

# Lab 3. Combinational Logic Design I

2017 Fall Logic Design Lab

Department of Computer Science and Engineering

Seoul National University

# Outline

---

1. 가산기(adder)의 기초
2. 반가산기(half adder)와 전가산기(full adder)
3. Ripple carry adder와 carry select adder

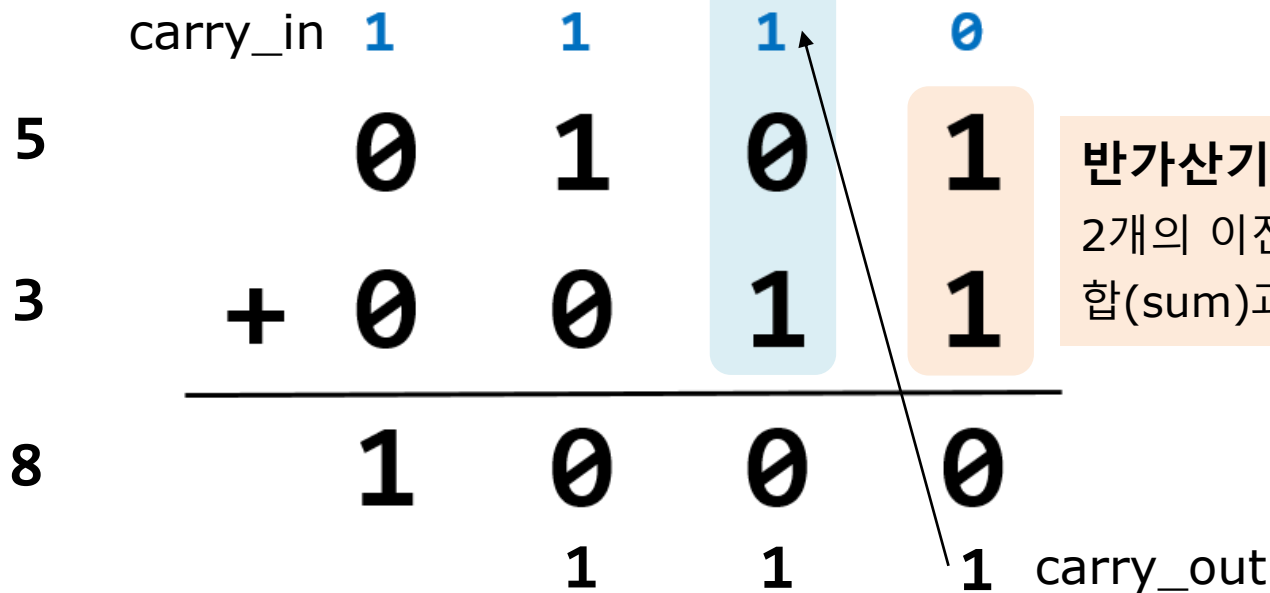
# 가산기의 기초

## ■ 가산기(adder)

- 이진수 덧셈 연산을 수행하기 위한 기초 회로
- ALU (arithmetic logic units) 의 가장 기본이 되는 덧셈 연산을 수행

### 전가산기(full adder)

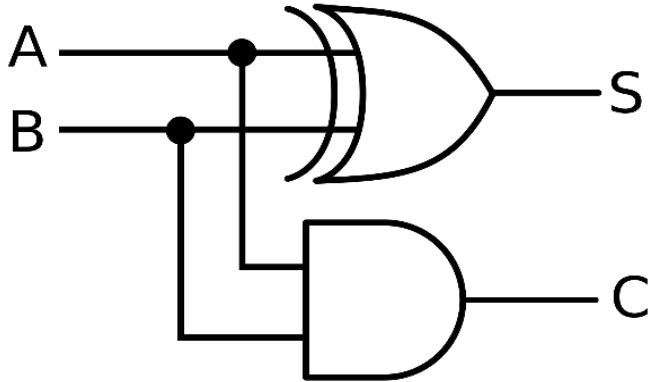
3개의 이진수 입력 A, B와 carry\_in을 받아  
합(sum)과 올림(carry\_out)을 출력



### 반가산기(half adder)

2개의 이진수 입력 A, B를 받아  
합(sum)과 올림(carry)을 출력

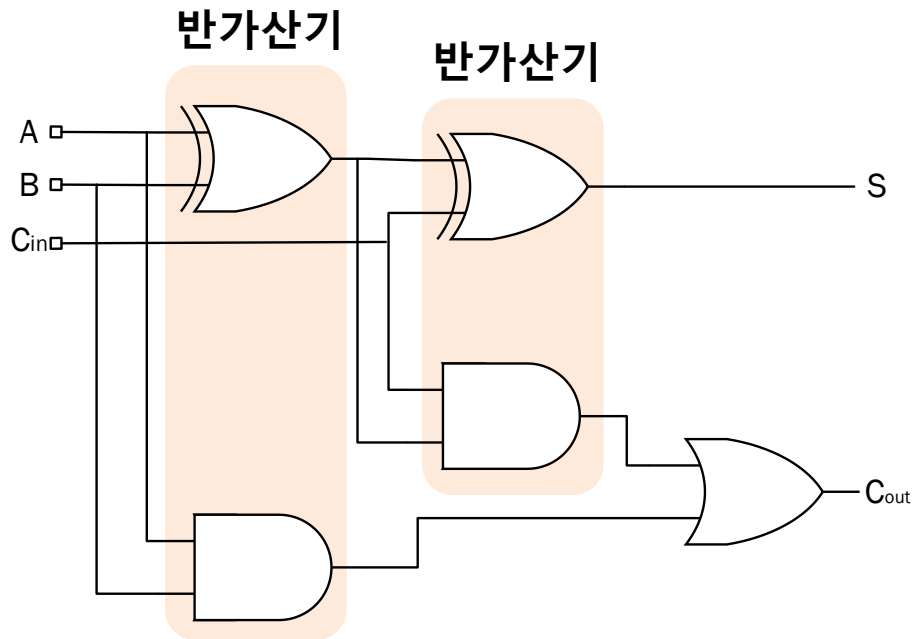
# 반가산기



Input		Output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- 두 이진수 A, B를 입력 받아 그 합(sum)과 올림(carry)을 출력
  - $S = X \oplus Y$
  - $C = X \cdot Y$
  - 실제 이진수의 덧셈 연산 구현을 위해서는 전 자릿수에서 올라온 올림을 같이 고려할 필요성 있음

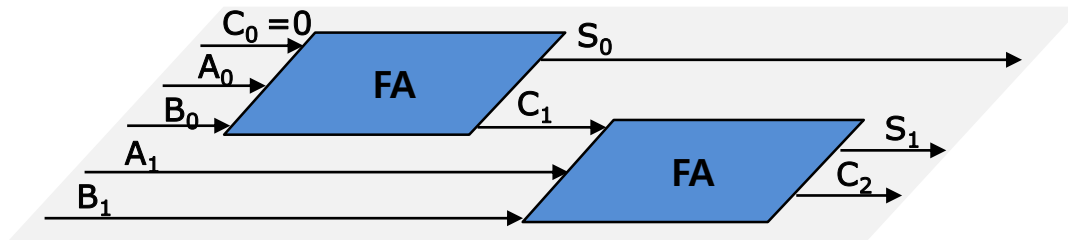
# 전가산기



Input			Output	
A	B	$C_{in}$	S	$C_{out}$
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

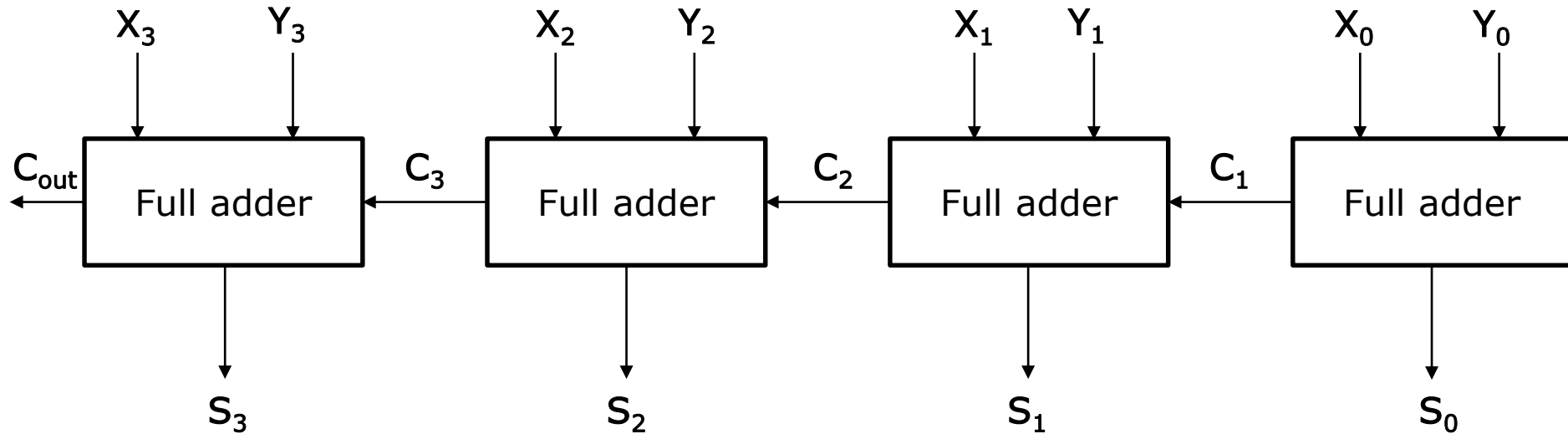
- 세 이진수 A, B,  $C_{in}$ 을 입력 받아 그 합(S)과 올림( $C_{out}$ )을 출력
  - $S = A \oplus B \oplus C_{in}$
  - $C_{out} = (A \cdot B) + (C_{in} \cdot (A \oplus B))$
  - 실제 이진수 덧셈 연산 구현의 기본 단위
  - 두 개의 반가산기와 OR 게이트로 구현 가능

