7주차 결과보고서

전공: 아트&테크놀로지학과 학년: 4학년 학번: 20191048 이름: 김도솔

**1.**

**1) Even Parity bit generator**

* 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| In A | In B | In C | In D | Out P |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

Even Parity bit generator는 입력 A, B, C, D 중에서 ‘1’의 개수가 홀수인 경우 Parity bit 값(P)으로 1을 출력하고, ‘1’의 개수가 짝수인 경우 Parity bit 값(P)으로 0을 출력한다. 이러한 특성을 반영해 진리표를 작성하면 위와 같은 결과가 나온다.

* K-map 작성

텍스트, 도표, 스크린샷, 디자인이(가) 표시된 사진

자동 생성된 설명

위의 진리표를 바탕으로 카르노맵을 작성한 결과이다. minimum SOP를 구하는 과정을 통해 라는 P에 대한 Boolean function을 얻을 수 있었다. 이를 XOR 연산자를 이용해 재구성하면 다음과 같이 더 간단하게 나타낼 수 있다.

* Verilog 코딩

|  |  |
| --- | --- |
| Design Source Code | Simulation Source Code |
| `timescale 1ns / 1ps  module pg(  input a, b, c, d,  output p  );  assign p = a^b^c^d;  endmodule | `timescale 1ns / 1ps  module pg\_tb;  reg aa, bb, cc, dd;  wire p;  pg u\_pg (  .a (aa ),  .b (bb ),  .c (cc ),  .d (dd ),  .p (p )  );  initial begin  aa = 1'b0;  bb = 1'b0;  cc = 1'b0;  dd = 1'b0;  end  always aa = #80 ~aa;  always bb = #40 ~bb;  always cc = #20 ~cc;  always dd = #10 ~dd;  initial begin  #1000  $finish;  end  endmodule |

위에서 K-map을 통해 구한 Boolean function을 바탕으로 Even Parity bit generator의 Verilog 코드를 작성한 결과이다. pg 모듈은 4개의 입력(a, b, c, d)과 1개의 출력(p)을 가지고 있다. 출력 p는 입력 a, b, c, d를 XOR 연산한 결과이다.

* Schematic Diagram

도표, 라인, 텍스트, 폰트이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드로 Schematic Diagram을 생성한 결과이다. 이를 통해 작성한 코드가 회로에 어떤 식으로 구현되는지 확인할 수 있었다.

* Simulation 결과

스크린샷이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 a, b, c, d의 값을 바꾸면 그에 따라 출력 p의 값이 변화하는 것을 확인할 수 있다. 입력 a, b, c, d 중 1의 개수가 홀수 개면 p 값이 1이고, 입력 a, b, c, d 중 1의 개수가 짝수 개면 p 값으로 0을 출력한다. 이를 통해 K-map 작성으로 얻은 minimum SOP가 의도한 대로 잘 동작한다는 것을 실험적으로 확인할 수 있었다.

**2) Even Parity bit checker**

* 진리표

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| In A | In B | In C | In D | P | Out E |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

앞서 본 Even Parity bit generator는 Parity bit를 포함한 전체 데이터에서 ‘1’의 개수가 짝수가 되도록 Parity bit를 설정한다. 이러한 성질을 이용해 Even Parity bit checker는 입력 A, B, C, D, P에서 ‘1’의 개수가 홀수인 경우 1을 출력하고, ‘1’의 개수가 짝수인 경우 0을 출력해 입력 데이터의 오류를 검출한다. 이때 0을 출력하는 경우는 입력 데이터에 오류가 없음을 나타내고, 1을 출력하는 경우는 입력 데이터에 오류가 있음을 나타낸다. 이러한 특성을 반영해 진리표를 작성하면 위와 같은 결과가 나온다.

* K-map 작성

텍스트, 도표, 스크린샷, 사각형이(가) 표시된 사진

자동 생성된 설명

위의 진리표를 바탕으로 카르노맵을 작성한 결과이다. minimum SOP를 구하는 과정을 통해 라는 E에 대한 Boolean function을 얻을 수 있었다. 이를 XOR 연산자를 이용해 재구성하면 다음과 같이 더 간단하게 나타낼 수 있다.

* Verilog 코딩

|  |  |
| --- | --- |
| Design Source Code | Simulation Source Code |
| `timescale 1ns / 1ps  module pc(  input a, b, c, d, p,  output e  );  assign e = a^b^c^d^p;  endmodule | `timescale 1ns / 1ps  module pc\_tb;  reg aa, bb, cc, dd, pp;  wire e;  pc u\_pc (  .a (aa ),  .b (bb ),  .c (cc ),  .d (dd ),  .p (pp ),  .e (e )  );  initial begin  aa = 1'b0;  bb = 1'b0;  cc = 1'b0;  dd = 1'b0;  pp = 1'b0;  end  always aa = #160 ~aa;  always bb = #80 ~bb;  always cc = #40 ~cc;  always dd = #20 ~dd;  always pp = #10 ~pp;  initial begin  #1000  $finish;  end  endmodule |

위에서 K-map을 통해 구한 Boolean function을 바탕으로 Even Parity bit checker의 Verilog 코드를 작성한 결과이다. pc 모듈은 5개의 입력(a, b, c, d, p)과 1개의 출력(e)을 가지고 있다. 출력 e는 입력 a, b, c, d, p를 XOR 연산한 결과이다.

