**컴퓨터공학 설계 및 실험Ⅱ**

7주차 결과보고서

서강대학교 공학부 컴퓨터공학 전공

20171646 박태윤

**1. Even Parity bit generator 및 checker 의 simulation 결과 및 과정에 대해서 설명하시오. (Truth table 작성 및 k-map 포함)**

(1) Even Parity bit generator

- Truth table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **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 |

- 카르노 맵

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AB / CD** | **00** | **01** | **11** | **10** |
| **00** | 0 | 1 | 0 | 1 |
| **01** | 1 | 0 | 1 | 0 |
| **11** | 0 | 1 | 0 | 1 |
| **10** | 1 | 0 | 1 | 0 |

P = A’B’C’D + A’B’CD’ + A’BC’D’ + A’BCD + ABC’D + ABCD’ + AB’C’D’ + AB’CD

= A’B’(C’D+CD’) + A’B(C’D’+CD) + AB(C’D+CD’) + AB’(C’D’+CD)

= A’B’(CD) + A’B(CD)’ + AB(CD) + AB’(CD)’

= (CD)(AB)’+(CD)’(AB)

= ABCD

- Verilog Code

|  |  |
| --- | --- |
| **even\_generator.v** | **even\_generator\_tb.v** |
| `timescale 1ns / 1ps  module even\_generator(ia,ib,ic,id,op);  input ia,ib,ic,id;  output op;  assign op = ia^ib^ic^id;  endmodule | `timescale 1ns / 1ps  module even\_generator\_tb;  reg a,b,c,d;  wire p;  even\_generator u\_even\_generator(  .ia(a), .ib(b), .ic(c), .id(d),  .op(p)  );  initial a = 1'b0; initial b = 1'b0; initial c = 1'b0; initial d = 1'b0;  always a = #80 ~a;  always b = #40 ~b;  always c = #20 ~c;  always d = #10 ~d;  initial begin |

- Simulation

스크린샷, 컴퓨터, 노트북, 모니터이(가) 표시된 사진

자동 생성된 설명

(2) Even Parity bit checker

- Truth table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **P** | **PEC** |
| 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 |

- 카르노 맵

(A=0)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BC / DP** | **00** | **01** | **11** | **10** |
| **00** | 0 | 1 | 0 | 1 |
| **01** | 1 | 0 | 1 | 0 |
| **11** | 0 | 1 | 0 | 1 |
| **10** | 1 | 0 | 1 | 0 |

(A=1)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BC / DP** | **00** | **01** | **11** | **10** |
| **00** | 1 | 0 | 1 | 0 |
| **01** | 0 | 1 | 0 | 1 |
| **11** | 1 | 0 | 1 | 0 |
| **10** | 0 | 1 | 0 | 1 |

