6주차 예비보고서

전공: 물리학과 학년: 3학년 학번: 20191286 이름: 김나현

1. 전 가산기 및 반 가산기에 대해 조사하시오(예시 포함).

조합 논리 회로는 이전의 출력에 관계없이, 현재의 입력 조합에 대해서만 출력이 결정되는 논리 회로로, 외부로부터 인가되는 입력뿐만 아니라 이전의 출력이 현재의 출력에 영향을 미치는 순서 논리 회로와 구분됩니다. 순서 논리 회로에는 플립플롭, 레지스터, 카운터 등이 있고, 조합 논리 회로에는 전가산기, 반가산기, 전감산기, 반감산기, 디코더, 인코더 등이 있습니다.

조합 논리 회로의 예시 중 전가산기와 반가산기에 대해 알아보겠습니다. 가산기란 여러 입력에 대해 덧셈 연산을 수행하는 논리 회로입니다. 전가산기(full adder)은 하위 자릿수에서 올라온 자리 올림수(cin)와 두 개의 입력 A, B를 포함하여 이진수의 한 자릿수를 연산하는 기능을 합니다. 이때, 입력으로는 A, B, cin가 있고 연산 결과, 생성되는 두 개의 출력은 cout과 s가 있습니다. cout은 상위 자릿수를 연산할 때 사용되는 값이며 s는 연산 결과, 나타나는 해당 자리의 자릿수를 의미합니다. 세 입력에 대해 참인 입력이 두 개 이상인 경우, cout은 1이 되고 s는 세 개의 입력 중 참인 입력이 짝수이면 0을, 홀수이면 1을 출력합니다. 이러한 연산의 진리표를 작성해보면 다음 <Table 1>과 같습니다.

<Table 1> 1 bit 전가산기의 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | cin | s | 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 |

이제 위의 <Table 1>과 카르노 맵을 이용하여 s와 cout에 대한 논리식을 찾아보겠습니다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| cin\AB | 00 | 01 | 11 | 10 |
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

<Figure 1> s에 대한 카르노 맵

s=A’Bcin’+AB’cin’+A’B’cin+ABcin이므로 x’y+xy’는 x⊕y라는 xor 연산을 이용하여 정리하면 s는

s=cin’(A’B+AB’)+cin(A’B’+AB)=cin’(A⊕B)+cin(A⊕B)’=cin⊕A⊕B라고 나타낼 수 있습니다. 이는 xor 연산의 성질인 입력 중 참인 입력의 개수가 홀수개일 때 1을 출력한다는 사실과 <Table 1>의 진리표를 비교해보았을 때, 올바른 논리식이라는 것을 알 수 있습니다.

마찬가지로, cout의 논리식을 얻기 위해 카르노 맵을 만들면 다음 <Figure 2>와 같습니다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| cin\AB | 00 | 01 | 11 | 10 |
| 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 |

<Figure 2> cout에 대한 카르노 맵

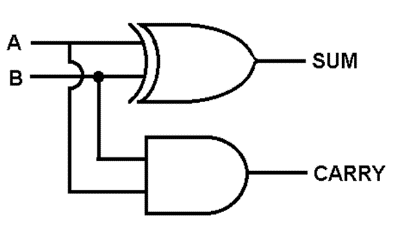
위의 <Figure 1>과 달리 2x1의 크기의 prime implicant을 얻을 수 있으므로, cout은 cout=Bcin+AB +Acin=AB+cin(A+B)(or AB+cin(A⊕B))이라고 정리할 수 있고 A, B, cin 중에 두 개 이상이 1이면 무조건 1이므로 올바른 논리식을 얻은 것을 확인할 수 있습니다.

반면, 반가산기(half adder)는 하위 자릿수에서 올라온 자리올림수 없이 두 입력 A, B를 이용해 상위 자릿수로 올림할 올림수 C와 해당 자리의 자릿수 S를 계산하여 출력하는 가산기입니다. 반가산기의 진리표는 다음 <Table 2>와 같이 나타낼 수 있습니다.

