2 장

부울 법칙과 드모르간 정리 및 로직 해저드

1. 실험 목표

▣ 부울 법칙에 대해 이해한다.

▣ 드모르간의 정리에 대해 이해한다.

▣ 글리치와 해저드의 개념을 이해하고, 해저드를 제거하는 방법을 알아본다.

▣ 부울 법칙과 드모르간의 정리를 Verilog 를 통해 시뮬레이션 해본다.

2. 실험 이론

가) 부울 법칙과 드모르간 정리

디지털 회로 설계의 목표는 최적의 성능과 최소의 비용을 갖는 회로를 구성하는 것으로, 부울 법칙은 주어진 로직 함수를 최소 항목으로 구성되는 함수로 변환시 적용되는 기본적인

법칙이며, 최적으로 간소화된 함수 여부를 판단하기 위해서 모든 가능한 경우를 조사한 후 최적의 결과를 얻을 수 있다. 따라서 최적으로 간소화된 함수 결과를 구할 수 있는 시스템적인 방법이 요구된다. 이러한 최적으로 간소화된 로직 회로를 구하기 위해서는 먼저 진리표를 구성하고, 각 출력에 대한 간소화된 부울 방정식을 구하여야 한다. 이러한 간소화된 부울 방정식을 만들기 위한 가장 기본적인 방법으로 부울 대수와 드모르간의 정리를 이용하는 방법이 있다. 다음은 기본적인 부울 대수 법칙을 나타내며, 이에 대한 Verolog 코드는 [그림 2-1]과 같다.

1) 부울 대수 법칙

① 교환법칙

ㄱ) A+B = B+A

ㄴ) AB = BA (A•B = B•A)

② 결합법칙

ㄱ) A+(B+C) = (A+B)+C

ㄴ) A(BC) = (AB)C

③ 분배법칙

ㄱ) A(B+C) = AB+AC

ㄴ) (A+B)(C+D) = AC+AD+BC+BD

④ 상수 '1' 또는 '0'과의 연산

ㄱ) A. 0 = 0, A+0 = A

ㄴ) A. 1 = A, A+1 = 1

⑤ 부울 변수 및 보수연산(A´= not A)

ㄱ) A. A = 0

ㄴ) A+A = A

ㄷ) A. A´ = 0

ㄹ) A+A´ = 1

⑥ 논리적 등식

ㄱ) A´´=A

ㄴ) A+A´B=A+B

ㄷ) A´+AB=A´+B

2) 드모르간의 정리

① $f'(X_1,X_2,...,X_n,0,1,+,\bullet) = f(X_1',X_2',...,X_n',1,0,\bullet,+)$: 변수 Xi 는 Xi′로 변환하고, 0 은 1 로, 1 은 0 으로, + 로직은 • 로직으로, • 로직은 + 로직으로 기계적인 변환을 수행한다.

② NOR EX: (A + B)′ = A′B′

③ NAND EX: (AB)′ = A′ + B′

```
module BOOL_LAW(A, B, C, COMM_R1, COMM_R2, ASSO_R1, ASSO_R2, DIST_R1, DIST_R2);
     input   A, B, C;
     output COMM_R1, COMM_R2, ASSO_R1, ASSO_R2, DIST_R1, DIST_R2;

     // 교환법칙
     assign COMM_R1 = A | B;
     assign COMM_R2 = B | A;

     // 결합법칙
     assign ASSO_R1 = A | (B | C);
     assign ASSO_R2 = (A | B) | C;

     // 분배법칙
     assign DIST_R1 = A & (B | C);
     assign DIST_R2 = (A & B) | (A & C);

endmodule
```

그림 2-1. 부울 대수 Verilog 코드.

그러나 주어진 문제에 대한 진리표를 구성하면 각 출력 방정식은 서로 다른 많은 결과가 존재한다. 따라서 주어진 진리표에 대하여 항상 같은 결과 방정식으로 표현할 수 있는 표준적인 표현 기법이 요구되며, 이러한 표기법을 표준형식 (Canonical form)이라 정의한다. 앞에서 언급하였듯이 진리표로부터 표준형식으로 각 출력 방정식을 구한 후, 최적화를 위하여 부울 법칙과 드모르간 정리를 적용하는 것 만으로는 간소화된 최적 논리 회로 설계를 구하기에 많은 어려움이 있다. 다소 복잡한 문제에 대해서 표준형식의 출력 방정식을 간소화 하는 과정에서 실수가 발생 할 수 있으며, 간소화 후 검증하기도 힘들다는 단점이 존재한다.