* Schematic Diagram

도표, 라인, 그래프, 평면도이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드로 Schematic Diagram을 생성한 결과이다. 이를 통해 작성한 코드가 회로에 어떤 식으로 구현되는지 확인할 수 있었다.

* Simulation 결과

스크린샷이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 a, b, c, d, p의 값을 바꾸면 그에 따라 출력 e의 값이 변화하는 것을 확인할 수 있다. 입력 a, b, c, d, p 중 1의 개수가 홀수 개면 e 값이 1이고, 입력 a, b, c, d 중 1의 개수가 짝수 개면 e 값으로 0을 출력한다. 이를 통해 K-map 작성으로 얻은 minimum SOP가 의도한 대로 잘 동작한다는 것을 실험적으로 확인할 수 있었다.

**2.**

**1) Odd Parity bit generator**

* 진리표

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| In A | In B | In C | In D | Out P |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Odd Parity bit generator는 입력 A, B, C, D 중에서 ‘1’의 개수가 홀수인 경우 Parity bit 값(P)으로 0을 출력하고, ‘1’의 개수가 짝수인 경우 Parity bit 값(P)으로 1을 출력한다. 이러한 특성을 반영해 진리표를 작성하면 위와 같은 결과가 나온다.

* K-map 작성

도표, 텍스트이(가) 표시된 사진

자동 생성된 설명

위의 진리표를 바탕으로 카르노맵을 작성한 결과이다. minimum SOP를 구하는 과정을 통해 라는 P에 대한 Boolean function을 얻을 수 있었다. 이를 XOR 연산자를 이용해 재구성하면 다음과 같이 더 간단하게 나타낼 수 있다.

* Verilog 코딩

|  |  |
| --- | --- |
| Design Source Code | Simulation Source Code |
| `timescale 1ns / 1ps  module pg(  input a, b, c, d,  output p  );  assign p = ~(a^b^c^d);  endmodule | `timescale 1ns / 1ps  module pg\_tb;  reg aa, bb, cc, dd;  wire p;  pg u\_pg (  .a (aa ),  .b (bb ),  .c (cc ),  .d (dd ),  .p (p )  );  initial begin  aa = 1'b0;  bb = 1'b0;  cc = 1'b0;  dd = 1'b0;  end  always aa = #80 ~aa;  always bb = #40 ~bb;  always cc = #20 ~cc;  always dd = #10 ~dd;  initial begin  #1000  $finish;  end  endmodule |

위에서 K-map을 통해 구한 Boolean function을 바탕으로 Odd Parity bit generator의 Verilog 코드를 작성한 결과이다. pg 모듈은 4개의 입력(a, b, c, d)과 1개의 출력(p)을 가지고 있다. 출력 p는 입력 a, b, c, d를 XOR 연산한 결과에 NOT을 취한 값이다.

* Schematic Diagram

도표, 라인, 폰트, 그래프이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드로 Schematic Diagram을 생성한 결과이다. 이를 통해 작성한 코드가 회로에 어떤 식으로 구현되는지 확인할 수 있었다.

* Simulation 결과

스크린샷, 다채로움이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 a, b, c, d의 값을 바꾸면 그에 따라 출력 p의 값이 변화하는 것을 확인할 수 있다. 입력 a, b, c, d 중 1의 개수가 홀수 개면 p 값이 0이고, 입력 a, b, c, d 중 1의 개수가 짝수 개면 p 값으로 1을 출력한다. 이를 통해 K-map 작성으로 얻은 minimum SOP가 의도한 대로 잘 동작한다는 것을 실험적으로 확인할 수 있었다.

**2) Odd Parity bit checker**

* 진리표

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| In A | In B | In C | In D | P | Out E |
| 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 |

앞서 본 Odd Parity bit generator는 Parity bit를 포함한 전체 데이터에서 ‘1’의 개수가 홀수가 되도록 Parity bit를 설정한다. 이러한 성질을 이용해 Odd Parity bit checker는 입력 A, B, C, D, P에서 ‘1’의 개수가 홀수인 경우 0을 출력하고, ‘1’의 개수가 짝수인 경우 1을 출력해 입력 데이터의 오류를 검출한다. 이때 0을 출력하는 경우는 입력 데이터에 오류가 없음을 나타내고, 1을 출력하는 경우는 입력 데이터에 오류가 있음을 나타낸다. 이러한 특성을 반영해 진리표를 작성하면 위와 같은 결과가 나온다.

