REPORT

보고서 작성 서약서

1. 나는 타학생의 보고서를 복사(Copy)하지 않았습니다.
2. 나는 타학생의 보고서를 인터넷에서 다운로드 하여 대체하지 않았습니다.
3. 나는 타인에게 보고서 제출 전에 보고서를 보여주지 않았습니다.
4. 보고서 제출 기한을 준수하였습니다.

나는 보고서 작성시 위법 행위를 하지 않고,
성.균.인으로서 나의 명예를 지킬 것을 약속합니다.

과 목 : 논리 회로 설계실험
과 제 명 : Design 4:2 Priority Encoder
담당교수 : 오 윤 호 교수님
학 과 : 전자 전기 공학부
학 년 : 3 학 년
학 번 : 2016311290
이 름 : 이 지 학
제 출 일 : 2021.04.03

4:2 Priority Encoder 설계

*이지학

*성균관대학교 전자전기공학부 dlwlgkr1@g.skku.edu

Design 4:2 Priority Encoder

*Ji-Hak Lee

*Dept. pf Electronic & Electrical Engineering, Sungkyunkwan University

목 차

1. 요 약 ********** 2p
2. 이론적 접근 ********** 2p
3. Verilog 구현 ********* 4p
4. 실험 결과 (고찰)******. 6p
5. 참 고 ********** 6p

*보고서 표지 포함 6 Page (표지 제외 5 Page)

1. 요 약 [Brief]

4:2 Priority Encoder 설계에 앞서 Encoder와 Decoder의 차이점을 명확하게 이해하고 설계를 진행한다. Encoder의 특수한 Case인 Priority Encoder는 어떤 H/W적 오류를 해결하기 위하여 착안된 Concept인지 확인하고 이를 이론적으로 점검한다.

세 가지 Modeling 기법인 Behavioral Modeling, Data Flow modeling, Gate-Level modeling을 각 순서에 맞게 구현하고 최종적으로 코드를 통해 구현한다. 실제 코드 구현에 앞서 설계 대상의 TRUTH TABLE과 KARNAUGH MAP을 통해 설계할 모듈이 어떻게 동작할지 예측하고 이를 실험적으로 규명한다.

각 모듈의 테스트를 위한 test-bench 코드를 작성할 때 주어진 모듈의 test point를 명확히 확인하고 주어진 test pattern이 설계한 모듈을 합리적으로 점검할 수 있는지 예측한다. 마지막으로 주어진 모듈과 test bench를 결합하여 시뮬레이션 함으로써 각 모듈이 적합하게 디자인되었고 그 결과가 앞서 TRUTH TABLE과 KARNAUGH MAP에서 예측한 결과와 맞는지 점검한다.

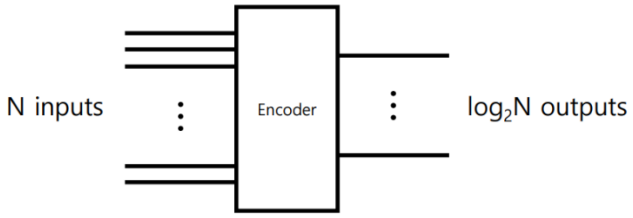
2. 이론적 접근

[Theoretical approach]

1) ENCODER

=> 다수의 입력 단자(N)에서 정해진 입력 값을 받아 출력(Log2N) 하는 논리 회로이다. ENCODER의 경우 N개의 입력 중 하나가 TRUE(1)가 되면 그 Input의 조합과 대응되는 위치의 Output 값을 출력한다. 밑의 Karnaugh map은 N=2인 경우의 예시다.

앞서 설명한 ENCODER의 동작은 이론적으로 제한된 INPUT에 대한 동작 구현이며(N개의 Input 중 1개의 Input 값만 'TRUE'), 실제 이 모듈을 H/W 적으로 설계하고 적용하였을 때에는 다른 상황이 발생할 수 있다. (N개의 Input 중 다수의 'TRUE' 발생) 이와 같은 상황을 해결하기 위한 것이 바로 이번 실험에서 설계할 Priority Encoder이다.