그러므로 부울 법칙과 드모르간 정리를 설계자 입장에서 좀 더 시각적이고 직관적인 형태로 활용하기 위하여 최적으로 간소화된 출력 방정식을 구할 수 있는 시스템적 방법으로 Karnaugh Map (K-map)을 사용하거나, Quine-McCluskey Method 를 적용한다. 이러한 방법은 근본적으로 진리표의 출력에 대해 참값을 제공하는 인접한 두 개의 입력변수 조합에 대하여

오직 한 개의 변수 값만 변화하는 경우 이를 간소화할 수 있는 기본적인 융합법칙 (A (B′ + B) = A)에 의거하여 간소화를 수행한다.

다음의 예시를 통해, 표준형식으로 구현한 경우, 부울 법칙 및 드모르간 정리를 이용해 간편화 한 경우와 K-map 을 이용해 최적화 하는 과정을 비교해본다.

3) 2-input multiplexer 예

2-input multiplexer 는 선택제어 신호의 값에 따라 A, B 입력 중 하나를 택하여 출력하는 로직이다. 마치 2-bit 메모리에서 1-bit 의 주소(address)를 통해 읽고자 하는 값을 선택하여 출력하는 경우와 비슷하다고 볼 수 있겠다.

흔히 불 대수식은 곱의 합(SOP: sum of product), 합의 곱(POS: product of sum)으로 표현하며, 다음은 2-input multiplexor 를 진리표와 최소항으로 나타낸 논리식 및 회로도는 아래 [그림 2-2]와 같다.

A	B	SEL	OUT
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

OUT = A′·B·SEL + A·B′·SEL′ + A·B·SEL′ + A·B·SEL

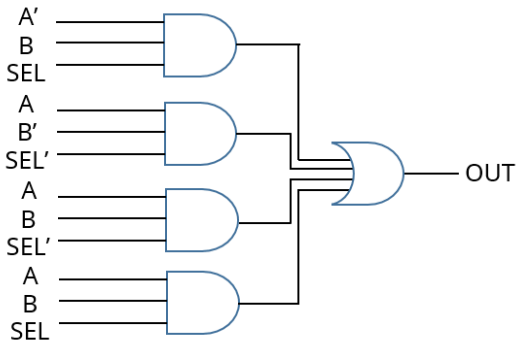
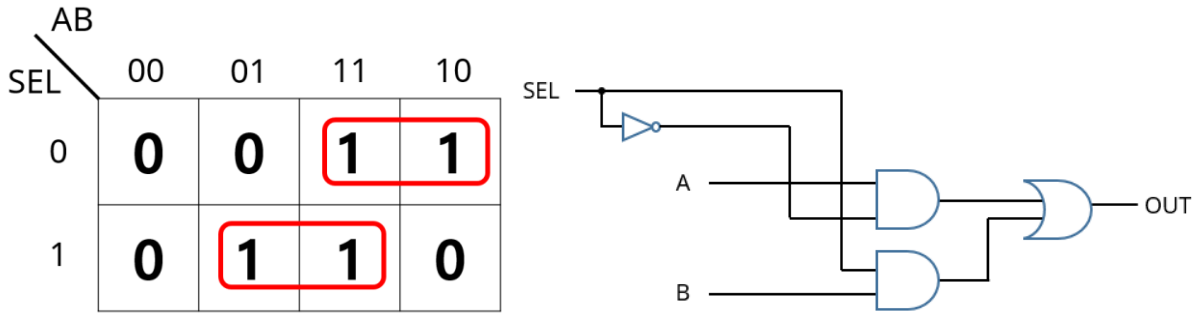


그림 2-2. 2-input Multiplexor 의 진리표와 부울방정식, 로직 다이아그램.

위의 논리식 및 회로도를 간소화 하기 위해서는 불 대수 및 드모르간의 법칙을 활용해야 하며, 위와 같이 간단한 logic 의 경우에서는 비교적 수월하게 진행할 수 있다. 다만, 4-input multiplexor 와 같이 64 개의 경우의 수가 존재하는 경우에는 그 작업이 매우 힘들게 된다. 따라서 일반적으로 5~6 변수 이하의 논리식을 간소화/최적화 하는 경우에는 아래와 같이 K-map 을 이용해 직관적인 방법으로 간소화를 진행한다.