<Table 2> 반가산기의 진리표

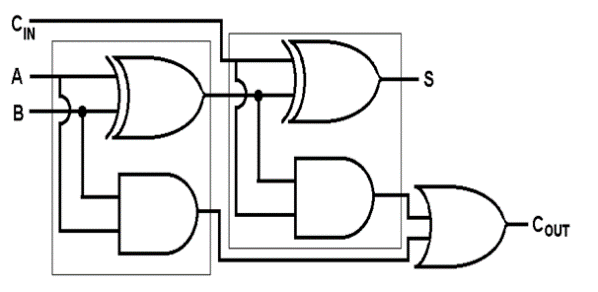
|  |  |  |  |
| --- | --- | --- | --- |
| A | B | S | C |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

이때, 두 입력의 합 S는 두 입력 중 1이 하나만 있을 때, 1을 출력하고 상위 자릿수로 올림할 올림수 C는 두 입력이 모두 1일 때만 1을 출력하고 두 출력의 논리식을 세워보면 S=A’B+AB’, C=AB라고 정리할 수 있습니다. 하지만 반가산기는 전가산기와 달리, 하위 올림수 없이 덧셈 연산을 하는 불완전한 가산기이기 때문에 반가산기라는 이름을 사용합니다.

반가산기는 XOR 게이트와 AND 게이트를 이용하여 다음 <Figure 3>과 같이 표현이 가능하고,

<Figure 3> 반가산기 회로

전가산기는 반가산기 회로를 2개 이용하여 다음 <Figure 4>와 같이 표현이 가능합니다.



<Figure 4> 전가산기 회로

1. 전 감산기 및 반 감산기에 대해 조사하시오(예시 포함).

조합 논리 회로의 예시로 전가산기, 반가산기 이외에도 뺄셈 연산을 담당하는 전감산기와 반감산기가 있습니다.

순서를 달리하여, 반감산기에 대해 먼저 설명해보겠습니다. 반감산기란 두 개의 입력 X, Y에 대해 D와 B를 출력하는 것으로, 이때 D는 두 입력의 차인 Difference의 약자이며 B는 두 입력을 뺄 때 상위 자릿수에서의 빌림수인 Borrow의 약자입니다. 따라서, 반감산기의 진리표를 먼저 작성해보면 다음 <Table 3>과 같이 나타낼 수 있습니다.

<Table 3> 반감산기의 진리표

|  |  |  |  |
| --- | --- | --- | --- |
| X | Y | B | D |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 |

두 출력의 논리식을 구해보면 B=X’Y이고, D=XY’+X’Y=X⊕Y라고 할 수 있습니다.

다음은 전감산기에 대해 알아보겠습니다. 전감산기는 두 개의 입력을 이용하는 불완전한 감산기인 반감산기와 달리, 하위 자릿수의 빌림수를 사용하여 뺄셈 연산을 하는 완전한 감산기라고 할 수 있습니다. 전감산기의 진리표를 작성해보면 다음 <Table 4>와 같습니다.

<Table 1> 1 bit 전감산기의 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | Bi | B | 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 |

따라서, B와 D의 논리식을 구하기 위해 카르노 맵을 작성해보면 다음과 같습니다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bi\XY | 00 | 01 | 11 | 10 |
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

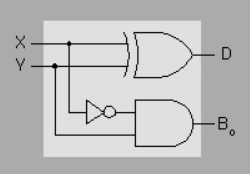
<Figure 5> D에 대한 카르노 맵

위의 <Figure 5>은 전감산기의 출력 중 s를 구하는 것과 동일하므로 D는 D=X⊕Y⊕Bi라고 표현할 수 있습니다. 이로부터 입력 중 참인 입력이 홀수 개이면 뺄셈 연산 결과, 해당 자릿수가 1이 됨을 알 수 있습니다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Bi\XY | 00 | 01 | 11 | 10 |
| 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 |

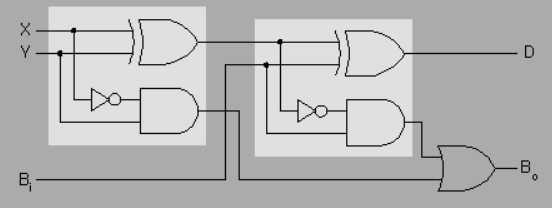
<Figure 6> B에 대한 카르노 맵

위의 <Figure 6>로부터 빌림수 B는 B=X’Bi+YBi+X’Y라고 정리할 수 있습니다. 반감산기는 XOR 게이트, NOT, AND 게이트를 이용하여 다음 <Figure 7>과 같이 표현이 가능하고,



<Figure 7> 반감산기 회로

반가산기 회로 두 개를 이용하여 전가산기를 표현할 수 있는 것과 동일하게, 전감산기는 반감산기 회로를 2개 이용하여 다음 <Figure 8>와 같이 표현이 가능합니다.



<Figure 8> 전감산기 회로

비트 연산에서 뺄셈은 덧셈과 달리 복잡합니다. A에서 B를 빼기 위해서는 B의 보수를 구해서 A-B를 A+(-B)와 같은 형태로 바꿔준 후 덧셈을 수행해야 합니다. 이진수의 여러 음수 표현 방법 중 컴퓨터의 뺄셈에서 사용하는 2의 보수를 구하기 위해서는 B의 모든 자릿수를 반전시킨 후, 1을 더해주면 됩니다.

