10주차 결과보고서

전공: 컴퓨터공학 학년: 2학년 학번: 20191629 이름: 이주헌

1. 4-bit binary parallel adder의 결과 및 simulation 과정에 대해서 설명하시오.

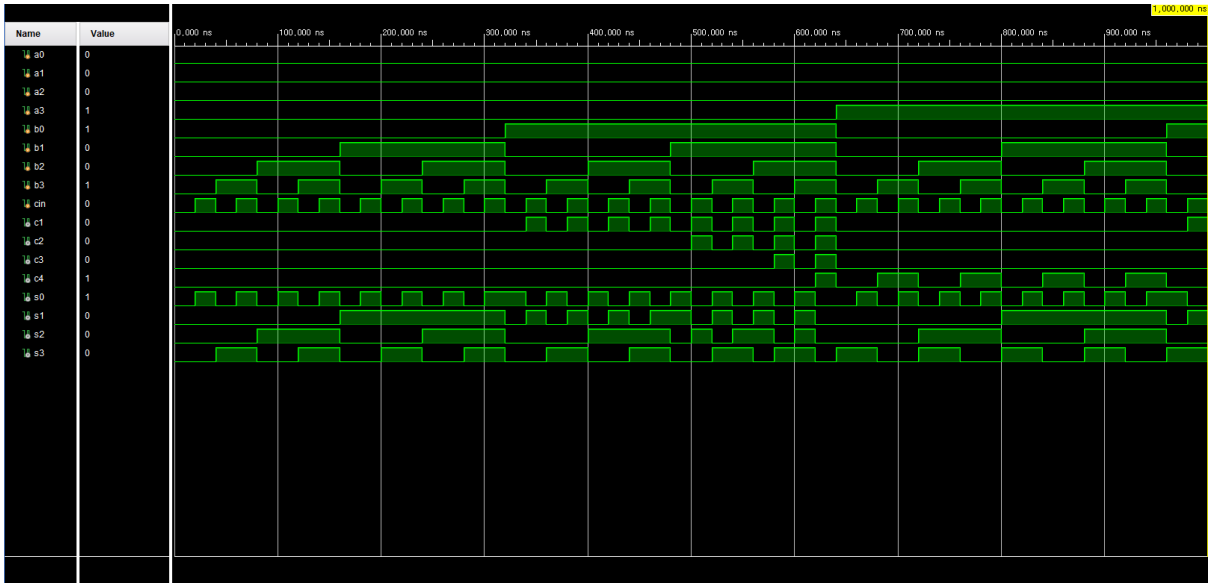
4비트 이진 병렬 가산기를 구하기 위해서 먼저 1비트 전가산기를 먼저 구현해주었다. 이 전가산기는 따로 모듈화하여 구현하였다. 다음은 1비트 전가산기의 소스코드이다.

```
`timescale 1ns / 1ps

module full_adder(a, b, cin, s, cout);
    input a, b, cin;
    output s, cout;

    assign s = a ^ b ^ cin;
    assign cout = (a & b) | (cin & (a ^ b));
endmodule
```

이제 이 전가산기를 4개를 엮어서 다음과 같은 방식으로 4비트 병렬 가산기를 만들 수 있다. 위 모듈을 사용하는 4비트 병렬 가산기의 코드는 다음과 같다.

```
`timescale 1ns / 1ps

module four_bit_parallel_adder(a0, a1, a2, a3, b0, b1, b2, b3, cin, c1,
c2, c3, c4, s0, s1, s2, s3);
    input a0, a1, a2, a3, b0, b1, b2, b3, cin;
    output c1, c2, c3, c4, s0, s1, s2, s3;

    full_adder f0(.a(a0), .b(b0), .cin(cin), .s(s0), .cout(c1));
    full_adder f1(.a(a1), .b(b1), .cin(c1), .s(s1), .cout(c2));
    full_adder f2(.a(a2), .b(b2), .cin(c2), .s(s2), .cout(c3));
    full_adder f3(.a(a3), .b(b3), .cin(c3), .s(s3), .cout(c4));
endmodule
```

위 코드는 각 입력 핀과 출력 핀을 전가산기에 연결하여 구현하기 때문에 훨씬 더 간결하고 읽기 좋은 코드로 구현할 수 있었다. 위 코드를 시뮬레이션하여 동작을 확인해 보면 다음과 같다.