* N=2: Input port 4개/Output port 2개인 경우

out0

dc \ ba	00	01	11	10
00	0	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	0	0	0

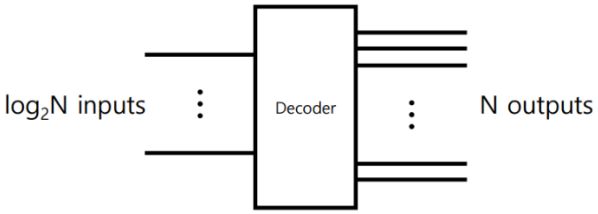
out1

dc \ ba	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	0	0	0	0
10	1	0	0	0

* 위의 예시에서는 undefined 된 값을 0으로 표기하였다.

2) DECODER

=> 소수의 입력 단자(Log2N)에서 받은 입력 값을 받아 출력(N) 하는 논리 회로이다. DECODER의 경우 지정된 입력 값에 대응하는 출력 값이 도출되며, 일반적으로 출력부의 output port 중 하나의 port가 지정되어 TRUE(1)가 된다. 밑의 Karnaugh map은 N=2인 경우의 예시다.



* N=2: Input port 2개/Output port 4개인 경우

out0

b \ a	0	1
0	1	0
1	0	0

out1

b \ a	0	1
0	0	1
1	0	0

out2

b \ a	0	1
0	0	0
1	1	0

out3

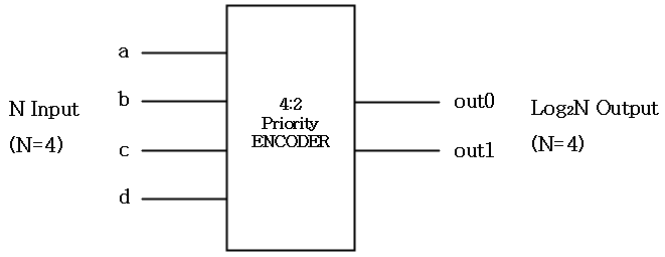
b \ a	0	1
0	0	0
1	0	1

3) 4:2 PRIORITY ENCODER

=> 4:2 Priority ENCODER는 일반적인 ENCODER와 비슷한 논리로 동작하지만, INPUT port에 여러 개의 값이 들어오더라도 정해진 Rule에 의해 출력 값을 정할 수 있다는 특징이 있다. 앞서 설명한 일반적인 ENCODER의 경우, 입력 port가 하나만 TRUE가 되어야 한다는 제한 조건이 있지만 Priority rule을 적용한 DECODER는 이를 완벽하게 보완할 수 있다. 따라서 실제 H/W 제작 및 프로젝트 구현 과정에서 발생할 수 있는 예기치 못한 Input에 대하여 오류 없이 정해진 동작을 수행할 수 있다.

* 설계 조건:

- Boolean expression must consist of AND, OR, NOT only.
- Other operators are not allowed and using parentheses is allowed.
- K maps must be correct



(1) Verbal Description (INPUT: a, b, c, d/ OUTPUT: out0, out1)

out0 = 0 and out1 = 0, when a = 1, b = 0, c = 0 and d = 0.

out0 = 1 and out1 = 0, when a = x, b = 1, c = 0 and d = 0.

out0 = 0 and out1 = 1, when a = x, b = x, c = 1 and d = 0.

out0 = 1 and out1 = 1, when a = x, b = x, c = x and d = 1.

* x는 don't care term을 의미합니다.

* priority rule:

a=0(b=0, c=0, d=0): out0=out1=0을 출력한다.

b=1(c=0, d=0): a의 값에 상관없이 out0=1, out1=0을 출력한다.

c=1(d=0): a, b의 값에 상관없이 out0=0, out1=1을 출력한다.

d=1: a, b, c의 값에 상관없이 out0=1, out1=1을 출력한다.

