Start = 0 일때는 계산이 적용되지 않고 계속 리셋만 되어 최종 결과가 0으로 리셋 및 지속 된다.

해당 시뮬레이션에서는 마지막 이 5자리의 숫자(Cout, S[3..0])를 7segment로 표현하기 위해 사용되는 segment decoder를 대신 할 수 있는 방법이 생각나지 않았고, 또한 K map을 이용해 표현하기에는 빵판의 공간이 매우 부족하여 하지 못하였다. 만약 Selector같은 ALU가 더 많거나 2 input primitive gate로 구성된 칩들의 Fan in 이 더 컸으면 아마 빵판 공간이 부족하지 않았을 듯 하다.

위 회로에서 핵심 포인트는 Addn\_Sub 이다. wire 이름의 의미와 같이 0일 땐 Add, 1일땐 Subtract를 수행하도록 하였다. Xor(7486) 을 이용하여 4bit Full adder에 들어갈 B를 계산하였다. 4'b0^B = B(Add), 4'b1^B = Bn(Sub) 으로 Subtract을 진행할 땐 B를 1's complement를 취하고 Cin =0 (add), Cin=1(Sub)를 통해 전체적으로 Subtract에서 2's complement로 계산되게 하였다.

|  |  |  |
| --- | --- | --- |
| bits | unsigned | signed |
| 5'b00000 | 0 |  |
| 5'b00001 | 1 |  |
| 5'b00010 | 2 |  |
| 5'b00011 | 3 |  |
| 5'b00100 | 4 |  |
| 5'b00101 | 5 |  |
| 5'b00110 | 6 |  |
| 5'b00111 | 7 |  |
| 5'b01000 | 8 |  |
| 5'b01001 | 9 |  |
| 5'b01010 | 10 |  |
| 5'b01011 | 11 |  |
| 5'b01100 | 12 |  |
| 5'b01101 | 13 |  |
| 5'b01110 | 14 |  |
| 5'b01111 | 15 |  |
| 5'b10000 | 16 |  |
| 5'b10001 | 17 |  |
| 5'b10010 | 18 |  |
| 5'b10011 |  |  |
| 5'b10100 |  |  |
| 5'b10101 |  |  |
| 5'b10110 |  |  |
| 5'b10111 | 23 | -9 |
| 5'b11000 | 24 | -8 |
| 5'b11001 | 25 | -7 |
| 5'b11010 | 26 | -6 |
| 5'b11011 | 27 | -5 |
| 5'b11100 | 28 | -4 |
| 5'b11101 | 29 | -3 |
| 5'b11110 | 30 | -2 |
| 5'b11111 | 31 | -1 |

위의 도표 대로 수를 사용하기 위해선 Addn\_Sub = 1일 때 즉 Subtract을 수행할 때 마지막에서 나오는 Bout과 Addn\_Sub를 Xor 해줘야 위 와 같은 결과를 도출 할 수 있다. Xor 계산을 하는 이유는 덧셈에서 나오는 Cout(MSB) = 1 (계산 결과 16이상) 은 unsigned로, 뺄셈을 진행 했을 때 나오는 Bout(MSB) = 1은 signed(2's complement)로 취급해야 하기 때문이다.

Case 1)