이 경우 혹시 문제가 있을 경우를 대비하여 carry 핀 역시 출력하도록 했는데, 모든 입력 결과에 대하여 알맞은 덧셈 결과와 받아올림 값을 출력한다는 것을 알 수 있다. 또, 실습 당시 FPGA를 이용한 실험에서도 정확한 결과가 출력된다는 것을 알 수 있었다.

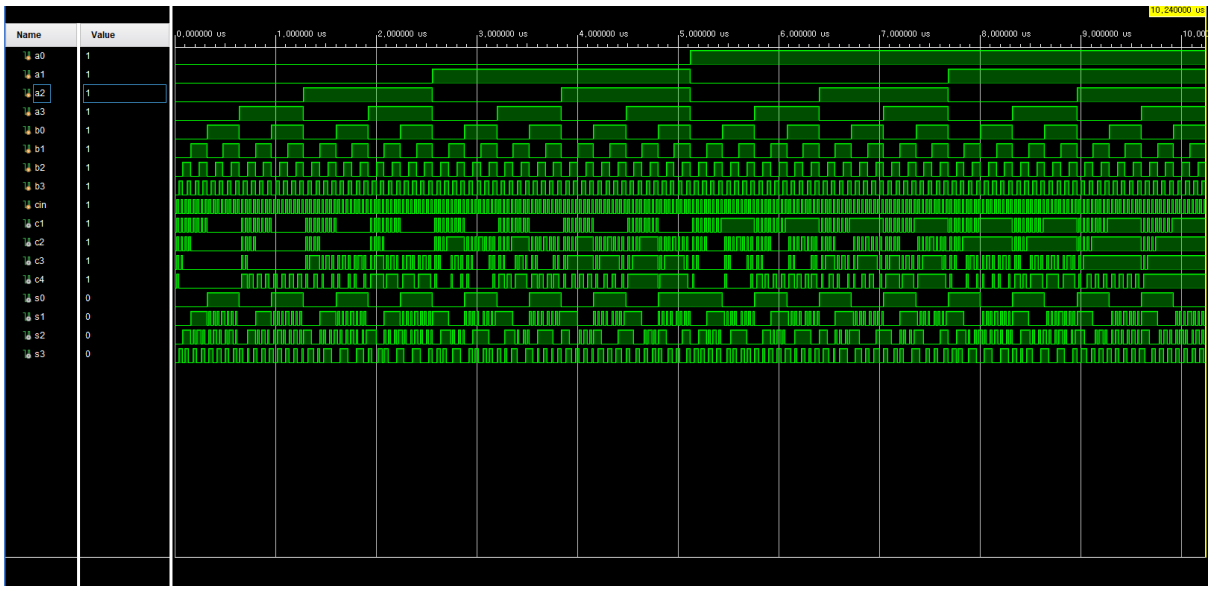
2. 4-bit binary parallel subtractor의 결과 및 simulation 과정에 대해서 설명하시오.

이 회로의 경우, 병렬 가산기와 마찬가지로 전감산기 모듈을 구현하여 계산하는 방식도 있으나, 예비보고서에서 설명한 바와 같이 XOR 게이트를 사용한 2의 보수법을 이용하여 구현하기로 하였다. 이렇게 하면 미리 만들어 둔 전가산기의 코드를 전부 재활용하면서 새로운 기능을 추가할 수 있게 된다. (가)감산기의 코드는 다음과 같다.

```
`timescale 1ns / 1ps

module four_bit_parallel_subtractor(a0, a1, a2, a3, b0, b1, b2, b3, cin,
c1, c2, c3, c4, s0, s1, s2, s3);
    input a0, a1, a2, a3, b0, b1, b2, b3, cin;
    output c1, c2, c3, c4, s0, s1, s2, s3;

    full_adder f0(.a(a0), .b((b0 ^ cin)), .cin(cin), .s(s0), .cout(c1));
    full_adder f1(.a(a1), .b((b1 ^ cin)), .cin(c1), .s(s1), .cout(c2));
    full_adder f2(.a(a2), .b((b2 ^ cin)), .cin(c2), .s(s2), .cout(c3));
    full_adder f3(.a(a3), .b((b3 ^ cin)), .cin(c3), .s(s3), .cout(c4));
endmodule
```

위에서 구현한 병렬 가산기의 코드와 비교해 보면, 대부분의 코드가 같으나 b의 입력과 cin을 XOR한다는 점이 다르다. 이 입력이 예비보고서에서 설명했던 M, 즉 덧셈/뺄셈

선택 핀이다. 여기서 cin을 항상 1으로 묶어주면 항상 뺄셈을 하는 감산기 회로가 완성된다. 이 회로를 시뮬레이션하여 동작을 확인해보면 다음과 같이 정확한 결과가 나온다는 것을 알 수 있다.



