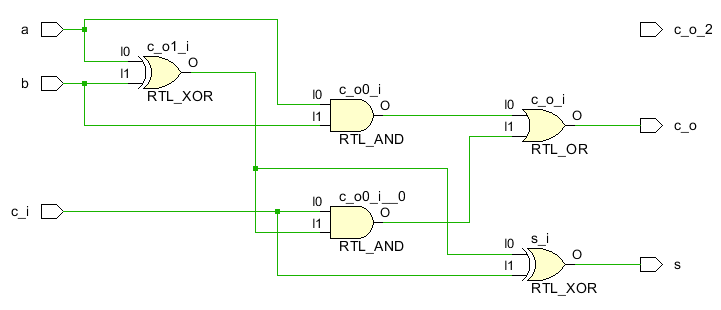
10주차 결과보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20191619 이름: 이동석

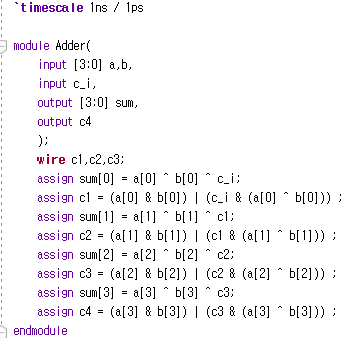
**1. 4bit Binary Parallel Adder**

4비트 병렬 이진 가산기는 1비트의 전가산기를 병렬적으로 연결한 가산기이다. 1비트의 전가산기는 6주차 실습에서 진행했다. 아래 사진은 당시 실습 때 사용했던 Verilog 코드이다. 기본적으로 1비트 전가산기는 이전 비트에서의 자리올림수(Carry)와 서로 더해줄 2개의 비트가 입력으로 들어온다. 각 s와 c\_o의 불 표현식을 구하는 방법은 앞선 주차에서 했으므로 본 보고서에서 설명하지 않겠다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위 코드를 병렬적으로 연결하면 4비트의 이진 병렬 가산기 논리회로 코드를 작성할 수 있다.

**텍스트이(가) 표시된 사진

자동 생성된 설명**

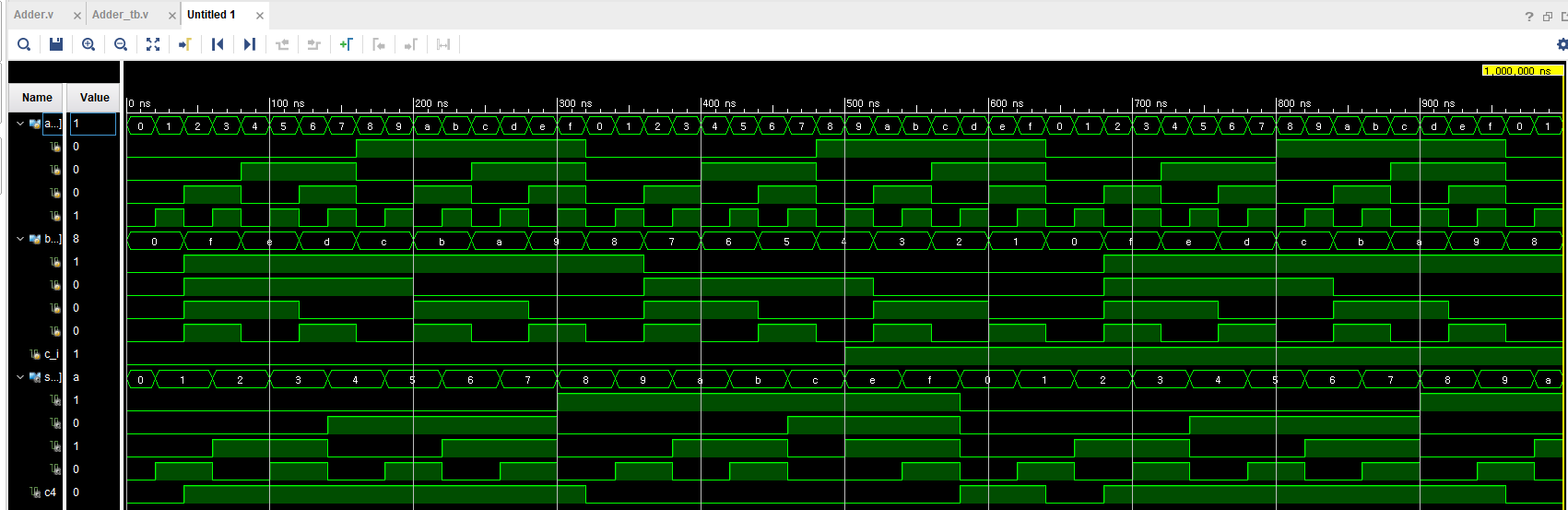
실습에서 변수 a와 b 각각 4비트와 자리올림수 c\_i까지 총 9개의 비트를 입력 받는다. 입력받을 2개의 4비트 a,b는 배열을 이용해 입력받았으며 이때, a[3]과 b[3]이 MSB가 되도록 코드를 작성했다. 출력은 a와 b의 덧셈을 출력할 4개의 비트 sum과 마지막으로 계산되는 자리올림수 c4로 총 5개만 출력하도록 했다. 중간 중간 계산되는 **자리올림수들(c1,c2,c3)은 출력되지 않도록 wire로 선언**해 실습을 진행했다. 출력할 4개의 sum비트 역시 배열로 출력했으며 sum[3]이 MSB가 된다.

