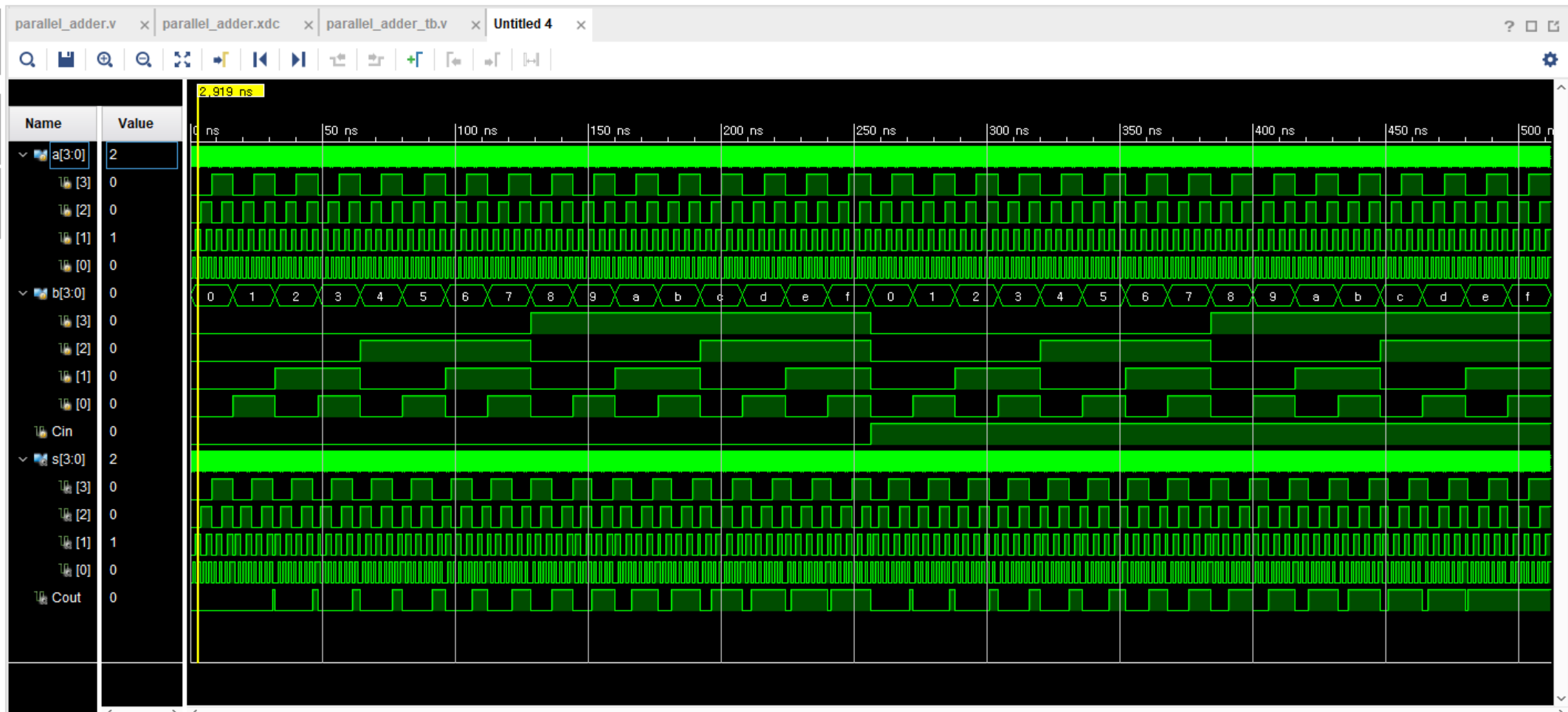
10주차 결과보고서

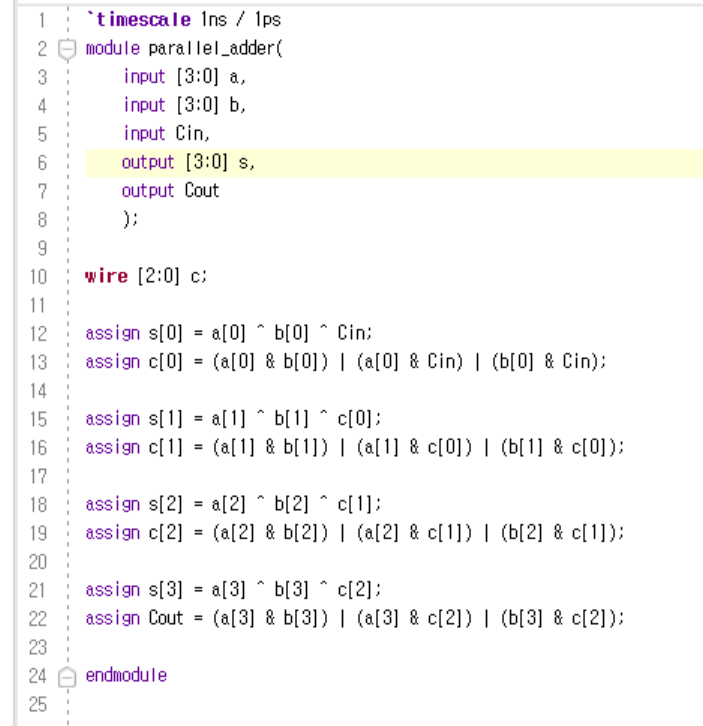
전공: 컴퓨터공학과 학년: 2학년 학번: 20231561 이름: 심소현

**1.**

.................



