|  |
| --- |
| **P1) CarWarningSystem (자동차 경고 시스템)에 대한 Gate-Level 설계 및 시뮬레이션** |
| * 목적: 자동차 시동이 걸려있을 떄, 내부에서 안전벨트가 착용되지 않을 경우 혹은 문이 제대로 닫히지 않았을 경우를 확인하여 알람을 보내는 회로를 제작 * 수행방법: 회로 설계를 위한 block diagram을 그린 후 dut에 대한 input과 output, wire와 gate를 선언한다. 회로가 제대로 설계되었는지 확인을 위해 tb를 생성하고 input에 여러 값을 대입하며 결과를 확인한다. |
| **설계를 위한 Block Diagram (각 Node의 신호명 포함)** |
| 제일 왼쪽의 세 개의 항목(Door, Ignition, Seatbelt)은 input에 해당한다. 이 변수들의 값에 따라 gate를 거쳐 최종 output이 결정된다. 순서대로 문닫힘, 시동 상태, 안전벨트 착용을 가리킨다. Door는 문이 닫혔을 경우 1, 열렸을 경우 0를 가리킨다. Ignition은 시동이 걸렸을 경우 1, 꺼져있을 경우 0이다. 마지막으로 seatbelt는 제대로 채워졌을 경우 1, 채워지지 않았을 경우 0이다. 이 회로도는 not을 제외한 첫 gate가 and이므로 두 개의 and gate에 모두 들어가는 Ignition이 1일 때 즉, 시동이 걸렸을 때를 기준으로 판단한다. 시동이 걸려있지 않다면 문이 열려있거나 벨트가 채워지지 않았더라도 알람이 울리지 않지만, 시동이 걸려있다면 문이 닫히지 않았거나 안전벨트가 채워지지 않았거나 이 둘 중 하나의 상황에라도 해당된다면 or gate를 통해 1이 출력되고 알람이 울린다. |
| **Main Code (수업에서 제공된 코드 부분은 넣지 말 것)** |
| * DUT  |  | | --- | |  |  * Test Bench  |  | | --- | |  | |
| **Schematic & Waveform or Display** |
| * Schematic : 직접 그린 block diagram과 input의 위치만을 제외하고는 똑같이 나왔다. 이 schematic에서 dut으로 네모박스가 있는 부분이 우리가 설계한 내용이며, input과 output은 사용자가 tb파일에서 직접 지정한 값, 혹은 그로 인한 output값이다. 또한, 각 gate들도 입력에 따라 그 값이 바뀔 수 있다.      * Waveform : 가장 먼저, 예제와 같이 100ns씩 delay를 주었을 때의 waveform이다. 가장 먼저 모든 변수를 0으로 초기한다. SeatBelt와 Door는 not gate를 통하여 DoorOpened, NoSeatBelt wire가 1로 변한다. 하지만 and gate에서는 Ignition의 값이 0이기 때문에 아무런 변화가 없다. 100ns동안 대기한 후 Ignition의 값이 1로 변화한다. 따라서 AND\_U1, AND\_U0 모두 두 개의 input들이 모두 1이므로 OR\_U0은 1,1의 인풋으로 1을 반환한다. 다시 100ns 대기 후 이번엔 Door와 SeatBelt의 값을 모두 1로 바꾼다. Not gate를 거쳐 두 개의 wire는 0으로 변환되고 따라서 두 개의 and gate는 각가 1, 0의 input을 받아 0을 반환한다. 알람이 울리지 않는다. 다시 100ns를 기다린 후 이번엔 SeatBelt의 값을 0으로 바꾼다. 이는 안전벨트가 채워지지 않았다는 신호이다. 따라서 NOT\_U1을 통해 noSeatBelt wire가 1이 되고, AND\_U1 gate도 1이 되어 다시 알람이 울린다. 다시 100ns 대기 후 종료한다.     이번엔 #숫자 -> 대기 시간을 이해하기 위해 약간 수정 후 다시 실행하였다. 첫 번째 대기시간 #100은 그대로 유지된다. 그 후 시동은 걸렸으나 문도 열려 있고 안전벨트도 채워지지 않은 상태가 #200 (200ns) 유지된다. 그리고 문도 닫고 안전벨트도 채워진 상황이 300ns 유지되며 마지막으로 안전벨트만 채워지지 않은 상황을 만들고 100ns를 유지한 후 종료한다. |
| **Comments (실험에 대한 고찰)** |
|  |

|  |
| --- |
| **A1) 2:1MUX Gate Level 설계 및 시뮬레이션** |
| * 목적: 2개의 input과 시그널을 이용하여 1개의 output를 출력하는 2:1MUX를 제작 * 수행방법: 진리표를 참고하여 맵과 회로도(block diagram)을 만든 후 이를 참고하여 dut, tb 파일을 만든다. |
| **Block Diagram (각 Node의 신호명 포함)** |
| 먼저 세개의 input과 한 개의 output이 있는 진리표를 참고하여 맵을 그린다. 인접한 항목들을 묶어 간소화한 식을 도출하고 이를 참고하여 회로도를 그린다.  Input은 2개의 input과 1개의 signal로 이루어져 있다. 따라서 총 세개의 input이 존재하며 signal의 경우 온전한 signal을 가리키는 s input과, not s를 가리키는 sWn wire가 있다. 다른 두 개의 input은 D0과 D1을 각각 d0, d1으로 나타낸다. 이어 d input들은 s input과 and gate를 거쳐 wA0, wA1 wire로 이어진다. 그리고 이 값들을 OR\_U0 gate를 이용하여 결과값 y로 출력한다. |
| **Main Code** |
