

Lab 4. Verilog 언어의 기초

2018 Fall Logic Design LAB

Department of Computer Science and Engineering

Seoul National University

Outline

1. Verilog를 이용한 간단한 하드웨어 구현
2. Xilinx ISE를 이용한 하드웨어 시뮬레이션

Verilog

- 하드웨어 기술 언어 (HDL : Hardware Description Language)
 - 1983년 하드웨어 기술언어인 HiLo와 C 언어의 특징을 기반으로 개발
 - Verilog 2005 : IEEE 1364-2005로 표준화됨
- 구조적 모델링 및 동작적 모델링 지원
 - 구조적 모델링 : 기본 게이트나 기존에 설계된 모듈들을 서로 연결하여 회로 설계
 - 동작적 모델링 : 회로의 내부 구조 대신 입력과 출력간의 관계(회로의 기능) 기술
- 언어 문법적 특징 – C 언어와 유사
 - 대소문자 구별(Case-sensitive), 세미콜론(;)으로 문장 종결
 - 흐름 제어(control flow) 문 : if/else, case, for
 - 주석 : // (한 줄), /* */ (여러 줄)

Verilog 기본 구조

■ 모듈(Module)

- 모듈은 Verilog에서 시스템을 표현하는 기본 구성 요소
- 모듈은 상위 계층에서 하위 계층의 모듈을 불러 재사용 가능
- 모듈간 인터페이스(포트)를 통해 커뮤니케이션 가능

```
module ModuleName(Port List...);
```

포트선언

데이터 타입 선언
(wire, reg, parameter)

모듈 동작 기술
(assign, always, initial,
하위 모듈 호출)

```
endmodule
```

```
module inverter(input a,  
                output reg o);
```

```
always @ (*)  
begin
```

```
    o = !a;
```

```
end  
endmodule
```

Verilog 문법 – 자료형 (1)

- 정수/논리 표현
 - [`<비트폭>'<진수>`]`<수치>`
 - e.g. `8'b0100_0000`, `4'd12`, `8'hff`
- Net data type
 - 다른 output으로 부터 연결되어(driven) 되어야만 함
 - wire : 논리적인 기능이 없는 컴포넌트간 물리적인 wire 연결 표현
- Variable data type
 - 값 할당 후 다음 할당 전까지 값 유지
 - reg : 실제 플립플롭으로 구현되는 물리적인 저장장소
 - integer : 정수 표현, 실제 하드웨어적인 레지스터가 아님, 합성 툴에서만 참조됨
 - 기본적으로 32-bit signed 값

Verilog 문법 – 자료형 (2)

- Parameter

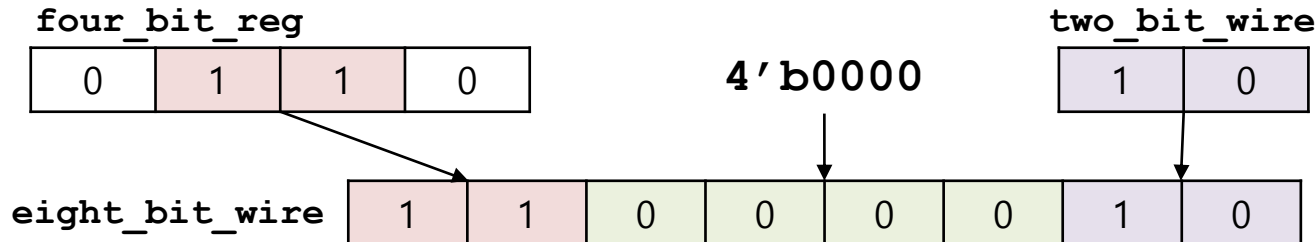
- Net이나 register는 아니며, 상수값 표현
- **parameter** msb = 7;

- Multi-bit wires 와 registers

- **wire** [3:0] four_bit_wire;
- **reg** [31:0] integer_register; // 32-bit register

- Bit 분리(separation) 및 연결(concatenation)

- **wire** [7:0] eight_bit_wire;
- eight_bit_wire = {four_bit_reg[2:1], 4'b0000, two_bit_wire};



구조적 모델링(Structural Modeling)

- 모듈간 연결(interconnect)의 집합(set)으로 회로 기술
 - 기본 게이트: and, nand, or, nor, xor, xnor, not, etc.

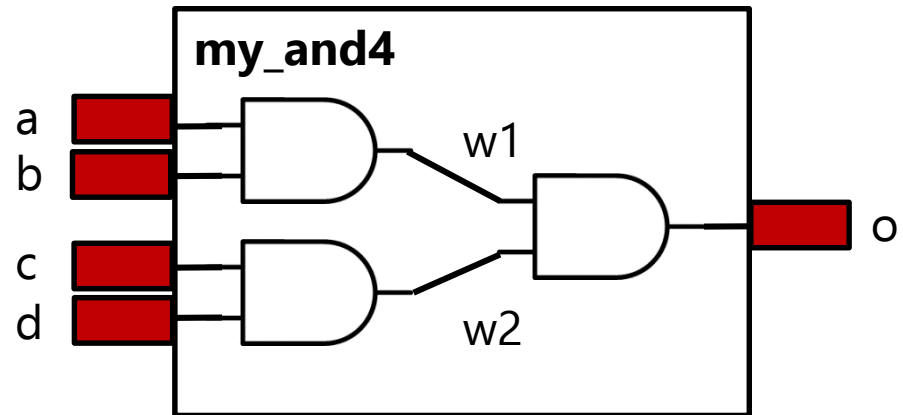
모듈 포트 리스트에 함께 선언 가능

```
module my_and4(a, b, c, d, o);  
    input a, b, c, d;  
    output o;
```



```
module my_and4(input x,  
               input y,  
               input z,  
               output f);
```

```
    wire w1, w2;  
  
    // Structural modeling  
    and g1(w1, a, b);  
    and g2(w2, c, d);  
    and g3(o, w1, w2);  
endmodule
```



