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 : 논리회로설계실험
과 제 명 : Design Full-Adder
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 : 2021.03.20

1-Bit Full Adder/4-Bit Full Adder 설계

*이지학

*성균관대학교 전자전기공학부 dlwlgkr1@g.skku.edu

Design 1-Bit Full Adder/4-Bit Full Adder

*Ji-Hak Lee

*Dept. pf Electronic & Electrical Engineering, Sungkyunkwan University

목 차

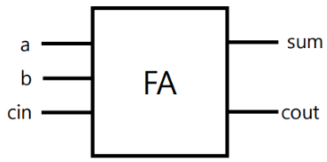
*보고서 표지 포함 6 Page (표지 제외 5 Page)

1. 요 약 [Brief]

1-Bit Full Adder 설계를 통하여 Full Adder와 Half Adder의 차이점을 명확하게 이해하고 이를 설계에 적용하여 본다. Data Flow modeling 기법과 Gate-Level modeling 두 가지 기법을 통해 각 Modeling의 차이점과 설계 시 고려해야 할 사항들을 확인한다. Modeling을 하기에 앞서 해당 Module의 Truth Table과 Karnaugh Map을 이론적으로 점검하고 이를 Module 설계 시 어떻게 적용할지 고찰한다. Multi-Bit component와 연산에 대한 완벽한 이해를 바탕으로 4 Bit Adder Design을 진행하고 1-Bit Adder를 활용한 4-Bit Adder 설계로 Module을 instantiation하는 방법을 설계에 직접 적용하여 본다. 마지막으로 완성된 각각의 Module을 Test 하기 위한 합리적인 Test Plan을 수립하고 이를 test-bench code 구현에 적용하여 본다.

2. 이론적 접근 [Theoretical approach]

1) 1-Bit Full Adder



=> 1-Bit Full Adder는 A bit와 B Bit 그리고 전 Column에서 넘어온 Carry(C_{in})을 합산하는 Combinational Circuit이다.

* 두 수의 연산 시 발생하는 Carry는 Cout으로 할당된다.

2) Truth Table / K-Map (S: Sum / Cout: Carry Out)

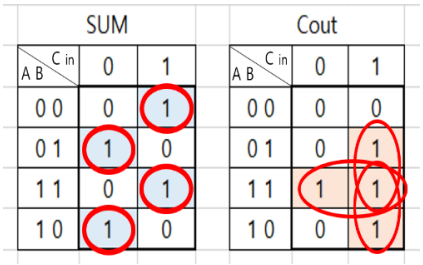
* 설계 조건:

- Boolean expression must consist of AND, OR, NOT, and XOR only.

- Other operators are not allowed and using parentheses is allowed.

- XOR available only for class 45.

Truth Table				
INPUT			OUTPUT	
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

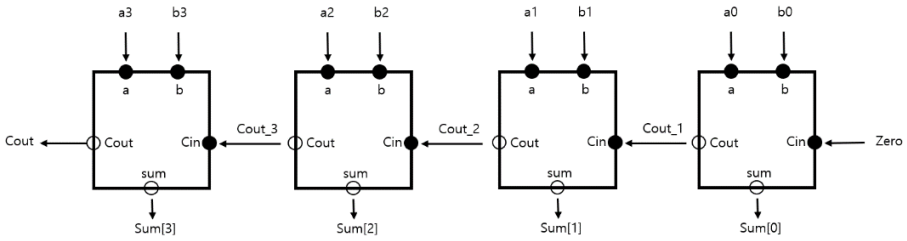


S *편의를 위하여 NOT 연산자를 A'으로 표현하였습니다.
= A'·B'·Cin + A'·B·C'in + A·B·Cin + A·B'·C'in
= Cin·(A'·B' + A·B) + C'in·(A'·B +A·B') 분배 법칙
= Cin·{(A+B) · (A'+B')} + C'in·(A'·B +A·B') 드모르간 법칙
= Cin·{A·B' + A'·B} + C'in·(A'·B +A·B') 전개
= Cin·{(A'+B) · (A+B')} + C'in·(A'·B +A·B') 드모르간 법칙
= Cin·{(A'·B +A·B')}' + C'in·(A'·B +A·B') 쌍대
= Cin ⊕ (A'·B + A·B')
= Cin ⊕ (A ⊕ B)

Cout
= A·B + Cin·A+ Cin·B
= A·B + Cin· (A+B)
* 이번 실험에서 사용할 OR Gate의 Input 수는 2개로 정해져 있기 때문에 분배 법칙 이용하여 정리.