감산기 역시 가산기의 경우와 마찬가지로, 문제가 있을 경우 디버깅하기 위해 carry 핀도 표시하도록 했는데, 모든 입력에 대하여 올바른 출력이 나오는 것을 확인할 수 있었다. FPGA에서 역시 올바른 결과가 나온다는 것을 확인할 수 있었다. 또, FPGA를 이용한 실험에서도 같은 결과가 나오는 것을 확인할 수 있었다.

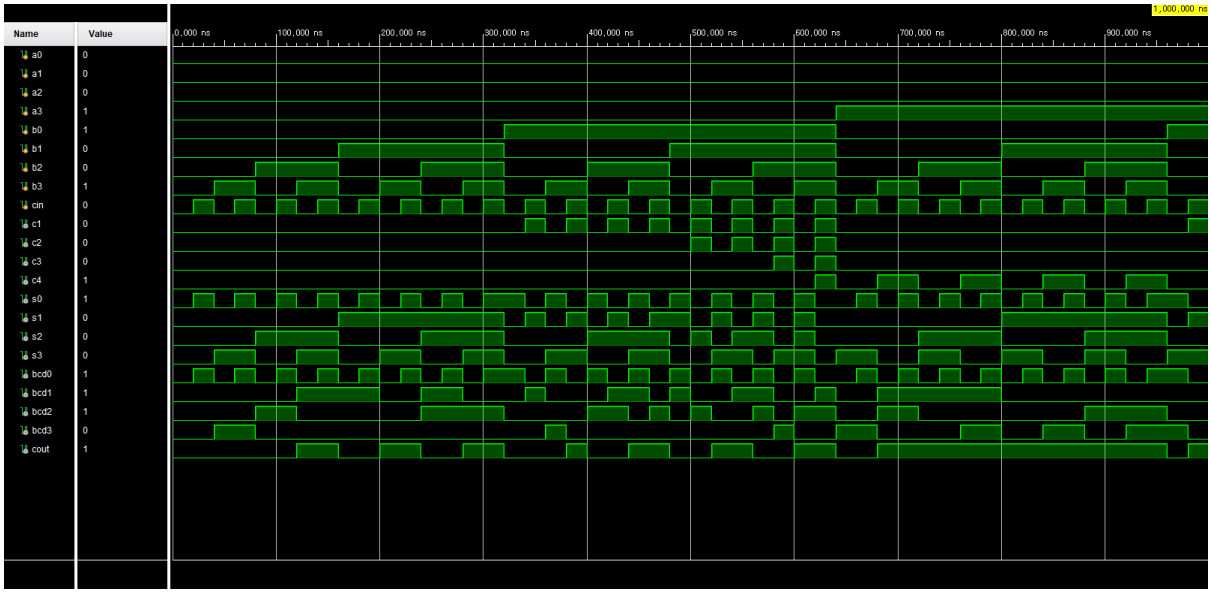
3. BCD adder의 결과 및 simulation 과정에 대해서 설명하시오.