| * DUT  |  | | --- | | module A1\_DUT #(  )( // 필요한 input과 output 선언      output  y   ,      input   s   ,      input   d1  ,      input   d0  );    // 필요한 wire 선언 (gate를 거친 값)      wire    wSn;    // s의 인버터 적용 후      wire    wA0;      wire    wA1;    // 필요한 gate 선언      not     NOT\_U0(wSn, s);      and     AND\_U0(wA0, d0, wSn);      and     AND\_U1(wA1, s, d1);      or      OR\_U0(y, wA0, wA1);  endmodule |  * Test Bench  |  | | --- | | `timescale 1ns/1ps  // A1\_DUT를 활용하여 test해 보기  module A1\_TB;    // 필요한 reg(input)와 wire(output) 선언      reg     s;      reg     d0;      reg     d1;      wire    y;  // A1\_DUT를 호출하여 test bench 진행  A1\_DUT           dut(      .y      (y  ),      .s      (s  ),      .d1     (d1  ),      .d0     (d0  ));      initial begin          // initial, 아래로 진리표 순서대로 적용하기          s   = 1'b0;          d1  = 1'b0;          d0  = 1'b0;          #100          s   = 1'b0;          d1  = 1'b0;          d0  = 1'b1;          #100          s   = 1'b0;          d1  = 1'b1;          d0  = 1'b0;          #100          s   = 1'b0;          d1  = 1'b1;          d0  = 1'b1;          #100          s   = 1'b1;          d1  = 1'b0;          d0  = 1'b0;          #100          s   = 1'b1;          d1  = 1'b0;          d0  = 1'b1;          #100          s   = 1'b1;          d1  = 1'b1;          d0  = 1'b0;          #100          s   = 1'b1;          d1  = 1'b1;          d0  = 1'b1;          #100          $finish;      end  endmodule | |
| **Waveform or Display** |
| * Waveform |
| **Comments** |
|  |

|  |
| --- |
| **A3) 2:1MUX Gate Level 설계 및 시뮬레이션** |
| * 목적: 4개의 input과 2개의 시그널을 이용하여 1개의 output를 출력하는 4:1MUX를 제작 * 수행방법: 회로도를 참고하여 간소화된 식을 만들고 이를 참고하여 dut, tb 파일을 만든다 |
| **Block Diagram (각 Node의 신호명 포함)** |
|  |
| **Main Code** |
| * DUT  |  | | --- | | module A2\_DUT #(  )(      output  y   ,      input   s1  ,      input   s0  ,      input   d0  ,      input   d1  ,      input   d2  ,      input   d3  );      wire    wS1n;    // s1의 인버터 적용 후      wire    wS0n;    // s0의 인버터 적용 후      wire    wA0;      wire    wA1;      wire    wA2;      wire    wA3;      not     NOT\_U0(wS1n, s1);      not     NOT\_U1(wS0n, s0);      and     AND\_U0(wA0, d0, wS1n, wS0n);      and     AND\_U1(wA1, d1, wS1n, s0);      and     AND\_U2(wA2, d2, s1, wS0n);      and     AND\_U3(wA3, d3, s1, s0);      or      OR\_U0(y, wA0, wA1, wA2, wA3);  endmodule |  * Test Bench  |  | | --- | | `timescale 1ns/1ps  module A2\_TB;      reg     s1;      reg     s0;      reg     d0;      reg     d1;      reg     d2;      reg     d3;      wire    y;  A2\_DUT          dut(      .y      (y  ),      .s1     (s1 ),      .s0     (s0 ),      .d0     (d0 ),      .d1     (d1 ),      .d2     (d2 ),      .d3     (d3 ));      initial begin          // initial          s1  = 1'b0;          s0  = 1'b0;          d0  = 1'b0;          d1  = 1'b0;          d2  = 1'b0;          d3  = 1'b0;            // D0 wrong          #100          s1  = 1'b1;          s0  = 1'b0;          d0  = 1'b1;          d1  = 1'b0;          d2  = 1'b0;          d3  = 1'b0;            // D0 wrong          #100          s1  = 1'b0;          s0  = 1'b1;          d0  = 1'b1;          d1  = 1'b0;          d2  = 1'b0;          d3  = 1'b0;            // D0 wrong          #100          s1  = 1'b1;          s0  = 1'b1;          d0  = 1'b1;          d1  = 1'b0;          d2  = 1'b0;          d3  = 1'b0;            // D0 right          #100          s1  = 1'b0;          s0  = 1'b0;          d0  = 1'b1;          