1. BCD 가산기에 대해 조사하시오.

BCD란 Binary Coded Decimal의 약자이고, 한국어로는 이진화 십진수라고 말할 수 있습니다. 0부터 9까지의 10진수를 0000부터 1001까지 우리가 잘 알고 있는 2진수 표기법으로 나타낸 것은 이러한 BCD 코드의 한 예시입니다.

10 이상의 10진수를 표현하기 위해 0부터 9까지의 4 bit 이진수를 사용한다고 가정하면 739는 0111 0011 1001이라고 저장될 것입니다. 이때, 빈칸으로 구분한 4 bit 이진수는 앞에서부터 백의 자리, 십의 자리, 일의 자리 값을 의미하므로 3 digit, 즉 백의 자리의 십진수를 나타내려면 12 bit의 이진수가 필요하다는 것을 알 수 있습니다.

앞서 말한 것처럼 우리가 2진수라고 말했을 때 바로 떠오르는 BCD 코드는 아래 <Figure 9>에서 8421 code입니다. 8421 code가 4비트의 상위 자릿수부터 8, 4, 2, 1이라는 자릿값(weight)을 부여하여 4비트의 2진수를 표현한 방식이라는 것을 이미 알고 있다면, 같은 방식으로 이진수를 나타내는 5421 code를 이해하는 것은 어렵지 않습니다. 2421 code를 이해하기 위해서는 1의 보수를 알고 있어야 하는데, 1의 보수란 각 자릿수의 비트를 반전시킨다고 이해하면 됩니다. 예를 들어, 8421 code의 8인 1000을 이용하여 1의 보수를 구해보면 0111이 됩니다. 하지만 2421 code의 10진수 n을 얻기 위해서는 8421 code의 n의 1의 보수를 구하는 것이 아니라 9-n의 1의 보수를 구하는 것입니다. Excess 3 code는 단순히 8421 code의 방법에 3을 더하여 0은 0011로, 9는 1100로 표현하는 방법입니다. 아래 <Figure 9> 이외에도 십진수를 이진수로 나타내는 방법은 다양합니다.

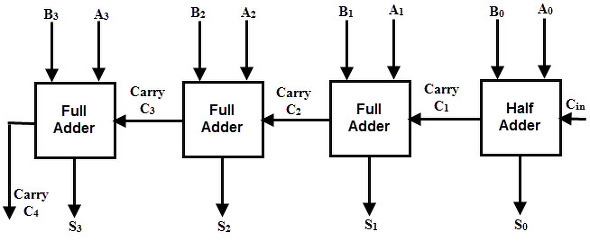
테이블이(가) 표시된 사진

자동 생성된 설명

<Figure 9> BCD codes

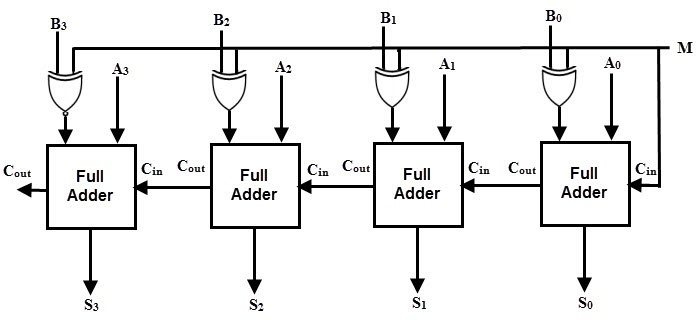
1. 병렬 가감산기에 대해 조사하시오.

병렬 가산기는 전가산기 여러 개를 병렬로 연결하여 여러 bit 덧셈 연산을 수행하는 회로입니다. 다음 아래 <Figure 10>은 4비트 병렬 가산기로, 이진수의 맨 아래 자릿수를 계산할 때에는 cin이 0이므로 아래 그림과 같이 반가산기를 이용하여도 되고, 전가산기를 사용하여 cin을 0이라고 하여도 무관합니다.