구조적 모델링(Structural Modeling) - assign

- 한 와이어를 다른 와이어 혹은 레지스터와 이어줄 때, assign 사용
- 한번 assign된 와이어의 연결을 다른 것으로 바꿀 수 없다.

```
module comp_2bit(a, b, a_eq_b, a_concat_b);  
    input [1:0] a, b;  
    output a_eq_b;  
    output [3:0] a_concat_b  
  
    assign a_eq_b = (a == b);  
    assign a_concat_b = { a[1:0], b[1:0] };  
endmodule
```

Logical operators

Symbol	Operation	Example (a=5, b=10)
<	Less than	(a < b) // 1
==	Equality	(a == b) // 0
&&	Logical and	((a < b) && (a == 5)) // 1

동작적 모델링(Behavioral Modeling) - always

- 회로의 동작을 표현
- 조건이 만족될 때마다 블록 내 문장 수행

```
reg a_and_b;  
always @ (a, b)  
begin  
    a_and_b = a & b;  
end
```

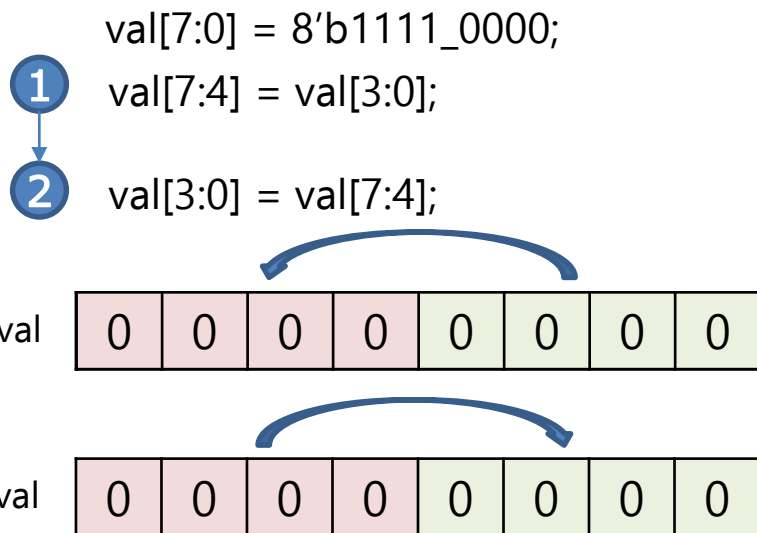
Sensitivity list

a 또는 b가 바뀔 때 마다 실행

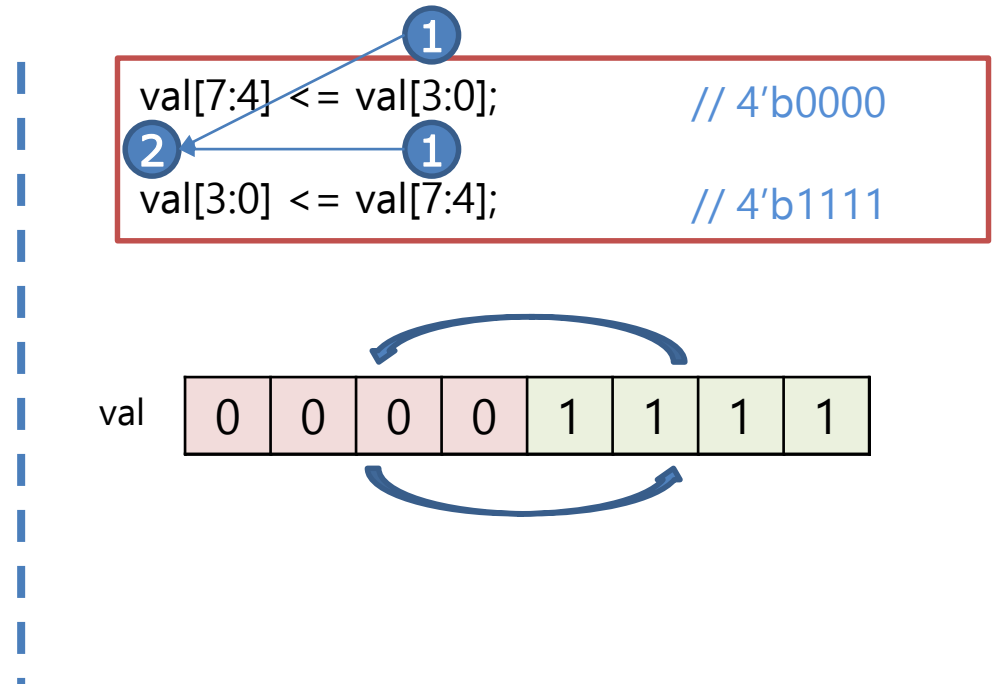
- 블록 내 모든 우변에 존재하는 net 또는 register가 변화할 때마다 수행하고 싶은 경우 (*) 사용가능
- 좌변에 할당되는 변수는 반드시 reg 타입이어야 함
- 실제 하드웨어에서 문장이 순차적으로 실행되는 것이 아님
 - 실행 결과가 같도록 합성 툴이 합성