## 2-Bit 가산기



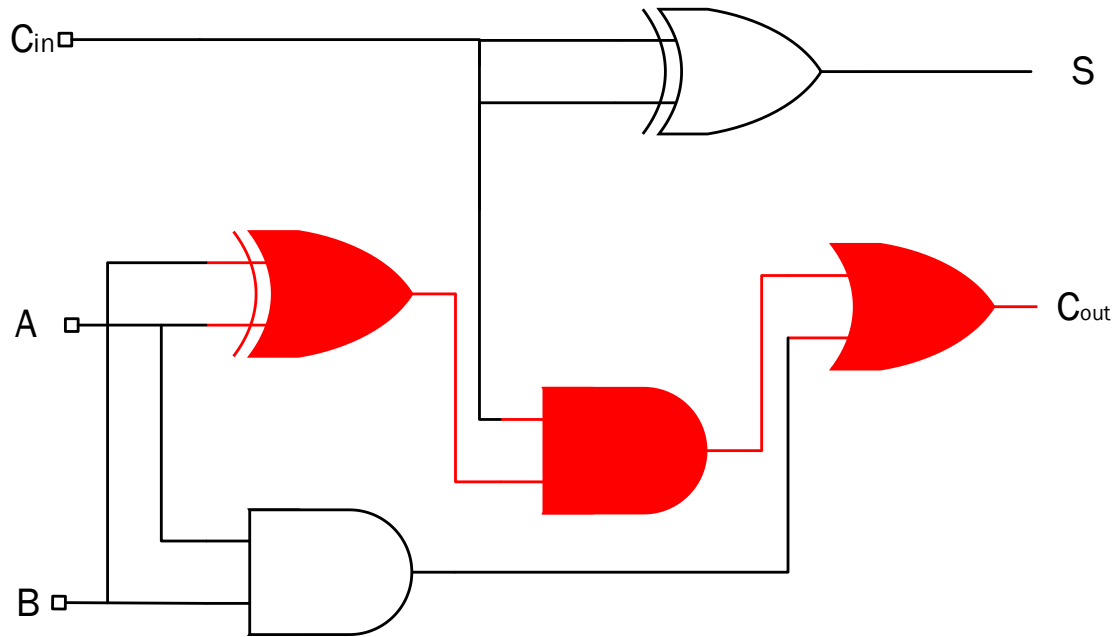
- 두 자리 이진수간의 덧셈은 두 개의 전가산기를 통해 구현 가능
  - $A_0, A_1, B_0, B_1$ 을 입력으로 받아 두 자리 결과값  $S_0, S_1$ 과 올림값  $C_2$ 를 출력
  - $A_0, B_0$  간에 계산된 올림 값이  $A_1, B_1$ 을 계산하기 위해 전달됨
  - $A_0, B_0$  가산기에서는 올림 값을 0으로 계산
  - 두 자리 이진수를 더했을 때 세 자리까지 결과가 나올 수 있음

# Ripple Carry Adder



- 여러 개의 전가산기를 연결하여 n-bit 이진수 덧셈을 수행하는 가산기 구현
  - 이 때,  $C_i$  (carry) 값이 순차적으로 다음 전가산기를 향해 물결(ripple) 치듯 전달되기 때문에 ripple carry adder로 호칭
  - 앞쪽 비트들의 계산이 완료되어 올림 값이 전달되기 전까지 다음 비트를 계산할 수 없음
  - 자릿수가 길 경우 이로 인한 딜레이가 길어질 수 있음

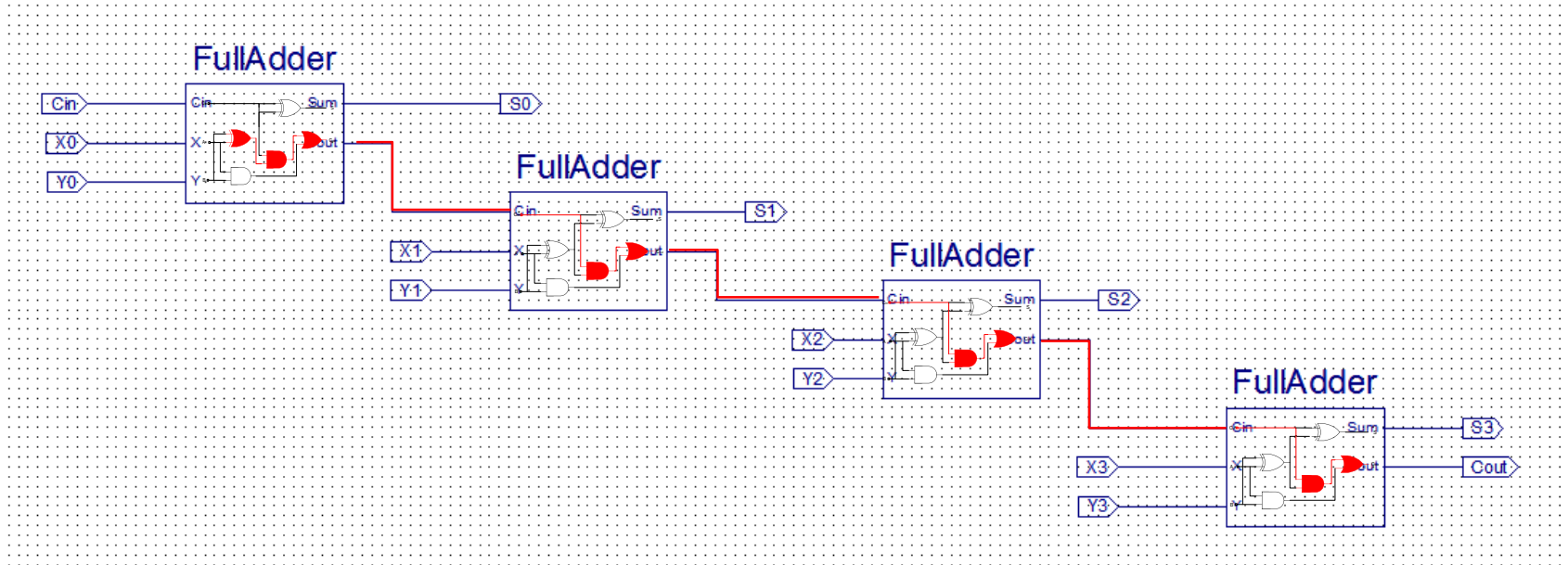
# 전달 지연(Propagation Delay)



- 논리 회로에 안정된 신호가 입력되는 순간부터, 안정되고 유효한 신호를 출력할 때까지 미세한 지연 시간이 발생
  - 논리 회로에서는 게이트 지연(gate delay) 이라고도 호칭
  - 전가산기의 경우,  $C_{out}$ 이 출력되기까지 3개의 게이트를 통과
    - 게이트 하나에서의 지연시간을  $x$ 라 표기할 경우  $3x$ 의 지연 시간 발생