<Figure 10> 4-bit 병렬 가산기

병렬 가감산기는 병렬 가산기 회로에서 입력을 일부 바꿔서 덧셈과 뺄셈이 모두 가능하도록 바꾼 회로입니다. 아래 <Figure 11>는 위의 <Figure 10> 4-bit 병렬 가산기의 입력 B를 바로 가산기에 입력하지 않고, M이라는 비트와 XOR 연산을 하여 입력해 줌으로써, M이 0일 때는 이 회로가 병렬 가산기로써 동작하도록, M이 1일 때는 이 회로가 병렬 감산기로 동작하도록 바꿔줄 수 있습니다. M이 1일 때, B가 0이면 출력은 1이 되고, B가 1이면 출력은 0이 되므로 M과의 XOR 연산을 통해 B의 값이 반전되는 것을 알 수 있습니다. 또한 <Figure 11>을 보면 위의 병렬 가산기 <Figure 10>와 달리, 맨 아래 자릿수를 계산하는 가산기 또한 전가산기이므로 M이 cin에 들어가 B의 모든 자릿수를 반전한 것과 1을 더함으로써 B의 2의 보수를 만들 수 있어 이러한 회로는 병렬 감산기로써 동작할 수 있게 됩니다.



<Figure 11> 4-bit 병렬 가감산기

1. Carry Look-Ahead Adder을 Ripple Carry Adder와 비교하여 설명하시오.

병렬 가산기 <Figure 10>은 ripple carry adder라고도 불립니다. ‘잔물결’, ‘파문처럼 번지다’라는 뜻을 가지고 있는 ripple처럼 병렬 가산기는 하위 자릿수에서의 출력이 상위 자릿수의 연산에 사용되기 때문에 ripple carry adder라고 불립니다. 하위 자릿수에서의 출력이 존재하지 않으면 해당 전가산기의 출력은 유효하지 않기 때문에 하위 자릿수에서의 출력 지연이 상위 자릿수의 연산에 영향을 미칩니다. 만약 n-bit의 ripple carry adder가 존재할 경우, 전체 시간 지연은 (2n+4)Δ라고 표현할 수 있습니다.

하지만 carry look ahead adder는 이름부터 알 수 있듯이 앞을 내다보는 가산기로, ripple carry adder에서의 시간 지연을 줄일 수 있다는 장점이 있습니다. Carry look ahead adder에서 입력 carry가 ci이고, xi+3+xi+2+xi+1+xi + yi+3yi+2yi+1yi를 수행할 때에는 자리올림수 생성 gi에 대한 논리식 gi=xiyi과 자리올림수 전달 pi에 대한 논리식 pi=xi⊕yi를 이용하여 si와 ci+1을 얻을 수 있습니다. si=pi⊕ci이며 ci+1=gi+(pi·ci)라는 논리식을 따라 가산기의 출력이 결정되기 때문에 하위 자릿수의 자리올림수가 계산되기까지 기다리지 않고 직접 처리할 수 있어서 하위 자릿수의 자리올림수가 결정될 때까지 기다려야 하는 ripple carry adder보다 빠르게 연산할 수 있습니다.

1. 기타 이론

이진수를 이용하여 음수를 표현하는 방법에 대해 알아보겠습니다. 우선, 지금까지 본 모든 수들은 양수이고, unsigned number였지만 signed number를 이용하면 음수와 양수를 모두 다룰 수 있습니다. 사람은 +와 -와 같은 부호로 양수와 음수를 구분하는 것이 편리하지만 컴퓨터는 이러한 방식을 사용하지 않기 때문에 컴퓨터가 음수를 나타내는 방법에 대해 이해해야 합니다. 컴퓨터가 음수를 표현하는 여러 가지 방법 중 Sign-magnitude representation에선 이진수에서 맨 첫 번째 bit를 sign indicator로써 사용합니다. 예를 들어, 8421 code로 +5는 0101로 표현이 되고, -5는 맨 첫 번째 bit를 1로 하여 1101로 표현이 가능합니다. 하지만 첫 번째 bit를 사용하지 않고 표현하고자 하는 십진수를 나타내야 하기 때문에 이러한 방식으로는 4 bit 이진수에서 +7에서부터 -7까지의 값만을 나타낼 수 있고, 0을 가리키는 숫자가 0000과 1000으로 두 개나 존재합니다. 하지만 이것보다 더욱 큰 문제점은 이 방법으로는 arithmetic이 복잡하다는 것입니다. 따라서, 보통 컴퓨터는 음수를 표현할 때 2의 보수 방법을 주로 사용합니다. 2의 보수로 숫자를 나타내기 위해서는 앞서 설명하였듯이 모든 자릿수의 비트를 반전시킨, 1의 보수에 1을 더하여 구할 수 있고, 이렇게 하면 +0은 0000으로, -0 또한 0000으로, 0이 딱 한 가지 방법으로만 표현 가능하다는 장점이 있을뿐 아니라 overflow를 제외하고는 arithmetic가 간단해집니다.