다음의 [그림 2-3]은 K-map 을 이용해 간소화한 회로의 다이어그램 및 Verilog 코드이다.




```
module mux2( A, B, SEL, OUT );
      input       A, B;
      input       SEL;
      wire        sel_a, sel_b, not_SEL;
      output      OUT;

      not   not0( not_SEL, SEL );

      or    or0( OUT, sel_a, sel_b );

      and and0( sel_a, A, not_SEL);
      and and1( sel_b, B, SEL);

endmodule
```

그림 2-3. 2-MUX 의 Verilog HDL 코드와 회로도.

나) 로직 해저드

조합회로에서는 일반적으로 글리치 (glitch)라는 원치 않은 신호들이 발생한다. 이것은 다른 경로를 지나온 신호들이 각기 다른 전달지연을 가지기 때문에 발생한다. 만일 글리치가 발생할 가능성을 가진 회로가 있다면 해저드 (Hazard)를 보유하는 회로라고 말할 수 있다.

해저드에는 정적 해저드와 동적 해저드가 있다. 정적 해저드는 특정 값이 입력된 경우 출력 값의 변화가 없어야 하는데, 한 번 변하는 경우를 정의하며, 동적 해저드란 특정 값이 입력된 경우 출력 값이 한번만 변해야 하는데, 두 번 이상 변하는 경우를 말한다.

이러한 해저드는 디지털회로에서는 다음 두 가지 문제를 가지고 있다. 첫 번째는 이전 조합회로의 출력이 아직 안정화 값을 갖기 전에 다음 회로에서 사용하는 경우다. 이 경우는 회로의 출력이 안정화될 때까지 충분히 기다린 후 출력 값을 이용할 수 있도록 해주어야 한다. 두 번째는 비동기 입력에 연결된 경우이다. 이 경우 기존 클럭에 동기되어 동작하지 않고 발생하는 글리치에 바로 영향을 받는다. 이 문제는 초기화 (RESET)을 제외한 모든 신호를

비동기 입력에 연결하지 않으면 해결된다. 다른 해결방안으로는 해저드가 생기지 않도록 회로를 설계하는 방법이 있다.

정적 해저드는 2 단계 회로에서 발생하는 것으로 입력이 변화하면 출력에 일시적으로 글리치가 발생하고 다시 원래 값으로 돌아가는 특징을 가진다. [그림 2-4]은 정적 해저드를 보여준다. 정적 해저드가 없는 회로 출력은 '1'또는 '0'의 값을 일정하게 유지해야 하는데, [그림 2-4(a)]는 입력 값이 변할 때 출력이 '1'에서 '0'이 되었다가 다시 '1'이 되는 경우고, (b)는 반대의 경우에 해당된다.

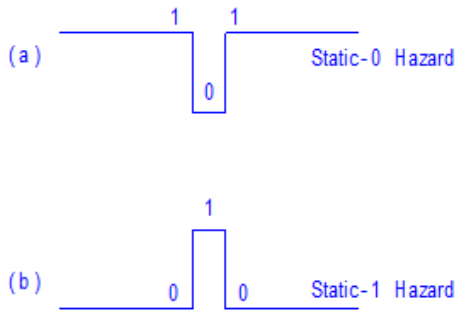


그림 2-4. 정적 해저드의 예.

[그림 2.5]는 해저드를 가지고 있는 회로와 K-map 을 보여준다.

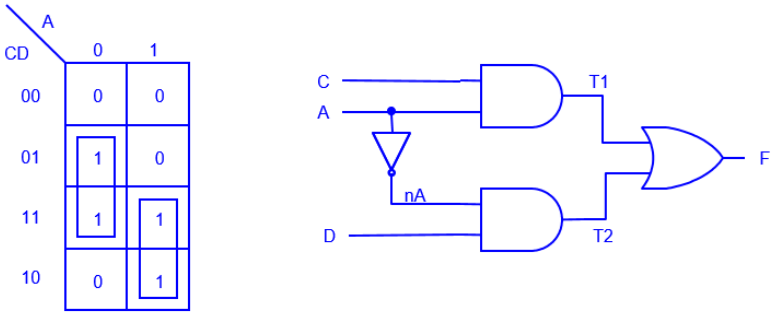


그림 2-5. 해저드를 가지고 있는 회로.

