7주차 예비보고서

전공: 컴퓨터공학과/생명과학과 학년: 9학기 학번: 20181435 이름: 박다희

**1.**

Parity bit은 이진 비트를 전송할 때 단일 에러 검출을 하기위해 데이터 끝에 추가된 1bit로, 데이터의 bit 크기가 작거나 에러가 발생할 확률이 낮을 때 주로 사용된다. Parity bit는 odd parity와 even parity 이렇게 2가지 종류로 나뉠 수 있는데 이는 논리 1의 개수에 따라 결정된다. Odd parity의 경우에는 논리 1의 개수가 홀수일 때 parity bit을 0으로, 짝수일 때는 parity bit를 1로 설정함으로써 논리 1의 개수를 늘 홀수로 만들어주는 것이고 반대로ㄴ even parity의 경우에는 논리 1의 개수가 짝수일 때 parity bit를 0으로, 홀수일 때는 parity bit를 1로 설정함으로써 논리 1의 개수를 늘 짝수로 만들어주는 것이다. 이러한 parity bit을 생성하는 생성기는 XOR gate를 사용하여 나타낸다. 만약 n개의 이진 bit가 있고 even parity bit을 사용한다고 할 때, n개의 입력 값을 XOR연산하여 parity bit을 생성하므로 (n-1)개의 XOR gate가 사용된다. 다음은 3-bit에 대한 even parity 생성기의 그림이다.

폰트, 도표, 화이트, 디자인이(가) 표시된 사진

자동 생성된 설명

Odd parity bit를 사용하고 n개의 이진 bit가 있을 때도 even parity bit 생성기와 비슷하다. 입력 값 n개를 XOR연산한 뒤에 그 출력 값에 보수를 취하면 되므로 (n-1)개의 XOR gate와 마지막에 NOT gate를 붙여서 구현하거나 또는 (n-2)개의 XOR gate와 1개의 XNOR gate를 이용하여 구현될 수 있고 XNOR gate로만 구성하고 싶을 때는 (n-1)개의 XNOR gate를 사용하면 된다. 밑의 사진은 3-bit odd parity 생성기이다.

스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명

**2.**

Parity bit 검사기는 받은 데이터의 parity bit를 확인하여 데이터가 에러없이 잘 전달됐는지 그 여부를 검사하는 회로로 이를 구현할 때는 XOR gate와 XNOR gate의 특성을 이용하면 된다. Even parity bit가 포함된 데이터를 에러 없이 전송 받았을 땐 그 데이터의 논리 1의 개수는 늘 짝수여야 하므로 검사할 때는 XOR gate를 사용하여 출력 값으로 0이 나오면 정상적으로 데이터가 전송됐다는 것을 알 수 있고, odd parity bit가 포함된 데이터를 에러 없이 전송 받았을 땐 그 데이터의 논리 1의 개수는 늘 홀수여야 하므로 검사할 때는 XNOR gate를 사용하여 입력 값에 대한 XNOR gate 출력 값이 0이 나왔을 때 정상적으로 데이터가 전송됐다는 것을 알 수 있다.

도표, 라인, 폰트, 그래프이(가) 표시된 사진

자동 생성된 설명도표, 폰트, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

위의 그림은 왼쪽부터 3-bit even parity bit 검사기, 3-bit odd parity bit 검사기이다. 그러나 parity bit 검사기는 짝수 개의 에러가 났을 경우에는 이를 알아낼 수 없고, 알아낸다 하더라도 어디에서 에러가 났는지 알 수 없다는 것이 단점이다.