위와 같이 변수를 선언한 뒤 가장 하위 비트, LSB부터 코드를 작성한다. 본 코드에서 LSB는 a[0], b[0], sum[0]이 된다. 앞서 1bit 전가산기 코드를 참고하여 작성하면 된다. 따라서, 첫 번째로 계산되는 sum[0]은 c\_i(이전 비트에서 계산된 자리올림수)와 a[0], b[0]의 XOR로 작성된다. 또한, 계산되는 자리올림수는 c1에 할당했다.

1. **sum[0] = c\_i ^ a[0] ^ b[0]**
2. **c1 = (a[0] & b[0] ) || c\_i & (a[0]^b[0])**

같은 방식으로, sum[1]과 c2, sum[2]와 c3, sum[3]와 c4를 작성하면 위 사진의 왼쪽과 같다.

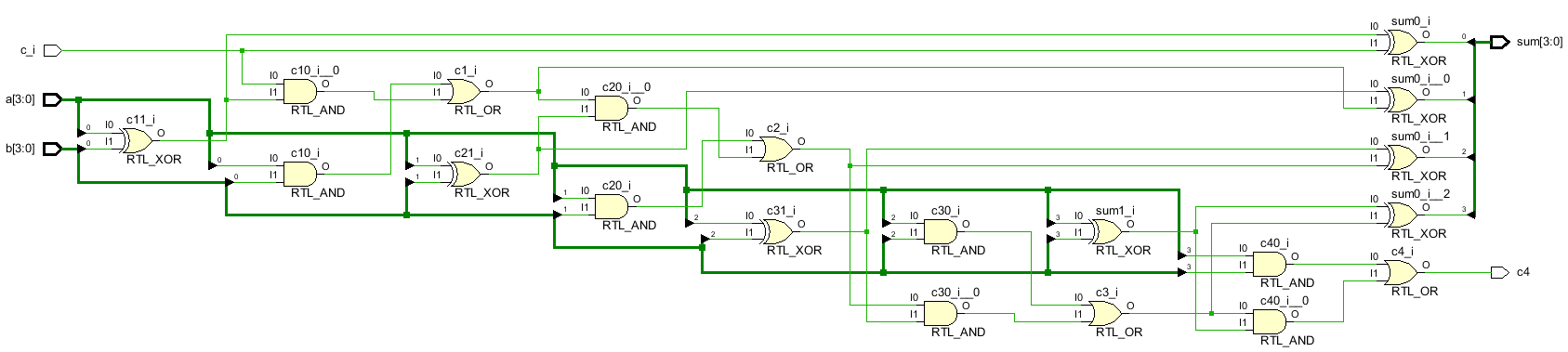
오른쪽 코드는 시뮬레이션 코드이다. a와 b는 4비트이므로 a <= $20 a + 4’b0001 과 같이 작성하였다. c\_i의 경우 0이 입력되는 경우와 1이 입력되는 경우로 나누었다. 아래는 위 시뮬레이션 코드의 결과이다.