(2) Truth Table

[TRUTH TABLE]

d	c	b	a	out1	out0
0	0	0	0	x	x
0	0	0	1	0	0
0	0	1	x	0	1
0	1	x	x	1	0
1	x	x	x	1	1

(3) Karnaugh map

[out0]

d c \ b a	0 0	0 1	1 1	1 0
0 0	x	0	1	1
0 1	0	0	0	0
1 1	1	1	1	1
1 0	1	1	1	1

d bc'

[out1]

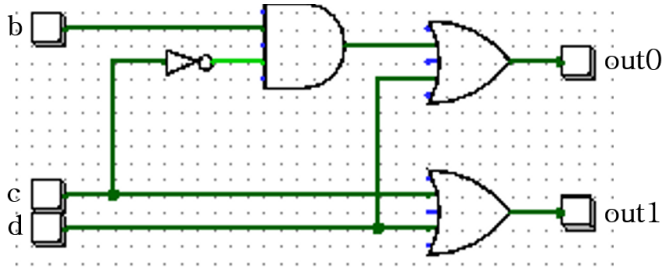
d c \ b a	0 0	0 1	1 1	1 0
0 0	x	0	0	0
0 1	1	1	1	1
1 1	1	1	1	1
1 0	1	1	1	1

d c

(4) Boolean Expression

out0 : b · c̄ + d

out1 : c + d



3. Verilog 구현

[Verilog Implementations]

1) Behavioral Modeling

* 설계 조건:

A hybrid implementation is not allowed.

* 설계 환경:

Model Sim-Intel FPGA Starter Edition/Verilog HDL

* 실험 세부:

(1) 4:2 Priority ENCODER의 Truth Table과 Karnaugh Map을 참고하여 어떤 조건에 어떤 동작을 수행하는지 확인한다.
(2) 해당 동작 조건에 적합한 Boolean expression을 표현하고, 이를 적합한 BERILOG operator(logical operators)를 사용하여 구현한다.

* 동작 조건(Verbal description)

```
case1, if input_b_4bit = 0001 => out_2bit = 00
case2, if input_b_4bit = 001x => out_2bit = 01
case3, if input_b_4bit = 01xx => out_2bit = 10
case4, if input_b_4bit = 1xxx => out_2bit = 11
```

* Boolean Expression

out0 : b · c̄ + d

out1 : c + d

* 코드 구현

```
module four_to_two_priority_encoder_behavioral_module (input_b_4bit, out0, out1);

        input [3:0] input_b_4bit; // input [3 2 1 0]
        output out0, out1; // output of this module.
        reg out0, out1; // Non blocking
        reg [1:0] out_2bit; // out [1 0] this line needs to use casex syntax
        assign out0 = out_2bit[0]; // LSB of output
        assign out1 = out_2bit[1]; // MSB of output

        always@(input_b_4bit)
        begin
                casex(input_b_4bit)
                4'b0001:out_2bit=2'b00; // case1, if input_b_4bit = 0001 => out_2bit = 00
                4'b001x:out_2bit=2'b01; // case2, if input_b_4bit = 001x => out_2bit = 01
                4'b01xx:out_2bit=2'b10; // case3, if input_b_4bit = 01xx => out_2bit = 10
                4'b1xxx:out_2bit=2'b11; // case4, if input_b_4bit = 1xxx => out_2bit = 11
                default:out_2bit=2'b00;
                endcase

        end
        //Fill this out.

endmodule
```

=> 'casex' 문을 사용하여 behavioral modeling을 진행하였다. 모듈의 output은 out0과 out1 두개의 단일 비트로 지정하였으며 'casex'문에서 두개의 단일 output을 2bit의 배열로 합성해 사용하기 위하여 2bit 크기의 새로운 wire, 'out_2_bit'를 reg로 지정하여 사용하였다.

2) Gate Level Modeling

* 설계 조건:

Use AND, OR, NOT gate only. and use the source codes of AND, OR, NOT gates, which are provided by us.

