10주차 예비보고서

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

**1.**

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

4-Bit Adder 및 Subtractor의 이진 병렬 연산이란 해당하는 회로들을 연결하여 나타내는 것을 통해 더 높은 bit의 Adder 및 Subtractor를 만들어 내는 것을 의미한다. 따라서 4-Bit의 연산을 하기 위해서 이전 실습에서 진행했던 Full Adder의 구현을 한 것을 하나의 블록으로 생각하여 각각의 출력으로 나오는 자리올림을 다음 Full Adder에 입력으로 넘겨주는 형태로 연결하는 구조이다. 각각의 Full Adder를 블록으로 표시하여 나타낸 도식은 다음과 같다.

도표, 스케치, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명

이러한 방식으로 구성했을 때 자리올림 수의 계산에 대해 고려해보면, 뒤로 갈수록 앞의 자리올림이 계산되고 난 후에나 계산이 이루어지는 것을 볼 수 있다. 따라서 Gate를 지날 때마다 delay가 발생하는데 1개의 Gate당 발생하는 delay를 ∆라고 하면 (2n+4) ∆만큼의 delay가 발생한다. Subtractor에 대한 예시 그림을 들어보면 아래와 같다.

도표, 스케치, 라인, 기술 도면이(가) 표시된 사진

자동 생성된 설명

이 경우에는 나머지는 Full Subtractor를 병렬로 연결하지만 첫 블록에는 Half Subtractor를 사용하는 것을 볼 수 있다.

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

**2.**

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

Look Ahead Carry는 앞에서 언급된 자리올림 계산에 있어서 발생하는 delay를 미리 식을 완성하는 방식으로 처리하는 방식이다. 각각 Carry generate와 Carry propagate를 이용하여 식을 정리하는데, 이는 각각 , 라고 했을 때 로, 로 표기할 수 있다. 이를 통해 자리올림을 순서대로 계산하면, 의 형태로 나타낼 수 있고 4 bit의 경우로 예시를 들어보면, 으로 첫 자리올림을 계산하고, 다음으로 으로 앞의 결과를 대입하여 정리할 수 있다. 다음으로 으로 정리할 수 있고 마지막으로 가 되는데, 이들을 최종 정리된 식으로 회로로 나타낸 것이 Look Ahead Carry이다. 이들은 앞의 입력을 기다리지 않고 회로를 연결해두었기 때문에 회로는 복잡해지지만 지연시간이 없어진다. 이를 블록으로 나타낸 형태는 다음과 같다.

텍스트, 도표, 폰트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

위의 그림에서는 c0가 첫 자리올림 입력으로 들어오므로 위의 식과는 c에서는 하나씩 숫자가 차이가 나는 모습을 볼 수 있다.

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

**3.**

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

우선 2’s complement 가감산은 뺄셈을 계산하는 것에 있어서 더하기로 바꿔서 계산하는 것을 의미한다. 이는 binary에서 음수를 표기 시에 2’s complement가 해당 수의 반대 부호의 수를 나타내는 규칙에 따라 성립한다. 양수와 양수의 뺄셈의 경우, - 기호 다음에 있는 수의 2’s complement를 구한 후 이를 앞의 수와 더하면 뺄셈을 한 것과 같은 것을 볼 수 있다. 다음으로 양수에서 음수를 빼는 경우, 음수의 2’s complement를 구하면 같은 절대값의 양수를 얻을 수 있어 가산으로 처리할 수 있다.

이를 활용한 것으로 XOR Gate를 활용하여 가감산이 모두 가능하도록 하는 것은 입력되는 두 수 중 하나에 또 다른 입력 M을 추가하여 이것과 1 bit씩 XOR 연산을 하도록 하는 방식이 있다. A와 B가 더할 두 수라고 했을 때 B에 XOR Gate를 연결한 후에 가감산이 이루어진다고 하면, 우선 M이 0일 때는 수가 바뀌지 않으므로 그대로 가산이 이루어진다. M이 1인 경우에는 B에서 1인 bit가 0이 되고 0인 bit가 1이 되는 것을 볼 수 있고, 이에 따라 B의 2의 보수의 형태와 A를 가산하는 형태가 되는데, 이는 뺄셈과 같은 효과를 가지므로 감산의 기능도 구현한다. 이를 블록 형태의 회로도로 나타내면 아래와 같다.

