6주차 예비보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20211533 이름: 남정연

**1.**

반 가산기(Half Adder)는 현재 단계에서 더할 숫자 X, Y에 대하여 그 합인 S와 다음 자릿수로 올라가는 수(Carry) C를 출력하는 함수이다. 아래와 같은 진리표를 가지며 논리식은 S=X XOR Y, C = XY와 같다.

|  |  |  |  |
| --- | --- | --- | --- |
| X | Y | S | C |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 |

전 가산기(Full Adder)는 반 가산기(Half Adder)를 연속적으로 사용할 수 있게 한 함수로, 현재 단계(i)에서 더할 숫자 X, Y 이외에도 이전 단계의 Carry C\_i-1를 추가적으로 입력받는다. 아래와 같은 진리표를 가지며 논리식은 S = X XOR Y XOR C\_i-1, C = XY + (X XOR Y)C\_i-1 이다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X | Y | C\_i-1 | S | C |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**2.**

반 감산기(Half Subtractor)는 입력 값 X와 Y에 대해서 X와 Y의 차인 D와 X-Y를 수행하였을 때 위 자리에서 빌려와야 하는 수(Borrow) B를 출력하는 함수이다. 아래와 같은 진리표를 가지며 논리식은 다음과 같다. D = X XOR Y, B = X’Y

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

반 가산기(Half Adder)와 비교하면 출력 값인 S와 D는 동일하고, C와 B의 경우 X에 NOT 게이트가 있는지 없는지 여부만 차이가 있음을 알 수 있다.

전 감산기(Full Subtractor)는 반 감산기(Half Subtractor)에서 이전 단계에서 발생한 Borrow B\_i-1를 추가적으로 입력 받아 현재 단계의 D와 B를 출력하는 함수이다. 아래와 같은 진리표를 가지며 논리식은 다음과 같다. D = X XOR Y XOR B\_i-1, B = X’Y + (X XOR Y)’B\_i-1

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

**3.**

BCD 가산기는 두 개의 BCD를 더한 값을 BCD로 출력하는 함수이다. BCD의 경우 10 이상의 수의 경우 새로운 비트가 추가되므로 두 값을 더하여 10 이상이 나왔다면 BCD 최댓값(1001)의 보수를 더하여 출력해야 한다. 즉 BCD Sum이 9 이하이면 FA 값을 그대로, 10 이상이면 FA + 6(0110)을 출력하는 함수이다.

**4.**

병렬 가감산기는 2의 보수의 특성을 이용하여 FA를 가지고 감산도 수행할 수 있게 설계한 논리 회로이다. 덧셈일 경우 0을, 뺄셈일 경우 1을 출력하는 M(mode)를 두고, 각 단계의 두 번째 입력 값 Y에 Y 대신 Y XOR M이 FA의 입력으로 사용되게 한다. 또한 첫 단계의 Carry C0에 M을 대입한다. 0과 XOR을 할 경우 본래의 값이 그대로, 1과 XOR 값을 할 경우 비트가 반전되는 결과가 나타나기 때문에 뺄셈일 경우 2의 보수 값이 더해지며 추가적으로 첫 단계의 Carry C0이 1을 더하게 되어 2의 보수 뺄셈이 이루어지게 된다.

**5.**

Ripple Carry Adder는 FA를 여러 개 이어 붙인 형태로 각 단계에서 이전 단계의 Carry C\_i-1이 입력되기를 기다려야 해 회로 상 지연 시간이 존재한다. Carry Look Ahead Adder는 이전 단계의 Carry C\_i-1이 재귀적으로 계산하면 각 단계의 비트에 의존함을 이용하여 이전 단계의 Carry 입력을 기다리지 않고 바로 계산을 수행하는 방식이다. FA에서 S = X XOR Y XOR C\_i-1, C = XY + (X XOR Y)C\_i-1 인데, XY 항을 이전 단계의 Carry와 무관하게 Carry를 발생시키는 항인 G(Carry Generator)=XY, 이전 단계의 Carry가 있으면 Carry를 발생시키는 항인 P(Carry Propagation)=X XOR Y를 새롭게 정의하여 S = P XOR C\_i-1, C = G + PC\_i-1로 생각한다. 즉 어떤 단계의 Carry C를 각 단계의 비트 값만 가지고 재귀적으로 구할 수 있게 된다. 다만 깊이가 깊어질수록(단계가 증가할 수록) gate에 들어가는 input의 개수가 많아져 fan-in 문제가 발생할 수 있다.

**6.**

BCD 코드 중 2421 코드는 4 이하의 수에 대해서는 8421과 동일한 값을, 5 이상의 수에 대해서는 4 이하의 수의 9에 대한 1의 보수를 취하여 생성한다. 즉 변환 표는 아래와 같다.

|  |  |  |
| --- | --- | --- |
| Decimal | 8421 Code | 2421 Code |
| 0 | 0000 | 0000 |
| 1 | 0001 | 0001 |
| 2 | 0010 | 0010 |
| 3 | 0011 | 0011 |
| 4 | 0100 | 0100 |
| 5 | 0101 | 1011 |
| 6 | 0110 | 1100 |
| 7 | 0111 | 1101 |
| 8 | 1000 | 1110 |
| 9 | 1001 | 1111 |

8421-2421 변환 로직은 2421 코드의 각 자리를 출력으로, 8421 코드의 두 자리씩을 묶어 입력으로 하는 카르노 맵을 그려 작성할 수 있다.