10주차 예비보고서

전공: 컴퓨터공학과 학년: 3학년 학번: 20191599 이름: 송경호

**1.**

**1-1) 4-bit Adder**

4-bit Adder의 설명에 앞서 먼저 과거에 실습했던 전가산기에 대해 간단히 복습하자면 전가산기는 두 개의 1 bit 입력을 Carry bit와 함께 계산해서 결과 비트와 Carry bit를 출력하는 회로이다. 결과 비트인 S(Sum) bit는 A ⊕ B ⊕ Cin으로 이루어지며 Cout 비트는 AB + Cin(A ⊕ B)로 얻어진다. 앞서 전가산기에 대한 실습을 진행했기 때문에 자세한 설명은 생략하며 회로와 진리표는 다음과 같다.

그래픽이(가) 표시된 사진

자동 생성된 설명

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | | | **Output** | |
| **A** | **B** | **Cin** | **Sum** | **Cout** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

전가산기는 Carry 비트를 입력 받아 다음 Carry 비트까지의 계산이 가능하므로 이를 연결해서 이어지는 연산이 가능하다. 따라서 4-Bit Adder는 4개의 전가산기를 병렬 연결해서 구성할 수 있다.

**텍스트, 도표, 폰트, 평면도이(가) 표시된 사진

자동 생성된 설명**

최초의 입력 a1과 b1 에 대한 s와 cout를 계산한 뒤 s bit는 출력하고 cout bit는 다음 full adder의 cin으로 입력된다. 이러한 과정은 4개의 full adder를 통과하며 일어나게 되며 결과적으로 두개의 4 bit 입력을 더하는 것이 가능해진다. 그러나 그림을 보면 알 수 있듯이, 상위 비트의 연산을 위해선 하위 비트의 carry 값이 필요하기 때문에 gate delay가 발생하고, 시간이 지연된다는 단점이 있다.

**1-2) 4-bit Subtractor**

4-bit Adder와 마찬가지로 과거에 실습했던 전감산기에 대해 간단히 복습하자면 전감산기는 두 개의 1 bit 입력을 Borrow bit와 함께 계산해서 결과 비트와 Borrow bit를 출력하는 회로이다. 결과 비트인 D(Difference) bit 역시 세 입력을 XOR 시킨 A ⊕ B ⊕ Br0으로 이루어지며 다음 Br 비트는 A’B + (A ⊕ B)‘Br0의 연산으로 얻어진다. 앞서 전감산기에 대한 실습은 진행했기 때문에 자세한 설명은 생략하며 회로와 진리표는 다음과 같다.

도표, 스케치, 라인, 평면도이(가) 표시된 사진

자동 생성된 설명

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | | | **Output** | |
| **A** | **B** | **Br0** | **Br** | **D** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

전감산기는 Borrow 비트를 입력 받아 다음 Borrow 비트까지의 계산이 가능하므로 이를 연결해서 이어지는 연산이 가능하다. 따라서 4-Bit Subtractor도 마찬가지로 4개의 전감산기를 병렬 연결해서 구성할 수 있다.

**도표, 평면도, 기술 도면, 스케치이(가) 표시된 사진

자동 생성된 설명**

최초의 입력 A1과 B1 에 대한 D와 Brout를 계산한 뒤 s bit는 출력하고 Brout bit는 다음 full subtractor의 Brin으로 입력된다. 이러한 과정은 4개의 full subtractor를 통과하며 일어나게 되며 결과적으로 두개의 4 bit 입력을 빼는 것이 가능해진다. 그러나 그림을 보면 알 수 있듯이, 상위 비트의 연산을 위해선 하위 비트의 borrow 값이 필요하기 때문에 gate delay가 발생하고, 시간이 지연된다는 단점이 있다.

**2.**

위에서 설명한 이진 병렬 연산은, 상위 bit에서 하위 bit의 값을 사용하기 때문에, 하위 bit에서의 연산을 기다리며 gate delay가 발생한다는 단점이 있다. 그러나 look ahead carry는 이름에서도 알 수 있듯이, 초기 입력을 바탕으로 하위 비트에서 전달될 값을 미리 예측하여 gate delay가 발생하지 않는다.