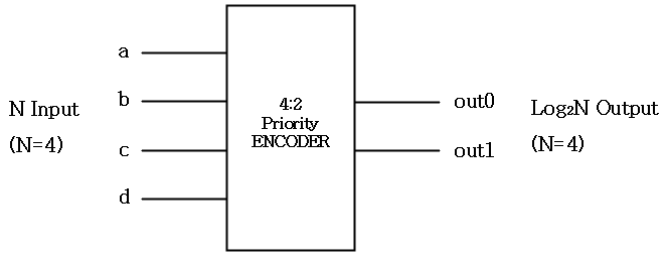
* 설계 환경:

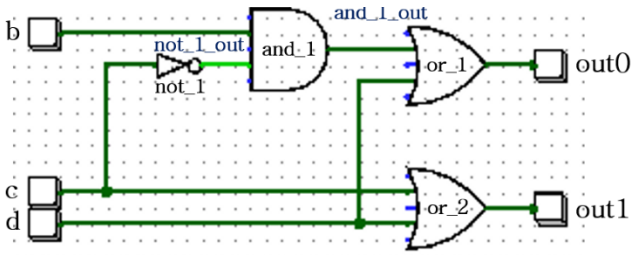
Model Sim-Intel FPGA Starter Edition/Verilog HDL

* 실험 세부:

(1) 4:2 Priority ENCODER의 Truth Table과 Karnaugh Map을 작성하고, 적합한 Boolean Expression을 도입한다.
(2) Boolean Expression을 기본 Gate로 표현해본다.
(3) Module화 된 각 Gate를 연결할 wire를 선언하고, Input과 Output을 고려하여 연결한다.

* Disassemble





* Verilog code 구현

```
module four_to_two_priority_encoder_gatelevel_module (a, b, c, d, out0, out1);
        input a, b, c, d;

        output out0, out1;
        wire not_1_out, and_1_out; // we have to conect two gates with these wires.

        //Fill this out
        not_gate not_1(.a(c), .out(not_1_out)); // c => c'
        and_gate and_1(.a(b), .b(not_1_out), .out(and_1_out)); // c' && b => and_1_ou
        or_gate or_1(.a(d), .b(and_1_out), .out(out0)); // d + (c' && b) => out0
        or_gate or_2(.a(c), .b(d), .out(out1)); // c + d => out1

endmodule
```

- Gate와 Gate 사이를 연결할 Wire를 정의하고 각 Gate 사이의 input과 output 관계를 유념하며 wire로 연결한다.

- 이전 실험에서 사용하였던 OR Gate의 경우 허용 INPUT 수가 2개 이므로, H/W spec을 고려하여 설계를 진행하였다.

3) Data Flow Modeling

* 설계 조건:

A hybrid implementation is not allowed.

* 설계 환경:

Model Sim-Intel FPGA Starter Edition/Verilog HDL

* 실험 세부:

(1) 4:2 Priority ENCODER의 Truth Table과 Karnaugh Map을 작성하고, 설계에 적합한 Boolean Expression을 도입한다.

(2) Boolean Expression을 참고하여 코드의 기본 Logic을 설계한다.

```
module four_to_two_priority_encoder_dataflow_module (a, b, c, d, out0, out1);
        input a, b, c, d;

        output out0, out1;

        //Fill this out.
        assign out0 = (b && !c) || d; // Boolean Expression : out0 = bc' + d
        assign out1 = c || d; // Boolean Expression : out1 = c + d
        //logical operators.

////////01/////////////////////////////////////////////////
//      assign out0 = (b==1'b1 && c==1'b0) || (d==1'b1);
//      assign out1 = (c==1'b1) || (d==1'b1);

endmodule
```

=> Dataflow modeling에서는 두가지 방법으로 코드를 구현하였다. 컴파일 후 코드 실행 시 두 방법 모두 같은 결과가 도출됨을 확인할 수 있다.

- out0에 **((b && !c) || d)**의 연산 결과를 할당한다.

- out1에 **(c || d)**의 연산 결과를 할당한다.

4) Test Bench

* 검증 요소

(1) 입력 값이 Module에 정확히 할당되는가?