PEC= A’B’C’D’P+A’B’C’DP’+A’B’CD’P’+A’B’CDP+A’BCD’P

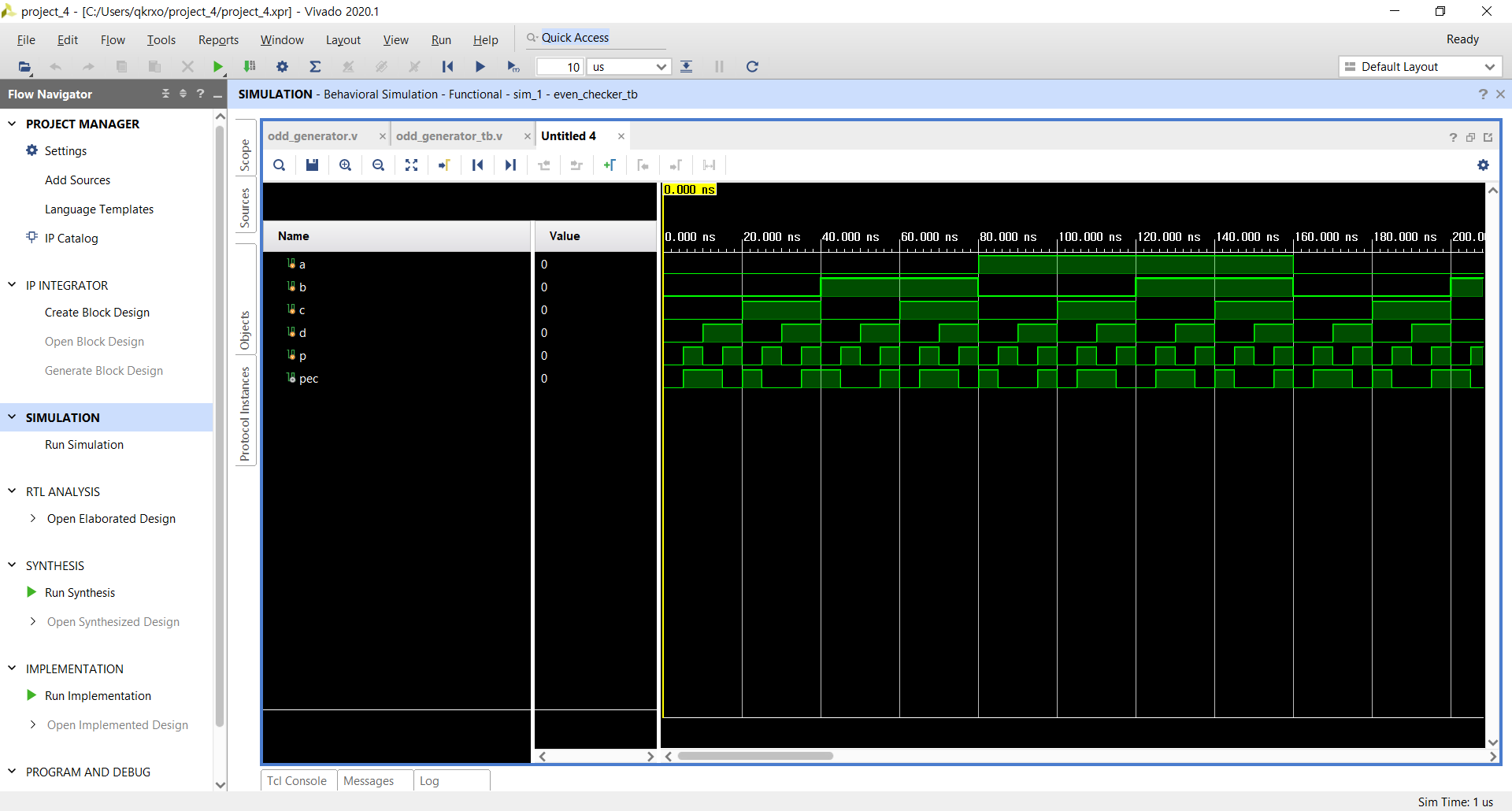
+A’BCDP’+A’BC’D’P’+A’BC’DP

= ABCDP

- Verilog Code

|  |  |
| --- | --- |
| **even\_checker.v** | **even\_checker\_tb.v** |
| `timescale 1ns / 1ps  module even\_checker(ia,ib,ic,id,ip,opec);  input ia,ib,ic,id,ip;  output opec;  assign opec = ia^ib^ic^id^ip;  endmodule | `timescale 1ns / 1ps  module even\_checker\_tb;  reg a,b,c,d,p;  wire pec;  even\_checker u\_even\_checker(  .ia(a), .ib(b), .ic(c), .id(d), .ip(p),  .opec(pec)  );  initial a = 1'b0; initial b = 1'b0; initial c = 1'b0; initial d = 1'b0; initial p = 1'b0;  always a = #80 ~a;  always b = #40 ~b;  always c = #20 ~c;  always d = #10 ~d;  always p = #5 ~p;  endmodule |

- Simulation



**2. Odd Parity bit generator 및 checker 의 simulation 결과 및 과정에 대해서 설명하시오. (Truth table 작성 및 k-map 포함)**

(1) Odd Parity bit generator

- Truth table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **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 |