3) 4-Bit Full Adder

4-Bit Full Adder는 1-Bit Full Adder의 조합으로 만들어진다. 1-Bit Full Adder의 Carry Out이 다음 Module의 Carry In으로 들어가며, 자리올림을 포함한 4Bit 연산을 가능하게 한다. 실험에서는 4Bit로 정의된 a[3:0], b[3:0]의 연산을 수행한다.



3. Verilog 구현 [Verilog Implementations]

1) 1-Bit Full Adder

[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.

- Use the source codes of XOR or XNOR gate that you implemented/ Using Built-in primitive gates in Verilog is not allowed.

* 설계 환경: Model Sim-Intel FPGA Starter Edition/Verilog HDL

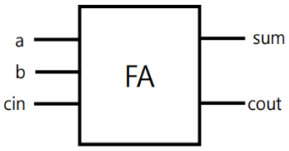
* 실험 세부:

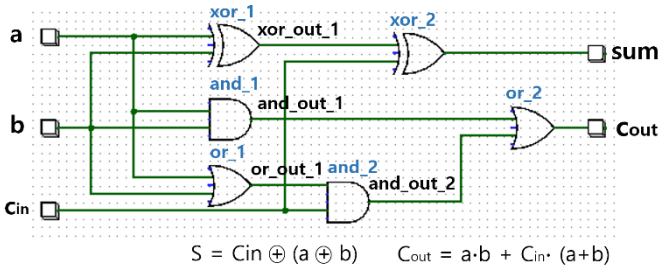
(1) 1Bit Full Adder의 Truth Table과 Karnaugh Map을 작성하고, 설계에 적합한 Boolean Expression을 도입하여 본다.

(2) Boolean Expression을 기본 Gate와 저번 실험에서 구현한 XOR Gate로 표현하여 본다.

(3) Module화 된 각 Gate를 연결할 wire를 선언하고, Input과 Output을 고려하여 연결한다.

- Disassemble






```
1   //This is for the 1-bit full adder module.
2
3   module full_adder_gatelevel_module (a, b, cin, sum, cout);
4           input a, b, cin;
5           output sum, cout;
6
7           wire xor_out_1;
8   //      wire xnor_out_1, xnor_out_2;
9   //      wire not_out_1;
10
11          wire and_out_1, and_out_2;
12          wire or_out_1;
13
14          //sum
15          xor_gatelevel_gate xor_1(.a(a), .b(b), .out(xor_out_1));
16          xor_gatelevel_gate xor_2(.a(xor_out_1), .b(cin), .out(sum));
17          //Fill this out
18
19          //cout
20          and_gate and_1 (.a(a), .b(b), .out(and_out_1));
21          or_gate or_1 (.a(a), .b(b), .out(or_out_1));
22          and_gate and_2 (.a(or_out_1), .b(cin), .out(and_out_2));
23          or_gate or_2 (.a(and_out_1), .b(and_out_2), .out(cout));
24          //Fill this out
25
26  endmodule
```

← Gate와 Gate 사이를 연결할 Wire를 정의한다.
* 사용하지 않는 Wire는 주석처리 하였습니다.

← 각 Gate 사이의 Input과 Output Wire를 할당한다.
* 이전 실험에서 만든 XOR 모듈 사용
* Sample Code의 구성을 최대한 따르는 방향으로 Gate Modeling을 진행하였습니다.

[Data Flow Modeling]

* 설계 환경: Model Sim-Intel FPGA Starter Edition/Verilog HDL

* 실험의 검증: Behavioral Modeling 기법으로 구현된 1-Bit Full Adder (sample code)와 Simulation 동작 결과를 대조하여 본다.

* 실험 세부:

(1) 1Bit Full Adder의 Truth Table과 Karnaugh Map을 작성하고, 설계에 적합한 Boolean Expression을 도입하여 본다.

(2) Boolean Expression을 참고하여 코드의 기본 Logic을 설계한다.

```
1
2   //This is for the 1-bit full adder module.
3
4   module full_adder_dataflow_module (a, b, cin, sum, cout);
5           input a, b, cin;
6           output sum, cout;
7
8           //sum
9           assign sum = (cin^(a^b));
10          //Fill this out
11
12          //cout
13          assign cout = ((a&&b)||(cin&&a)||(cin&&b));
14          //Fill this out
15
16  endmodule
```

← S = Cin ⊕ (a ⊕ b)

← Cout = a·b + Cin· (a+b)

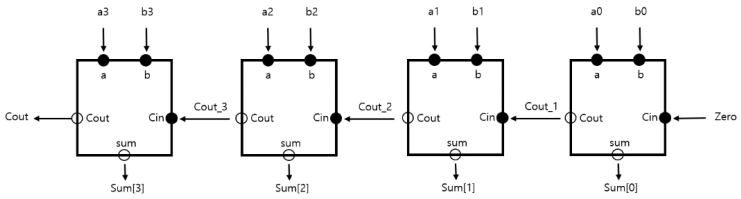
* Karnaugh map을 통해 구한 Boolean Expression을 Verilog Operator를 사용하여 구현한다.

