7주차 예비보고서

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

**1.**

Check Bit로도 알려진 Parity Bit는 데이터 전송 중 오류를 감지하기 위해 사용하는 중요한 방법중 하나이다. Parity Bit는 even parity와 odd parity 두 가지가 존재한다. even parity의 경우 parity bit를 포함한 전체 비트에서 1의 개수가 짝수가 되도록 설정된다. 예를 들어 데이터 비트의 1의 개수가 이미 짝수라면 even parity는 0으로 설정되며 홀수의 경우 1로 설정되는 방식이다. odd parity도 같은 방식으로 동작하며 parity bit를 포함한 전체 비트에서 1의 개수가 홀수가 되도록 설정해준다.

이렇게 parity bit는 데이터의 1의 개수에 대한 정보를 담고 있기 때문에 데이터 수신측에서 데이터 비트와 parity bit를 검사하여 데이터의 오류를 검사할 수 있다. 그러나 parity bit는 단순히 1의 개수의 짝수와 홀수로 오류를 판단하기 때문에 두개 이상의 비트에 오류가 생기는 경우 검출에 어려움이 생긴다. 또한 1010010이라는 7 비트 데이터를 전송한다고 할 때, even parity bit라고 가정하면 11010010의 데이터를 전송하는 식으로 작동하므로 자연스레 전송되는 비트의 수가 증가한다. 때문에 데이터 전송 효율성이 떨어진다는 단점도 존재한다. 따라서 복잡한 통신 시스템에서는 보다 신뢰성 있는 오류 검출 알고리즘이 필요하다.

Parity bit 생성기는 데이터 비트의 1의 개수에 따라 parity bit를 생성하므로 XOR gate를 사용한다. Even parity bit 생성기의 경우 1의 입력 개수가 홀수일 경우 1이 되므로 XOR gate를 그대로 사용하며 odd parity bit 생성기의 경우 반대인 XNOR gate를 사용한다. 아래는 데이터 비트가 4bit일 때의 parity bit 생성기의 예시이다. 좌측 그림이 even, 우측 그림이 odd parity bit 생성기 임을 알 수 있다.

텍스트, 도표, 폰트, 스케치이(가) 표시된 사진

자동 생성된 설명텍스트, 도표, 폰트, 스케치이(가) 표시된 사진

자동 생성된 설명

**2.**

전달받은 parity bit를 검사하여 데이터 전송 중에 오류의 발생 여부를 확인하는 회로이다. 송신자가 데이터에 parity bit를 추가하고 수신자가 parity bit 검사기를 통해 even parity와 odd parity 중 한 방식을 선택하여 검사를 수행한다.

**2-1) Even parity**

Even parity 검사기에서는 수신 받은 데이터 비트에 1의 개수를 검사하여 1의 개수가 짝수인지 확인한다. 만약 데이터의 전송 중 오류가 발생하여 1의 개수가 홀수로 바뀌었다면, 검사기는 오류를 감지하게 된다. 앞서 설명했듯이 이 방식은 단일 비트 오류를 감지할 수 있지만 여러 비트 오류에 대해서는 민감하지 않다.

**2-2) Odd parity**

Even parity 검사기와 동일하지만 1의 개수가 홀수인지 확인한다. 만약 데이터의 전송 중 오류가 발생하여 1의 개수가 짝수로 바뀌었다면, 검사기는 오류를 감지하게 된다.당 연히 여러 비트 오류에 대해서는 민감하지 않다.

**2-3) 예시**

수신한 데이터가 4 비트일 경우, 첫 비트가 parity 비트이며 뒤의 세 비트가 전송된 데이터이다. 전송 데이터와 parity 비트에 따른 검사기의 동작을 진리표로 나타내면 아래와 같다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **송신** | | | | **수신** | |
| **A** | **B** | **C** | **Parity** | **Even** | **Odd** |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 |

**3.**

**3-1) Cyclic Redundancy Check**

순환 중복 검사 (CRC)는 다항식 코드를 사용하여 CRC 값을 추가하고, 수신 측에서 이를 다시 계산하여 오류 여부를 확인하는 방식이다. Parity 비트와 동작 방식이 유사하지만 단순히 데이터 비트의 1의 개수만을 이용하는 것이 아니라 보다 복잡한 방법으로 다중 비트 오류를 검사할 수 있다. CRC는 다음과 같은 원리로 동작한다.

CRC는 다항식을 통해 제수를 구한다. 다항식의 차수가 곧 제수의 비트의 자리가 되며 특정 자수의 항이 존재하면 해당 비트는 1이 된다. 예시는 아래와 같다.

라인, 도표, 스크린샷, 그래프이(가) 표시된 사진

자동 생성된 설명

이후 해당 제수를 이용하여 n비트의 전송 데이터를 나눠주고 나머지를 구한다. 해당 나머지 값이 CRC 값이 되며 이를 데이터에 붙여 전송한다. 수신 측은 받은 데이터에 대해 CRC를 계산하여 전송된 CRC와 동일한지 확인한다. 만약 두 CRC가 다를 경우 전송된 데이터에 오류가 발생했음을 의미한다.