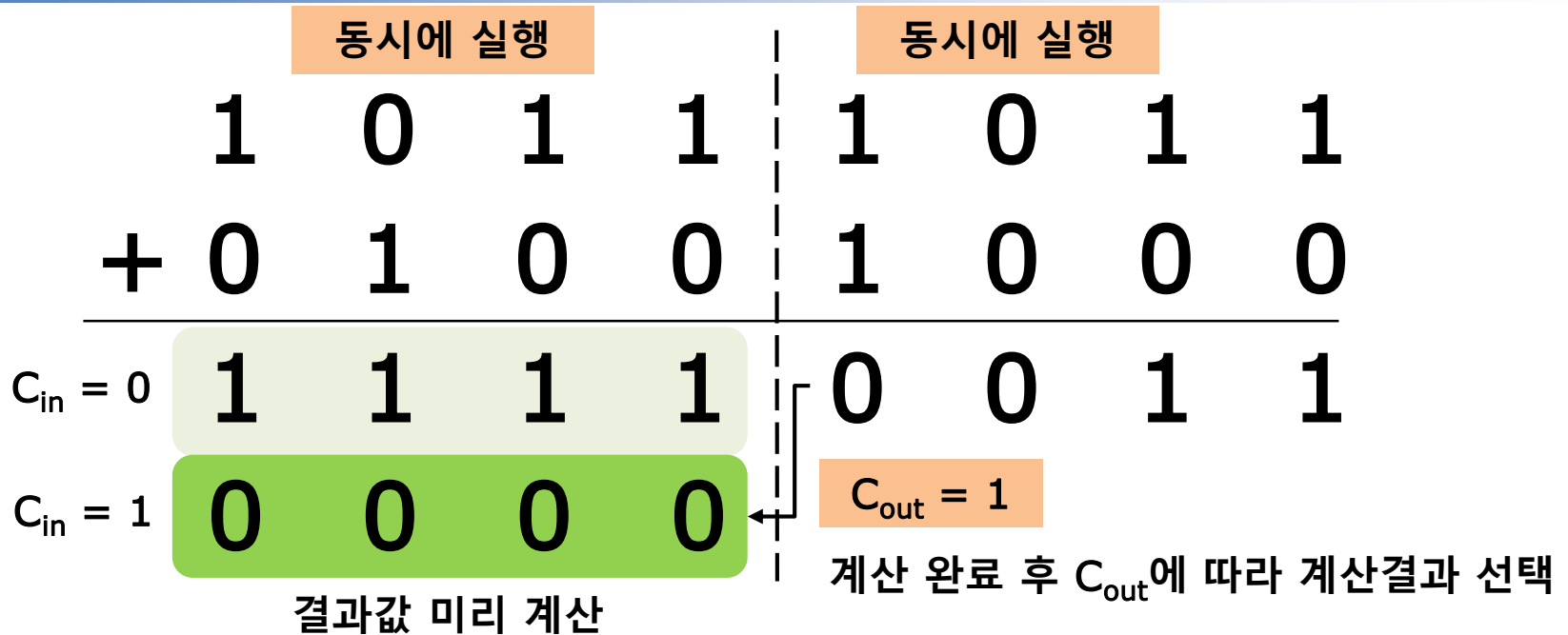


# Ripple Carry Adder의 전달 지연



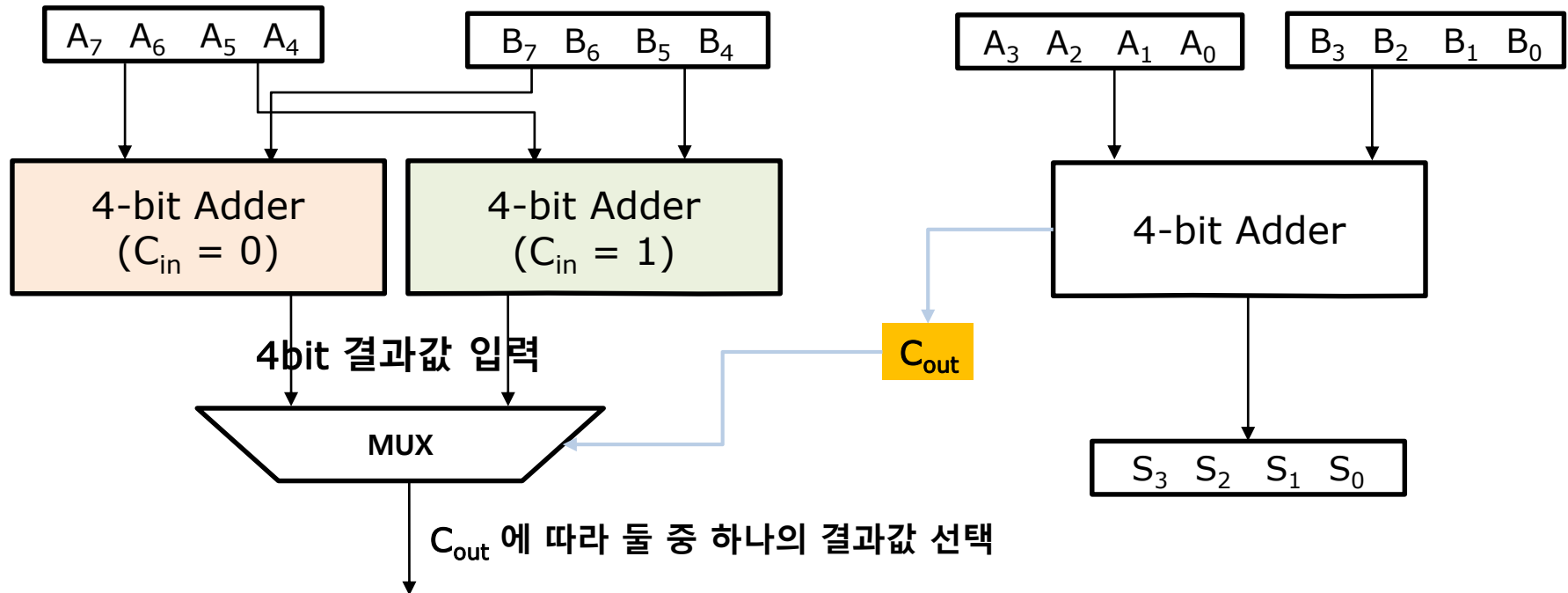
- 실제 여러 비트 가산기를 Ripple carry adder를 사용하여 구현할 경우 단계에 따라 전달 지연 시간이 가중됨
  - 하나의 전가산기 계산시마다 2개의 게이트를 통과해야 Cout값 출력
  - 게이트 하나에서의 지연시간을  $x$ 라 표기할 경우 4-bit 가산기의 최종 Cout값이 출력되려면  $3x + 2x \times 3 = 9x$ 의 지연 시간 발생

# Carry Select Adder



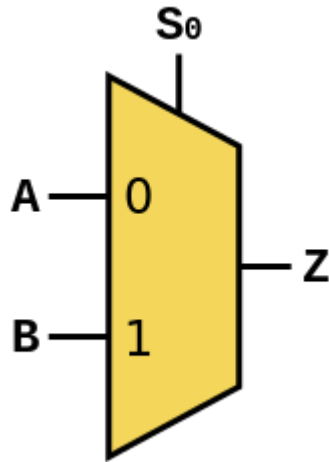
- Ripple carry adder의 경우 앞쪽 계산에서 carry가 전달된 후 계산이 이루어지기 때문에 긴 자릿수를 연산할 경우 딜레이 발생
  - 올림값은 0 또는 1 이므로 각 경우에 따라 계산을 미리 해놓고 실제 값에 의해 선택

# Carry Select Adder (Cont'd)



- 실제 구현을 위해서는  $C_{in}$ 의 입력값 0, 1에 따라 서로 다른 두 개의 Adder를 동시에 사용
  - 사용하는 전자 회로를 늘려서 동작 시간을 단축시키기 위한 기법
  - Tradeoff: 동작 성능(시간) vs 회로의 크기
  - 입력 값에 따라 두 입력 중 하나를 선택하는 MUX (multiplexer) 회로 사용 필요

# Multiplexer

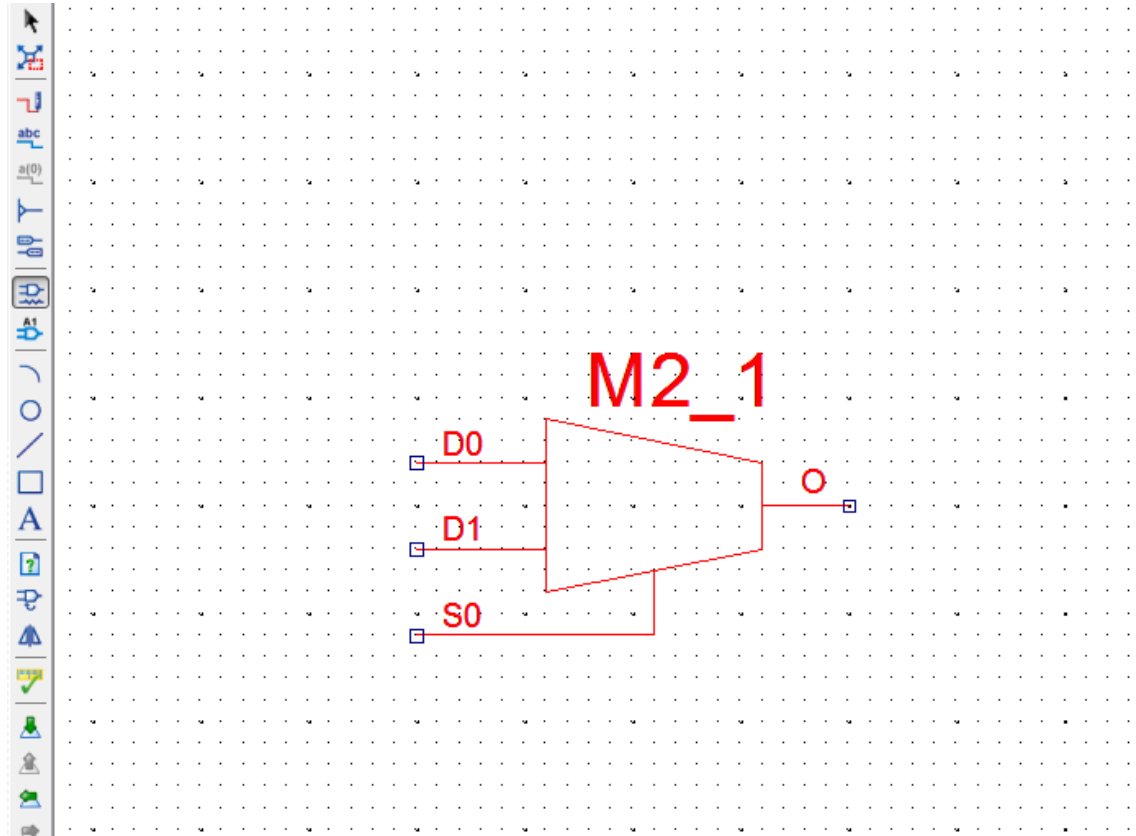
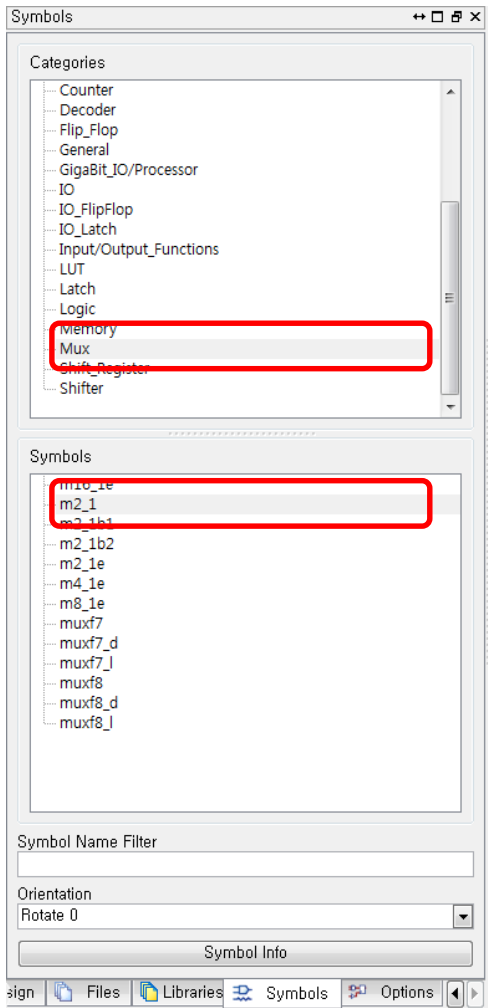


Input			Output
$S_0$	A	B	Z
0	0	0	0
	0	1	0
	1	0	1
	1	1	1
1	0	0	0
	0	1	1
	1	0	0
	1	1	1

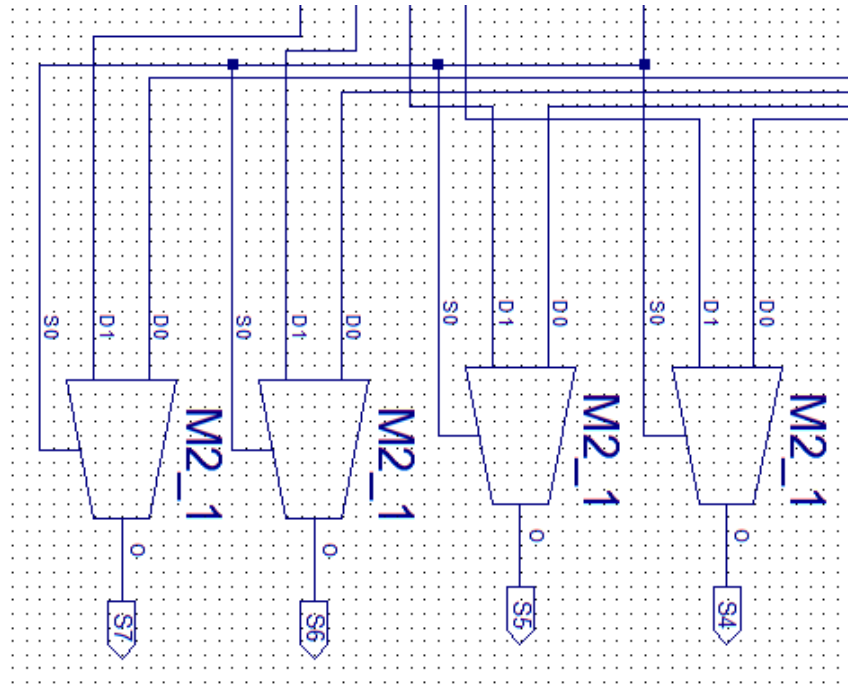
- S의 신호값에 따라 출력 값을 선택적으로 결정하는 회로
  - $A_0 \dots A_{2^n-1}$ 의 input에 대하여  $S_0 \dots S_{n-1}$ 의 select 신호로 output 선택
  - (input 개수):(output 개수) MUX 로 표현
  - 이후 실험에서 자세히 다룰 예정이므로 여기에서는 개념만 확인