A = 0

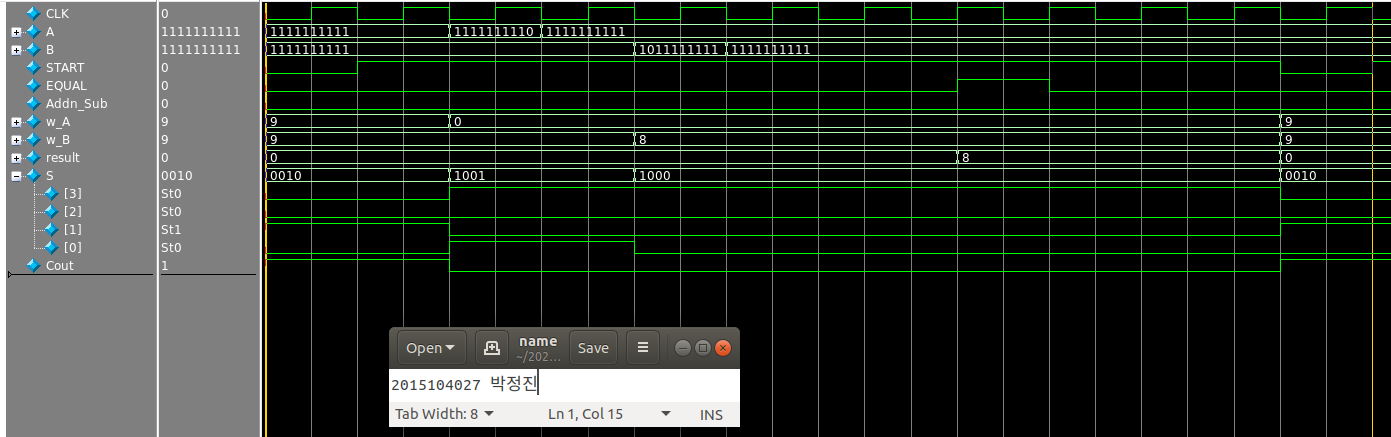
B = 8

Addn\_Sub = 1'b0

결과 예상 : 8

Equal rising(0->1) 과 함께 계산 결과 result(C\_out, S) 가 정확히 8(5'b01000) 이 나오는 것을 확인 할 수 있다.

Start Falling(1->0) 과 함게 계산 결과 result 가 5'b00000으로 리셋 되는 것을 확인 할 수 있다.



Case 2)

A = 7

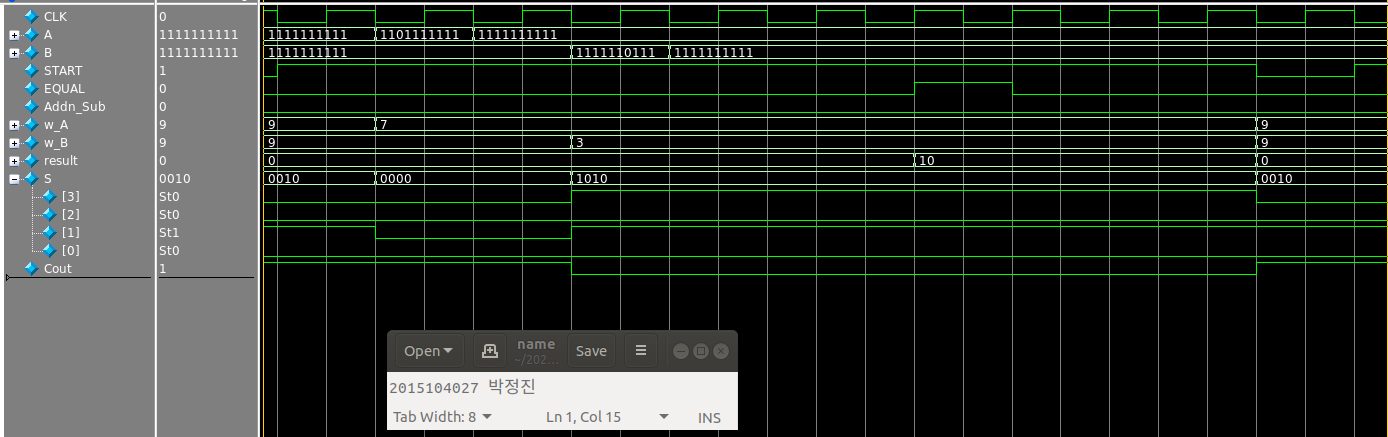
B = 3

Addn\_Sub = 1'b0

결과 예상 : 10

Equal rising(0->1) 과 함께 계산 결과 result(C\_out, S) 가 정확히 8(5'b01010) 이 나오는 것을 확인 할 수 있다.