우선 a는 0000부터 0001 .. 1111 순으로 1비트씩 증가한다. 반면, b의 경우 0000에서 1111, 1110..으로 1비트씩 감소한다. 너무 많은 경우의 수가 생기기 때문에 모든 경우를 시뮬레이션에 담지 못하였다.

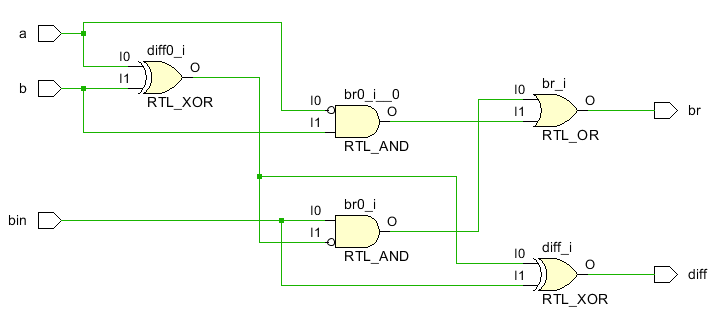
c\_i가 0일 때를 확인해보자. 처음 0000과 0000의 합이 0000이됨을 확인할 수 있다. 0001과 0000의 합 역시 0001로 잘 나오는 것을 확인할 수 있다. b가 1111이 되었을 때, a는 0010이다. 십진수로 바꾸어 계산해보면, 15 + 2 = 17 이며, 이는 이진수로 1 0001 이다. 시뮬레이션 상에서 sum은 0001이고 c4가 1인 것을 확인할 수 있다.

이번에는 c\_i가 1이되는 부분을 확인해보자. 이때, a는 1001, b는 0100임을 확인 할 수 있다. 둘이 더하게 될 경우 9 + 4 = 13으로 이진수 0 1101 이다. 그러나, 추가적으로 c\_i를 더해줘야 하므로 최종 출력되어야 하는 것은 14로 이진수 0 1110 이다. 시뮬레이션에서 sum이 1110이고 c4가 0이므로 올바른 출력값을 가지는 것을 알 수 있다. c\_4가 1이 되는 경우를 확인해보자. a가 1101 이고 b가 0010일 때, 둘의 합은 13 + 2로 15이다. 하지만 현재 c\_i가 1이므로 1111 + 0001 = 1 0000 을 출력해야한다. 시뮬레이션 상에서 c\_4는 1이고 sum의 모든 비트가 0으로 역시 올바른 출력값을 가진다. 시뮬레이션의 결과를 통해 4비트 이진병렬 가산기의 코드는 올바르다고 할 수 있다. 아래 그림은 위 코드의 스케메틱이다.

**2. 4bit Binary Parallel Subtractor**

4비트 병렬 이진 감산기 역시 1비트의 전감산기를 병렬적으로 연결한 가산기이다. 아래 사진은 실습 때 사용했던 Verilog 코드이다. 기본적으로 1비트 전감산기는 이전 비트에서의 빌림수(bin)와 a에서 빼줄 b, 총 2개의 비트가 입력으로 들어온다. 각 diff와 br의 불 표현식을 구하는 방법은 앞선 주차에서 했으므로 본 보고서에서 설명하지 않겠다.

