|  |
| --- |
| 7주차 예비보고서 |
| |  | | --- | | 1. Parity Bit  - Parity Bit 생성기  - Parity Bit 검사기  2. 그 외의 오류 검출기와 정정기  3. N Bit 비교기 & IC 7485 비교기 | |
| 20141196 김성희 |
| 1. Parity Bit |
| 1-1. 생성기  [설명]  - 정보의 전달 과정에서 오류가 생겼는지를 검사하기 위해 덧붙이는 bit를 parity bit라고 한다.  - 하나의 bit를 사용하며 방식은 2가지가 있다. Even parity bit와 Odd parity bit가 있다.  - Even parity bit는 1의 개수를 짝수로 Odd parity bit는 홀수로 맞춰 준다.  - 이제 논리식을 만들어 보자.  \* 방식은 같기에 Even parity bit에 대해서 4비트만을 가지고 만들기로 한다.  [설계]  1. 4비트 생성기 진리표   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **입력** | | | | **출력** | min  term | | **I3** | **I2** | **I1** | **I0** | **Even parity bit** | | 0 | 0 | 0 | 0 | 0 | m0 | | 0 | 0 | 0 | 1 | 1 | m1 | | 0 | 0 | 1 | 0 | 1 | m2 | | 0 | 0 | 1 | 1 | 0 | m3 | | 0 | 1 | 0 | 0 | 1 | m4 | | 0 | 1 | 0 | 1 | 0 | m5 | | 0 | 1 | 1 | 0 | 0 | m6 | | 0 | 1 | 1 | 1 | 1 | m7 | | 1 | 0 | 0 | 0 | 1 | m8 | | 1 | 0 | 0 | 1 | 0 | m9 | | 1 | 0 | 1 | 0 | 0 | m10 | | 1 | 0 | 1 | 1 | 1 | m11 | | 1 | 1 | 0 | 0 | 0 | m12 | | 1 | 1 | 0 | 1 | 1 | m13 | | 1 | 1 | 1 | 0 | 1 | m14 | | 1 | 1 | 1 | 1 | 0 | m15 |   2. 4비트 생성기 K-map   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **I1 I0**  **I3 I2** | | **I1’I0’** | **I1’I0** | **I1 I0** | **I1 I0’** | | **00** | **01** | **11** | **10** | | **I3’I2’** | **00** | **0** | **1** | **0** | **1** | | **I3’I2** | **01** | **1** | **0** | **1** | **0** | | **I3 I2** | **11** | **0** | **1** | **0** | **1** | | **I3 I2’** | **10** | **1** | **0** | **1** | **0** |   3. 4비트 생성기 논리식  Pe = I3’I2’I1’I0 + I3’I2’I1 I0’ + I3’I2 I1’I0’ + I3’I2 I1 I0  +I3 I2 I1’I0 + I3 I2 I1 I0’ + I3 I2’I1’I0’ + I3 I2’I1 I0  = I3’I2’(I1’I0 + I1 I0’) + I3’I2 (I1’I0’ + I1 I0)  + I3 I2 (I1’I0 + I1 I0’) + I3 I2’(I1’I0’ + I1 I0)  = (I3’I2’ + I3 I2)(I1’I0 + I1 I0’) + (I3’I2 + I3 I2’)(I1’I0’ + I1 I0)  = (I3’I2 + I3 I2’) XOR (I1’I0 + I1 I0’)  = (I3 XOR I2) XOR (I1 XOR I0)  = I3 XOR I2 XOR I1 XOR I0  3. 4비트 생성기 회로도  C:\Users\KSH\Desktop\even_perity_generator.png  4. 참고 사항  Odd parity bit의 경우  Pe = ( (I3 XOR I2)’ XOR (I1 XOR I0)’ )’ 이다. 즉 I3 NXOR I2 NXOR I1 NXOR I0  (혹은 Pe = I3’ XOR I2 XOR I1 XOR I0)  1-2. 