블록킹 할당문과 비블록킹 할당문

- 블록킹(blocking) 할당문(=)블록 내의 할당문을 순서대로 수행
- 비블록킹(nonblocking) 할당문(<=) : 순서 관계없이 할당문 병렬 수행
 - 블록 내 우변이 모두 평가된 후 할당됨
- 상위 4 bit, 하위 4 bit swap



※ 임시변수를 활용하여 swap 가능



Verilog 문법 – 조건문

- If-else 문

```
if (in == 1)
begin
    out <= 0;
end
else
begin
    out <= 1;
end
```

- Case 문

```
case (sel[1:0])
    2'b00 : out <= 4'b0000;
    2'b01 : out <= 4'b0001;
    2'b11 :
begin
    out <= 4'b1111;
end
    default : out <= 4'b0101;
endcase
```

Verilog 문법 – 반복문

- For 문

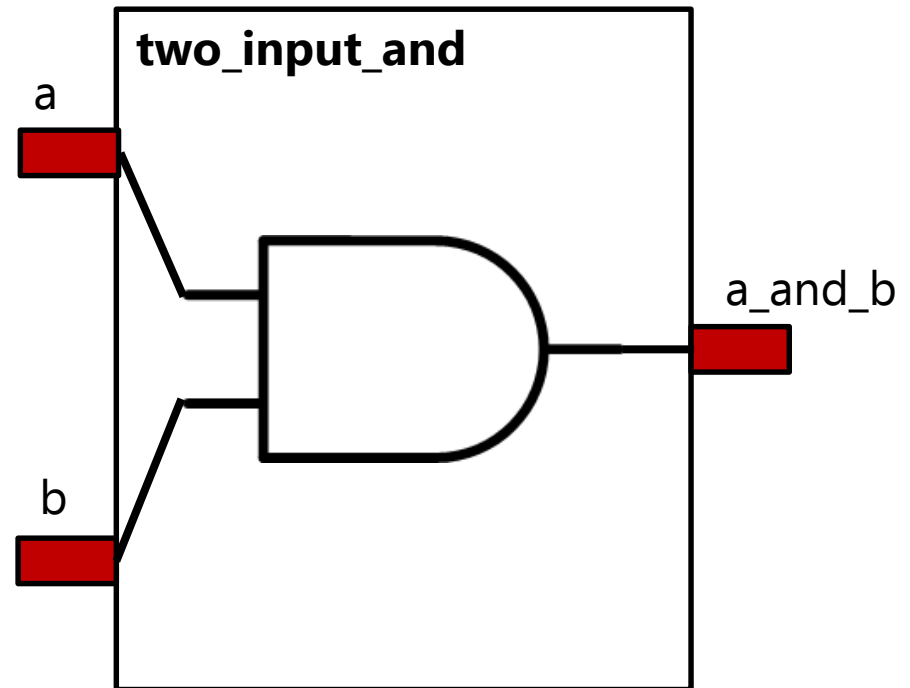
```
integer i;  
for (i = 0; i < 10; i = i + 1)  
begin  
    vector[i] <= 1'b0;  
end
```



```
vector[9:0] <= 10'd0
```

예제 1. 2-input AND 게이트

```
module two_input_and(  
    input a,  
    input b,  
    output reg a_and_b);  
  
    always @ (*)  
    begin  
        a_and_b = a & b;  
    end  
endmodule
```