텍스트이(가) 표시된 사진

자동 생성된 설명

4비트 병렬 이진 가산기를 구현했던 것처럼 위 코드를 병렬적으로 연결하면 4비트 병렬 이진 감산기를 구현할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

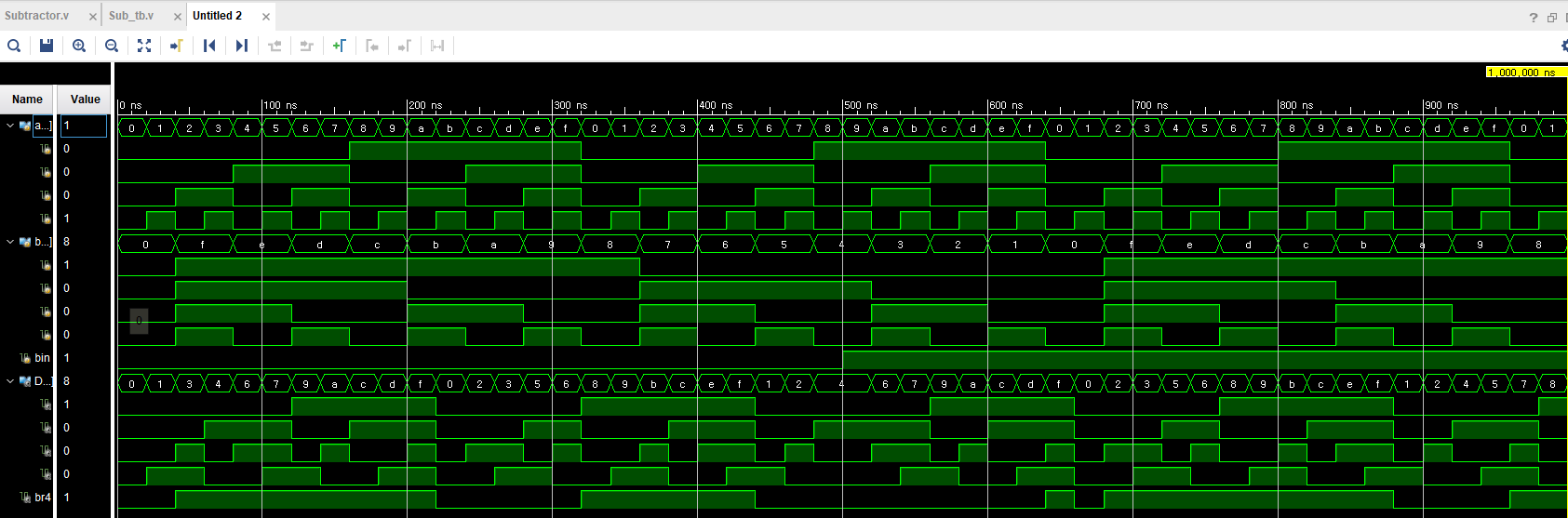
자동 생성된 설명

비슷하게 실습에서 변수 a와 b 각각 4비트와 이전 비트의 빌림수 bin까지 총 9개의 비트를 입력 받는다. 입력받을 2개의 4비트 a,b는 배열을 이용해 입력받았으며 이때, a[3]과 b[3]이 MSB가 되도록 코드를 작성했다. a와 b의 뺄셈을 출력할 4개의 비트 D와 마지막으로 계산되는 빌림수 br4로 총 5개만 출력한다. **빌림수들(br1, br2, br3)은 출력되지 않도록 wire로 선언**해 실습을 진행했다. D의 4개 비트도 배열로 출력했으며 D[3]이 MSB가 된다.

위와 같이 변수를 선언한 뒤 가장 하위 비트, LSB부터 코드를 작성한다. 첫 번째로 계산되는 D[0]은 bin와 a[0], b[0]의 XOR로 작성된다. 계산되는 빌림수는 br1에 할당했다. 이때 a에서 b를 빼는 것이므로 a와 b를 주의해서 작성해야 한다.

