10주차 결과보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20211522 이름: 김정환

**1.**

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

4bit Binary Parallel Adder를 구현하기 위해서 우선 1bit Full Adder 먼저 구현해보았다. 해당 코드는 아래와 같다.

텍스트, 폰트, 스크린샷, 화이트이(가) 표시된 사진

자동 생성된 설명

이를 이용하여 4bit Adder를 구현하는데, 위에서 사용한 식을 참조하여 각각의 bit별로 똑 같은 구조의 식을 사용하여 구현하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

여기서 a, b는 각각 더해야 하는 4 bit 수를 의미하며, input으로 주어지는데 4bit이므로 [3:0]의 배열로 주어졌고, input으로 들어오는 cin은 하나만 주어지므로 이전의 다른 변수들처럼 하나의 변수로 놔두었다. output으로는 각 bit별로 sum과 자리올림이 있으므로 각각은 [3:0]의 배열로 s, c로 표기하였다. 식의 구조는 위에서 구현한 s와 cout을 사용한 것을 볼 수 있다. 또한 c가 0부터 계산되며 다음으로 값을 넘겨주며 연산되는 Ripple Carry 방식을 사용하였다.

Testbench 코드는 다음 코드를 사용하였다.

텍스트, 스크린샷, 영수증이(가) 표시된 사진

자동 생성된 설명

해당 코드를 Simulation한 결과는 아래와 같다.

스크린샷, 디스플레이이(가) 표시된 사진

자동 생성된 설명스크린샷, 회로이(가) 표시된 사진

자동 생성된 설명스크린샷, 텍스트, 디스플레이이(가) 표시된 사진

자동 생성된 설명

해당 Simulation을 살펴보면, A=0101, B=0010, CIN=0의 입력이 주어졌을 때, S=0111로 나오는 것을 확인할 수 있고, 해당 경우에는 자리올림이 일어나지 않으므로 C는 모두 0이 나오는 점을 볼 수 있다. 다른 경우에 대해 알아보면, A=1001, B=0011, CIN=0의 입력이 주어졌을 때 S=1100으로 나오는 것을 볼 수 있고 자리올림이 2 bit에서 일어나므로 C는 0011로 나타나는 점을 확인할 수 있어 Adder가 맞게 구현된 것을 볼 수 있다. 또한 FPGA로 연결하여 결과 확인 시에도 Simulation과 동일하게 나오는 것을 확인하였다.

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

**2.**

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

4bit Binary Parallel Substractor를 구현하기 위해 1bit Full Substractor를 먼저 구현해보면 다음과 같다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이 식을 바탕으로 4bit로 확장하여 나타내면 다음과 같은 Verilog 코드로 나타낼 수 있다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Adder일 때와 구조에서 크게 바뀌는 점이 없기에 변수명은 동일하게 사용하였으나 cin은 처음 입력으로 들어오는 빌림수를 의미하고, output인 s와 c는 각각 4bit의 차와 4개의 빌림수를 의미한다. Adder와의 비교를 해보면, 차는 합과 연산이 동일하고 빌림수의 경우에 ~이 두 곳에 추가되는 점에 차이가 있다. 이 역시 Ripple Carry 즉 빌림수가 첫 자리부터 다음으로 결과를 넘겨주는 방식으로 구현하였다.

다음으로 Simulation 시 사용한 Testbench 코드는 다음과 같다.

텍스트, 영수증, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Simulation의 결과는 다음과 같다.

스크린샷이(가) 표시된 사진

자동 생성된 설명스크린샷, 전자제품이(가) 표시된 사진

자동 생성된 설명스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명

우선 A=1011, B=1011, CIN=0으로 입력이 들어오는 지점을 살펴보면, S=0000으로 나오는 것으로 같은 경우에 0으로 결과가 정확히 나오는 것을 확인할 수 있다. 이외에는 A=1000, B=0101, CIN=0으로 입력이 주어질 때 S=0011로 나오는 것을 확인하였고 자리내림 역시도 0, 1, 1, 1로 잘 동작하는 것도 볼 수 있다. FPGA에서도 동일하게 동작하는 것을 확인했다.