- 카르노 맵

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **AB / CD** | **00** | **01** | **11** | **10** |
| **00** | 1 | 0 | 1 | 0 |
| **01** | 0 | 1 | 0 | 1 |
| **11** | 1 | 0 | 1 | 0 |
| **10** | 0 | 1 | 0 | 1 |

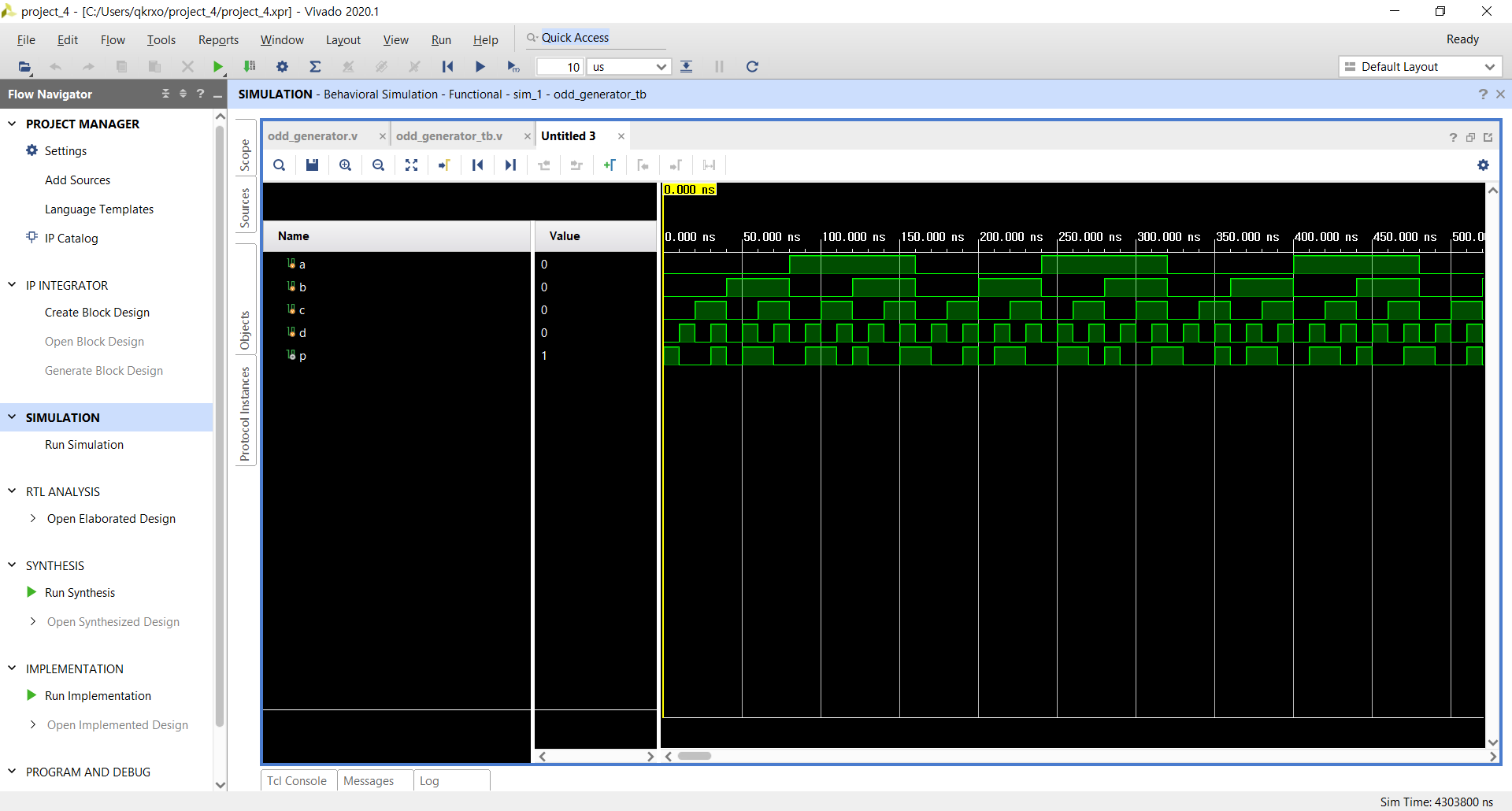
P = A’B’C’D’ + A’B’CD + A’BC’D + A’BCD’ + ABC’D’ + ABCD + AB’C’D + AB’CD’