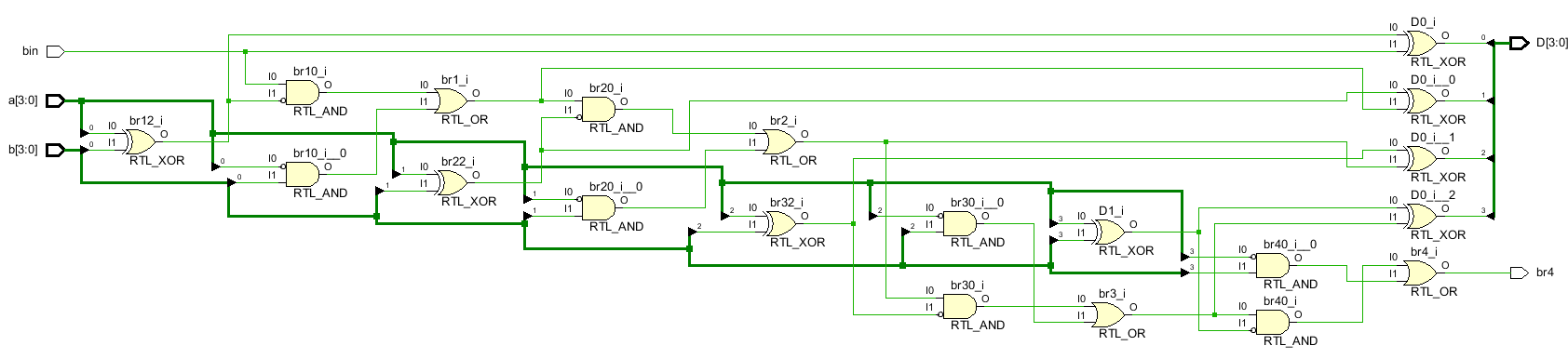
1. **D[0] = bin ^ a[0] ^ b[0]**
2. **br1 = (~a[0] & b[0] ) || bin & ~(a[0]^b[0])**

같은 방식으로, D[1]과 br2, D[2]와 br3, D[3]와 br4를 작성하면 위 사진의 왼쪽과 같다.

오른쪽의 시뮬레이션 코드는 4비트 이진 병렬 가산기와 변수명을 제외하면 동일하게 작성했다. 

모든 경우를 확인해보는 것은 힘드므로, 몇 가지의 경우만 확인해 보도록 하겠다. 우선, 빌림수 bin이 0일 때 확인해보자. 0000에서 0000을 빼면 0000이다. 즉, a가 어떤 이진수던지 b가 0000이면 출력은 a 그대로 나온다. a가 0010이 되면, b는 1111이다. 이때, a가 더 작으므로, 빌림수가 생길 것이다. 따라서 br4가 1이 되어야 하고 이는 1 0010 에서 1111을 빼는 것처럼 생각할 수 있다. 1 0010은 십진수로 18이고 1111은 15이므로 최종 D는 0011을 출력해야 한다. 시뮬레이션 상으로 확인했을 때 올바르게 출력하는 것을 알 수 있다.

이번에는 bin이 1일 때 확인해보자. 바뀌는 시점에 a는 1001이고, b는 0100의 값을 가진다. a가 b보다 더 크므로, 빌림수는 0일 것이다. 또한, 1001은 9이며 0100은 4이다. 따라서 차이는 5로 D는 0101될 것이다. 하지만, 이는 bin이 0일 때로 지금은 bin의 값이 1이기 때문에 추가로 1비트를 빼주어야 한다. 따라서, D의 값이 0100이 될 것이며, 시뮬레이션 상에서 0100인 것을 확인할 수 있다. 빌림수가 생기는 경우를 확인해 보면, 0000과 0000의 뺄셈이다. 앞선 bin이 0일때는 그대로 0000이었지만, bin이 1이기 때문에 0000 – 0001 로 생각해야 한다. 따라서, a가 더 작은값이므로 br4가 1이 될 것이며, D는 1 0000 – 0001의 결과를 출력할 것이다. 이때 값은 15로 1111이며, 시뮬레이션 상에서 br4가 1이며, D도 1111을 올바르게 출력하는 것을 확인할 수 있다. 이 외에도 모두 올바른 값을 출력한다. 따라서, 이진 병렬 감산기 역시 올바르게 구현했다고 할 수 있다. 아래 그림은 스케메틱이다.



**3. BCD Adder**

**텍스트이(가) 표시된 사진

자동 생성된 설명텍스트이(가) 표시된 사진

자동 생성된 설명**

BCD는 예비보고서에서 조사했듯이 십진수로 표현했을 때 값이 10이 넘어갈 경우 각 자릿수를 4비트의 이진수로 표현한다. 즉, 12의 경우 0001 0010으로 표현한다. 기본적으로 0~9까지의 결과 값은 4비트 이진 병렬 가산기와 동일하므로, 이를 사용한 구조로 회로를 구현할 수 있다.