예제 2. 2-input NAND 게이트

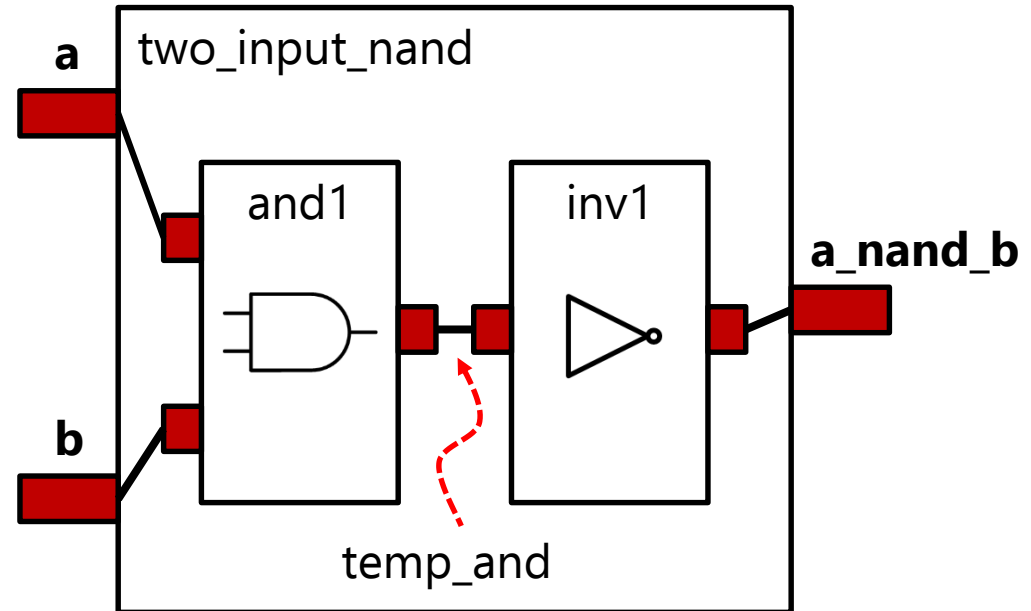
```
module two_input_nand(  
    input a,  
    input b,  
    output a_nand_b);
```

```
    wire temp_and;
```

```
    two_input_and and1(a, b, temp_and);
```

```
    not inv1(a_nand_b, temp_and);
```

```
endmodule
```



예제 3. Half Adder – 구조적/동작적 모델링 비교

```
module half_adder(  
    input x,  
    input y,  
    output s,  
    output c);  
  
xor xor1(s, x, y);  
and and1(c, x, y);  
  
endmodule
```

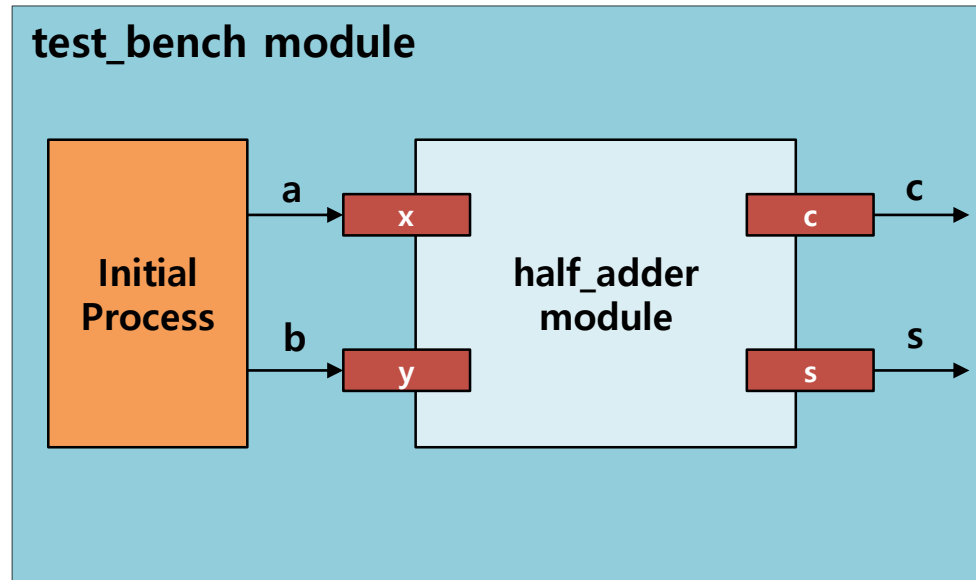
<Structural>

```
module half_adder(  
    input x,  
    input y,  
    output reg s,  
    output reg c);  
  
always @ (*)  
begin  
    s = x ^ y;  
    c = x & y;  
  
end  
endmodule
```

<Behavioral>

테스트 벤치

- Verilog에서 시뮬레이션을 위한 특수목적의 모듈
- Initial 블록은 처음 단 한번만 수행되며 모듈에 입력되는 초기 신호들을 설정함
- 여러 개의 initial 문이 있으면 독립적으로 수행됨



예제 4. 반가산기를 위한 테스트벤치

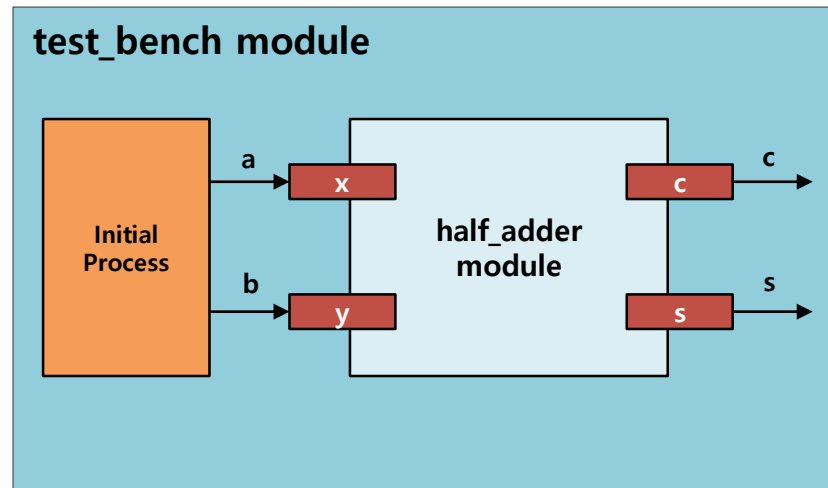
```
`timescale 1ns / 1ps
module half_adder_test;
reg a, b;
wire sum, carry;
```