Start Falling(1->0) 과 함게 계산 결과 result 가 5'b00000으로 리셋 되는 것을 확인 할 수 있다.



Case 3)

A = 9

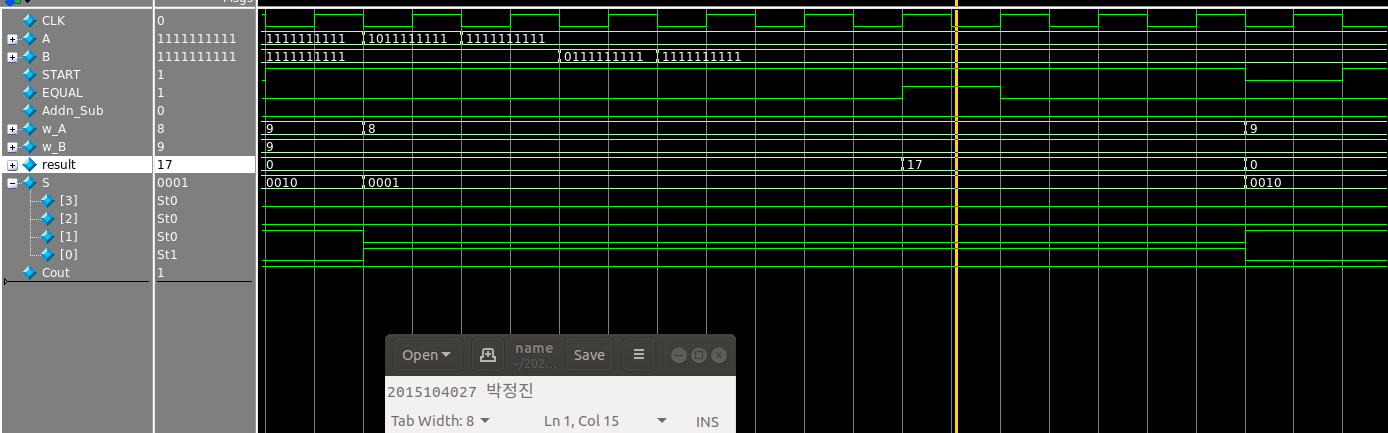
B = 8

Addn\_Sub = 1'b0

결과 예상 : 17

Equal rising(0->1) 과 함께 계산 결과 result(C\_out, S) 가 정확히 8(5'b10001)(unsigned) 이 나오는 것을 확인 할 수 있다.

Start Falling(1->0) 과 함게 계산 결과 result 가 5'b00000으로 리셋 되는 것을 확인 할 수 있다.



Case 4)