* K-map 작성

텍스트, 도표, 스크린샷, 평면도이(가) 표시된 사진

자동 생성된 설명

위의 진리표를 바탕으로 카르노맵을 작성한 결과이다. minimum SOP를 구하는 과정을 통해 라는 E에 대한 Boolean function을 얻을 수 있었다. 이를 XOR 연산자를 이용해 재구성하면 다음과 같이 더 간단하게 나타낼 수 있다.

* Verilog 코딩

|  |  |
| --- | --- |
| Design Source Code | Simulation Source Code |
| `timescale 1ns / 1ps  module pc(  input a, b, c, d, p,  output e  );  assign e = ~(a^b^c^d^p);  endmodule | `timescale 1ns / 1ps  module pc\_tb;  reg aa, bb, cc, dd, pp;  wire e;  pc u\_pc (  .a (aa ),  .b (bb ),  .c (cc ),  .d (dd ),  .p (pp ),  .e (e )  );  initial begin  aa = 1'b0;  bb = 1'b0;  cc = 1'b0;  dd = 1'b0;  pp = 1'b0;  end  always aa = #160 ~aa;  always bb = #80 ~bb;  always cc = #40 ~cc;  always dd = #20 ~dd;  always pp = #10 ~pp;  initial begin  #1000  $finish;  end  endmodule |

위에서 K-map을 통해 구한 Boolean function을 바탕으로 Odd Parity bit checker의 Verilog 코드를 작성한 결과이다. pc 모듈은 5개의 입력(a, b, c, d, p)과 1개의 출력(e)을 가지고 있다. 출력 e는 입력 a, b, c, d, p를 XOR 연산한 결과에 NOT을 취한 값이다.

* Schematic Diagram

도표, 라인, 그래프, 폰트이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드로 Schematic Diagram을 생성한 결과이다. 이를 통해 작성한 코드가 회로에 어떤 식으로 구현되는지 확인할 수 있었다.

* Simulation 결과

스크린샷이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 a, b, c, d, p의 값을 바꾸면 그에 따라 출력 e의 값이 변화하는 것을 확인할 수 있다. 입력 a, b, c, d, p 중 1의 개수가 홀수 개면 e 값이 0이고, 입력 a, b, c, d 중 1의 개수가 짝수 개면 e 값으로 1을 출력한다. 이를 통해 K-map 작성으로 얻은 minimum SOP가 의도한 대로 잘 동작한다는 것을 실험적으로 확인할 수 있었다.

**3.**

**2-bit binary comparator**

* 진리표

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

2-bit binary comparator는 2 bit로 구성된 두 개의 이진수를 비교한 결과를 세 가지 경우로 나누어 출력하는 논리 회로이다. 위의 진리표에서 , 는 A의 각 자리 수를 나타내고, , 는 B의 각 자리 수를 나타낸다. 은 A > B인 경우, 는 A = B인 경우, 는 A < B인 경우 1을 출력한다. 이러한 특성을 반영해 진리표를 작성하면 위와 같은 결과가 나온다.

* K-map 작성

텍스트, 도표, 스크린샷, 번호이(가) 표시된 사진

자동 생성된 설명도표, 텍스트, 디자인이(가) 표시된 사진

자동 생성된 설명

위의 진리표를 바탕으로 카르노맵을 작성한 결과이다. minimum SOP를 구하는 과정을 통해 다음과 같은 3개의 Boolean Function을 얻을 수 있었다.

이때 의 경우 XOR 연산자를 이용해 식을 재구성하면 다음과 같이 더 간단하게 나타낼 수 있다.

* Verilog 코딩

|  |  |
| --- | --- |
| Design Source Code | Simulation Source Code |
| `timescale 1ns / 1ps  module cmp(  input a1, a2, b1, b2,  output f1, f2, f3  );  assign f1 = (a1&~b1)|(a2&~b1&~b2)|(a1&a2&~b2);  assign f2 = ~(a1^b1)&~(a2^b2);  assign f3 = (~a1&b1)|(~a1&~a2&b2)|(~a2&b1&b2);  endmodule | `timescale 1ns / 1ps  module cmp\_tb;  reg a1, a2, b1, b2;  wire f1, f2, f3;  cmp u\_cmp (  .a1 (a1 ),  .a2 (a2 ),  .b1 (b1 ),  .b2 (b2 ),  .f1 (f1 ),  .f2 (f2 ),  .f3 (f3 )  );  initial begin  a1 = 1'b0;  a2 = 1'b0;  b1 = 1'b0;  b2 = 1'b0;  end  always a1 = #80 ~a1;  always a2 = #40 ~a2;  always b1 = #20 ~b1;  always b2 = #10 ~b2;  initial begin  #1000  $finish;  end  endmodule |

위에서 K-map을 통해 구한 Boolean function을 바탕으로 2-bit binary comparator의 Verilog 코드를 작성한 결과이다. cmp 모듈은 4개의 입력(a1, a2, b1, b2)과 3개의 출력(f1, f2, f3)을 가지고 있다. f1은 (a1&~b1)|(a2&~b1&~b2)|(a1&a2&~b2)를 연산한 결과이고, f2는 ~(a1^b1)&~(a2^b2)의 결과이고, f3는 (~a1&b1)|(~a1&~a2&b2)|(~a2&b1&b2)의 결과이다.