명명된 포트 매핑

```
half_adder uut (.a(a), .b(b), .sum(sum), .carry(carry));
initial
begin
```

```
    a = 0; b = 0;
    #100;
    a = 0; b = 1;
    #100;
    a = 1; b = 0;
    #100;
    a = 1; b = 1;
```

```
end
endmodule
```

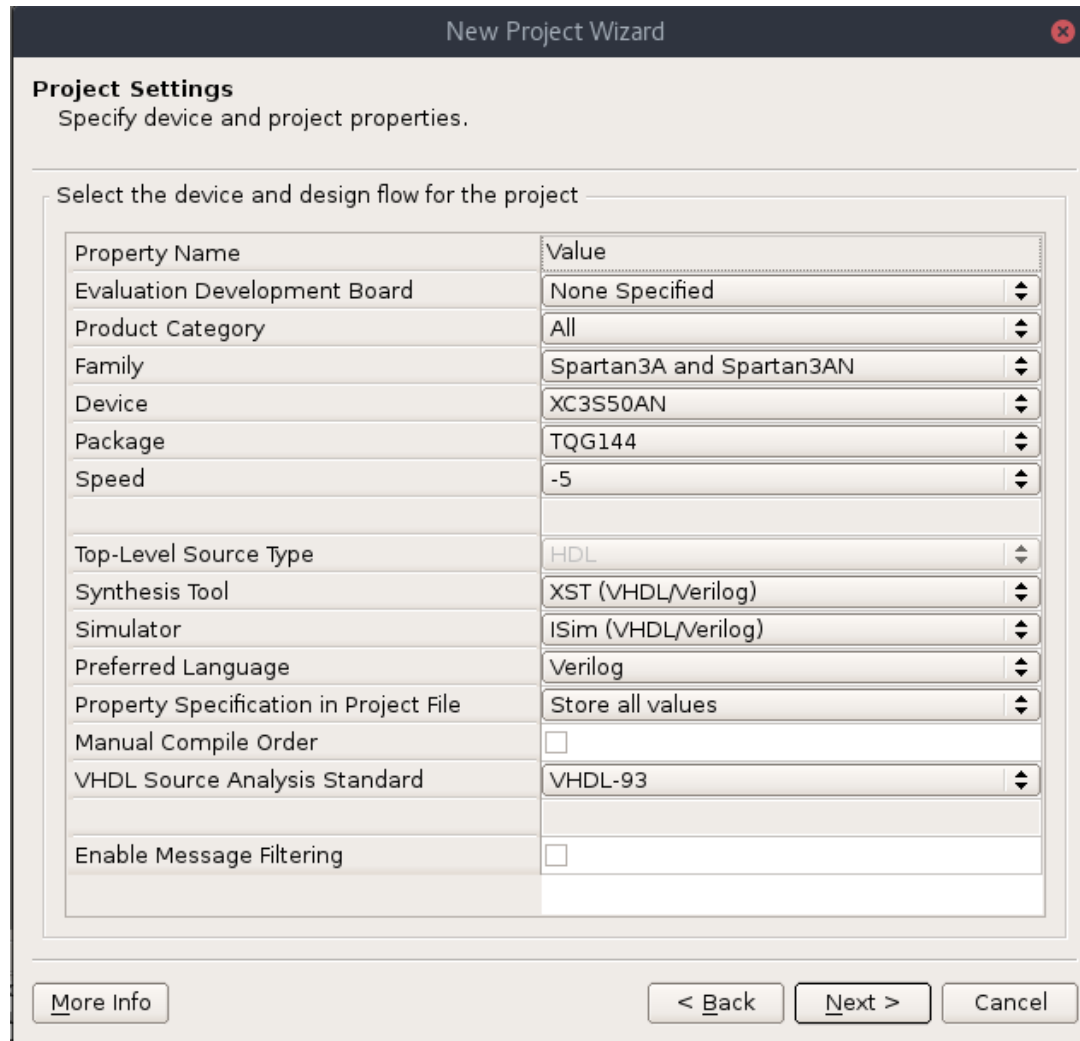


Xilinx ISE 실습 – Half Adder

- 목표
 - Xilinx ISE에서 Verilog를 이용한 모듈 생성 방법 습득
 - Verilog를 사용하여 half adder 구현 및 테스트
- 실험 내용
 - 프로젝트 생성 후, Verilog 파일 생성
 - 주어진 코드를 입력하여 half adder 구현
 - 구현 후 test fixture를 통해 결과값 확인
- 제출 사항 없음