# MUX 심볼 사용방법

- Symbol 탭에서 MUX - M2\_1 심볼 선택하여 입력

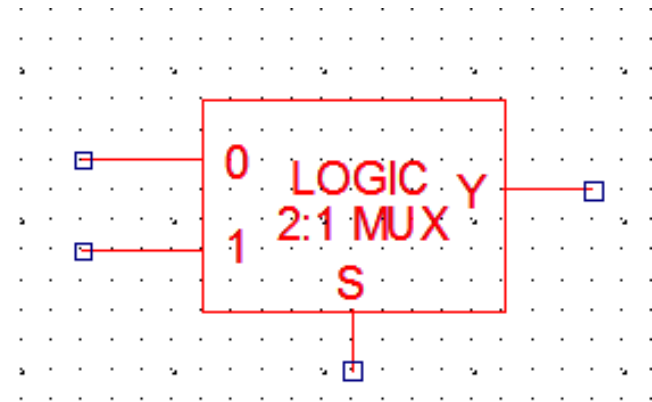
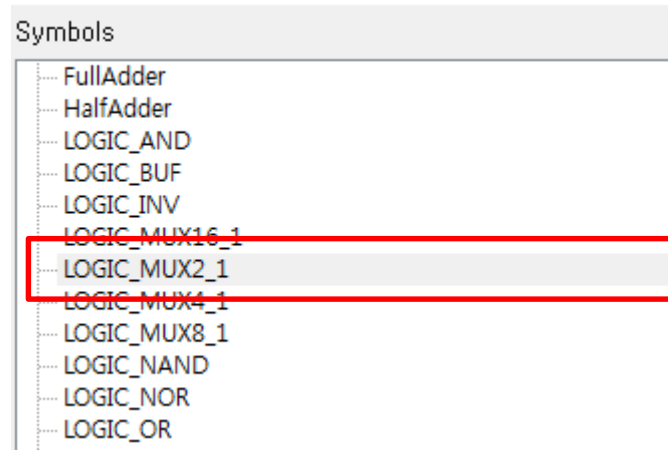


# 4-Bit MUX 회로의 구성



- 실제 두 개의 4 bit 입력간 선택하는 회로를 만들기 위해서는 2:1 MUX 4 개를 사용하여 구현
  - 같은 Select 신호(회로도의 S0) 공유
  - 입력에 따라 각 2:1 MUX에서 D0, D1 값 중 하나를 선택
  - D0값은 Carry가 0인 경우, D1값은 Carry가 1인 경우

# LOGIC\_MUX Symbols



- 실제 MUX에서도 전달 지연 현상이 발생
  - 특히 MUX의 경우 회로의 복잡성으로 인해 일반적인 게이트보다 긴 딜레이 발생
  - 이를 표현하기 위해 LOGIC\_MUX2\_1 심볼을 별도로 사용
    - 2 ns의 딜레이를 가지고 있도록 구현
  - MUX를 사용하면서도 ripple carry adder에 비해 carry select adder의 성능이 빠를 수 있는지 실험을 통해 확인

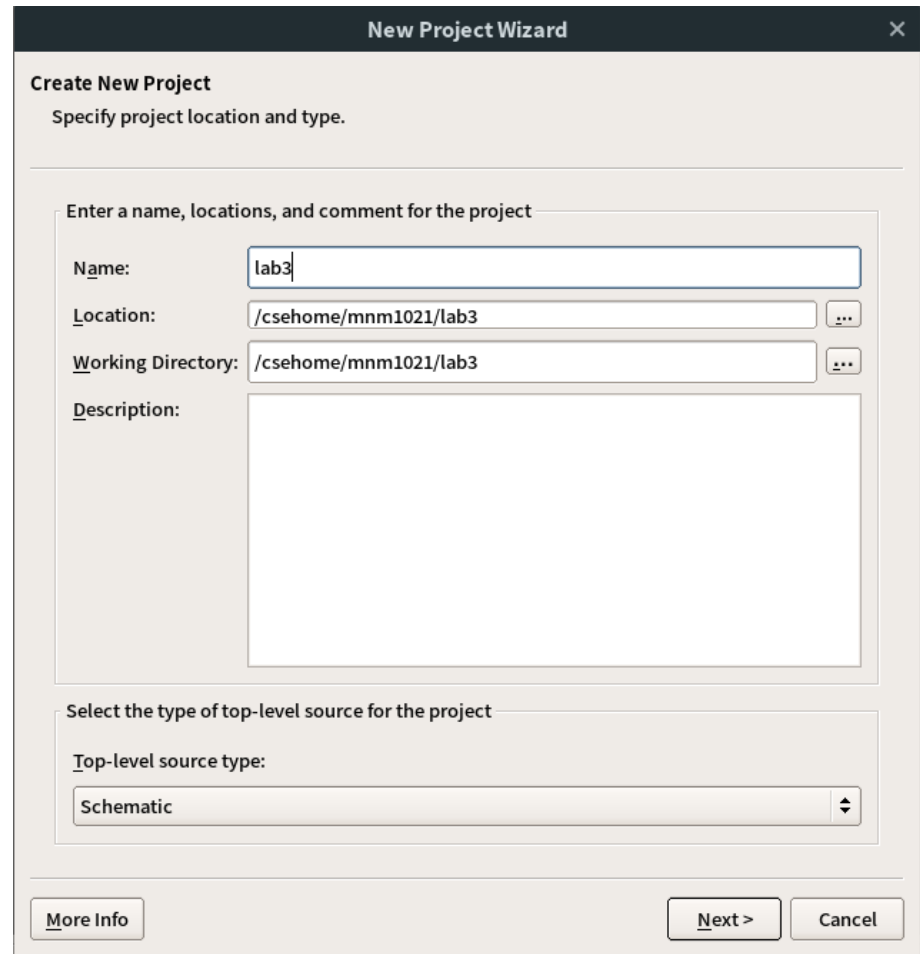
# 실험 1 – 반가산기 구현 및 사용자 심볼 등록

- 목표
  - 반가산기 schematic 구현 및 테스트
  - 구현한 반가산기 schematic을 사용자 심볼으로 등록
- 실험 내용
  - 프로젝트 생성 후, 반가산기 schematic을 구현
  - 구현 후 test fixture를 통해 결과값 확인
  - 구현 및 테스트 완료한 반가산기 schematic을 사용자 심볼으로 등록
- 제출 사항
  - 없음



# 실험 1 – 프로젝트 생성

- 프로젝트 생성
  - New project
  - Name 입력(e.g., lab3)
  - Schematic으로 설정 후 Next



The image shows a 'New Project Wizard' dialog box with a dark title bar and a close button. The main area is titled 'Create New Project' with the instruction 'Specify project location and type.' Below this, there are two sections. The first section, 'Enter a name, locations, and comment for the project', contains four fields: 'Name' (with 'lab3' entered), 'Location' (with '/csehome/mnm1021/lab3' and a browse button), 'Working Directory' (with '/csehome/mnm1021/lab3' and a browse button), and 'Description' (a large text area). The second section, 'Select the type of top-level source for the project', contains a 'Top-level source type' dropdown menu set to 'Schematic'. At the bottom, there are three buttons: 'More Info', 'Next >', and 'Cancel'.

New Project Wizard

Create New Project  
Specify project location and type.

Enter a name, locations, and comment for the project

Name: lab3

Location: /csehome/mnm1021/lab3

Working Directory: /csehome/mnm1021/lab3

Description:

Select the type of top-level source for the project

Top-level source type:  
Schematic

More Info Next > Cancel

# 실험 1 – 프로젝트 생성

## ■ 프로젝트 생성

- Family tab: Spartan3A and Spartan3AN
- Device: XC3S50AN
- Package: TQG144
- Synthesis Tool: XST
- Simulator: ISim
- Preferred Language: Verilog
- 설정 후 Next, Finish 클릭해 생성 완료

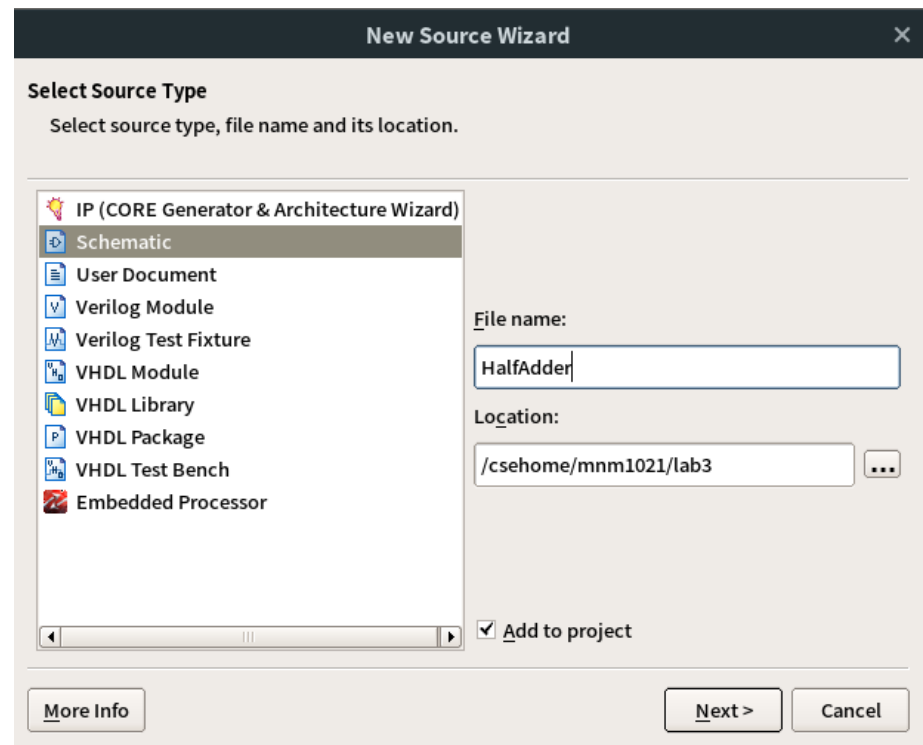
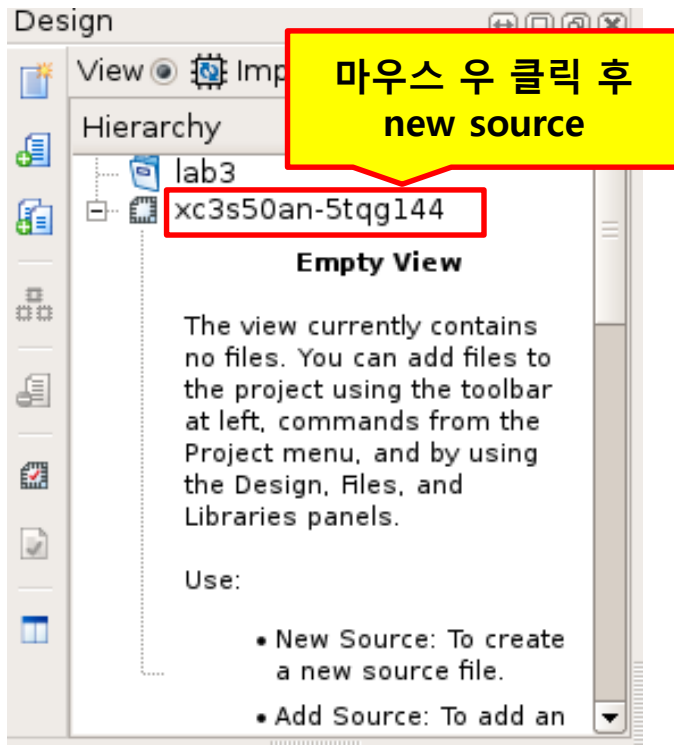
The image shows the 'New Project Wizard' dialog box in a software development environment. The title bar is 'New Project Wizard' with a close button. The main section is 'Project Settings' with the instruction 'Specify device and project properties.' Below this is a section titled 'Select the device and design flow for the project' which contains a table of properties.