검사기  [설명]  - parity bit를 덧붙인 패턴으로부터 에러를 검출해 내는 데 사용한다.  - 마찬가지의 이유로 4비트 input + Even parity bit에 대해서 설명하기로 한다.  - 생성기와 검사기의 논리식과 회로도는 같은 방식이다.  - 예를 들면 생성기는 I3, I2, I1, I0 를 통해서 Pe를 만들었고 검사기에서는 I3, I2, I1, I0, Pe를 통해 C를 만든다. 그리고 결과값 C의 값으로 에러를 판별한다. C를 검사 부호라고 한다.  - C의 값이 1이라면 에러가 있다고 판단하고 0이라면 에러가 없다고 판단한다.  [설계]  1. 논리식  C = I3 XOR I2 XOR I1 XOR I0 XOR Pe  2. 논리 회로도  C:\Users\KSH\Desktop\검사기.png  1-3 논의사항  [Odd parity bit]  - Odd parity bit는 마찬가지로 xor 대신 nxor를 사용하면 된다. 검사 부호 C가 1일 때 오류 검출, 0일 때 오류가 없다고 판단.  [문제점]  - parity bit 방식에는 문제가 있다.  - Even parity bit 방식의 경우 에러 비트 개수가 홀수라면 검출이 가능하나 짝수라면 검출이 불가능하다. 반대로 Odd parity bit 방식의 경우 에러 비트 개수가 홀수라면 검출이 불가능하다. |

|  |
| --- |
| 2. 그 외의 오류 검출기 및 정정기 |
| [오류 검출 코드]  1. Parity Check  2. Block Sum Check  - 여러 개의 비트를 모아서 2차원 배열형태로 각 행의 오류를 검출해 주는 홀수 패리티 방식과 각 열의 오류를 검출해 주는 짝수 패리티 방식을 섞어서 사용한다. 다음은 예시이다.   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |   보라색 비트들은 각각이 해당 되는 행들의(수평) 비트에서 오류를 검출해 주는 odd parity bit  푸른색 비트들은 각각이 해당 되는 열들의(수직) 비트에서 오류를 검출해 주는 even parity bit  3. CRC(Cyclic Redundancy Check) Code  - 나머지 연산을 이용한 오류 검출 코드다.  - 미리 정해놓은 CRC 발생 코드를 선정하고 이를 이용해 나머지 계산을 활용한다.  - 사용법은 다음과 같다. 전송하고 싶은 데이터를 110010, CRC 발생 코드를 1101이라고 하자.  1. CRC 발생 코드의 최고 차수를 계산하자 -> 4자리수, 최고 차수는 3이다.  2. 전송하고 싶은 데이터 뒤에 최고 차수만큼의 0을 붙인다. -> 110010 + 000 = 110010000  C:\Users\KSH\Desktop\KakaoTalk_20200427_214958225.jpg3. 2에서 만든 코드를 CRC 발생 코드로 나누고 전송데이터 + 나머지 = 전송코드를 완성한다..    => 전송 데이터 + ‘0’ 3개(최고 차수)    => 위 두 수를 비교해서 같은 수면 0을, 다른 수면 1이 튀어 나온다.  즉 +나 –가 아니라 XOR 계산.  => 나머지 ‘100’을 전송 데이터에 붙이면 전송할 코드 완성.  4. 전송 측에는 3에서와 같은 방식으로 전송코드 110010100을 CRC 발생코드 1101로 나눈다.  5. 나머지가 0(00…0)이면 error가 없다는 뜻이다.  [오류 정정 코드]  1. Hamming Code (Even parity bit 이용)  - 해밍 코드는 여러 개의 패리티 비트를 넣어서 어느 부분이 에러가 생겼는지 알려 준다.  (위키피디아)  C:\Users\KSH\Desktop\HammingCode.png  - 우선 표를 해석해보자. Bit position은 말 그대로 bit의 위치를 의미한다. Encoded data bits는 패리티 비트와 데이터 비트를 이름으로 구분한다. p는 패리티 비트, d는 데이터 비트를 의미한다. Parity bit coverage는 각 패리티 비트가 어느 데이터 비트들의 오류를 검출해주는지 나타내준다. x표시가 있는 데이터 비트를 고려한다. 