Xilinx ISE 새 프로젝트 생성

- 새 프로젝트 생성 클릭 후 아래와 같이 설정



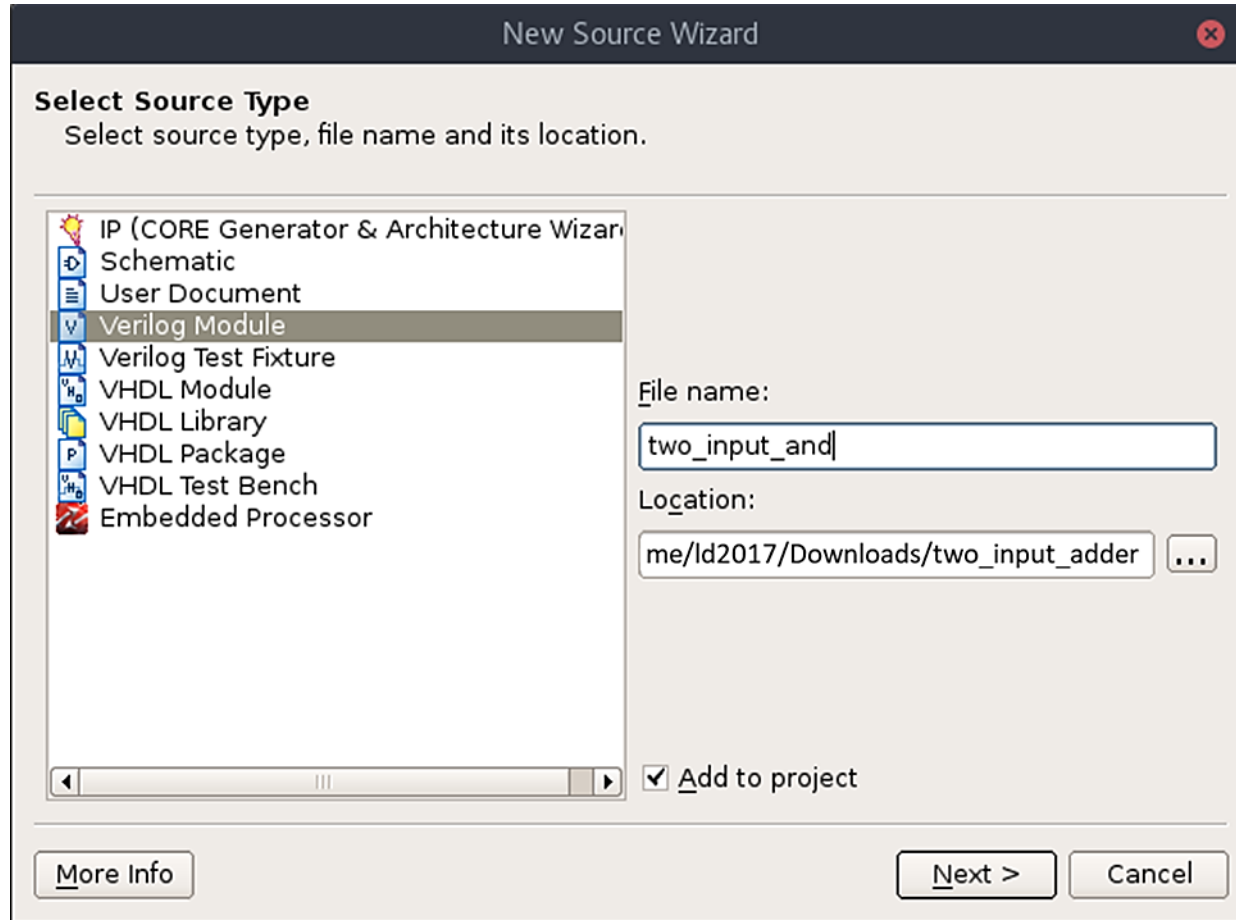
The image shows the 'New Project Wizard' dialog box in Xilinx ISE. The title bar is 'New Project Wizard'. The main section is 'Project Settings' with the subtitle 'Specify device and project properties.' Below this is a section 'Select the device and design flow for the project' containing a table of properties.

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3A and Spartan3AN
Device	XC3S50AN
Package	TQG144
Speed	-5
Top-Level Source Type	HDL
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
Enable Message Filtering	<input type="checkbox"/>

At the bottom of the dialog, there are four buttons: 'More Info', '< Back', 'Next >', and 'Cancel'.

Verilog Module 파일 추가

- “Verilog Module” 선택 후 파일명 입력



모듈 정의

- 포트 이름 및 입출력 방향 설정

The image shows a 'New Source Wizard' dialog box with the title 'Define Module' and the instruction 'Specify ports for module.' The 'Module name' field contains 'two_input_and'. Below this is a table for defining ports.

Port Name	Direction	Bus	MSB	LSB
a	input	<input type="checkbox"/>		
b	input	<input type="checkbox"/>		
a_and_b	output	<input type="checkbox"/>		
	input	<input type="checkbox"/>		
	input	<input type="checkbox"/>		
	input	<input type="checkbox"/>		
	input	<input type="checkbox"/>		
	input	<input type="checkbox"/>		
	input	<input type="checkbox"/>		
	input	<input type="checkbox"/>		
	input	<input type="checkbox"/>		

At the bottom of the dialog are four buttons: 'More Info', '< Back', 'Next >', and 'Cancel'.