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

**3.**

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

BCD Adder는 일반적인 Adder에 더해서 고려해야 할 점이 추가된다. 우선 0~9는 그대로 나타내지만 10 이상의 결과에 대해서는 10에 대해서는 앞의 한 자리로 빼고 나머지 남은 숫자를 표기하는 방식이다. 10을 넘는 것을 판정하는 변수를 cout으로 표기했는데, 이는 마지막 자리올림이 1이거나, 합의 결과의 제일 높은 2 bit가 둘 다 1이거나, 제일 높은 bit와 2를 나타내는 bit가 1인 경우에 넘으므로 이 경우들을 OR Gate로 묶어서 표기해주었다. 그리고 10 이상인 경우들은 일의 자리를 얻기 위해서는 6, 즉 0110을 더하는 형태로 표기된다. 따라서 해당 부분에 대한 연산까지 구현하는데, 이는 이전 실습에서 사용한 Adder를 이용하였다. 따라서 이전의 Adder의 결과 s와 0(cout)(cout)0의 합이 이루어지는데, 이는 10이 넘는 경우 0110, 아닌 경우는 0000으로 합해지는 구조로 이루어진다. 해당 Verilog 코드는 아래와 같다.

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

해당 코드에서 cout 변수의 식 아래가 BCD로 보정해주는 보상회로에 대한 부분이다. 다음으로 Simulation을 위한 Testbench 코드는 다음과 같다.

텍스트, 스크린샷, 영수증, 폰트이(가) 표시된 사진

자동 생성된 설명

이를 이용해 Simulation한 결과는 아래와 같다.

스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명스크린샷, 컴퓨터이(가) 표시된 사진

자동 생성된 설명스크린샷, 텍스트, 디스플레이이(가) 표시된 사진

자동 생성된 설명

이를 이용하여 진리표를 작성하면 아래와 같다. 여기서 K는 16 이상을 표기하기 위한 추가 bit이고, C는 BCD에서 Carry를 나타내기 위한 핀이다.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| K | Z8 | Z4 | Z2 | Z1 | C | S8 | S4 | S2 | S1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |

19보다 큰 수에 대해서는 Carry는 1bit로 설정하였으므로 의미가 없으므로 해당하는 부분까지만 나타냈다.

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

**4.**

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

본 실험에서는 1bit Full Adder를 이용하여 4bit Binary Parallel Adder를 구현하는 것과 1bit Full Subtractor를 사용하여 4bit Binary Parallel Subtractor를 구현하였다. 또한 추가적인 보상회로를 이용하여 4bit Binary의 합을 BCD로 표기할 수 있도록 하는 회로를 구현하였다. 해당 실습에서 처음으로 Array를 사용하여 논리식을 작성하였고, 따라서 Verilog에서 Array를 어떤 방식으로 사용하는지에 대해서 학습할 수 있었다. 이번 실습에서는 Array는 사용하였지만 동일하게 반복되는 식에 대해서 다른 언어들의 함수처럼 모듈을 생성하는 과정은 거치지 않았는데 해당 방법에 대해서도 알아보고 이를 통해 더욱 간단히 코드를 표기할 수 있는 방향으로 개선할 점이 존재한다.

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

**5.**

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

Adder를 활용하여 만들 수 있는 회로 중에서 곱셈기도 존재한다. 각각 곱하는 두수를 A1A0, B1B0의 2bit Binary로 가정하고 곱하는 과정을 살펴보면

텍스트, 폰트, 스크린샷, 도표이(가) 표시된 사진

자동 생성된 설명

위의 과정처럼 식을 나타낼 수 있다. 이는 각 자리별로 AND 연산을 진행한 후 칸을 한 칸씩 밀어준 후 더해주는 과정이라고 할 수 있다. 따라서 이를 회로도를 이용하여 그려보면 Half Adder 부분은 블록으로 표시했을 때 아래와 같다.

도표, 기술 도면, 평면도, 개략도이(가) 표시된 사진

자동 생성된 설명

2 bit끼리의 곱셈이 아닌 3bit, 4bit 또는 그 이상으로 갔을 경우, Half Adder, Full Adder, AND Gate 등이 더 많이 사용되어 구현된다. 이에 대해서는 위에서 식으로 나타낸 것처럼 계산하는 것을 이용하여 구현하는 방식이다.

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

참고자료)

<https://electbros.com/multiplier/>

https://beginagain22.tistory.com/15