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3A and Spartan3AN
Device	XC3S50AN
Package	TQG144
Speed	-4
Top-Level Source Type	Schematic
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	Verilog
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93

At the bottom of the dialog, there are three buttons: 'More Info', '< Back', and 'Next >', and a 'Cancel' button on the far right.

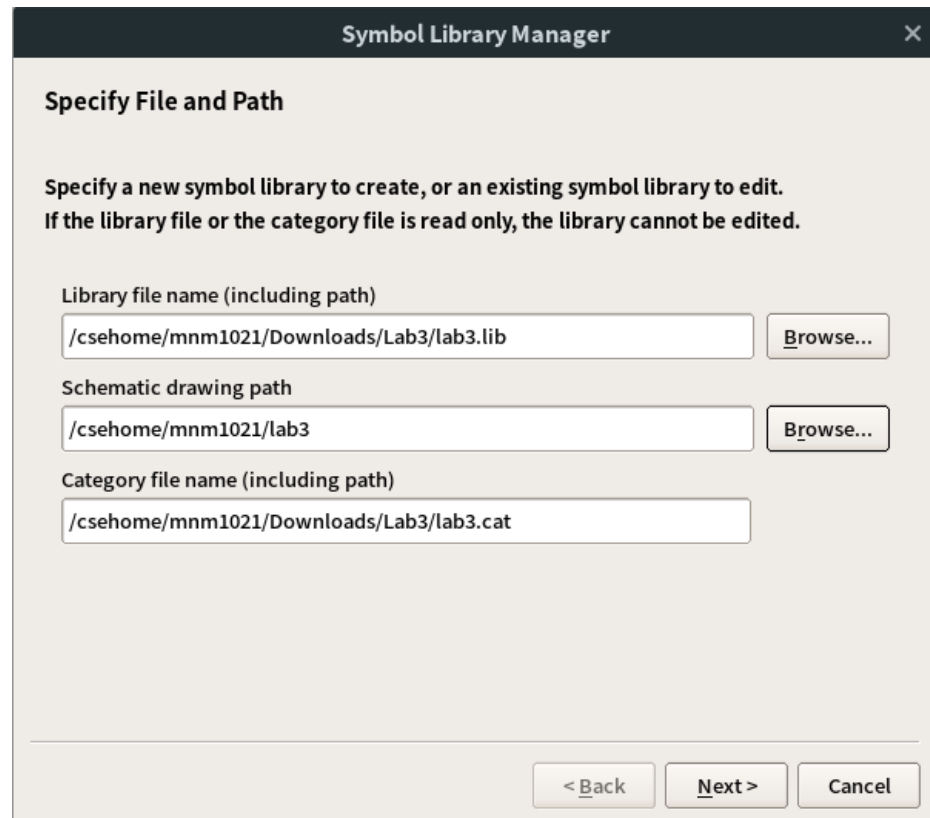
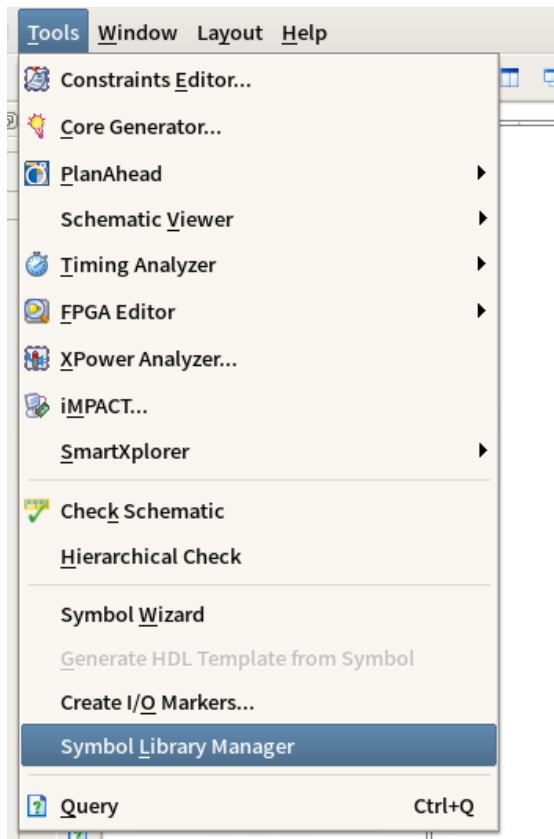
# 실험 1 – 프로젝트 생성

- Schematic 파일 생성
  - New source
  - Schematic 선택
  - 파일 이름(e.g., HalfAdder) 입력 후 Next, Finish 클릭으로 생성 완료



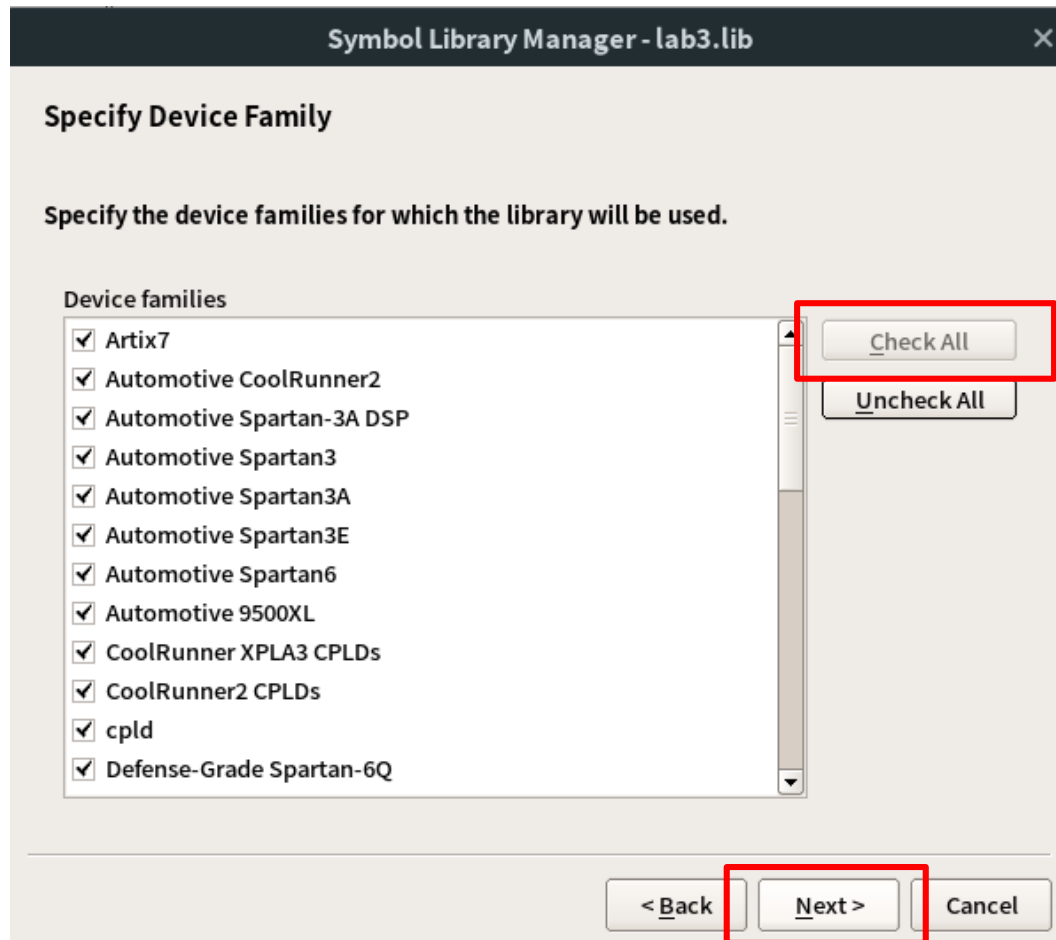
# 실험 1 – 딜레이가 적용된 Logic Gate 추가

- Tools tab – Symbol Library Manager 선택하여 아래와 같이 지정
  - lab3.lib 다운받은 위치
  - 프로젝트를 생성한 위치 설정