Two_input_and 모듈 작성

- 아래와 같이 입력하여 two_input_and 모듈 작성

```
module two_input_and (  
    input a,  
    input b,  
    output reg a_and_b);  
  
    always @ (*)  
    begin  
        a_and_b = a & b;  
    end  
endmodule
```

Two_input_xor 모듈 작성

- Two_input_and 모듈과 같은 방법으로 xor을 위한 모듈 작성

```
module two_input_xor (  
    input a,  
    input b,  
    output reg a_xor_b);  
  
    always @ (*)  
    begin  
        a_xor_b = a ^ b;  
    end  
endmodule
```

반가산기 구현

- 구현한 XOR, AND 게이트 모듈을 이용하여 반가산기 구현

```
module half_adder (  
    input a,  
    input b,  
    output sum,  
    output carry);
```

```
    two_input_and and1(a, b, carry);  
    two_input_xor xor1(a, b, sum);
```

```
endmodule
```


Test Fixture 코드

- 다음과 같이 Test Fixture 코드 작성

```
module half_adder_test;
    // Inputs
    reg a;
    reg b;

    // Outputs
    wire sum;
    wire carry;

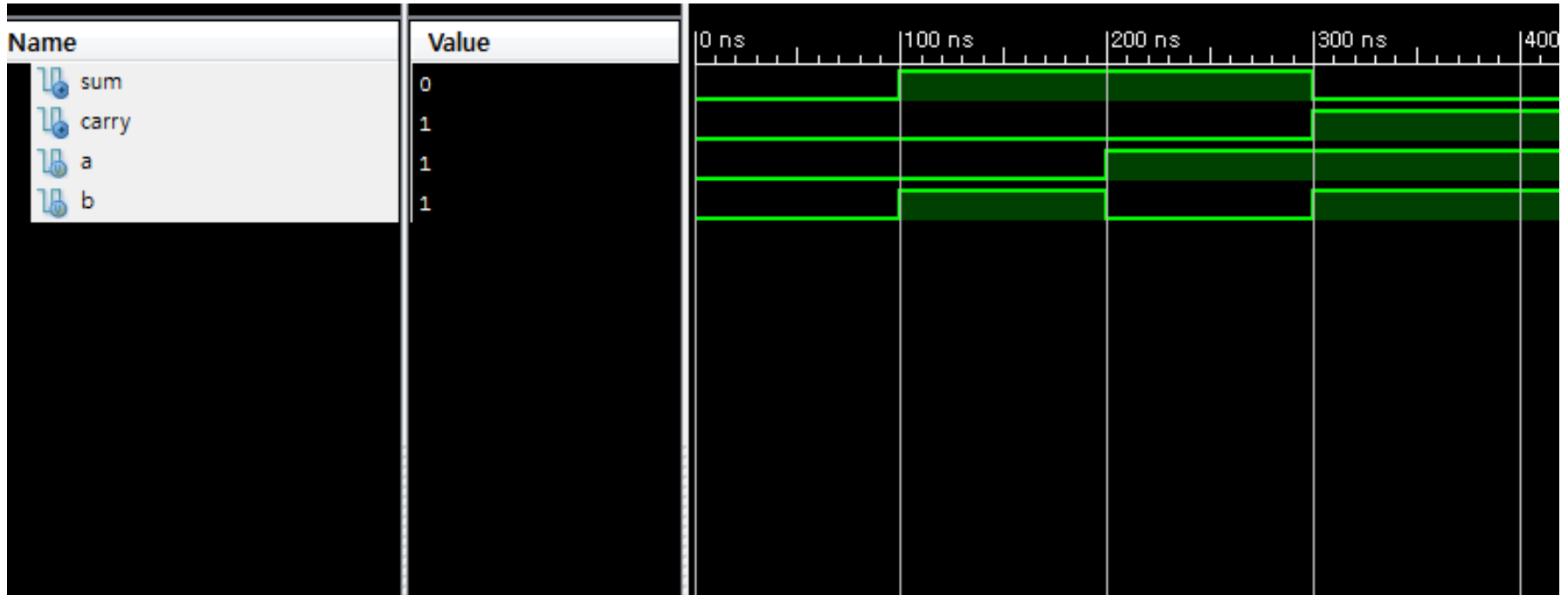
    // Instantiate the Unit
    Under Test (UUT)
    half_adder uut (
        .a(a),
        .b(b),
        .sum(sum),
        .carry(carry)
    );
```

```
initial
begin
    // Initialize Inputs
    a = 0;
    b = 0;
    #100;
    a = 0;
    b = 1;
    #100;
    a = 1;
    b = 0;
    #100;
    a = 1;
    b = 1;

end
endmodule
```

시뮬레이션 확인

- 반가산기 입력에 따라 sum, carry 결과 올바른지 확인



실험 1 – 8-bit Carry Select Adder 구현

- 목표
 - 주어진 4-bit adder를 이용하여 8-bit carry select adder 구현
- 실험 내용
 - 주어진 코드를 이용해 4-bit adder 구현
 - 4-bit adder를 이용하여 8-bit carry select adder 구현
 - test fixture를 통해 결과 값 확인
- 제출 사항
 - 실습지 참고