위의 사진은 4-bit Binary Parallel Adder의 simualtion 사진이다.



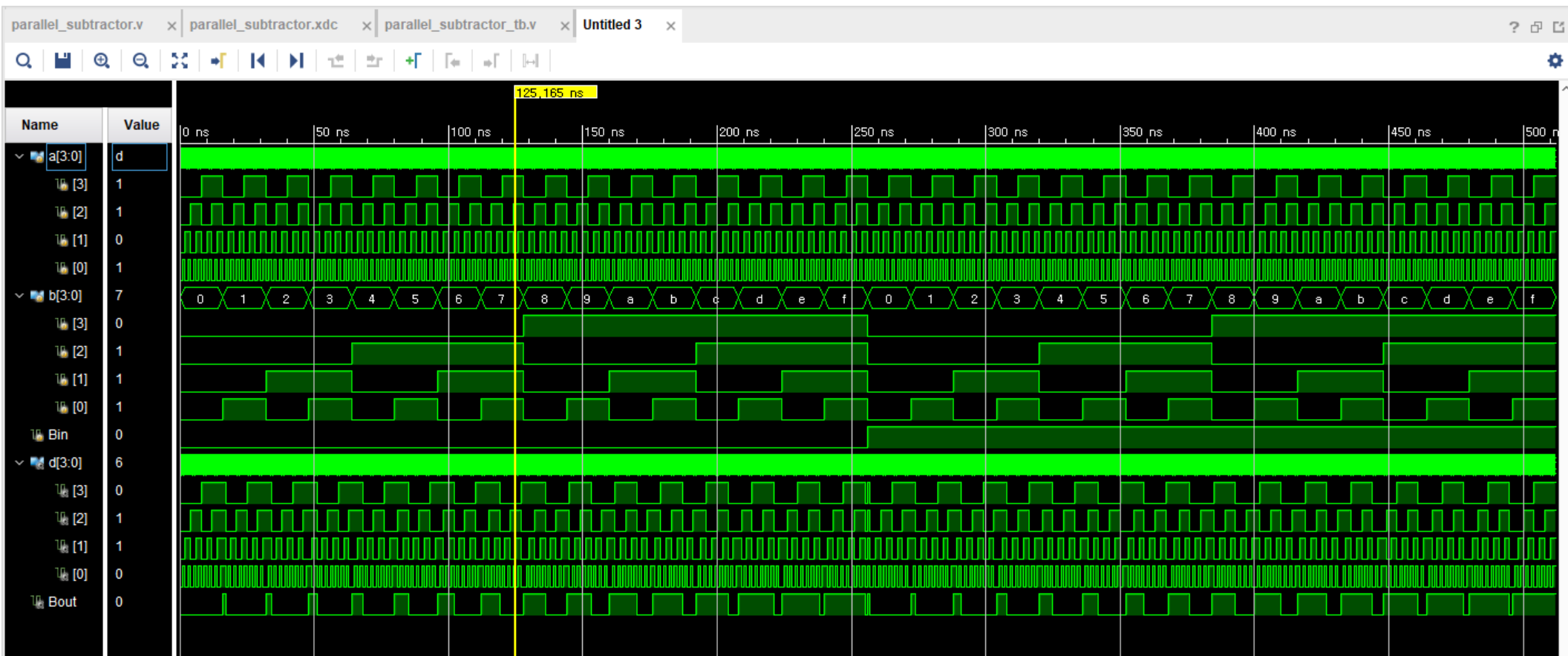
위는 4-bit Binary parallel adder의 verilog design 코드이다.