도표, 라인, 텍스트, 번호이(가) 표시된 사진

자동 생성된 설명

위의 그림에서 보면 신호로 들어오는 M이 첫 자리올림(내림수)로도 입력에 들어가는 점도 확인할 수 있다.

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

**4.**

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

BCD는 10진수의 각 자리를 4 bit 이진수로 나타내는 코드를 의미한다. 따라서 이는 일반적인 8-4-2-1 BCD의 경우에 4 bit의 이진수 중에서 0000부터 1001까지의 수만 사용하고 나머지는 사용하지 않는다. 이 경우에는 이진수를 더해주는 경우에 9를 넘는 크기의 연산이 일어날 때 앞에 4 bit를 추가하여 다음 자리를 표기할 수 있도록 하는 것이 필요하다. 예시를 들어보면 8 + 5 = 13의 연산을 표기하는 경우, 1000 + 0101로 나타내지만 1101로 표기하는 것이 아닌 0001 0011의 형태로 표기하는 것을 볼 수 있다. 다른 형태의 BCD 역시도 그대로 더하는 것이 아닌 자리 수에 맞게 4bit씩으로 나타내는 규칙으로 표기한다.

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

**5.**

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

ALU는 arithmetic logic unit을 줄인 것으로 가감산과 같은 산술연산과 논리연산을 계산할 수 있는 회로이다. 구조를 간단하게 나타내면 아래와 같은 그림으로 나타낼 수 있다.

텍스트, 도표, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

Opcode를 통해서 어떤 연산을 할지를 입력된다. 이는 ALU의 목적에 따라서 달라지는데, 여기에는 덧셈뿐 아니라 곱셈이나 나눗셈 역시도 포함될 수 있다. 또한 Status에서는 overflow가 일어났는지 여부와 같은 연산 결과의 상태를 나타낸다. Overflow 외에는 계산 결과의 음수 여부, 0 여부 등 종류에 따라 더 많은 정보를 표기하기도 한다. 위에서 말한 기능을 담당하는 구성요소들로는 우선 보수를 만드는 보수기, 오버플로우를 검출하는 오버플로우 검출기 등이 있다.

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

**6.**

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

ALU에 오버플로우 검출기가 있다고 했는데, 여기서는 가산 시에 오버플로우를 어떤 방식으로 검출하는지에 대해서 알아보았다. 가산 시의 오버플로우는 n bit의 수끼리 더하였을 때 n+1 bit가 되는 경우라고 할 수 있을 것이다. 즉 양수끼리 더해서 음수가 나오거나 음수끼리 더해서 양수가 나오는 경우가 해당한다. 4 bit끼리의 덧셈을 예시로 해보면, 0111 + 0001은 7 + 1이지만 덧셈 시 1000으로 -8이 나온다. 다음으로 1000 + 1111은 (-8)+(-1)임에도 0111로 7이 나온다. 이러한 방식으로 경우를 따져볼 때, 식을 세워보면 자리올림 수 중에서 마지막 2개가 서로 다를 때 오버플로우가 발생하는 것을 볼 수 있다. 따라서 검출기가 오버플로우 발생 시 1을 출력한다고 하면, 아래와 같은 회로도를 그릴 수 있다.

도표, 기술 도면, 평면도, 라인이(가) 표시된 사진

자동 생성된 설명

XOR Gate를 활용하여 위와 같이 오버플로우를 검출한다. 이는 더 큰 bit에서도 적용할 수 있고 마지막 두 자리올림으로 비교하는 점을 똑같이 적용하면 된다.

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