d1  = 1'b0;          d2  = 1'b0;          d3  = 1'b0;            // D0 don't care          #100          s1  = 1'b0;          s0  = 1'b0;          d0  = 1'b1;          d1  = 1'b1;          d2  = 1'b0;          d3  = 1'b0;          // D0 don't care          #100          s1  = 1'b0;          s0  = 1'b0;          d0  = 1'b1;          d1  = 1'b1;          d2  = 1'b0;          d3  = 1'b1;            // D0 wrong          #100          s1  = 1'b1;          s0  = 1'b0;          d0  = 1'b1;          d1  = 1'b0;          d2  = 1'b0;          d3  = 1'b0;            // D1 right          #100          s1  = 1'b0;          s0  = 1'b1;          d0  = 1'b0;          d1  = 1'b1;          d2  = 1'b0;          d3  = 1'b0;            // D1 wrong          #100          s1  = 1'b1;          s0  = 1'b1;          d0  = 1'b0;          d1  = 1'b1;          d2  = 1'b0;          d3  = 1'b0;            // D1 wrong          #100          s1  = 1'b0;          s0  = 1'b0;          d0  = 1'b0;          d1  = 1'b1;          d2  = 1'b0;          d3  = 1'b0;            // D1 wrong          #100          s1  = 1'b1;          s0  = 1'b0;          d0  = 1'b0;          d1  = 1'b1;          d2  = 1'b0;          d3  = 1'b0;            // D2 right          #100          s1  = 1'b1;          s0  = 1'b0;          d0  = 1'b0;          d1  = 1'b0;          d2  = 1'b1;          d3  = 1'b0;            // D2 wrong          #100          s1  = 1'b0;          s0  = 1'b0;          d0  = 1'b0;          d1  = 1'b0;          d2  = 1'b1;          d3  = 1'b0;            // D2 wrong          #100          s1  = 1'b1;          s0  = 1'b1;          d0  = 1'b0;          d1  = 1'b0;          d2  = 1'b1;          d3  = 1'b0;            // D2 wrong          #100          s1  = 1'b0;          s0  = 1'b1;          d0  = 1'b0;          d1  = 1'b0;          d2  = 1'b1;          d3  = 1'b0;            // D3 right          #100          s1  = 1'b1;          s0  = 1'b1;          d0  = 1'b0;          d1  = 1'b0;          d2  = 1'b0;          d3  = 1'b1;            // D3 wrong          #100          s1  = 1'b1;          s0  = 1'b0;          d0  = 1'b0;          d1  = 1'b0;          d2  = 1'b0;          d3  = 1'b1;            // D3 wrong          #100          s1  = 1'b0;          s0  = 1'b1;          d0  = 1'b0;          d1  = 1'b0;          d2  = 1'b0;          d3  = 1'b1;            // D3 wrong          #100          s1  = 1'b0;          s0  = 1'b0;          d0  = 1'b0;          d1  = 1'b0;          d2  = 1'b0;          d3  = 1'b1;          #100          $finish;      end  endmodule | |
| **Waveform or Display** |
| * Waveform |
| **Comments** |
|  |

|  |
| --- |
| **A3) Visual Studio Code 관련 학습 내용 정리** |
| * 목적: verilog를 사용하는 과정에서 소스 코드 제작 및 편집의 편리함을 위해 코드 작성은 visual studio code ide를 활용한다. 이를 더 효율적으로 사용하기 위해 단축키나 유용한 사용법을 알아보고 정리한다. * 수행방법: 구글링, 교재 찾아보기 |
| **Block Diagram (각 Node의 신호명 포함)** |
|  |
| **Main Code** |
| * DUT  |  | | --- | | **[Code Description]** |  * Test Bench  |  | | --- | | **[Code Description]** | |
| **Waveform or Display** |
| * Waveform |
| **Comments** |
|  |