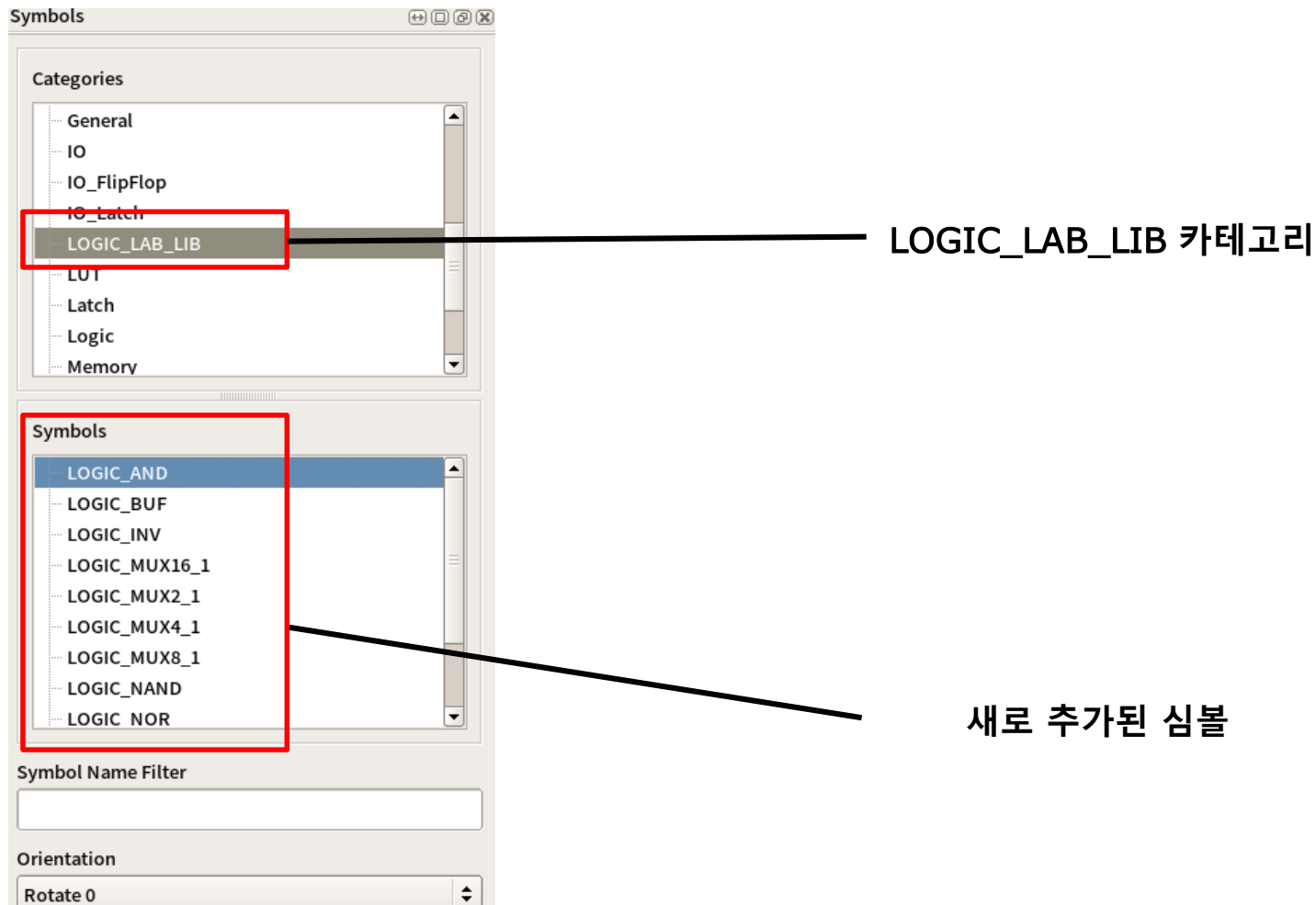
# 실험 1 – 딜레이가 적용된 Logic Gate 추가

- 연속적으로 Next 클릭 후 Specify Device Family 항목에서는 Check All 설정 후 Next 클릭



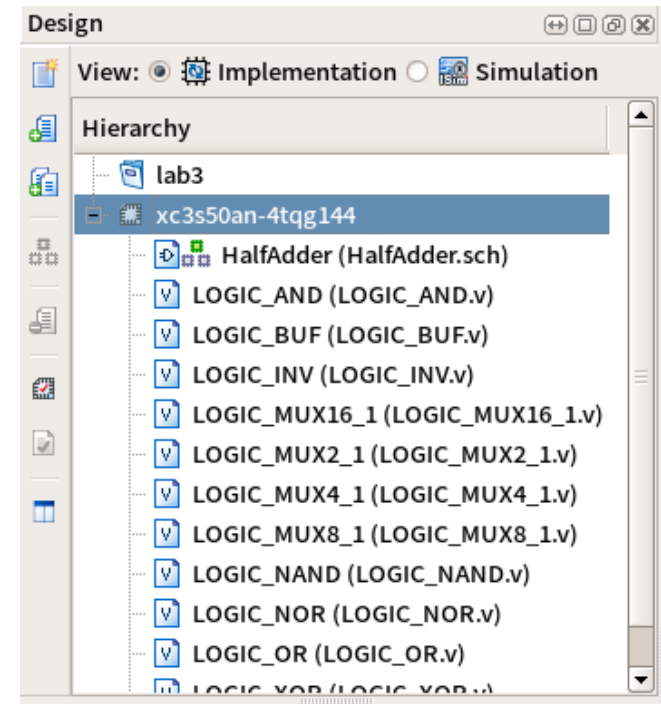
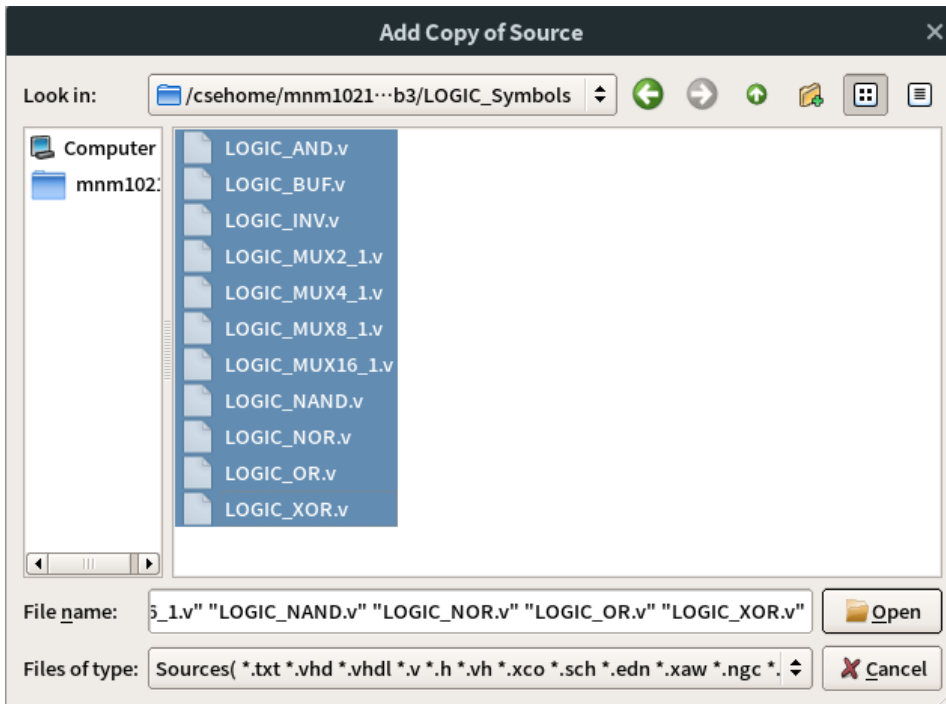
# 실험 1 – 딜레이가 적용된 Logic Gate 추가

- Symbol 탭에서 새로 추가된 LOGIC\_LAB\_LIB 카테고리 및 LOGIC\_XXX 형태의 추가된 심볼 확인 가능



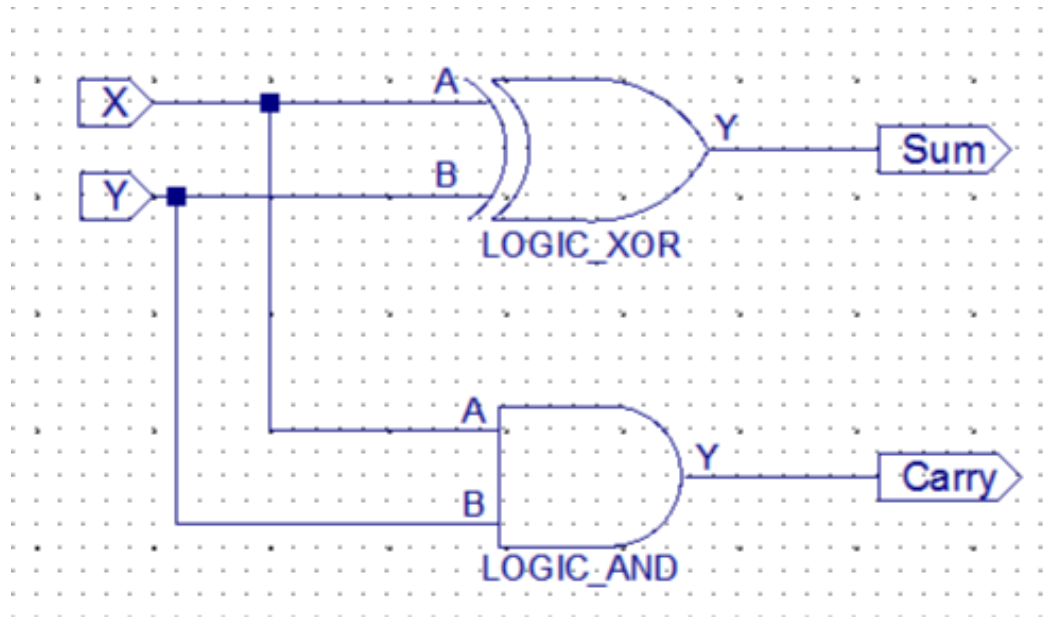
# 실험 1 – 딜레이가 적용된 Logic Gate 추가

- 실제 심볼이 정상 동작하기 위해서는 심볼 동작을 기술하기 위한 하드웨어 모듈 코드를 불러오는 작업이 필요함
  - Add Copy of Source 명령을 통해 프로젝트에 ETL을 통해 배포된 Xilinx\_Symbol.zip 파일로 압축되어 있는 LOGIC\_XXX.v 파일들을 모두 import
  - Design tab에서 추가된 심볼들 확인