예시 상황을 들어 A에 1001, B에 1001이 입력된 경우를 들겠다. 이렇게 입력이 되면 Sum 결과는 0010 이며, carry out은 1이 된다. 최종 출력 output은 Cout S3 S2 S1 S0의 순서라고 하면 10010이 되는 것이다. 이 계산 과정은 우선 각 비트가 위치 별로 두 입력 값을 더한다. 하위 Adder에서 발생한 carry는 다음 Adder로 전달되어 비트 입력과 함께 Adder에서 사용된다. 이와 같은 과정이 연결되다가 마지막 4번째 비트들을 계산하는 Adder에서 발생하는 carry는 최종 Cout으로 출력된다. 이를 다룬 모든 경우의 수를 위의 simulation이 담고 있다.

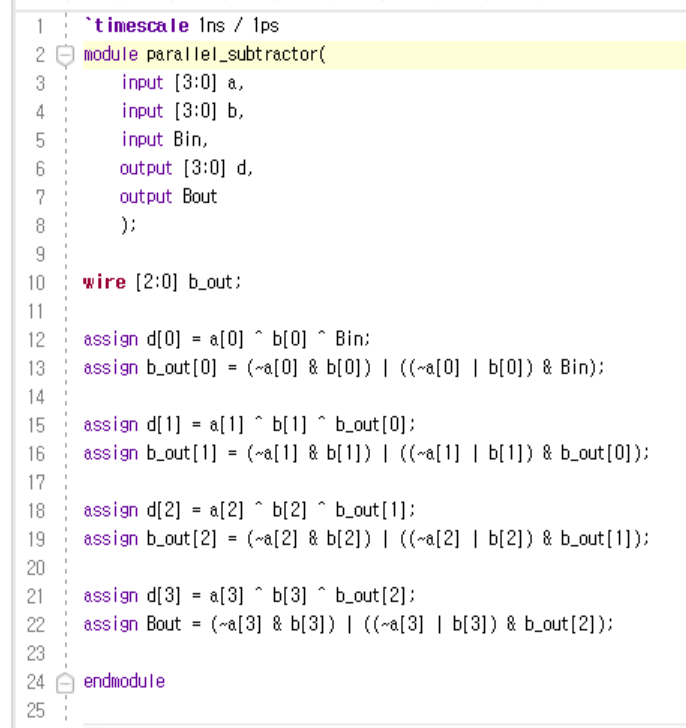
................

**2.**

.......................



위는 4-bit Binary parallel subtractor의 simulation 사진이다.



위는 4-bit Binary parallel subtractor의 verilog design 코드이다.

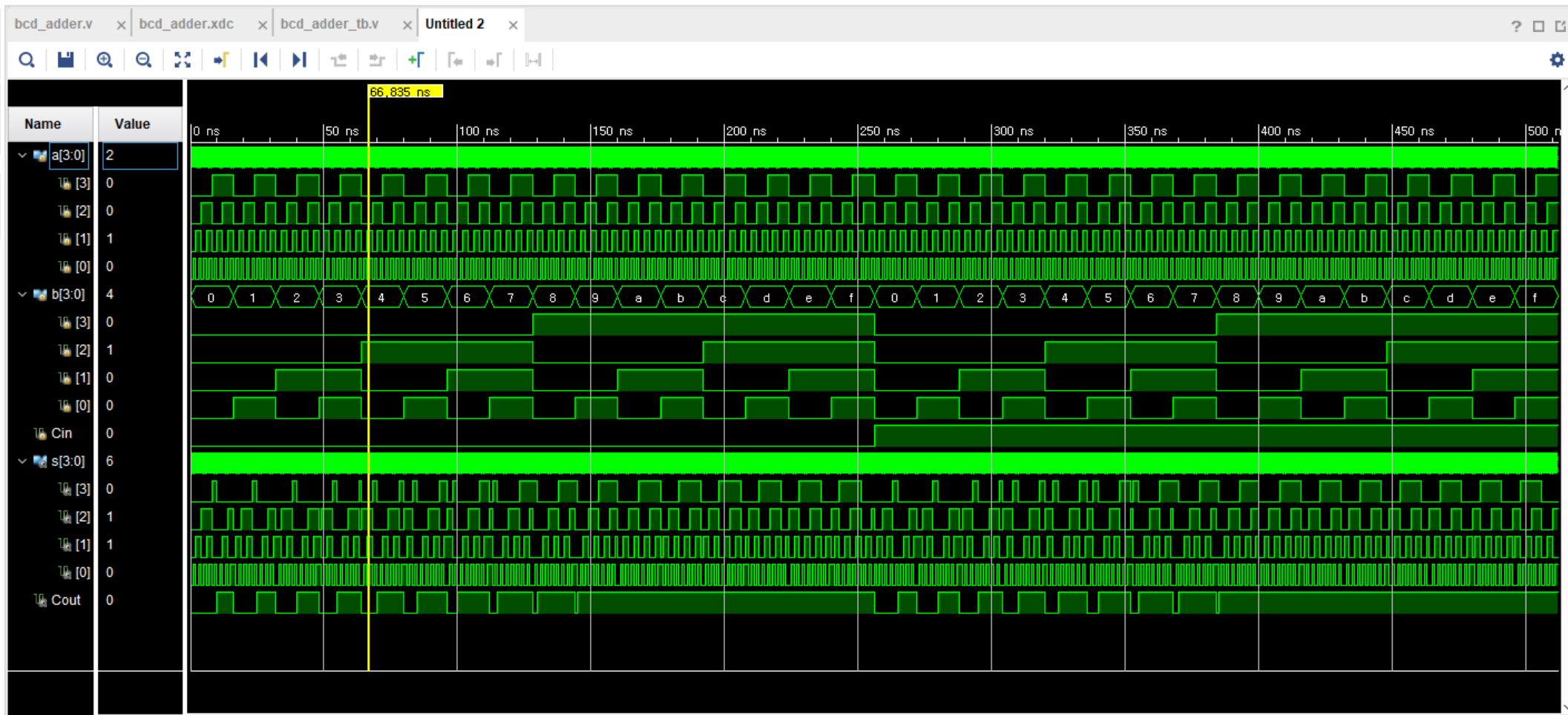
위는 4-bit Binary parallel adder의 verilog design 코드이다.