= (ABCD)’

- Verilog Code

|  |  |
| --- | --- |
| **odd\_generator.v** | **odd\_generator\_tb.v** |
| `timescale 1ns / 1ps  module odd\_generator(ia,ib,ic,id,op);  input ia,ib,ic,id;  output op;  assign op = ~(ia^ib^ic^id);  endmodule | `timescale 1ns / 1ps  module odd\_generator\_tb;  reg a,b,c,d;  wire p;  odd\_generator u\_odd\_generator(  .ia(a), .ib(b), .ic(c), .id(d),  .op(p)  );  initial a = 1'b0; initial b = 1'b0; initial c = 1'b0; initial d = 1'b0;  always a = #80 ~a;  always b = #40 ~b;  always c = #20 ~c;  always d = #10 ~d;  endmodule |

- Simulation



(2) Odd Parity bit checker

- Truth table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **P** | **PEC** |
| 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 |

- 카르노 맵

(A=0)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BC / DP** | **00** | **01** | **11** | **10** |
| **00** | 1 | 0 | 1 | 0 |
| **01** | 0 | 1 | 0 | 1 |
| **11** | 1 | 0 | 1 | 0 |
| **10** | 0 | 1 | 0 | 1 |

(A=1)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **BC / DP** | **00** | **01** | **11** | **10** |
| **00** | 0 | 1 | 0 | 1 |
| **01** | 1 | 0 | 1 | 0 |
| **11** | 0 | 1 | 0 | 1 |
| **10** | 1 | 0 | 1 | 0 |

PEC= A’B’C’D’P’+A’B’C’DP+A’B’CD’P+A’B’CDP’+A’BCD’P’