예를 들면 p1은 d1, d2, d4, d5, d7, …의 데이터 비트들 중에서 오류를 검출해 준다.  - 그렇다면 어떻게 오류 비트를 콕 찝어 주는 지 알아보자.  1. 가장 먼저 각 Parity bit coverage 마다 error가 유무를 판별한다. (ex p1=1, p2=0, …)  2. Parity bit coverage 중 parity bit가 1인 행들 중 x표시가 공통적으로 표시된 데이터 비트를 선택한다.  3. 선택한 데이터 비트들 중 Parity bit coverage의 parity bit가 0인 행들에 해당하는 데이터 비트를 지운다.  4. 남아 있는 데이터 비트가 에러가 있는 부분이다.  - 구체적인 예시를 이용해서 사용법을 자세히 알아보자. 총 7비트(p1, p2, …, d4)를 사용한다. 비트 패턴을 1001110이라고 하자. 여기서 +를 xor로 사용하겠다.  1. p1 : p1+d1+d2+d4=0, p2 : p2+d1+d3+d4=1, p4 : p4+d2+d3+d4=1  2. 결과 값이 1인 p2, p4의 공통된 데이터 비트는 d3과 d4이다.  3. d3과 d4 중에서 나머지 p1에 해당하는 데이터인 d4를 제외한다. 즉 d3만 남았다.  4. 남아 있는 데이터 비트인 d3이 error가 난 비트이다.  즉 d3 = !d3;을 수행하면 검출 및 정정 완료.  - 참고로 Hamming Distance란 것이 있는데 이는 오류 비트의 개수를 의미한다. 이를 HD라고할 때 오류를 검출하기 위해선 HD+1개의 비트가 필요하고 정정하기 위해선 2HD+1개의 비트가 필요하다.  2. Cyclic codes 3. BCH codes 4. Reed-Solomon codes, 5.LDPC codes |
| 3. N bit 비교기 & IC 7485 비교기 |
| [N bit 비교기]  - N bit 비교기란 N개의 비트를 가진 두 수의 크기를 비교하는 회로를 말한다.  - 회로의 사용법은 다음과 같다.  0. A, B의 최상위 비트로 접근  1. 두 비트를 비교  A비트>B비트 : O(A>B) = 1 AND 종료  A비트<B비트 : O(A<B) = 1 AND 종료  A비트=B비트 : 2번으로.  2. 마지막 비트인가  맞으면 O(A=B) = 1 AND 종료  아니면 두 비트 모두 다음(하위) 비트로 접근 AND 1번으로.  [IC 7485 비교기]  - 해당 비교기는 4비트 비교기이다.  - 4비트 비교기로 8, 16, … 비트 비교기를 구현하기 위해 input 비트가 2\*4+3=11 비트다.  - 즉 A, B 각각 4개의 비트와 앞선 4비트 비교기의 결과를 입력으로 받을 수 있는 I(A>B) I(A<B) I(A=B) input 3개가 추가되어 총 11개의 input이 존재한다.  C:\Users\KSH\Desktop\비교기.png 왼쪽 비교기에서 보통 I(A?B)는 현재 비교하는 비트들의  자리수보다 낮은 자리수의 비교 결과가 들어온다..   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **입력** | | | | | | | **출력** | | | | **A3 , B3** | **A2 , B2** | **A1 , B1** | **A0 , B0** | **IA>B** | **IA<B** | **IA=B** | **OA>B** | **OA<B** | **OA=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 | 1 | 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 | 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 |   Xi = Ai ☉ Bi = Ai NXOR Bi  OA=B = X3 X2 X1 X0 IA=B  OA>B = A3B3’+ X3 A2 B2’+ X3 X2 A1 B1’+ X3 X2 X1 A0 B0’+ X3 X2 X1 X0 IA>B+ X3 X2 X1 X0 IA>B’IA<B’ IA=B’  OA>B = A3’B3+ X3 A2’B2+ X3 X2 A1’B1+ X3 X2 X1 A0’B0+ X3 X2 X1 X0 IA<B+ X3 X2 X1 X0 IA>B’IA<B’ IA=B’ |