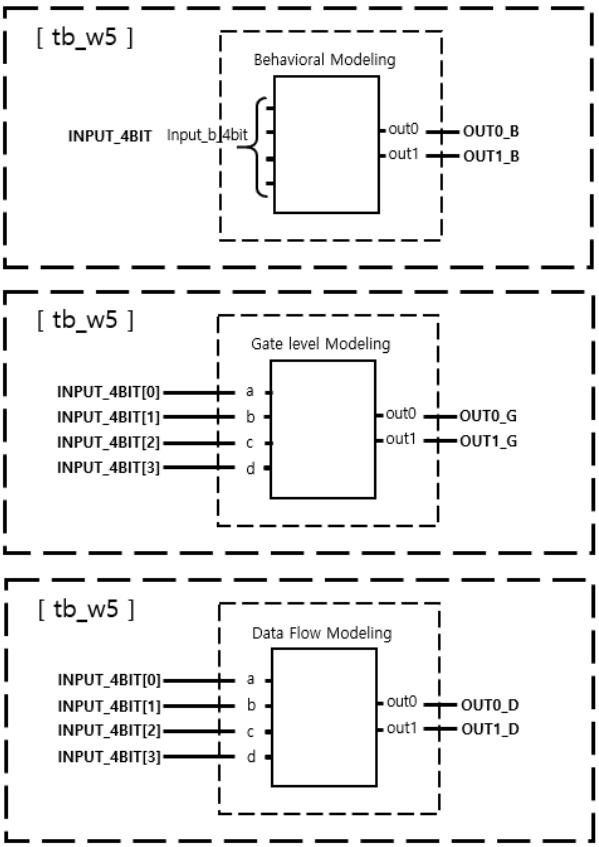
(2) Module 출력 값이 Truth Table과 일치하는가? (올바른 동작)

(3) Test pattern이 output의 모든 경우를 발생시킬 수 있는가?

* input, output wire/register 선언:

```
//4:2 Priority Encoder
//input
//Fill this out
reg [3:0] INPUT_4BIT; // INPUT of this test bench

//output
//Fill this out
wire OUT0_B, OUT1_B; // output wire of behavioral modeling
wire OUT0_D, OUT1_D; // output wire of dataflow modeling
wire OUT0_G, OUT1_G; // output wire of gatelevel modeling

//temporal variable for loop
//You may use it to create your test plan for the 4:2 priority module
integer count;
```

* 구현한 module과 test bench 상의 port 연결:



* test pattern 동작을 위한 initializing:

```
initial
begin
        A_4_TO_2 = 1'b0; B_4_TO_2 = 1'b0; C_4_TO_2 = 1'b0; D_4_TO_2 = 1'b0;
        A_2_TO_4 = 1'b0; B_2_TO_4 = 1'b0;
        //Fill this out
        INPUT_4BIT = 4'b0000;// Initializing of the input bits [0000]
end
```

* test pattern 동작 구현:

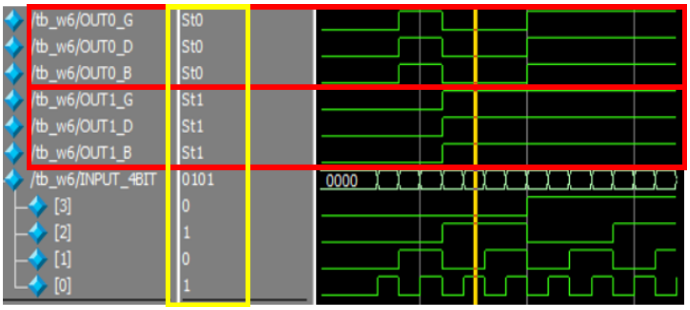
```
////THE FIRST WAY OF TEST PATTERN///////////
    for (count=0; count<16 ; count=count+1)
            #10 INPUT_4BIT = count;

////THE SECOND WAY OF TEST PATTERN//////////

    #10 INPUT_4BIT = 4'b0000;
    #10 INPUT_4BIT = 4'b0001;
    #10 INPUT_4BIT = 4'b0010;
    #10 INPUT_4BIT = 4'b0011;
    #10 INPUT_4BIT = 4'b0100;
    #10 INPUT_4BIT = 4'b0101;
    #10 INPUT_4BIT = 4'b0110;
    #10 INPUT_4BIT = 4'b0111;
    #10 INPUT_4BIT = 4'b1000;
    #10 INPUT_4BIT = 4'b1001;
    #10 INPUT_4BIT = 4'b1010;
    #10 INPUT_4BIT = 4'b1011;
    #10 INPUT_4BIT = 4'b1100;
    #10 INPUT_4BIT = 4'b1101;
```