예시 상황을 들어 A에 0010, B에 1010이 입력된 경우를 들겠다. 이렇게 입력이 되면 Difference 결과는 1000 이며, borrow out은 1이 된다. 최종 출력 output은 Bout D3 D2 D1 D0의 순서라고 하면 11000이 되는 것이다. 이 계산 과정은 우선 각 비트에서 계산이 일어나는데, 차이 계산에 Borrow가 필요할 경우 Borrow를 가져온다. 첫 번째 감산기에서는 B0=0, D0=0, 두 번째 감산기에서는 B1=0, D1=0, 세 번째 감산기에서는 B2=0, D2=0, 마지막 네 번째 감산기에서는 Bout=1, D3=1이 되는 것이다. 이를 다룬 모든 경우의 수를 위의 simulation이 담고 있다.

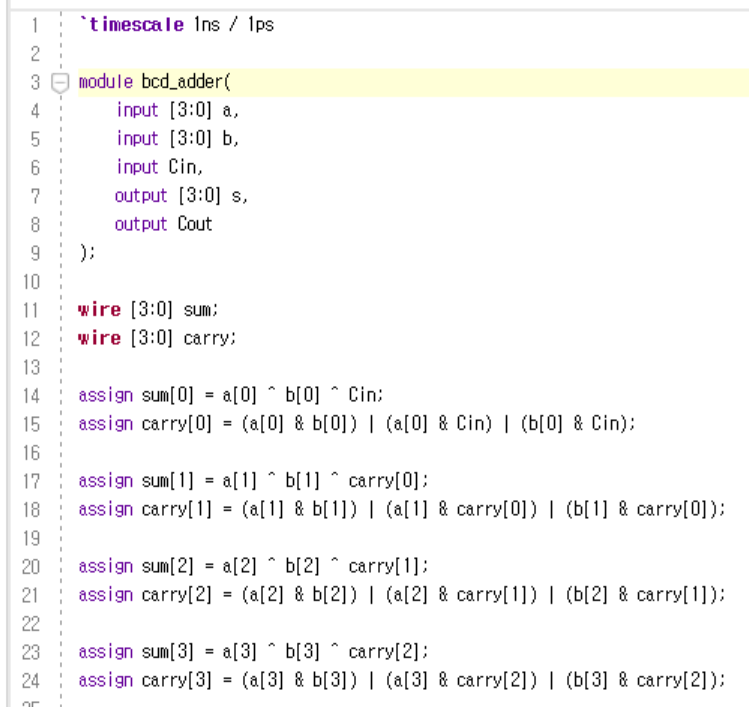
.........................