# 실험 1 – 반가산기 구현하고 사용자 심볼 등록

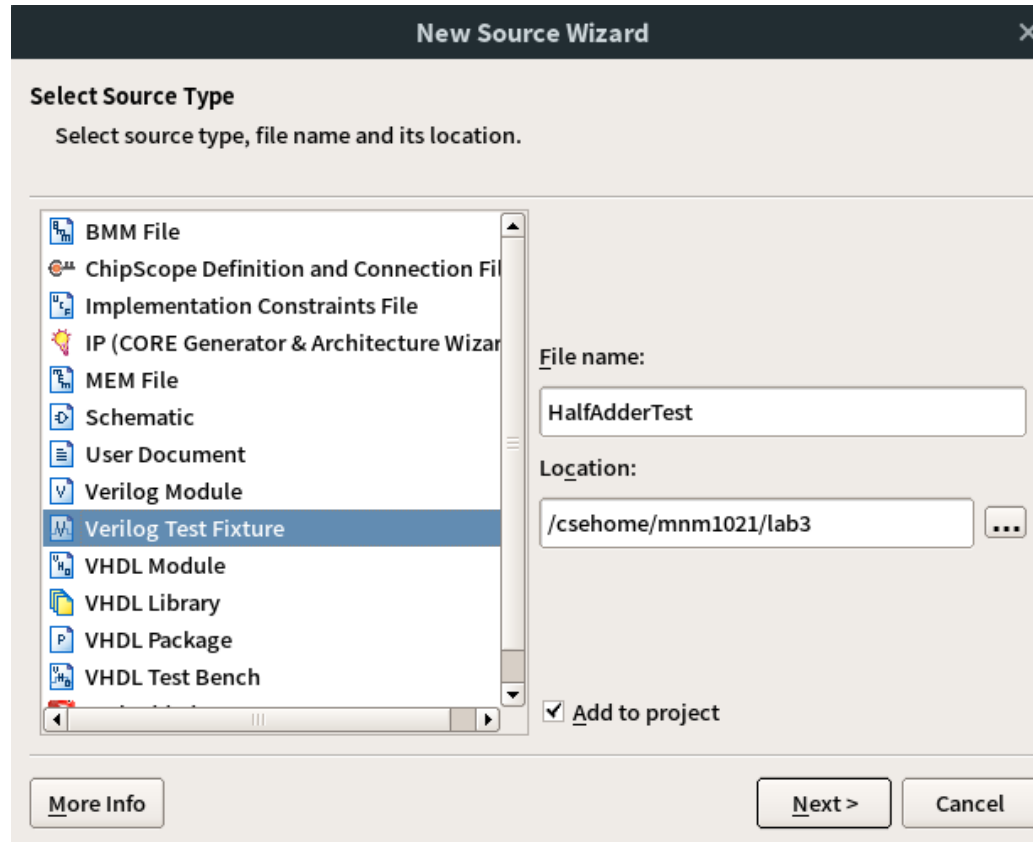
- 추가한 logic gate를 이용해 아래와 같이 half adder 만들기





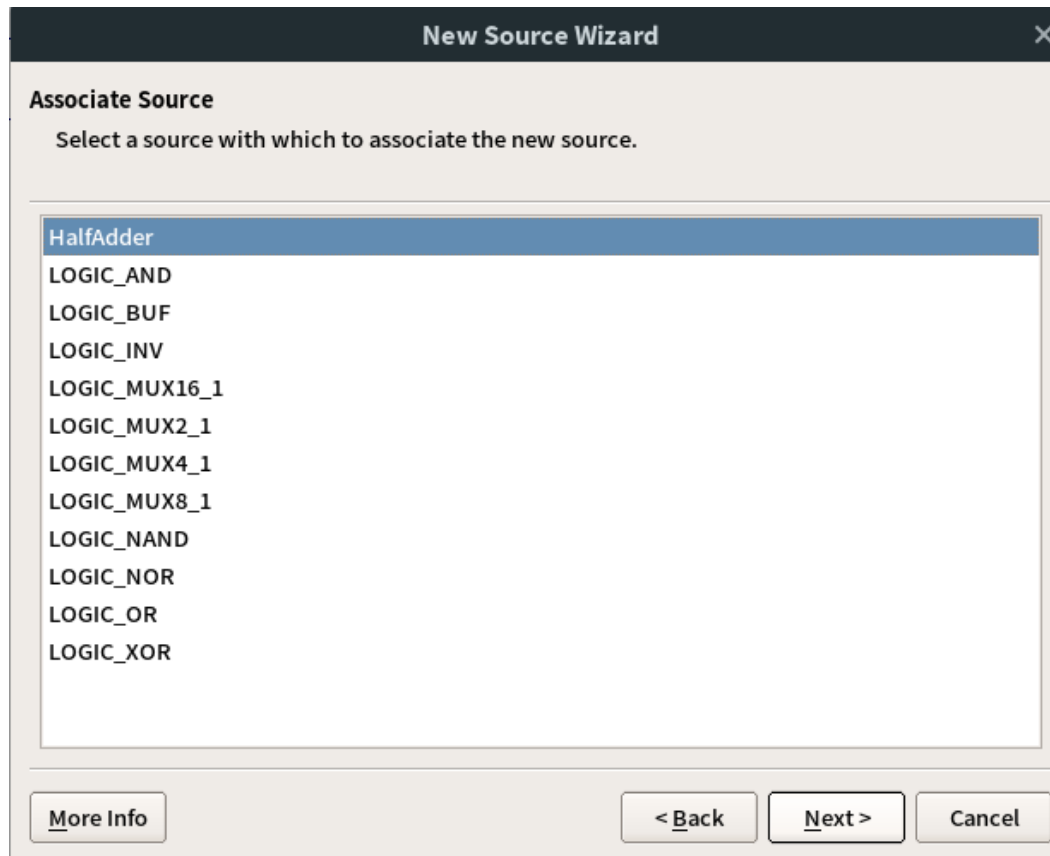
# 실험 1 – 반가산기 구현하고 사용자 심볼 등록

- New source – Verilog Test fixture 메뉴 선택
- 파일 이름 입력 후 Next



# 실험 1 – 반가산기 구현하고 사용자 심볼 등록

- 방금 만든 half adder 파일을 선택 후 Next, Finish를 클릭해 생성 완료

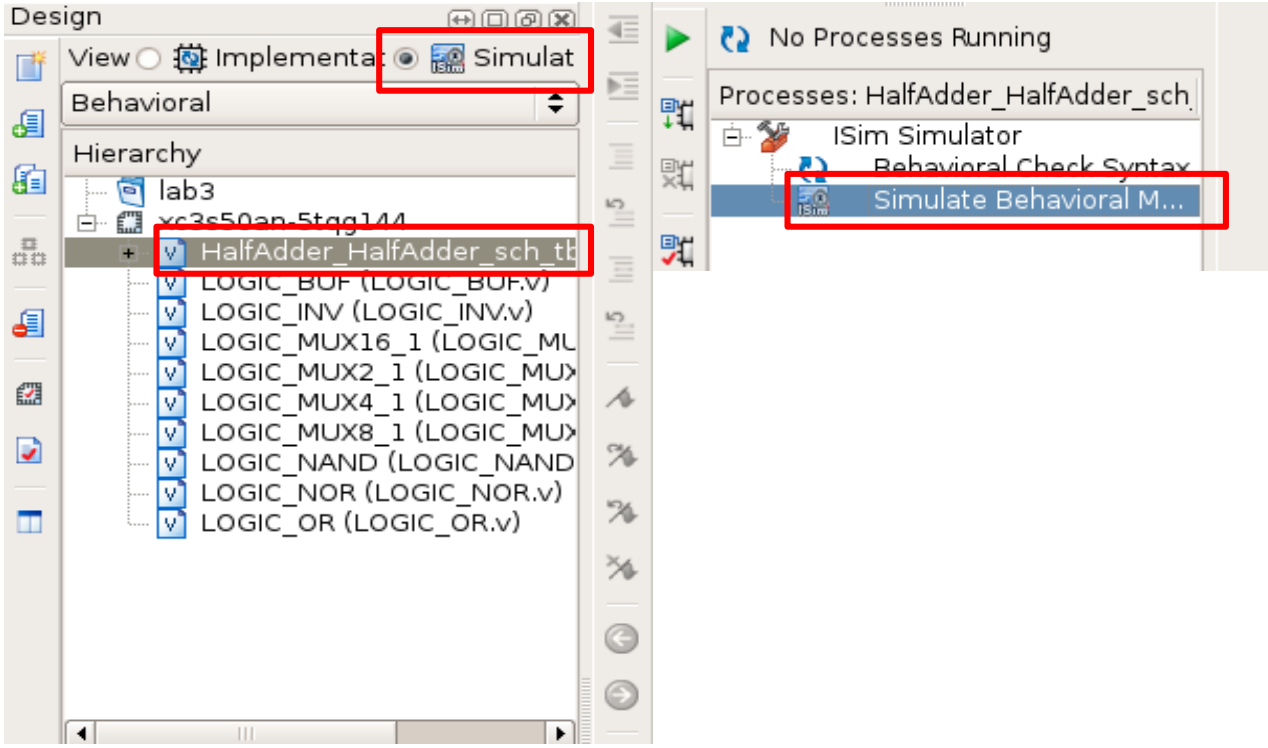


# 실험 1 – 반가산기 구현하고 사용자 심볼 등록

## ■ Verilog Test Fixture 파일 작성 후 Simulation

- 기본적인 initialize 코드는 자동으로 채워짐
- initial begin 아래에 그림과 같이 시간에 따라 입력 값을 변화 시키도록 작성
- Simulation radio 버튼 클릭
- test fixture 파일 선택 후 아래 Simulate Behavioral Model 더블클릭으로 실행

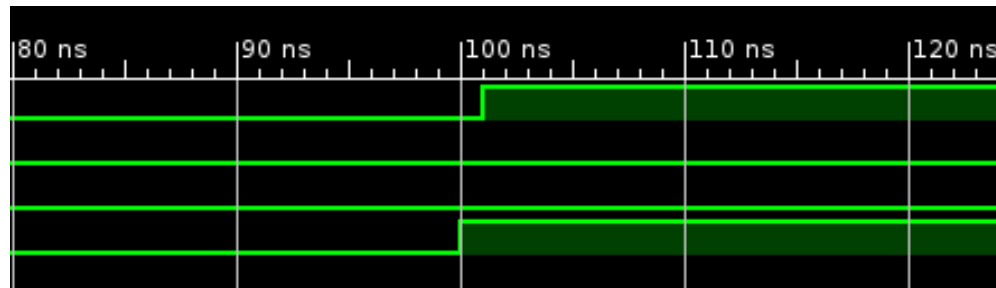
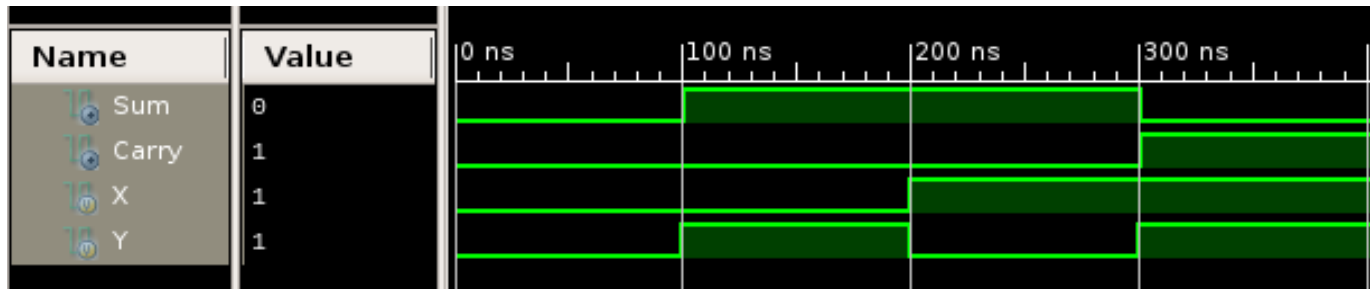
```
24 // Initialize Inputs
25
26 initial begin
27     X = 0;
28     Y = 0;
29     #100;
30     X = 0;
31     Y = 1;
32     #100;
33     X = 1;
34     Y = 0;
35     #100;
36     X = 1;
37     Y = 1;
38 end
39 endmodule
```



The screenshot shows the EDA tool interface during simulation. The Design window displays the hierarchy with 'HalfAdder\_HalfAdder\_sch.tb' selected. The View menu has 'Simulate' checked. The Processes window shows 'Simulate Behavioral Model' as the active process.