A = 0

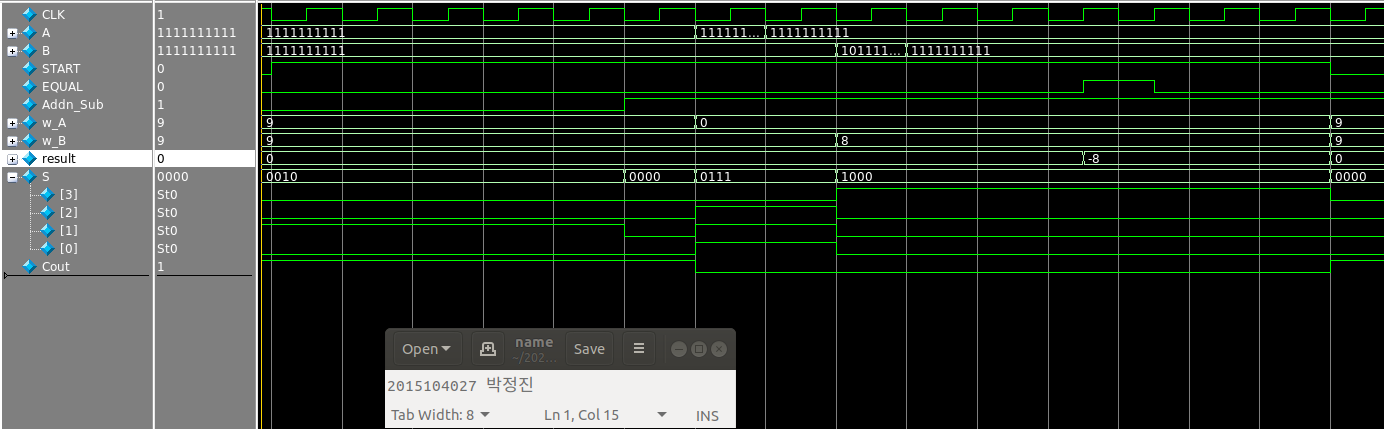
B = 8

Addn\_Sub = 1'b1

결과 예상 : -8

Equal rising(0->1) 과 함께 계산 결과 result(C\_out^Addn\_Sub, S) 가 정확히 8(5'b11000)(Signed)이 나오는 것을 확인 할 수 있다.

Start Falling(1->0) 과 함게 계산 결과 result 가 5'b00000으로 리셋 되는 것을 확인 할 수 있다.



Case 5)

A = 7

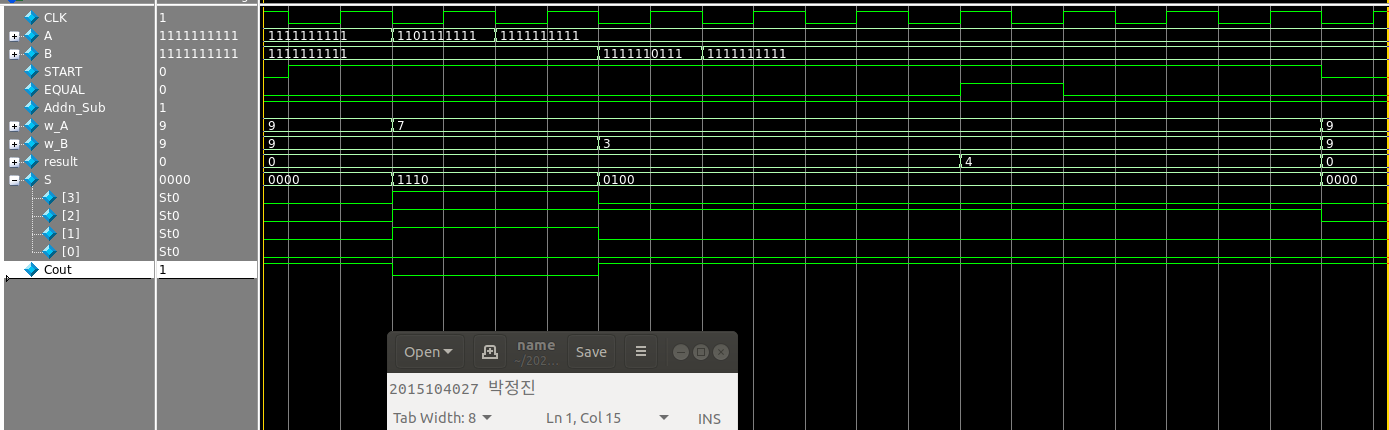
B = 3

Addn\_Sub = 1'b1

결과 예상 : 4

Equal rising(0->1) 과 함께 계산 결과 result(C\_out^Addn\_Sub, S) 가 정확히 8(5'b00100)(Signed)이 나오는 것을 확인 할 수 있다.

Start Falling(1->0) 과 함게 계산 결과 result 가 5'b00000으로 리셋 되는 것을 확인 할 수 있다.



Case 6)

A = 8

B = 9

Addn\_Sub = 1'b1

결과 예상 : -1

Equal rising(0->1) 과 함께 계산 결과 result(C\_out^Addn\_Sub, S) 가 정확히 8(5'b11111)(Signed)이 나오는 것을 확인 할 수 있다.

Start Falling(1->0) 과 함게 계산 결과 result 가 5'b00000으로 리셋 되는 것을 확인 할 수 있다.