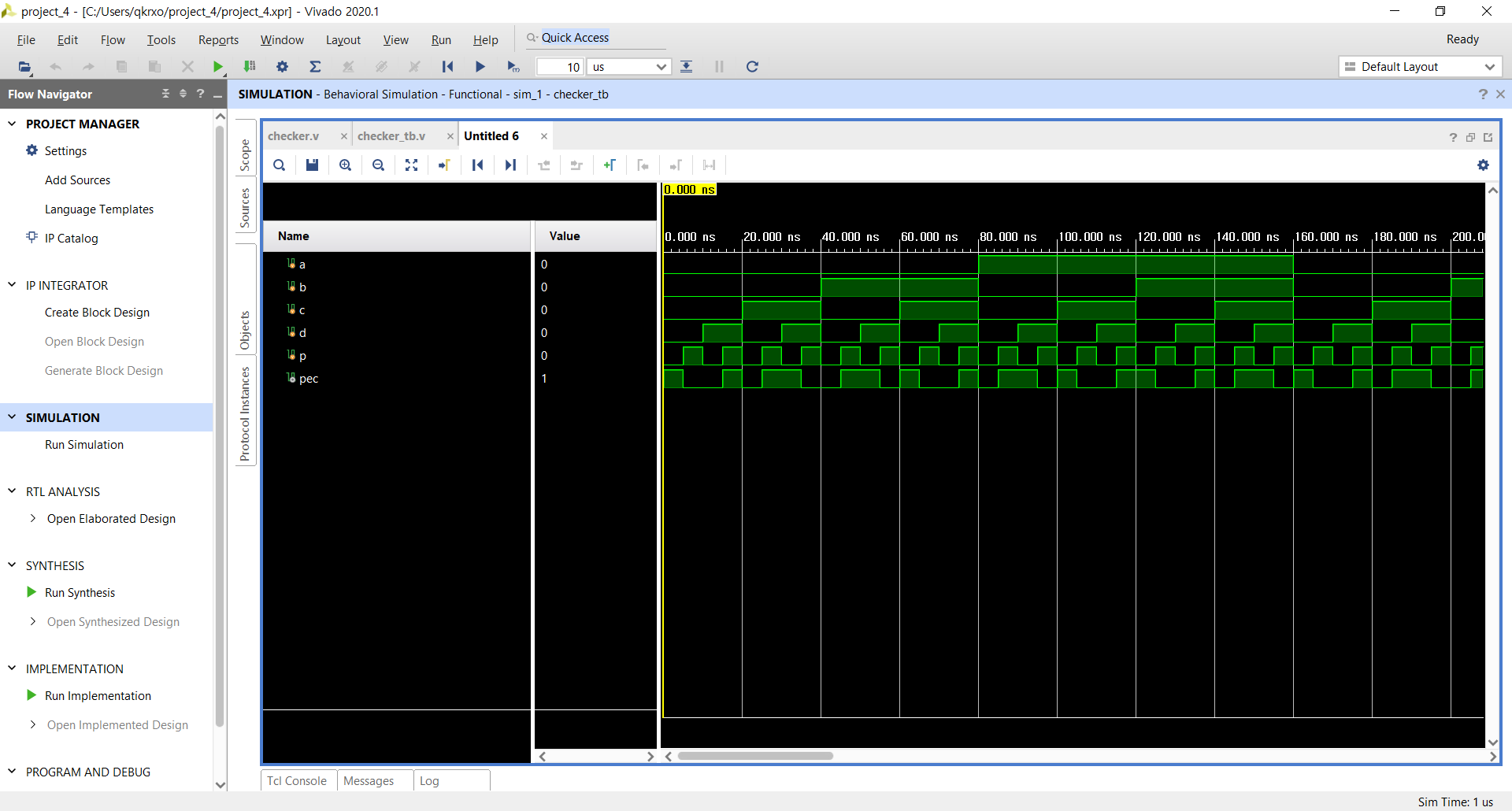
+A’BCDP+A’BC’D’P+A’BC’DP’

= (ABCDP)’

- Verilog Code

|  |  |
| --- | --- |
| **odd\_checker.v** | **odd\_checker\_tb.v** |
| `timescale 1ns / 1ps  module odd\_checker(ia,ib,ic,id,ip,opec);  input ia,ib,ic,id,ip;  output opec;  assign opec = ~(ia^ib^ic^id^ip);  endmodule | `timescale 1ns / 1ps  module odd\_checker\_tb;  reg a,b,c,d,p;  wire pec;  odd\_checker u\_odd\_checker(  .ia(a), .ib(b), .ic(c), .id(d), .ip(p),  .opec(pec)  );  initial a = 1'b0; initial b = 1'b0; initial c = 1'b0; initial d = 1'b0; initial p = 1'b0;  always a = #80 ~a;  always b = #40 ~b;  always c = #20 ~c;  always d = #10 ~d;  always p = #5 ~p;  endmodule |

- Simulation



**3. 2-bit binary comparator simulation 결과 및 과정에 대해서 설명하시오. (Truth table 작성 및 k-map 포함)**

- Truth table

(input)

a,b : 2-bit of A

c,d : 2-bit of B

(output)

L : A > B 일 때 1출력

S : A < B 일 때 1출력

E : A = B 일 때 1출력

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **a** | **b** | **c** | **d** | **L** | **S** | **E** |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 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 | 0 | 1 |

- 카르노 맵

(L)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ab / cd** | **00** | **01** | **11** | **10** |
| **00** | 0 | 0 | 0 | 0 |
| **01** | 1 | 0 | 0 | 0 |
| **11** | 1 | 1 | 0 | 1 |
| **10** | 1 | 1 | 0 | 0 |

L = ac’ + bc’d’ + abd’