우선, 앞서 구현했던 4비트 이진 병렬 가산기로 두 개의 4비트 a,b의 합과 자리올림수(K)를 구해준다. 이제, 십진수로 9보다 큰 값을 가지는 이진수에 대해 보상회로를 추가해주어야한다. 그렇다면 우선, 10이상의 값을 판별해야한다.

가장 쉽게 알 수 있는 첫번째 방법은 K의 값이 1인지 아닌지를 확인하는 것이다. K는 2^4에 해당하는 비트로 1이라면 항상 10보다 크다. 두 번째는, 4비트 이진수가 10이상이기 위해서는 MSB가 항상 1어야 한다는 점을 이용하는 것이다. 또한, 4비트 이진수에서 적어도 S2나 S1에 해당하는 부분이 1어야 10보다 크다. 따라서, 이를 AND게이트를 사용해 구현할 수 있다. 종합적으로 위의 방법들을 OR게이트를 사용하여 c\_out = K || (S3 & S2) || (S3 & S1) 으로 표현 가능하다.

이제 c\_out이 1일 때는 두 4비트 합의 결과가 10이상이다. 또한, BCD로 표현한 것과 이진 수로 표현한 것의 차이가 0110인 것을 알고 있다. 그러므로, 추가적인 4비트 병렬이진 가산기로 앞선 결과값에 0c\_outc\_out0 을 더해줌으로서 구현할 수 있다. 어떤 이진수에 0000을 더해도 그 결과는 자기 자신이다. 만약 c\_out이 0이라면 0000을 더하는 것이며, c\_out이 1이라면 0110을 더해 원하는 결과를 얻을 수 있다.

왼쪽 코드를 확인해보면 입력은 크기가 4인 배열 a,b와 자리올림수 c\_i이다. 이후, wire로 선언된 S3S2S1S0에 중간 계산 결과를 할당한다. assign K 부분까지는 4비트 이진 병렬 가산기와 동일한 것을 볼 수 있다. 이제 c\_out은 앞서 말한대로 K || (S3 & S2) || (S3 & S1)이다. 이제, 0c\_outc\_out0과 S3S2S1S0의 4비트 이진 병렬 가산기를 다시 코딩해주면 끝이다.