(2) 4-Bit Full Adder

* 실험 세부:

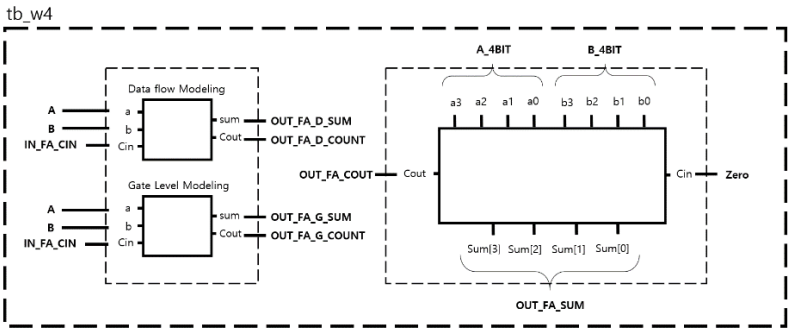
(1) 이전 실험에서 작성한 1-Bit Full Adder (Gate Level Modeling) Module 4개를 사용하여 4 Bit Full Adder를 구현한다.

(2) 각 Module의 Input 값과 Output 값을 정해진 순서에 따라 할당하고 그 동작을 검증한다.




```
1   //This is for the 4-bit full adder module.
2
3   module four_bit_full_adder_module (a, b, cin, sum, cout);
4           input [3:0] a, b;        ← Module의 input 값인 a, b는 4bit의 vector a[3:0], b[3:0]으로 지정한다.
5           input cin;
6           output [3:0] sum;        ← Module의 output 값인 sum은 4bit의 vector sum[3:0]으로 지정한다.
7           output cout;
8
9           wire cout_1, cout_2, cout_3;
10
11          full_adder_gatelevel_module f_1 (.a(a[0]), .b(b[0]), .cin(cin), .sum(sum[0]), .cout(cout_1));
12          full_adder_gatelevel_module f_2 (.a(a[1]), .b(b[1]), .cin(cout_1), .sum(sum[1]), .cout(cout_2));
13          full_adder_gatelevel_module f_3 (.a(a[2]), .b(b[2]), .cin(cout_2), .sum(sum[2]), .cout(cout_3));
14          full_adder_gatelevel_module f_4 (.a(a[3]), .b(b[3]), .cin(cout_3), .sum(sum[3]), .cout(cout));
15
16          //Fill this out                    4-Bit Full Adder 내의 4 개의 1-Bit Full Adder의 연결은 위의 Gate 연결을 참고하여 주십시오.
17
18
19  endmodule
```

(3) Test Bench



검증 요소

(1) 입력 값이 Module에 정확히 할당되는가?

(2) Module 출력 값이 Truth Table과 일치하는가? (올바른 동작을 보이는가?)

(3) Test pattern의 input 값으로 일반화된 output의 모든 경우를 도출할 수 있는가?

- 1-Bit Full Adder (Gate Level Modeling/ Data Flow Modeling)

Truth Table에 따라 8가지 경우(발생할 수 있는 모든 경우)를 Test하고 그 결과를 확인한다.

A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

```
38      initial
39      begin
40              A = 1'b0; B = 1'b0; IN_FA_CIN = 1'b0; ZERO = 1'b0;
41              #10 A = 1'b0; B = 1'b0; IN_FA_CIN = 1'b1;
42              #10 A = 1'b0; B = 1'b1; IN_FA_CIN = 1'b0;
43              #10 A = 1'b0; B = 1'b1; IN_FA_CIN = 1'b1;
44              #10 A = 1'b1; B = 1'b0; IN_FA_CIN = 1'b0;
45              #10 A = 1'b1; B = 1'b0; IN_FA_CIN = 1'b1;
46              #10 A = 1'b1; B = 1'b1; IN_FA_CIN = 1'b0;
47              #10 A = 1'b1; B = 1'b1; IN_FA_CIN = 1'b1;
48      end
```

- 4-Bit Full Adder

A-4BIT와 B_4BIT의 조합으로 나올 수 있는 경우의 수는 총 256가지이다. 실험적 한계로 모든 결과를 도출하기 어려워 가능한 Output 쌍을 일반화하여 정리하였다.