* Schematic Diagram

도표, 텍스트, 평면도, 라인이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드로 Schematic Diagram을 생성한 결과이다. 이를 통해 작성한 코드가 회로에 어떤 식으로 구현되는지 확인할 수 있었다.

* Simulation 결과

스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명

위에서 작성한 Verilog 코드를 시뮬레이션 돌린 결과이다. 시간에 따라 입력 a1, a2, b1, b, 2의 값을 바꾸면 그에 따라 출력 f1, f2, f3의 값이 변화하는 것을 확인할 수 있다. 입력 a1, a2가 하나의 2-bit짜리 이진수 A를 구성하고, 입력 b1, b2가 하나의 2-bit짜리 이진수 B를 구성한다고 하자. 이때 a2와 b2는 LSB를 나타낸다. A > B인 경우 f1이 1을 출력하고 A = B인 경우 f2가 1을 출력하고 A < B인 경우 f3가 1을 출력하는 것을 위의 시뮬레이션을 통해 알 수 있다. 이를 통해 K-map 작성으로 얻은 minimum SOP가 의도한 대로 잘 동작한다는 것을 실험적으로 확인할 수 있었다.

**4.**

카르노맵으로 구한 Even/Odd parity bit generator와 checker의 minimum SOP 식은 매우 복잡하게 나왔는데 이를 XOR 연산을 이용해 간단하게 나타낼 수 있다는 점이 흥미로웠다. 이번 실습에서는 조교님이 힌트를 주셔서 식을 XOR로 간소화 해야겠다는 아이디어가 빠르게 떠올랐지만 다른 경우에는 바로 생각해내기 어려울 것 같다. 따라서 특정 규칙을 가진 진리표나 카르노 맵을 보고 직관적으로 식을 생각해보는 연습을 하면 좋을 것 같다.

이번 실험에서 다룬 2-bit binary comparator는 지난 실습에서 다룬 1-bit binary comparator보다복잡했지만 작동 방법이 동일해 진리표를 작성하는데 오래 걸리지 않았다. 또한 저번 실습에서 한번 카르노맵 그리는 연습을 해봐서 그런지 minimum SOP 구하는데도 오랜 시간을 투자하지 않았다. 앞으로 카르노맵 그리는 연습을 더 하면 실습하는데 걸리는 시간을 더욱 줄일 수 있을 것 같다.

**5.**

**Checksum**은 데이터 무결성을 검증하기 위한 간단한 오류 탐지 기술이다. 주로 데이터 전송 중에 발생하는 임의의 오류를 감지하고 수정하기 위해 사용되며 주로 컴퓨터 네트워크, 데이터 저장 장치 및 데이터 통신에서 중요한 역할을 한다. Checksum과 Parity Bit는 둘다 기존 데이터를 분석해 오류 검출을 위한 추가적인 비트를 덧붙여 보낸다는 점에서 공통점을 가진다. 그러나 Checksum은 데이터의 모든 비트나 바이트를 조합하여 특정 규칙 또는 알고리즘을 사용하여 계산한다. 일반적으로 여러 비트에 대한 계산을 수행하므로 데이터 무결성을 더 정확하게 판단할 수 있다. 예를 들어, 모든 바이트의 합을 계산하는 방식이 있다. 반면 이번 실습에서 진행한 Parity Bit는 보다 간단한 방식으로 동작한다. 홀수 패리티인 경우에는 데이터 비트들의 합이 홀수가 되도록 조정하고, 짝수 패리티인 경우에는 합을 짝수로 만든다. 따라서 Checksum의 경우 보통 네트워크 통신, 데이터 저장 및 파일 전송과 같이 비교적 안정적인 환경에서 사용된다. 데이터 무결성이 중요한 곳에서 사용되며, 복잡한 오류 패턴을 감지할 필요가 있는 경우에 적합하다. Parity Bit는 주로 단순한 환경 또는 하드웨어 수준에서 사용된다. 메모리 모듈의 오류 검출이나 특정 하드웨어 장치에서 사용될 수 있으며, 비용과 오버헤드를 최소화해야 하는 경우에 유용하다.