전공: 컴퓨터공학 학년: 2학년 학번: 20161603 이름: 신민준

1. Even parity bit generator 및 checker의 simulation 결과 및 과정에 대해서 설명하시오. (Truth table 작성 및 K-map 포함)

* Parity bit generator

Parity bit generator의 truth table은 다음과 같다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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 |

진리표를 사용해 K-map을 그리고, 함수에 대한 식을 정리하면 다음과 같다.

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

이 K-map에서 얻은 식을 기반으로 Design Source를 작성했다.

|  |
| --- |
| `timescale 1ns / 1ps  module ParityGen(      input a, b, c, d,      output p      );      assign p = (a^b)^(c^d);  endmodule |

Design Source의 Schematic은 다음과 같다.

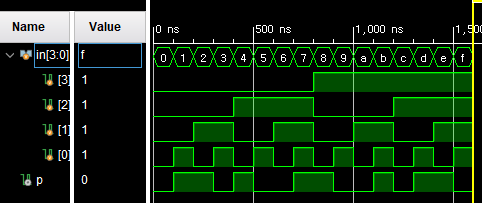
텍스트, 지도이(가) 표시된 사진

자동 생성된 설명

이 Design source를 simulation하기 위해 다음과 같은 Simulation Source를 작성했다.

|  |
| --- |
| `timescale 1ns / 1ps  module parity\_sim;      reg [3:0] in;      wire p;      ParityGen pg(in[3], in[2], in[1], in[0], p);      initial in = 4'b0000;      always in = #100 in+1;      initial begin          #1600          $finish;      end  endmodule |

Simulation 결과는 다음과 같았다.



* Parity bit checker

Parity bit checker의 K-map은 다음과 같은 5-variable K-map으로 나타내어진다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| p = 0 | | | | |
| 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 = 1 | | | | |
| 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 |

위 K-map을 정리한 식은 다음과 같다.

이 식을 Design source로 구현하기 위해 다음과 같은 Verilog Code를 작성했다.

|  |
| --- |
| `timescale 1ns / 1ps  module ParityCheck(      input a, b, c, d, p,      output pec      );      assign pec = a^b^c^d^p;  endmodule |

이 Verilog code의 Schematic은 다음과 같다.

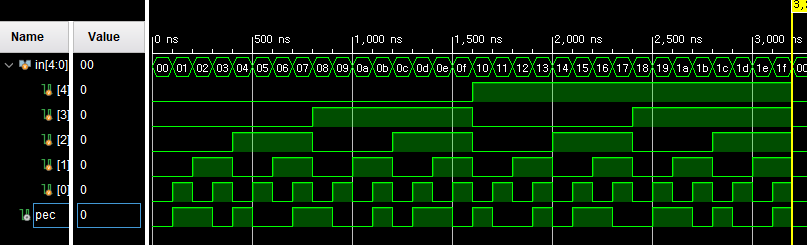
텍스트이(가) 표시된 사진

자동 생성된 설명

Simulation을 위해 다음과 같은 simulation source를 작성했다.

|  |
| --- |
| `timescale 1ns / 1ps  module pc\_sim;      reg [4:0] in;      wire pec;      ParityCheck pc(in[4], in[3], in[2], in[1], in[0], pec);      initial in = 4'b0000;      always in = #100 in+1;      initial begin          #1600          $finish;      end  endmodule |

이 Simulation의 결과는 다음과 같다.



1. Odd parity bit generator 및 checker의 simulation 결과 및 과정에 대해서 설명하시오. (Truth table 작성 및 K-map 포함)
2. 2-bit binary comparator 의 simulation 결과 및 과정에 대해서 설명하시오. (Truth table 작성 및 K-map 포함)

2-bit binary comparator의 Truth Table은 다음과 같다.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **In A** | **In B** | **In C** | **In D** | **Out F1** | **Out F2** | **Out F3** |
| 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 |

위 진리표를 K-map으로 나타내고 식을 정리하면 다음과 같다.

* F1(A>B)

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

* F2(A=B)

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

* F3(A<B)

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

위의 세 식을 구현하기 위해 다음과 같이 4-input & 3-output Design source를 Verilog code로 작성하였다.

|  |
| --- |
| `timescale 1ns / 1ps  module cmp(      input a, b, c, d,      output lg, eq, le      );      assign lg = (a&~c)+(b&~c&~d)+(a&b&~d);      assign eq = ~((b^d)|(a^c));      assign le = (~a&c)+(~b&c&d)+(~a&~b&d);  endmodule |

이 Design source의 Schematic은 다음과 같다.

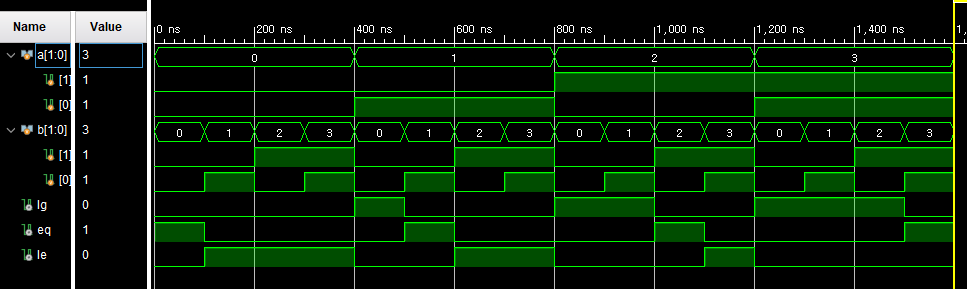
텍스트, 지도이(가) 표시된 사진

자동 생성된 설명

Simulation을 위해 다음과 같이 simulation source를 작성하였다.

|  |
| --- |
| `timescale 1ns / 1ps  module cmp\_sim;      reg [1:0] a, b;      wire lg, eq, le;      cmp cs(a[1], a[0], b[1], b[0], lg, eq, le);      initial a = 2'b00;      initial b = 2'b00;      always a = #400 a+1;      always b = #100 b+1;      initial begin          #1600          $finish;      end  endmodule |

이 simulation의 결과는 다음과 같이 나왔다.



1. 결과 검토 및 논의 사항

세 Verilog Simulation의 결과가 모두 의도했던 진리표 상의 결과와 맞아떨어지는 것으로 보아, 각 함수의 구현은 정확하게 이루어졌음을 확인할 수 있었다.

특히, parity bit generator/checker에서 K-map의 결과 식이 XOR 회로로 귀결되는 것, 즉 변수의 1의 갯수가 짝수/홀수인지 여부를 표시하는 회로가 되는 것을 보였기 때문에, 식을 적절하게 정리했다고 볼 수 있다.

2-bit comparator에서 결과는 정확하게 나왔으나, RTL Schematic이 매우 복잡하게 구성되었기에 식에 추가적인 변형을 가해 최적화해야 할 필요가 있다 생각한다.

1. 추가 이론 조사 및 작성

* 2-bit comparator를 제작할 때, 만약 각 가 공유된 term들을 갖도록 식을 정리하고, 이를 사용해 회로를 구성했다면 더 간단하고 적은 cost가 드는 논리 회로가 완성되었을 수 있을 것이다.
* 또는, 1-bit comparator를 n개 연결한 형태의 cascading design을 사용해 n-bit comparator를 구현할 수도 있을 것이다. 이 경우, n의 값에 따라 각 comparator를 새로 구성할 필요가 없기 때문에 구현이 간편할 것이라 생각한다.