**3.**

해밍코드(Hamming Code)는 데이터를 전달 받는 쪽에서 에러가 발생한 bit를 검출한 뒤에 수정까지 하는 방법이다. 이 방법으로는 1-bit에 해당하는 에러만 수정이 가능하고 잉여bit가 필요하다. 해밍 코드에서는 송신한 데이터와 수신한 데이터를 비교했을 때 차이가 있는 bit의 수를 해밍 거리(Hamming Distance)라고 하고, 2진수의 데이터 중에서 2의 거듭제곱자리의 bit를 parity bit로 사용한다. 만약 2^5에 해당하는 bit 데이터를 전송 받았다면 (5+1)개의 bit는 parity bit이 되고, 나머지 bit들은 원래의 데이터를 담고 있는 bit가 되는 것이다. 구체적으로 예시를 들어 4-bit의 0101 데이터가 있다고 가정하고 2의 거듭제곱 수 자리를 제외한 칸에 데이터를 넣으면 다음과 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | 0 |  | 1 | 0 | 1 |
| bit 1 | bit 2 | bit 3 | bit 4 | bit 5 | bit 6 | bit 7 |

이렇게 데이터를 넣은 후에 나머지 자리에 들어갈 parity bit를 구해보면 다음과 같다. 먼저 bit 1자리는 bit 3, 5, 7에 있는 값들의 even parity bit이고, bit 2자리는 bit 3, 6, 7에 있는 값들의 even parity bit, bit 4자리는 bit 5, 6, 7에 있는 값들의 even parity bit이다. 최종적으로 십진수로 5, 곧 이진수로 0101을 에러 없이 전송 받았을 때의 해밍 코드는 0100101입니다. 그러나 만약 전송 과정에서 bit 5의 데이터가 0으로 바뀌었다고 가정하면 이 코드를 받은 수신기의 오류 검출기로는 bit 1, 3, 5, 7을 even parity bit check하여 에러가 있으므로 1을 반환, bit 2, 3, 6, 7을 even parity bit check하여 에러가 없으므로 0을 반환, bit 4, 5, 6, 7을 even parity bit check하여 에러가 있으므로 1을 반환한다. 이렇게 checker가 반환한 값을 역순으로 적으면 101이 되어 bit 5자리에서 에러가 생겼음을 알 수 있고 bit 5의 값을 0에서 1로 올바르게 수정한다.

**4.**

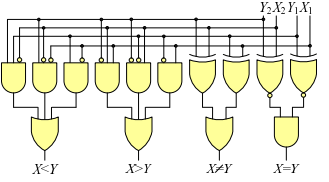
N(N은 2이상의 수)개의 bit로 구성된 두 수를 비교하며 대소 비교와, 일치 여부를 출력하는 것이다. 그 전에 1 bit 비교기에 해당하는 진리표와 논리 회로는 밑의 그림과 같다.

테이블이(가) 표시된 사진

자동 생성된 설명

만약 N이 2라면 입력 값은 00, 01, 10, 11중 2개의 조합이 되고 이 두 수를 X, Y고 했을 때 해당하는 진리표를 작성하면 다음과 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X1X2(=X) | Y1Y2(=Y) | X == Y | X != Y | X > Y | X < Y |
| 00 | 00 | 1 | 0 | 0 | 0 |
| 01 | 0 | 1 | 0 | 1 |
| 10 | 0 | 1 | 0 | 1 |
| 11 | 0 | 1 | 0 | 1 |
| 01 | 00 | 0 | 1 | 1 | 0 |
| 01 | 1 | 0 | 0 | 0 |
| 10 | 0 | 1 | 0 | 1 |
| 11 | 0 | 1 | 0 | 1 |
| 10 | 00 | 0 | 1 | 1 | 0 |
| 01 | 0 | 1 | 1 | 0 |
| 10 | 1 | 0 | 0 | 0 |
| 11 | 0 | 1 | 0 | 1 |
| 11 | 00 | 0 | 1 | 1 | 0 |
| 01 | 0 | 1 | 1 | 0 |
| 10 | 0 | 1 | 1 | 0 |
| 11 | 1 | 0 | 0 | 0 |

 위의 진리표를 바탕으로 카르노맵을 작성한 뒤에 논리 함수를 찾아서 2-bit 비교기를 구현해보면 다음과 같다.