A	B	A	B	Sum	Carry Out
FALSE	FALSE	0 0 0 0	0 0 0 0	FALSE	0
TRUE	TRUE	1 0 0 0	1 0 0 0	FALSE	1
FALSE	TRUE	0 0 0 0	0 0 0 1		
TRUE	FALSE	0 0 0 1	0 0 0 0	TRUE	0
TRUE	TRUE	0 0 0 1	0 0 0 1		
TRUE	TRUE	1 1 1 1	0 0 1 0	TRUE	1

*4Bit에서 0000을 제외한 나머지 값은 모두 TRUE로 가정

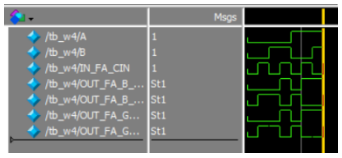
* FALSE = 0 0 0 0

```
49
50
51
52
53
54
55
56
57
```

```
initial
begin
        A_4BIT = 4'b0000; B_4BIT = 4'b0000;
        #10 A_4BIT = 4'b1000; B_4BIT = 4'b1000;
        #10 A_4BIT = 4'b0000; B_4BIT = 4'b0001;
        #10 A_4BIT = 4'b0001; B_4BIT = 4'b0000;
        #10 A_4BIT = 4'b0001; B_4BIT = 4'b0001;
        #10 A_4BIT = 4'b1111; B_4BIT = 4'b0010;
end
```

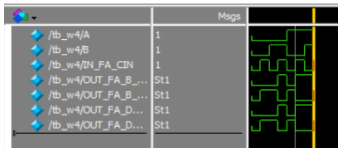
4. 실험 결과 (고찰) [Experiment result]

(1) 1-Bit Full Adder (Gate Level Modeling)



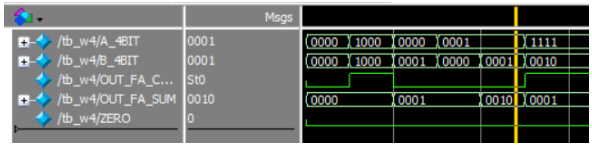
직접 설계한 1-Bit Full Adder와 sample code로 구현된 덧셈기가 동일한 파형으로 동작함을 확인할 수 있다. 또한 test pattern에서 지정한 모든 test point가 정상 동작함을 확인할 수 있고 덧셈에 대한 결과 값이 Truth Table의 값과 일치함을 확인할 수 있다.

(2) 1-Bit Full Adder (Data Flow Modeling)



직접 설계한 1-Bit Full Adder와 sample code로 구현된 덧셈기가 동일한 파형으로 동작함을 확인할 수 있다. 또한 test pattern에서 지정한 모든 test point가 정상 동작함을 확인할 수 있고 덧셈에 대한 결과 값이 Truth Table의 값과 일치함을 확인할 수 있다.

(3) 4-Bit Full Adder



* Overflow 발생시 Cout이 1이 됨을 확인할 수 있다.

1 Bit Full Adder를 instantiation하여 구현한 4-Bit Full Adder의 결과 값이 Truth Table 값과 일치함을 확인할 수 있다. 발생할 수 있는 모든 Output을 일반화하여 6가지 Case로 test를 진행하였고 Input과 Output의 test point에서 예측한 값이 정상적으로 도출됨을 확인할 수 있다.

(4) 고찰

이번 실험에서는 다양한 Modeling 기법을 1-Bit Full Adder 설계에 적용하여 구현하는 방법을 탐구하였다. 1-Bit Full Adder를 설계하며 이전 실험에서 구현한 XOR gate module을 성공적으로 Instantiation하였으며 만들어진 Module을 새로운 Project에 도입하여 적용하는 방법을 탐구하였다. 또한 Multi-Bit의 요소와 연산에 대한 완벽한 이해를 바탕으로 4-Bit Full Adder를 설계하였고 4Bit의 연산을 수행하는 Full Adder의 세부 동작에 대해 이해할 수 있다. 설계를 진행하며 Gate Level Modeling시 선행되는 Boolean Expression이 어떤 형태로 조합되는지에 따라 각기 다른 H/W Gate를 조합해야 한다는 사실을 배웠으며, 설계 초기에 정해진 H/W spec (reg, wire의 수 등)을 고려하여 설계에 적용해 나가는 프로세스가 필요함을 알 수 있었다. 모든 Module의 구현 후에 각 Module을 정확하게 Test 할 수 있는 test bench code를 작성하며, 발생할 수 있는 모든 경우의 수를 test하기 어려운 경우, 각 도출 값을 합리적인 형태로 일반화하여 정해진 Case별로 Testing 해야 함을 알 수 있었다.

5. 참 고

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