(S)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ab / cd** | **00** | **01** | **11** | **10** |
| **00** | 0 | 1 | 1 | 1 |
| **01** | 0 | 0 | 1 | 1 |
| **11** | 0 | 0 | 0 | 0 |
| **10** | 0 | 0 | 1 | 0 |

S = a’c + a’b’d + b’cd

(E)

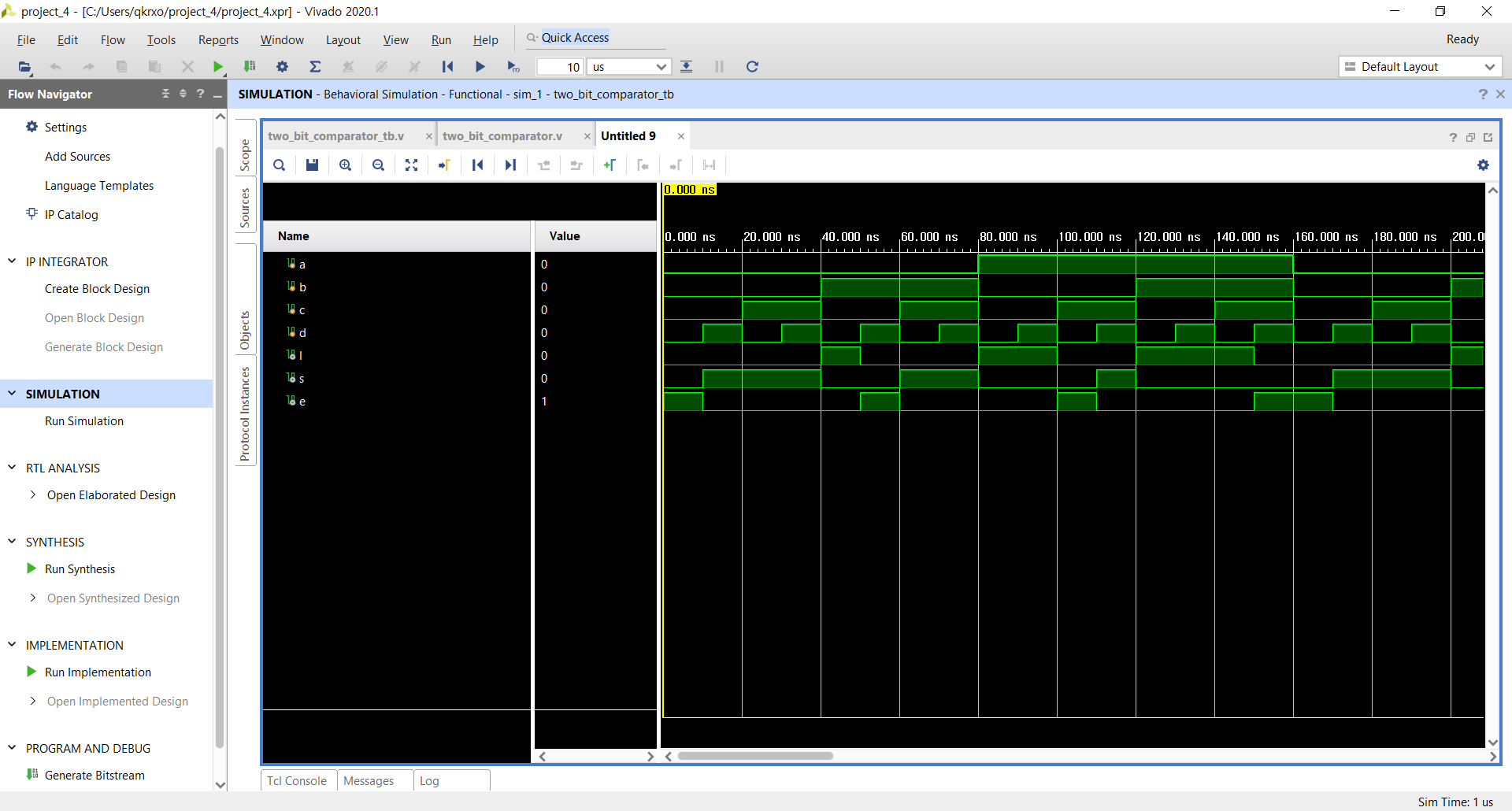
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ab / cd** | **00** | **01** | **11** | **10** |
| **00** | 1 | 0 | 0 | 0 |
| **01** | 0 | 1 | 0 | 0 |
| **11** | 0 | 0 | 1 | 0 |
| **10** | 0 | 0 | 0 | 1 |