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

**3-2) Hamming Code**

이전 Parity code와 CRC는 오류의 검출만 가능하지 오류 정정은 수행할 수 없다. 그러나 해밍 코드는 오류 정정까지 가능하다. 해밍 코드는 parity bit를 여러 개 만들어 전송하는 방식이다. Parity bit의 개수는 data bit의 길이에 따라 달라지며

**2p-1 >= d+p**

에 따라 parity bit(p)를 구한다. d는 data bit의 길이를 의미한다.

생성된 parity bit는 data bit 사이에 배치된다. 아래의 예시처럼 가장 좌측 자리부터 시작하여 2의 거듭제곱 자리에 배치된다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

각 parity bit는 특정 자리의 bit의 1의 개수에 따라 결정된다.

스크린샷, 텍스트, 사각형, 다채로움이(가) 표시된 사진

자동 생성된 설명

각 parity bit가 확인하는 자리는 위와 같다. 예를 들어 p1의 경우 3, 5, 7, 9, 11 자리의 bit를 확인해야 하며 even parity bit라고 가정했을 때 p1=p3⊕p5⊕p7⊕p9⊕p11로 p1이 결정된다. 수신 측에서는 받은 parity bit들을 모두 검사하며 오류를 검출할 수 있다. 예를 들어 p2와 p4 parity bit에서 오류가 검출되었다면 두 영역이 겹치는 10번 bit에서 오류가 발생했음을 알 수 있고 오류의 수정까지 가능하다. 그러나 parity bit가 여러 개 포함되면서 전송해야 할 데이터의 길이가 길어지는 단점이 있다.

**3-3) Block Sum Check**

블록 합 검사 역시 parity bit의 원리를 이용하며 이를 보완한 방식이다. 단일 parity bit가 아닌 2차원 parity를 구하는 방식으로 문자 블록의 수평 parity와 수직 parity를 모두 검사하여 다중 비트의 오류 검출을 가능하게끔 만든다. 아래 예시와 같이 여러 데이터를 일정한 크기의 블록으로 묶고 행단위와 열단위의 parity bit를 구한다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

**4.**

N bit 비교기란, 두 개의 N 비트 이진수의 크기를 비교하는 회로를 의미한다. 두 개의 N 비트 입력값을 가지므로 총 2N개의 입력을 가지며 각 이진수를 A와 B라고 할 때, A=B, A≠B, A>B, A<B 연산의 결과값을 출력한다.

1 비트 비교기를 예시로 진리표와 회로는 다음과 같다.

도표, 라인, 폰트, 평면도이(가) 표시된 사진

자동 생성된 설명

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **X** | **Y** | **X = Y** | **X ≠ Y** | **X > Y** | **X < Y** |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |

진리표를 통해 Boolean 식을 도출 해보면 X와 Y가 0, 0이고 1, 1일 때 X=Y이므로 이는 X’Y’ + XY로 혹은 XNOR 회로를 통해 구할 수 있다. 반대로 X≠Y의 경우 당연히 XOR 함수를 통해 구할 수 있다. 또한 X > Y는 1, 0일 때만 1, X < Y는 0, 1일 때만 1이 되므로 각 각 XY’와 X’Y로 나타낼 수 있다.

N 비트의 경우 위의 연산을 높은 자리의 비트 시작하여 대소 관계를 출력한다. 동일한 비트의 경우 다음 비트로 넘어가 해당 연산을 다시 수행하여 N개의 비트를 비교할 수 있다. 단순히 모든 경우의 수를 확인한다고 했을 때 2N \* 2N 개의 경우의 수를 확인해야 한다.

**5.**

IC 7485 비교기는 4 bit 이진 비교기로 위에서 설명한 N-bit 비교기 중 N=4인 경우이다. IC 7485 비교기는 여느 비교기처럼 두 개의 4 비트 입력 X와 Y를 받아서 두 숫자를 비교하고 결과를 출력한다. 또한 추가적인 입력을 통해 비교 연산의 결과를 확장하거나 특정 용도에 사용할 수 있다.

추가 입력에는 AGBO, ALBO, AEBO, EQO 등이 있는데 각 각, X가 Y보다 큰 경우, X가 Y보다 작은 경우, X와 Y가 같을 때 출력 되는 신호이다.