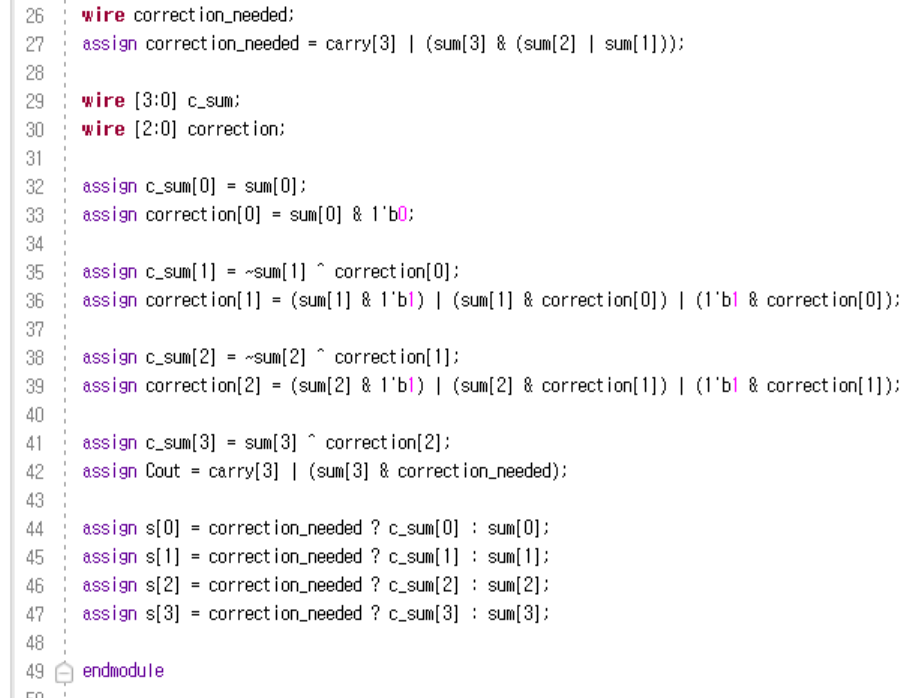
**3.**

.......................



위는 BCD Adder의 simulation 사진이다.





위는 BCD Adder의 verilog design 코드이다.

BCD Adder에서는 십진수를 각 자릿수 별로 4-bit 이진수로 표현하기 때문에 0에서 9까지로 각 4-bit 이진수의 범위가 정해져 있다. 출력에서도 마찬가지이다. 만약 합의 결과가 9를 초과한다면 보정이 발생하는데 6, 즉 이진수로 0110을 더하게 된다. 위의 코드 중 위의 사진은 보정이 필요없는 경우의 출력 비트를 나타내고, 아래의 사진은 보정이 필요한 경우로 보정이 들어간 출력 비트를 표현한다. 처리 과정의 예시로 A = 1001, B = 0011이라고 해보겠다. A는 십진수로 9, B는 십진수로 3이다. 그렇다면 합의 결과는 12로 9를 초과하게 된다. 그렇다면 이를 판별한 코드는 carry를 발생시키고 0110을 결과 값에 더해 Cout = 1, Sum = 0010으로 나타나게 만든다. 최종 출력은 CoutS3S2S1S0이라고 했을 때 10010이라는 것을 확인할 수 있게 된다. 이를 다룬 모든 경우의 수를 위의 simulation이 담고 있다.

.........................

**4.**

.......................

위의 실습들로 병렬 가감산기와 BCD Adder의 작동 원리와 설계 과정에 대하여 알게 되었다. 위의 실습들에서는 사용하는 비트의 수가 이전 실험들에 비하여 많아 simulation 설정에서 애를 먹었다. 더 많은 비트의 수는 어떻게 다뤄야 할지 생각하게 되어 이를 보완할 방법이 필요하다고 여겨진다. 이번에는 각각 입력이 4비트로 주어져 전파 지연에 대해서 크게 느껴지지 않았지만 더 많은 데이터의 경우에서는 느껴질 수 있을 듯하여 이에 대한 논의가 필요할 것으로 보인다.

.........................

**5.**

.......................

이번 실습에서 발생하는 지연을 보완한 회로가 Carry Lookahead Adder라는 것을 알게 되었다. 줄여서 CLA라고 부르는 이 덧셈기는 각 자리의 덧셈 결과에 따라 carry를 계산하여 전달하는 방식이 아니라 carry를 빠르게 예측하여 병렬 처리를 강화하여 속도를 향상시켜준다. 고성능이 필요한 연산 회로에서는 binary parallel adder보다는 CLA를 사용하면 큰 효과를 볼 수 있을 것으로 예상된다.

.........................