Look ahead carry에는 G와 P라는 임의의 값이 필요하다. 이때 입력 비트 A와 B에 대해 G = AB이며 P = A ⊕ B 이다. 먼저 G는 자리 올림 값이 반드시 발생하는지를 확인하는 비트이다. 만약입력 비트 A와 B가 모두 1이라면 반드시 자리 올림 값이 발생하므로 G는 두 입력의 AND 연산으로 구할 수 있다. 다음으로 P는 추가적인 자리 올림 값이 발생하는 지를 확인하는 비트이다. 두 입력 A, B와 함께 이전 단계의 carry 비트가 더해질 것이므로 A와 B가 다른 경우에만 carry 비트를 통해 추가적인 자리 올림 값이 발생할 수 있다. 따라서 P는 두 입력의 XOR 연산으로 구할 수 있다.

다음으로 G와 P를 바탕으로 Si와 Ci+1을 구할 수 있다. 이때 Si와 Ci+1은 전가산기에서 사용했던 출력과 다음 Carry를 의미하며 Si = Pi⊕Ci, Ci+1= Gi + (Pi⦁Ci)로 나타낼 수 있다.

위 두 수식을 바탕으로 4-Bit의 연산을 계산한다고 할 때, 각 각의 Carry bit는 다음과 같이 수식으로 나타낼 수 있다.

텍스트, 폰트, 화이트, 대수학이(가) 표시된 사진

자동 생성된 설명

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

다시 말해 모든 Ci+k 비트는 Ci를 통해 구할 수 있으며, 다시 말해 C1~C4를 C0를 통해 구할 수 있다는 것을 의미한다. 결과적으로 앞서 언급되었던 gate delay 문제를 해결할 수 있어, 빠른 시간안에 비트 연산이 가능해진다. 그러나 위에서 언급된 연산들을 모두 수행해야 하기 때문에 회로의 복잡도 및 면적이 증가한다는 단점이 여전히 존재한다.

**3.**

**도표, 스케치, 라인, 기술 도면이(가) 표시된 사진

자동 생성된 설명**

2’s complement 가감산은 전가산기 4개를 연결한 Full Adder에 XOR 연산을 활용하여 가산과 감산이 모두 발생할 수 있도록 한 회로이다. 회로의 그림에서 우측 위의 S bit에 숫자에 따라 가산기로 활용될지 감산기로 활용될지 결정되며 sign bit가 0일 경우 가산기, 1일 경우 감산기이다.

이러한 구현은 XOR의 성질을 통해 구현된다. XOR은 X⊕0=X 이고, X⊕1=X’인 성질을 가지고 있고 Sign Bit는 모든 XOR 회로에 연결되어 있기 때문에 만약 Sign bit가 0일 경우 Bi 비트는 그대로 Full Adder에 입력 되지만, Sign bit가 1일 경우 Bi의 비트가 뒤집어진 상태로 Full Adder에 입력된다.

추가적으로 Sign bit가 Full Adder의 C0, 즉 첫번째 Carry bit로 입력되는 것에 주목해야 한다. 만약 Sign bit가 0일 경우 A0과 B0의 가산이 정상적으로 이루어진다. 만약 Sign bit가 1일 경우 Full Adder에는 A0, B0’ 그리고 1이 입력된다. 이때 B0’의 보수에 1을 더해주는 것은 B0에 2의 보수를 취하는 것과 같은 의미이므로 결과적으로 Full Adder에 A0와 B0에 2의 보수를 취한 값이 입력되며, 이들의 가산은 A0와 B0의 감산과 같은 효과를 발생시킨다.

이와 같이 XOR을 활용하여 하나의 회로로 가산과 감산을 모두 수행이 가능하다.