[그림 2-5]의 회로는 2 개의 AND 게이트 입력인 A 를 NOT 게이트를 통과한 nA (NOT A)에 의한 시간 지연차 때문에 해저드가 발생하는 예이다. [그림 2-6]는 해저드가 발생하는 과정을 입력의 변화에 따라 보여준다. [그림 2-6(a)]는 회로의 특정 입력 값에 대한 초기 상태이고, (b)는 A 의 값이 '0'으로 바뀐 상태이다. (b)에서는 NOT 게이트 지연 때문에 nA 가 아직 '0'의 값을 갖고 있어서 출력 F 가 일시적으로 '0'이 된다. (c)는 NOT 게이트 지연이 충분히 지나가서 출력 F 가 안정화된 것을 보여준다.

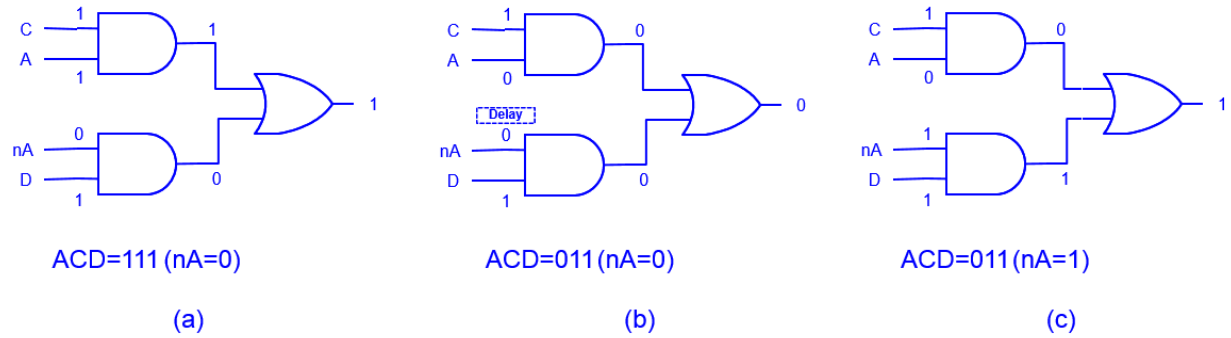


그림 2-6. 해저드 진행 과정.

[그림 2-7]은 위에서 제시한 해저드 발생회로에 대한 Verilog 코드를 보여준다.

```
module HAZARD( A, C, D, F);
      input       A, C, D;
      wire        nA, T1, T2;
      output      F;

      assign nA = ~A;
      assign T1 = A & C;
      assign T2 = nA & D;
      assign F  = T1 | T2;

endmodule
```

그림 2-7. 해저드 발생 실험을 위한 HDL 코드.

[그림 2.7]은 [그림 2.5]의 회로를 Verilog 로 기술한 것이다. Verilog 에서 보통 assign 문에 @n 식의 방식으로 지연을 주나, 본 실험 환경에서는 그 구성 방식이 상이하므로 실험 여건에 따라 조교의 지시대로 수행하기를 바란다. [그림 2-8]은 [그림 2-7]의 회로의 파형 신호를 보여준다. 이 회로에서는 입력 A 의 값이 '1'에서 '0'이 될 때 글리치가 발생한다.

[그림 2-9]는 해저드를 제거하기 위한 K-map, 회로, Verilog 코드를 보여준다. K-map 에서 점선 부분을 회로에 추가하면 글리치가 발생하지 않는다.

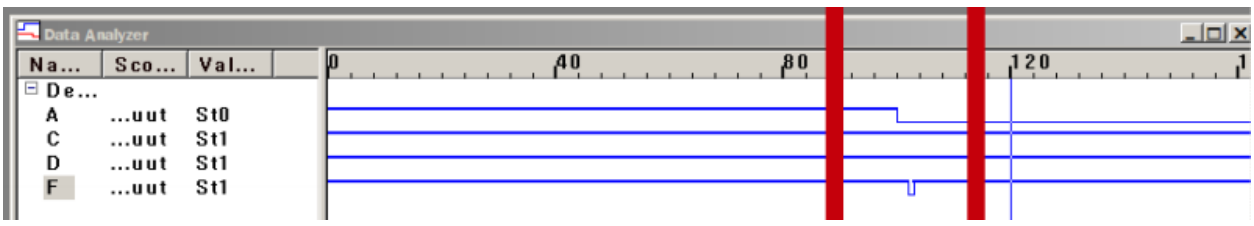


그림 2-8. 글리치 발생 파형의 예.