BCD adder는 위에서 구현했던 4비트 병렬 가산기에 BCD 변환 회로를 추가하는 것으로 쉽게 구현할 수 있다. 따라서, 위에서 작성한 four_bit_parallel_adder의 코드를 완전히 재활용하고, 새로운 출력을 추가하는 것으로 쉽게 BCD 가산기를 구현할 수 있었다. 코드는 다음과 같다.

```
`timescale 1ns / 1ps

module bcd_adder(a0, a1, a2, a3, b0, b1, b2, b3, cin, c1, c2, c3, c4,
s0, s1, s2, s3, bcd0, bcd1, bcd2, bcd3, cout);
    input a0, a1, a2, a3, b0, b1, b2, b3, cin;
    output c1, c2, c3, c4, s0, s1, s2, s3, bcd0, bcd1, bcd2, bcd3, cout;

    four_bit_parallel_adder adder(
        .a0(a0), .a1(a1), .a2(a2), .a3(a3),
```

```
        .b0(b0), .b1(b1), .b2(b2), .b3(b3),
        .cin(cin), .c1(c1), .c2(c2), .c3(c3), .c4(c4),
        .s0(s0), .s1(s1), .s2(s2), .s3(s3)
    );

    assign cout = c4 | (s2 & s3) | (s3 & s1);
    assign bcd0 = s0;
    assign bcd1 = s1 ^ cout;
    assign bcd2 = s2 ^ cout ^ (s1 & cout);
    assign bcd3 = s3 ^ ((s2 & cout) | ((s1 & cout) & (s2 ^ cout)));
endmodule
```

먼저 4비트 병렬 가산기를 이용하여 계산을 한 뒤, 각 4비트 값이 10 이상일 경우 다음 비트로 받아올리는 것을 구현하는 것으로 간단하게 가산기를 만들었다. 이 회로의 작동을 검증하기 위해 시뮬레이션해 보면 다음과 같은 결과를 얻을 수 있다.



시뮬레이션의 결과와 실습 시에 확인했던 FPGA의 동작을 비교해 보면, 이미 사용했던 모듈을 재활용하는 방식으로 구현해도 전혀 문제가 없이 정확한 결과가 나온다는 것을 알 수 있었다.

4. 결과 검토 및 논의사항

이번 실습에서는 4비트 가산기와 감산기를 제작하였는데, 지난 6주차에서 작성한 1비트 전가산기를 4개 연결하여 가산기와 감산기를 제작하였다. 또, 감산기를 구현할 때에는 똑같이 6주차 실습에서 작성한 1비트 전감산기를 4개 연결하여 구현할 수도 있었으나, 이번

예비보고서에서 조사한 방법대로 XOR를 이용한 2의 보수법을 이용하여 뺄셈을 구현해 주었다. 감산기를 사용하는 편이 간단할 수는 있으나, 이 방법을 사용하면 하나의 회로를 이용하여 4비트 덧셈과 뺄셈을 모두 구현할 수 있기 때문에 이 방법을 구현해 보았다.

마지막으로 작성했던 BCD 가산기의 경우 일반적인 4비트 연산을 진행한 이후 10진수로 표현하였을 때 10 이상의 값이 나온다면 다음 비트로 받아올리는 방식으로 구현하였다. 십진수 수는 0부터 9까지만 표현할 수 있기 때문에 표현하기 위해서는 4비트가 필요하다. 그런 점에서 이번에 구현한 BCD 가산기는 실습 1번과 2번에서의 1비트 전가산기와 하는 역할이 비슷하다고 할 수 있다. 즉, 이 BCD 가산기를 여러 개 엮어 여러 자릿수를 가진 십진수 수를 더할 수 있다.

5. 추가 이론 조사 및 작성

BCD adder가 있다면 당연히 BCD subtractor도 존재한다. 물론 모든 경우의 수에 대한 진리표를 그려 BCD 감산기를 구현할 수도 있지만, 4비트 감산기를 구현할 때와 같은 이유로, 이번에는 BCD 가산기를 이용하여 BCD 감산기를 구현하고자 한다.

BCD는 기본적으로 십진수 자릿수 하나를 4비트를 이용해 나타내는 것이므로, 만약 감산기를 구현하려면 4비트 감산기 때와 달리 2의 보수를 사용할 수 없다. 대신, 10의 보수를 사용해야 한다. 10의 보수를 사용하는 방법은 2의 보수를 사용할 때와 비슷하다.

1. 자릿수 하나를 고른다. 이 경우에는 현재 BCD 가산기에 두 번째로 입력된 4비트 수이다.
2. 9에서 현재 자릿수를 뺀다. 이것이 9의 보수이다.
3. 9의 보수 결과값에 1을 더한다. 이것이 10의 보수이다.
4. 첫 번째로 입력된 4비트 수와 두 번째로 입력된 4비트 수를 더한다.
5. 여기서 발생하는 받아올림은 필요없는 받아올림이므로 버린다.

이를 회로도로 구현하면 다음과 같다.