E = a’b’c’d’ + a’bc’d + abcd + ab’cd’

- Verilog Code

|  |
| --- |
| **two\_bit\_comparator** |
| `timescale 1ns / 1ps  module two\_bit\_comparator(ia,ib,ic,id,ol,os,oe);  input ia,ib,ic,id;  output ol,os,oe;  assign ol = (ia&(~ic))|(ib&(~ic)&(~id))|(ia&ib&(~id));  assign os = ((~ia)&ic)|((~ia)&(~ib)&id)|((~ib)&ic&id);  assign oe = ((~ia)&(~ib)&(~ic)&(~id))|((~ia)&ib&(~ic)&id)|(ia&ib&ic&id)|(ia&(~ib)&ic&(~id));  endmodule |
| **two\_bit\_comparator\_tb** |
| `timescale 1ns / 1ps  module two\_bit\_comparator\_tb;  reg a,b,c,d;  wire l,s,e;  two\_bit\_comparator u\_two\_bit\_comparator(  .ia(a), .ib(b), .ic(c), .id(d),  .ol(l), .os(s), .oe(e)  );  initial a = 1'b0; initial b = 1'b0; initial c = 1'b0; initial d = 1'b0;  always a = #80 ~a;  always b = #40 ~b;  always c = #20 ~c;  always d = #10 ~d;  endmodule |

- Simulation



**4. 결과 검토 및 논의 사항.**

even parity bit generator와 checker, odd parity bit generator와 checker의 진리표를 작성하고 카르노맵으로 식을 간소화했으며 이후 Verilog로 구현하여 Simulation을 통해 최종 결과를 확인하였다. even parity bit generator(checker)와 odd parity bit generator(checker)는 서로 inversion한 결과임을 간소화된 식과 Simulation을 통해 알아내었다. 두 가지의 parity bit generator와 checker에 대한 카르노 맵을 작성하면서 카르노 맵은 간소화된 SOP 또는 POS형태만 나타낼 수 있을 뿐 XOR로 간소화 할 수 있는 경우에는 카르노 맵에서 구한 식에서 추가적인 과정을 통해 간소화를 진행하여야했다.

2-bit binary comparator또한 진리표를 작성한 뒤 카르노맵을 만들고 이를 토대로 Verilog로 구현한 뒤 Simulation을 통해 결과를 확인하였다. 여기서 이진수 A와 B의 각 bit를 나타내는 a,b 와 c,d를 입력으로 받아 총 3개의 출력 l,s,e 중 A가 B보다 크면 l, B가 A보다 크면 s, A와 B가 같으면 e에서 1을 출력하는 방식으로 구현하였다.

**5. 추가 이론 조사 및 작성.**

하나의 Parity bit로는 에러 발생 유무만 확인이 가능하고 에러의 정정은 불가능하다. Parity bit를 사용하여 에러를 검출한 뒤 정정을 할 수 있도록 하는 방법이 있는데 이를 Parrel Parity라고 한다. Parity bit를 가로, 세로로 구성되는 블록에 적용하여 검출 및 정정을 진행한다.

|  |  |
| --- | --- |
|  |  |

Parity bit를 가로, 세로에 각각 표현하였기 때문에 오류가 발생하면 가로줄과 세로줄이 겹치는 부분이 오류가 발생한 bit임을 확인할 수 있다. 오류가 발생한 bit의 위치를 알 수 있기 때문에 이를 정정할 수 있다