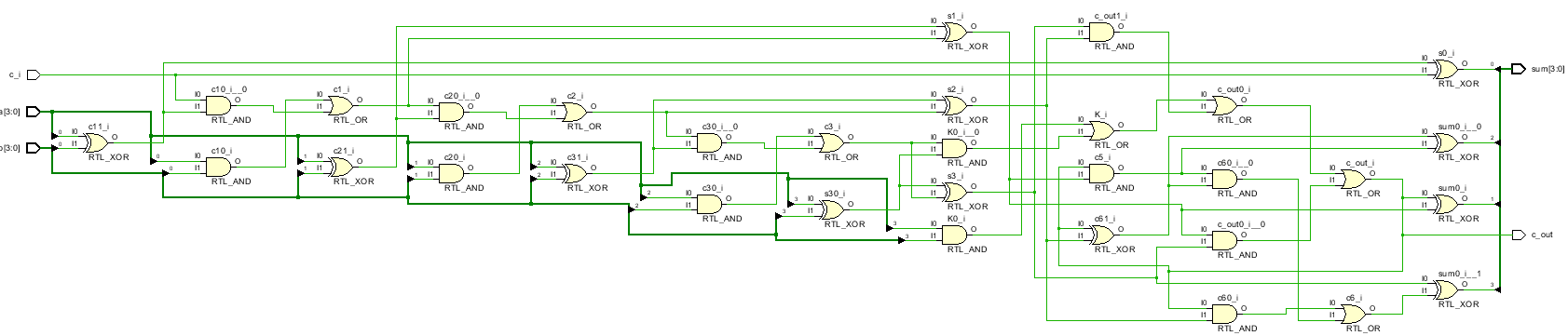
**텍스트이(가) 표시된 사진

자동 생성된 설명**

우선, c\_i가 0일 때의 예시를 확인해보면 0000과 0000의 합은 0000으로 올바르게 출력되었다. 또한, 0001과 0000의 합 역시 9보다 작으므로 0001로 출력되었다. 두 경우 c\_out의 값 역시 0이 출력되었다. 이제 a가 0010이고, b가 1111값을 가질 때 출력되야 하는 값을 생각해보자. 십진수의 덧셈 결과는 2 + 15 = 17 이므로 c\_out은 1이 출력된다. 또한, 이는 BCD로 0001 0111이므로, sum의 결과는 뒤의 4비트인 0111일 것이다. 확인해보면, 올바르게 출력됨을 알 수 있다. 마찬가지로 0011 과 1111의 합의 결과는 1000이다. 0010과 0111을 더하는 경우에는 9를 넘지 않아 c\_out은 0이고 1001을 출력하는 반면, 0011과 0111을 더하는 경우 9를 넘기 때문에 출력의 결과가 0000이 되며 c\_out의 값이 1임을 알 수 있다.

c\_i가 1이되는 경우도 확인해보자. a의 값이 1001이고 b가 0100이다. 마찬가지로 c\_out도 1이 출력되며, 이때 sum은 c\_i가 1이기 때문에 0100을 출력함을 볼 수 있다. 나머지 경우에 대해서도 모두 확인할 수 있으며 시뮬레이션을 바탕으로 아래 진리표를 작성하였다.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **K** | **Z8** | **Z4** | **Z2** | **Z1** | **C** | **S8** | **S4** | **S2** | **S1** | **Dec** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 3 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 4 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 5 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 6 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 7 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 8 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 9 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 10 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 11 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 12 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 13 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 14 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 15 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 16 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 17 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 18 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 19 |

****

**4. 결과 및 논의**

4비트 병렬 이진 가산기와 감산기는 1비트의 감산기와 가산기를 병렬적으로 연결해 구현했다. 이때, 이번 실습에서는 배열의 사용법을 익히고 wire을 사용해 출력하지 않아도 되는 변수를 코드에서 할당하는 방법을 배울 수 있었다. C언어와 문법이 유사하여 이해하는데 크게 어려움은 없었다.

하지만, 예비 보고서에서 조사했던 XOR 2’s complement로 구현했으면 더 좋았을 것 같은 아쉬움이 있다. 코드를 보면, sum과 D는 XOR게이트로 구현되며 변수만 다른 동일한 코드를 가지는 것을 알 수 있다. 이러한 특징과 보수와 XOR게이트의 특징을 이용해 구현한 것이 아래 그림의 시그널 비트 S값에 따라 가산기와 감산기의 역할을 둘 다 할 수 있는 병렬 감가산기이다.

벽, 실내, 방이(가) 표시된 사진

자동 생성된 설명

BCD 가산기는 4비트 이진 병렬 가산기를 이용해 구현하였다. 실습 때 많이 헷갈려서 조금 고전하였다. 하지만, 보고서로 작성하면서 하나씩 확인해보니 보상회로가 어떻게 구현된건지와 왜 2개의 가산기를 사용했는지 회로에 대해 좀 더 이해할 수 있었다. 예비보고서에서 BCD의 감산기에 대해서도 조금 공부했었는데, 이는 가산기 보다 더욱 복잡한 회로를 가질 것이라 생각된다.

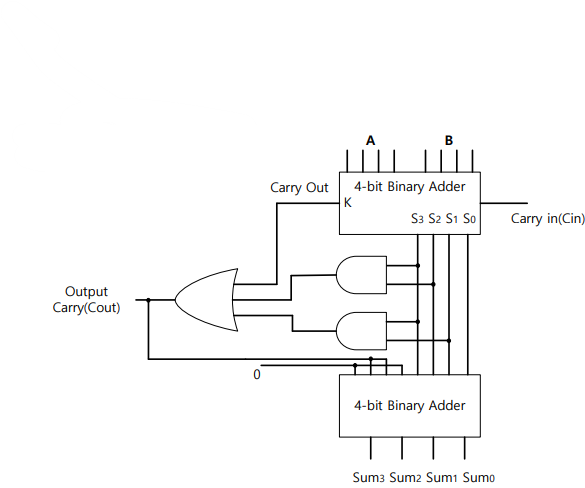
또한 배열을 사용할 경우 시뮬레이션 상에서 위에 비트를 십진수로 변환한 값을 알려준다. 덕분에 값을 확인할 때 일일이 총 8개의 비트를 확인할 필요가 없었다. 앞으로의 코딩에서 배열을 적극적으로 활용하는 것이 여러방면에서 편리하고 좋을 것이라 생각된다.