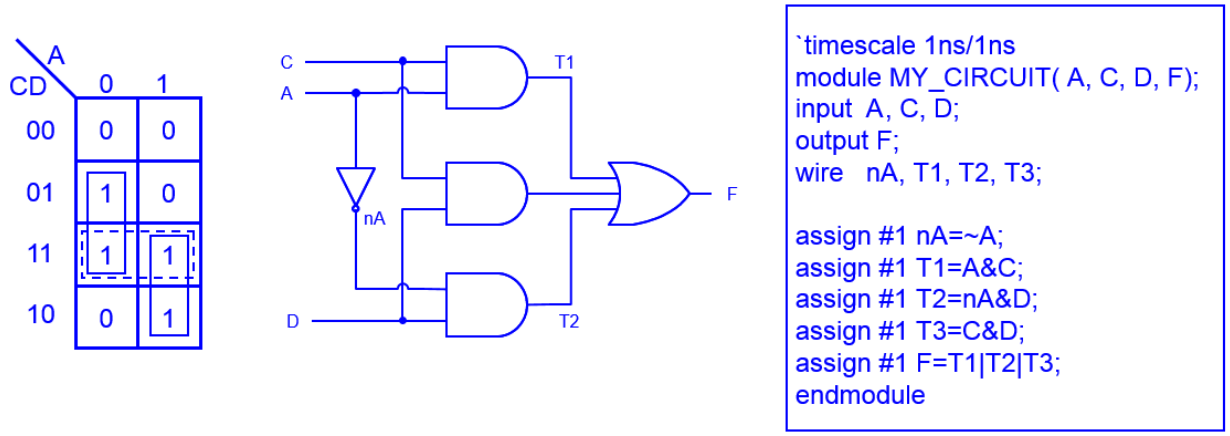


그림 2-9. 해저드를 제거한 회로.

3. 예비보고서 작성

▣ [그림 2.1]의 Verilog 코드와 같은 형식으로 드모르간의 정리를 Verilog 코드로 작성하시오.

▣ 동적 해저드의 원인과 해결 방안을 조사하고, 동적 해저드를 발생시키는 회로를 제시하시오.

▣ 결과 보고서 3 번째 항목에서 제시한 회로를 설계하는 방안에 대하여 미리 생각해보시오. (세부적인 동작 사항에 대해서 서술할 필요는 없으며, 어떻게 설계해야 목표로 하는 동작을 수행할 수 있을지에 대해서만 간단히 고려해보시오.)

4. 실험에 필요한 기기

▣ 실험용 PC

▣ MAX PLUS II 소프트웨어

5. 실험 내용과 과정

① 실험의 기본 내용을 이해하고자 하는 경우 그림 2-1, 2-3 에 제시된 Verilog 코드를 작성하여 검증해보거나, 개념에 대해 충분한 이해를 갖고 있을 경우 [6. 실험 결과 보고서]에서 진행할 ①, ②, ③ 실험의 논리 회로를 구상한다.

② MAX PLUS II 를 실행하고 각 모듈들에 해당되는 Verilog 코드 파일들 및 Project 들을 생성한다.

③ MAX PLUS II 의 Text Editor 를 통해 Verilog 코드를 작성한다.

④ MAX PLUS II 상단의 메뉴에서 Compiler 를 선택하여 Verilog 코드를 컴파일 한다.

⑤ MAX PLUS II 상단의 메뉴에서 Waveform Editor 를 선택하여, 입출력 Node 를 등록한 뒤 Timeline 에 맞추어 입력 값을 정한다.

⑥ MAX PLUS II 상단의 메뉴에서 Simulator 를 선택하여 수행한 후, 해당 Verilog 코드가 Waveform Editor 에서 입력한 신호에 맞추어 올바른 결과값을 보이는지 확인한다.

6. 실험 결과 보고서

① 예비 보고서에서 작성한, 드모르간의 정리를 실험하기 위한 Verilog 코드의 동작을 검증하시오.

② [그림 2.7]과 [그림 2.9] 회로에서 제시된 해저드가 존재하는/존재하지 않는 회로를 직접 검증해보시오. 그리고 해저드가 제거된다면, 그것이 가능한 이유에 대해서 고찰해보시오.

③ 아래 그림과 같이 A, B, C 각 1bit 의 입력이 주어질 시, 각 A', B', C' 를 출력하는 회로를 구현하고, 동작을 검증하시오. (단, 지난 실험에서의 조건과 달리, XOR Gate 는 사용할 수 없음. 오로지 NOT gate 2 개와 무수히 많은 AND, OR gate 만을 이용해서 구현해야 함)