=> 4bit 크기의 input reg(INPUT_4BIT)를 선언하였으며 for문을 사용해 발생할 수 있는 모든 경우의 수를 구현하였다. 두번째는 각 Bit의 조합 결과를 모두 서술하고 각 line에 delay를 따로 주는 방법으로 구현하였으며 시뮬레이션 결과 동일한 동작을 수행함을 확인할 수 있었다.

4. 실험 결과 (고찰)

[Experiment result]

(1) 2:4 Priority ENCODER



붉은 색으로 표시한 부분은 각각의 OUTPUT을 서로 다른 Modeling 기법으로 설계하여 시뮬레이션 한 결과이다. 총 3가지 방법으로 구현된 Module은 시뮬레이션 결과, 동일한 파형으로 나타남을 알 수 있다. 또한 두 개의 out(out0, out1)에 표현될 수 있는 1(TRUE)과 0(FALSE)의 모든 경우를 test pattern으로 사용하였고 이를 시뮬레이션 결과로 확인할 수 있다.

노란색으로 표시한 부분은 주어진 각 Module이 설계 전 검증한 TRUTH TABLE과 Boolean expression 결과와 부합하는지 검증하기 위한 point이다. 4 bits의 Input bits인 'INPUT_4BIT'가 1010인 경우 각 모듈의 out0은 0이 되어야 하고, out1은 1이 되어야 하는데 시뮬레이션 결과, 합당한 결과가 도출됨을 알 수 있다. 또한 위의 예시에서 a3=1이 된 경우 하위 비트가 어떤 값을 갖던 주어진 출력을 출력하는 'priority rule'이 정확하게 적용되었음을 확인할 수 있다.

(2) 고찰

이번 실험에서는 세 가지의 modeling 기법을 통해 설계를 진행하였고 각 Module의 Truth Table과 Karnaugh Map을 이론적으로 점검하고 이를 Module 설계 시 어떻게 적용할지 고찰하였다. 'casex'문장을 사용하며 문장 내에 선언되는 Default line에 대해 탐구하여 보았다. Default line은 하드웨어적으로 중요한 요소인데, 함수 내에서 선언되지 않는 경우의 수를 default로 지정하지 않는다면 실제 H/W 구현 시 이상 동작을 일으킬 수 있기 때문이다.

Test-bench 코드 구현 시 4개의 Input 값을 하나의 line으로 구현하기 위하여 4bits 벡터를 선언하여 사용하였다. 하여 Behavioral modeling의 경우 input port에 4bits를 그대로 연결하여 테스트하였고, 나머지 modeling에서는 벡터 내의 요소(1bit)를 각각의 input port에 연결하여 테스트를 진행하였다.

모든 Module의 구현 후에 각 Module을 정확하게 Test 할 수 있는 test bench code인지 고찰했고 발생할 수 있는 모든 경우의 수를 test하였다. 또한 Test pattern 구현 시 검증해야 하는 test point를 명확하게 지정해야 하고 이는 모듈의 동작 검증에서 매우 중요함을 알 수 있었다.

5. 참 고

오윤호 (2021). *Logic Design Laboratory(ICE2005)* Week3.Sungkyunkwan Univ.

Charles H. Roth, JR. Larry L. Kinney. (2014). *Fundamentals of Logic Design.* Cengage Engineering, a part of Cengage Learning.