**4.**

BCD는 Binary Coded Decimal의 약자로, 십진수를 이진 코드로 나타낸 것을 의미한다. 10진수를 이진수로 변환했을 때, 각 자리의 숫자가 앞에서부터 8,4,2,1의 가중치를 갖는 것과 마찬가지로 BCD도 가중치를 갖게 되어 8421 코드라고도 불린다.

|  |  |
| --- | --- |
| **십진수** | **BCD 코드(4bits)** |
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |

한가지 주목해야 할 점은 BCD 코드는 4 bit임에도 불구하고 9까지의 십진수만 나타낸다는 점이다. 따라서 BCD 코드를 사용하는 연산은 일반적인 이진법 연산과는 다른 형태를 띈다. 우선 십진수가 9 이상일 경우 BCD 코드는 십진수의 각 자리 수를 그대로 나타낸다. 예를 들어 13의 경우 십의 자리 수 1과 일의 자리 수 3이므로, BCD 코드로 나타냈을 때 0001 0011로 변환된다.

다음으로 BCD의 연산 과정에 대해 13 + 7을 예로 든다. 13과 7을 BCD 코드로 표현 시. 각 각 0001 0011과 0000 0111으로 나타나진다. 이때 두 bit를 더할 경우 0001 1010을 얻을 수 있다. 이때 BCD 코드에는 1010이 존재하지 않으므로 6에 대응되는 숫자인 0110을 더한다.

6을 더하는 이유는 4 bit의 이진수로 나타낼 수 있는 숫자는 16가지이고 BCD 코드는 10개의 숫자를 나타낼 수 있으므로 Carry를 위해 6을 더해야 하기 때문이다. 따라서 0001 1010에 0110을 더하면 carry가 발생하며 0010 0000이 되며 이는 20에 대응되는 것을 알 수 있다. 또한 13 + 7의 합이 20이므로 정상적인 연산이 수행됐 음을 확인할 수 있다

**5.**

ALU는 쉽게 말해 계산기로, 입력 받은 자료에 대해 산술, 논리, 비트, 수의 대소 판단등의 연산을 처리한다. 이때 ALU는 반드시 레지스터들과 조합해야 데이터 처리가 가능하다.



또한 ALU는 산술 연산 장치를 중심으로 논술 연산 장치, 보수기, 검출기 등으로 구성되어 있다.

텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

산술 연산 장치는 덧셈, 뺄셈, 곱셈, 나눗셈과 같은 산술 연산을 수행하며, 내부는 전가산기를 연결한 병렬 가산기로 구현되어 있다.



논리 연산 장치는 OR, AND, XOR, NOT과 같은 논리 연산을 수행하며 멀티플렉서를 사용하여 하나의 논리 연산 결과값을 생성한다.



상태 레지스터는 여러 비트로 이루어져 있으며 연산 결과에 의해 set 된다. overflow bit는 산술 연산에 대한 결과가 레지스터의 자리수를 초과할 때 1로 set되며 overflow가 발생했음을 알린다. zero bit는 연산 결과가 0일 경우 set 된다. sign bit는 결과의 부호를 나타내는 비트로써 0이면 양수, 1이면 음수를 나타낸다.



보수기는 감산 시에 빼는 수를 보수로 변환하고 가산기에 입력해야 되는 경우 사용된다.



**6.**

**6-1) Single Level CLA**

도표, 폰트, 타이포그래피이(가) 표시된 사진

자동 생성된 설명

전가산기를 4개 연결하여 4 bit adder를 구현하듯이 4-bit CLA를 연결하여 16-bit CLA를 구현할 수 있다. 회로가 직관적이고 단순히 연결하기 때문에 구현이 쉽지만 Full Adder를 이용했을 때 발생했던 Gate Delay가 발생하게 된다.

**6-2) 2 Level CLA**

텍스트, 도표, 스케치이(가) 표시된 사진

자동 생성된 설명

Single Level CLA의 단점인 Gate Delay를 보완한 CLA로써 하나의 계층을 추가로 두어 추가적인 연산과정을 거쳐 carry bit를 예상하게 된다. Single Level CLA보다 빠른 연산 속도를 갖지만 동시에 회로의 면적이 증가하고 복잡도가 높아진 다는 단점이 있다. 추가적인 계층에 대한 예상식은 아래와 같이 구성된다.

텍스트, 폰트, 스크린샷, 친필이(가) 표시된 사진

자동 생성된 설명