아래의 진리표에서 색칠된 부분은 비정상적인 입력을 표시한다. 또한 AGBI, ALBI, AEBI는 4비트 이상의 비교에 사용되는 확장 입력을 나타낸다. 주목해야 할 점은 밑 쪽의 AGBO, ALBO, AEBO의 출력이 윗단에 AGBI, ALBI, AEBI에 연결되어 입력으로 다시 들어오게 된다 점이다. 이에 따라 IC 7485 비교기에서의 입력은 input X, Y뿐 아니라 AGBI, ALBI, AEBI까지 입력으로 받게 된다.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Input** | | | | | **Output** | | | | |
| **A3, B3** | **A2, B2** | **A1, B1** | **A0, B0** | **AGBI** | **ALBI** | **AEBI** | **AGBO**  **A>B** | **ALBO**  **A<B** | **AEBO**  **A=B** |
| A3>B3 | X | X | X | X | X | X | 1 | 0 | 0 |
| A3<B3 | X | X | X | X | X | X | 0 | 1 | 0 |
| A3=B3 | A2>B2 | X | X | X | X | X | 1 | 0 | 0 |
| A3=B3 | A2<B2 | X | X | X | X | X | 0 | 1 | 0 |
| A3=B3 | A2=B2 | A1>B1 | X | X | X | X | 1 | 0 | 0 |
| A3=B3 | A2=B2 | A1<B1 | X | X | X | X | 0 | 1 | 0 |
| A3=B3 | A2=B2 | A1=B1 | A0>B0 | X | X | X | 1 | 0 | 0 |
| A3=B3 | A2=B2 | A1=B1 | A0<B0 | X | X | X | 0 | 1 | 0 |
| A3=B3 | A2=B2 | A1=B1 | A0=B0 | 1 | 0 | 0 | 1 | 0 | 0 |
| A3=B3 | A2=B2 | A1=B1 | A0=B0 | 0 | 1 | 0 | 0 | 1 | 0 |
| A3=B3 | A2=B2 | A1=B1 | A0=B0 | 0 | 0 | 1 | 0 | 0 | 1 |
| A3=B3 | A2=B2 | A1=B1 | A0=B0 | 0 | 1 | 1 | 0 | 0 | 1 |
| A3=B3 | A2=B2 | A1=B1 | A0=B0 | 1 | 0 | 1 | 0 | 0 | 1 |
| A3=B3 | A2=B2 | A1=B1 | A0=B0 | 1 | 1 | 1 | 0 | 0 | 1 |
| A3=B3 | A2=B2 | A1=B1 | A0=B0 | 1 | 1 | 0 | 0 | 0 | 0 |
| A3=B3 | A2=B2 | A1=B1 | A0=B0 | 0 | 0 | 0 | 1 | 1 | 0 |

IC 7485 비교기도 위에서 설명한 N-bit 비교기와 마찬가지로 제일 큰 자리를 나타내는 A3, B3부터 비교를 시작하여 두 값이 같지 않은 경우에는 하위 비트들을 비교할 필요 없이 출력이 나오게 된다는 것을 진리표를 통해 확인할 수 있다. 또한 두 값이 같다면 다음 자리로 넘어와서 비교를 한다는 것을 확인하며 이를 재귀적으로 수행한다. 밑에 색칠한 부분이 비정상 입력인 이유는 A>B, A=B, A<B 은 한번에 여러 개가 성립할 수 없으며, 무조건 한 개만 성립한다는 특징을 가지고 있기 때문이다. 즉, AGBI, ALBI, AEBI 중 1이 2개 이상이거나 아예 없으면 비정상적인 입력이 된다.

**6.**

**6-1) BEC &FEC**

위에서 확인한 해밍 코드를 제외한 오류 검출기들은 오류를 정정할 수 있는 기능은 없었다. 때문에 Parity 검사나 CRC, Block sum check은 오류를 검출하였을 때, 오류가 없는 데이터를 받을 때 까지 송신측에 재전송을 요구하게 된다. 이러한 방식을 역방향 오류 정정(BEC)라고 부른다.

반면, 해밍 코드와 같이 재전송 없이 검출과 동시에 오류의 정정이 가능한데 이를 순방향 오류 정정(FEC)라고 부른다. 해밍 코드를 보면 알 수 있듯이 오류를 검출하기 위해선 특정 비트를 계산하여 추가로 전송해야 하기 때문에 전송의 효율이 떨어진다는 단점이 있다. 따라서 데이터의 전송 크기와 속도 등을 적절히 고려하여 BEC와 FEC 방식을 선택해야 한다.

**6-2) Gray code**

그레이 코드는 하나의 비트만이 차이가 나는 연속적인 값을 오류 감지에 사용하는 데이터 코드이다. 예를 들어 3-bit Gray Code는 0, 1, 3, 2, 6, 7와 같이 진행된다. 1은 이진수로 01이고 3은 이진수로 11이므로 두 수는 하나의 비트만이 차이가 나지만 2는 이진수로 10이기 때문에 1과 두개의 비트가 차이가 난다. 따라서 그레이 코드로 1 다음에는 3이 온다. 이러한 특성을 이용하여 데이터 전송 중 발생한 오류를 쉽게 식별할 수 있다. 오류가 발생하면 인접한 Gray Code 사이에 두 개 이상의 비트 변화가 발생하기 떄문이다.

그레이 코드는 재귀적 방식으로 생성할 수 있고, 순환 패턴을 갖고 있어 순환성이 필요한 리모컨이나 회전 엔코더 등에 유용하다. 또한 카운터, 상태 기계 및 인코더 디자인에 널리 사용된다.

텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명