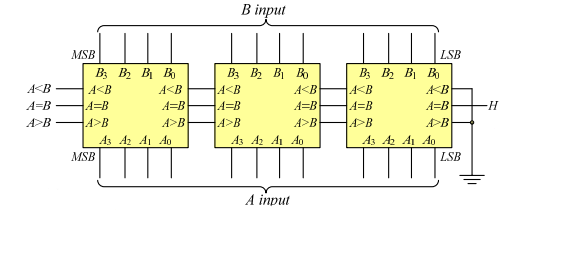
N이 커지더라도 4-bit 비교기를 여러 개 이용하는 방식으로 구현할 수 있게 된다.

**5.**

IC 7485 비교기는 4-bit 비교기로 두 4-bit 수를 비교하여 대소비교와 일치여부를 판단한다. 예를 들어 와 이렇게 4-bit 수가 입력 값으로 들어왔다고 했을 때 OA>B, OA<B, OA=B 값이 맞으면 1로 출력하는 비교기이다. 또한 입력 값으로 4-bit 수 뿐만 아니라 cascading input인 IA>B, IA<B, IA=B도 입력 값으로 들어 간다. Cascading input이란 아래 자릿수에서 올라오는 입력 값으로 4-bit 비교기를 사용하여 더 큰 수 비트 비교기(12-bit, 16-bit)를 나타낼 때 사용된다. 4-bit 비교기를 진리표로 나타내면 다음과 같다.

테이블이(가) 표시된 사진

자동 생성된 설명

 위의 진리표를 이용하여 각 출력 값에 해당하는 논리 함수를 구해보면 다음과 같다. OA=B은 Ai와 Bi를 XNOR gate 실행한 출력 값인 Ai’Bi’+AiBi (=Xi)라고 한다면 각 입력 값에 대한 X3, X2, X1, X0을 논리곱을 통해 구할 수 있고, OA>B은 A3B3’ + X3A2B2’ + X3X2A1B1’ + X3X2X1A0B0’, OA<B은 A3’B3 + X3A2’B2 + X3X2A1’B1 + X3X2X1X0A0’B0로 나타낼 수 있다. 이렇게 구한 4-bit 비교기를 3개 이어 연결하면 12-bit 비교기도 구현할 수 있다.

**6.**

Decoder는 n개의 입력 값을 받아서 개의 출력 값을 내보내는 논리 회로로 만약 한 개의 출력 값에서 1을 가지면 남은 출력 값들은 0을 가진다는 특징을 갖는다. 만약 XY가 00이면 D0이 1, XY가 01이면 D1이 1, XY가 10이면 D2가 1, 마지막으로 XY가 11이면 D3이 1의 출력 값을 갖는다. Encoder는 decoder와 상반되는 개념으로 개의 입력 값을 받아서 n개의 출력 값을 내보내는 논리 회로이다. 입력 값의 개수가 출력 값의 개수보다 더 많기 때문에 암호화 과정이므로 Encoder라고 불리는 것이다. Encoder는 입력 값 중에서 1개의 값만 1을 갖고 남은 입력 값들은 0을 가져야 한다.

Multiplexer(MUX)는 n개의 입력 값 중에서 한 개를 택하여 출력 값으로 내보내는 논리 회로이다. 이 논리 회로는 여러 개의 host computer나 여러 개의 단말기에서 생성된 데이터를 오직 한 개의 통신회선을 이용하여 보낼 수 있게 해준다는 점에서 효율을 높이고 비용은 절감할 수 있다. 이와 반대로 demultiplexer도 존재하는데 이는 MUX와는 반대로 한 개의 입력 값을 가지고 여러 개의 출력 값을 내보내는 논리 회로이다 multiplexer와 demultiplexer의 진리표는 다음과 같다.

스크린샷, 텍스트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명스크린샷, 텍스트, 소프트웨어이(가) 표시된 사진

자동 생성된 설명