# 실험 1 – 반가산기 구현하고 사용자 심볼 등록

- Simulation결과 확인
  - X, Y 입력 값에 따라 Sum, Carry가 맞게 변화 하는지 확인
  - 더 확대해보면 아래와 같이 딜레이가 적용되고 있는 것을 확인할 수 있음



# 실험 1 – 반가산기 구현하고 사용자 심볼 등록

- 사용자 심볼 등록
  - Tools tab – Symbol Wizard 클릭
  - Using schematic 선택
  - 방금 만든 schematic파일 선택
  - Next 클릭

Symbol Wizard

Source Page

Select the source for pin names and the symbol shape.

Pin name source

☐ Specify manually

☒ Using schematic HalfAdder

☐ Using symbol

☐ Import symbol attributes

Shape

☒ Do not use reference symbol

☒ Rectangle

☐ Square

☐ Use reference symbol

Browse...

More Info Next > Cancel

# 실험 1 – 반가산기 구현하고 사용자 심볼 등록

- 선택한 schematic에 맞게 핀이 자동으로 설정이 되어있음
- 만약 Output 핀의 순서를 바꾸고 싶으면 Order 값을 바꾸어 설정
- Next, Finish로 심볼 등록 완료
- Symbol 탭에 지금 만든 HalfAdder라는 심볼이 추가 되어 있는 것을 확인

**Symbol Wizard**

**Pin Page**  
Define the pins to be placed on the symbol shape.

Symbol name:  
HalfAdder

Pin definitions:

	Name	Polarity	Side	Order
1	Carry	Output	Right	2
2	Sum	Output	Right	1
3	X	Input	Left	1
4	Y	Input	Left	2

**Keyboard Usage Tips:**  
-Use Tab and Shift-Tab to go to previous/next cell inside grid  
-Use Tab and Shift-Tab to change focus among controls

**Buttons:** Add Pin, Remove Pin/Spacer, Insert Spacer, Move Spacer Up, Move Spacer Down

**Navigation:** More Info, < Back, Next >, Cancel

**Symbol Wizard**

**Layout Page**  
Define the layout for various elements of the symbol.

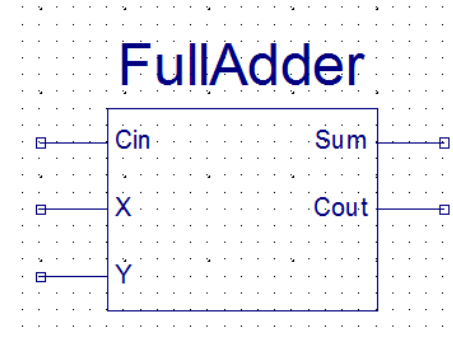
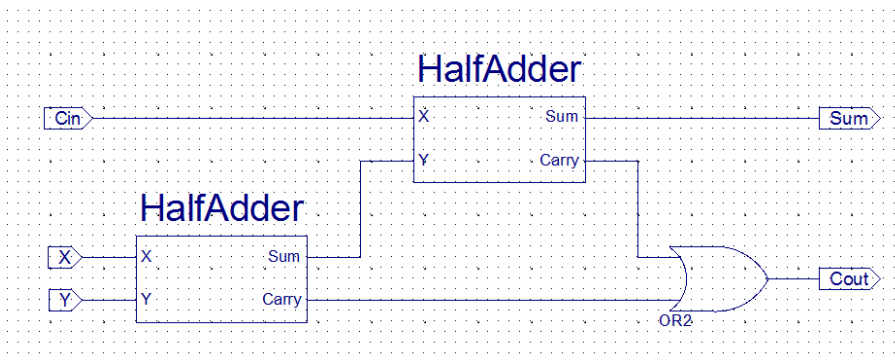
Symbol name font size: 56  
Pin name font size: 24  
Pin length: 64  
Pin space: 64  
Pin edge: 32  
Symbol width: 256  
Symbol origin: Left Bottom

**Diagram:** A schematic diagram of a Full Adder (FD) symbol. It shows a central box with inputs X and Y on the left, and outputs Sum and Carry on the right. The symbol is labeled 'FD' at the top. Dimensions and positions are indicated with arrows and labels: 'symbol name font size' for the 'FD' text, 'pin name font size' for the input/output labels, 'pin length' for the vertical lines, 'pin space' for the gap between pins, 'symbol width' for the total width, and 'symbol origin' for the bottom-left corner.

**Navigation:** More Info, < Back, Next >, Cancel

# 실험 2 – 전가산기 구현 및 사용자 심볼 등록

- 목표
  - 전가산기 schematic 구현
  - 사용자 심볼 등록
- 실험 내용
  - 새로운 schematic 파일 생성
  - 실험 1에서 완성한 half adder 심볼을 사용하여 전가산기 구현
  - eTL에서 받은 FullAdderTest.v 파일을 통해 시뮬레이터 동작 확인 후, 사용자 심볼로 등록
- 제출 사항
  - 없음
- Schematic



# 멀티 비트 입출력 회로의 테스트

reg[2:0] count;

wire X;

wire Y;

wire Cin;

assign X = count[0];

assign Y = count[1];

assign Cin = count[2];

// Output

wire Sum;

wire Cout;

// Instantiate the UUT

...

// Initialize Inputs

initial begin

count = 0;

#20;

repeat (8) begin

count = count + 1;

#20;

end

end

3 bit 각각 떨어져 있는 출력 값을 하나의 이진수 숫자로 보기 위해 vector 타입을 사용 가능

- <데이터 타입> [최상위비트:최하위 비트] <변수 명>

repeat 구문과 reg (레지스터)를 사용하여 반복되는 구문 테스트 가능



# 실험 3 – 4-Bit Ripple Carry Adder 구현

## ■ 목표

- 4-Bit Ripple Carry Adder schematic 구현

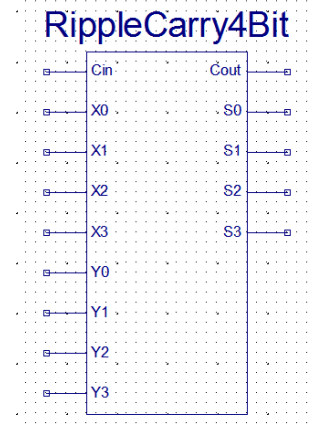
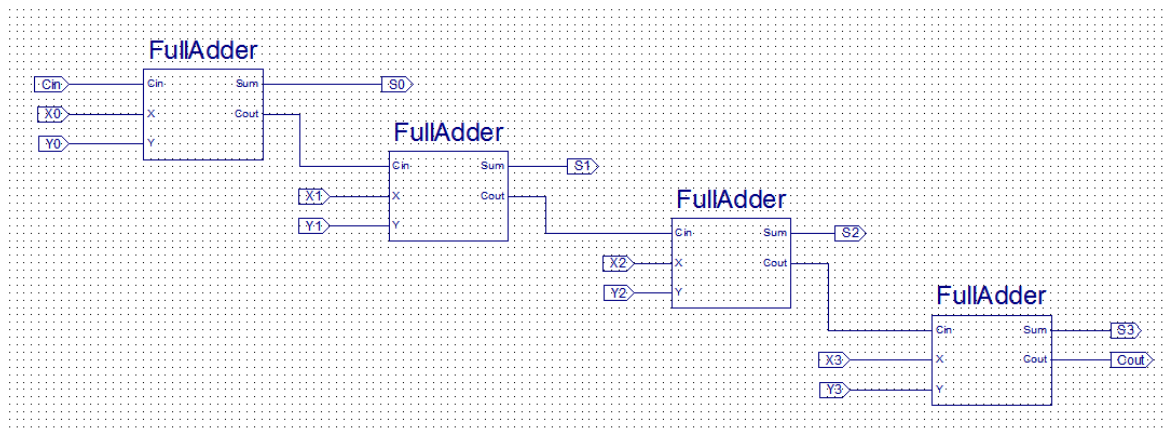
## ■ 실험 내용

- 실험 2에서 완성한 전가산기 심볼을 사용하여 ripple carry adder 구현
- eTL에서 받은 RippleCarryTest.v 파일을 통해 시뮬레이터 동작 확인 후 **조교에게 검증**
- 'RippleCarry4Bit' 심볼로 등록하여 실험 4에 활용

## ■ 제출 사항

- 없음

## ■ Schematic



# 실험 4 – 8-Bit Carry Select Adder 구현

## ■ 목표

- 8-bit ripple carry adder와 8-bit carry select adder 구현 및 동작 비교

## ■ 실험 내용

- 4-bit ripple carry adder 두 개를 연결하여 8-bit ripple carry adder를 대조 군으로 구현
- LOGIC 2:1 MUX를 활용하여 8-bit carry select adder 구현
- 시뮬레이터를 통해 두 회로 간의 딜레이 차이 및 실제 덧셈 연산 동작 확인

## ■ 제출 사항

- 완성된 8-bit ripple carry adder와 8-bit carry select adder의 회로도
  - 이 때, 두 회로는 같은 입력을 공유하도록 회로 구성
- 같은 입력을 두 회로에 동시에 인가했을 때 발생하는 시뮬레이션 결과
  - 활동지에 모든 case가 아닌 8-bit ripple carry adder에서 가장 시간 지연이 길 것으로 예상되는 case(A[7:0], B[7:0]값)를 선정하고 이유를 서술한 후, 해당 case의 시뮬레이션 결과 스크린샷과 함께 제출

# 실험과제 제출 안내

## ■ 보고서 포함 사항

- 실험 4 결과물
  - 8-bit ripple carry adder와 8-bit carry select adder의 회로도 스크린샷
  - 시뮬레이션 결과 스크린샷
  - 8-bit ripple carry adder 에서 가장 딜레이가 클 것으로 예상되는 case 및 그 이유 서술
- 추가적인 내용은 자유롭게 작성
- 하나의 문서로 정리

## ■ 제출 방법 및 기한

- ETL 과제 게시판에 팀별로 제출
- 일요일 오후 6시까지