**5. 추가이론**

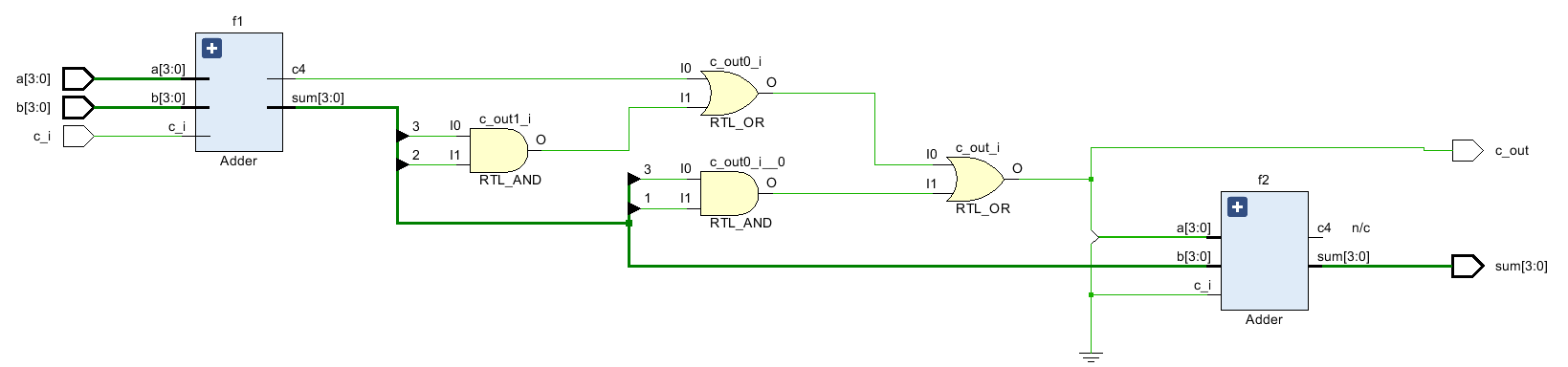
C언어에서 사용자 정의 함수를 만들어 사용하는 것처럼, Verilog에서도 비슷하게 코드를 작성하는 방법이 있다. 앞서 작성한 BCD코드의 스케메틱은 보기 불편하지만, 이런 방식으로 작성할 경우 훨씬 더 깔끔하게 볼 수 있다.

4비트 이진 병렬 가산기의 코드를 그대로 활용하여 아래와 같은 코드로 BCD 가산기를 작성할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

아래 스케메틱을 보면, 조금 더 이해하기 쉽다.



강의자료에서 처럼 우선, a와 b 그리고 c\_in을 입력받는다. 이때 f1은 4비트 이진 병렬 감산기이다. 이때 계산되는 값은 wire로 선언된 s 배열에 할당된다. 이제 carry값 K와 보상회로의 연산으로 c\_out을 얻을 수 있다. 이제 마지막으로 f2에서 다시 4비트 이진 병렬 연산을 하게 된다. 더해줄 4비트는 0 c\_out c\_out 0 이 되므로, wire로 선언한 arr에 값을 할당을 해준다음 f2의 a에 해당하는 파라미터로 넘겨준다. 이제 0c\_outc\_out0과 s와의 병렬 연산을 통해 최종 출력 값 sum을 얻을 수 있다.

BCD뿐만 아니라, 4비트 이진 병렬 감가산기 역시 같은 방식으로 작성 가능하다. 1비트의 전 가산기를 먼저 작성 후, 위와 비슷한 방식으로 구현하면 된다.