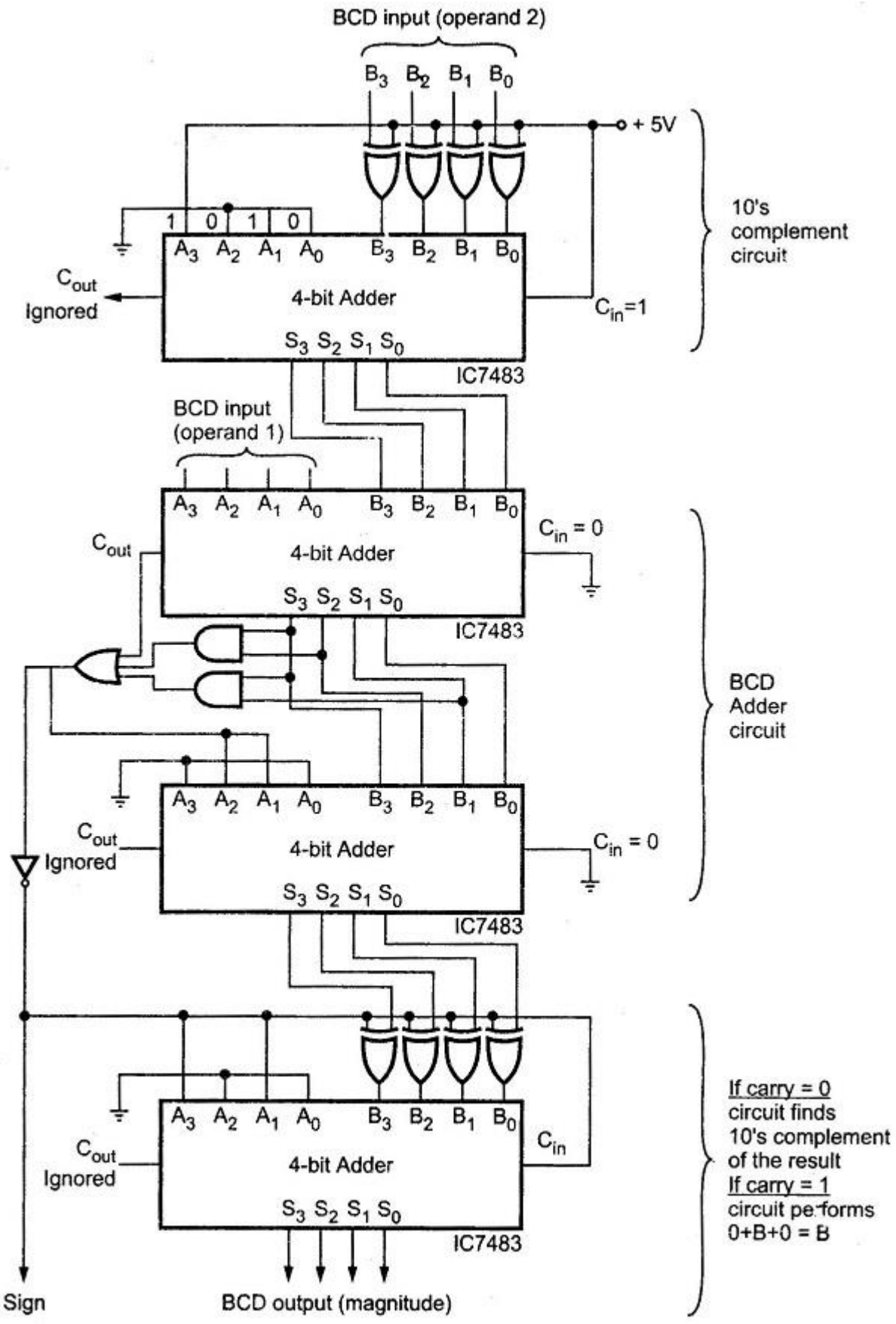


Fig. 3.35 4-bit BCD subtractor using 10's complement method

이 회로는 기본적으로 BCD 가산기를 내장하고 있는 회로이기 때문에, 필요하지 않은 회로는 비활성화하고, 뺄셈을 할 때만 10의 보수를 계산하는 회로를 활성화하는 방식으로 구현하게 되면 우리가 2번 실습에서 만들었던 것과 같은 범용 BCD 가감산기를 구현할 수 있다.