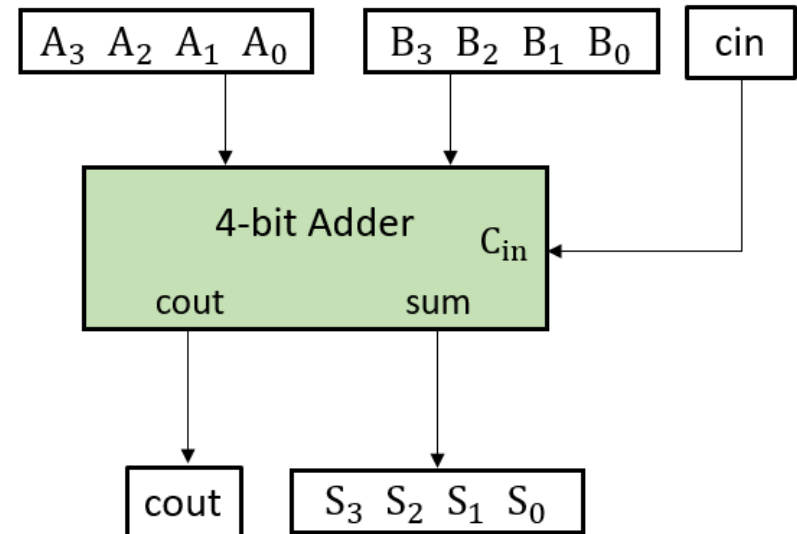
실험 1 – 8-bit Carry Select Adder 구현 (cont'd)

- 아래 주어진 코드를 이용해 4-bit adder를 구현한 후, 이를 이용하여 다음 슬라이드의 8-bit carry select adder 구현

- 4-bit adder 코드

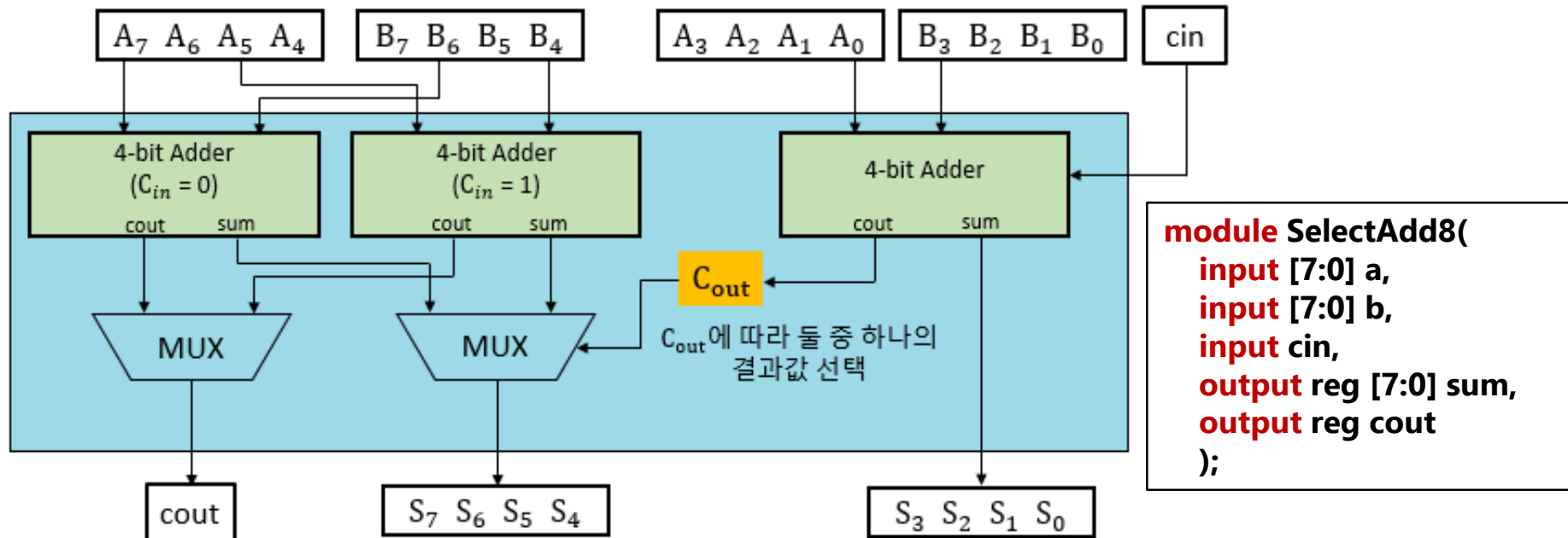
```
module add4(  
  input [3:0] a,  
  input [3:0] b,  
  input cin,  
  output [3:0] sum,  
  output cout  
);  
  
assign {cout, sum} = a + b + cin;  
  
endmodule
```

- 4-bit adder 구조



실험 1 – 8-bit Carry Select Adder 구현 (cont'd)

- 8-bit carry select adder를 구현하고 시뮬레이션을 통해 결과를 확인하라. 실습지의 지시사항에 따라 프로젝트와 스크린샷을 제출
- 8-bit carry Select Adder 구조 및 인터페이스



과제 3

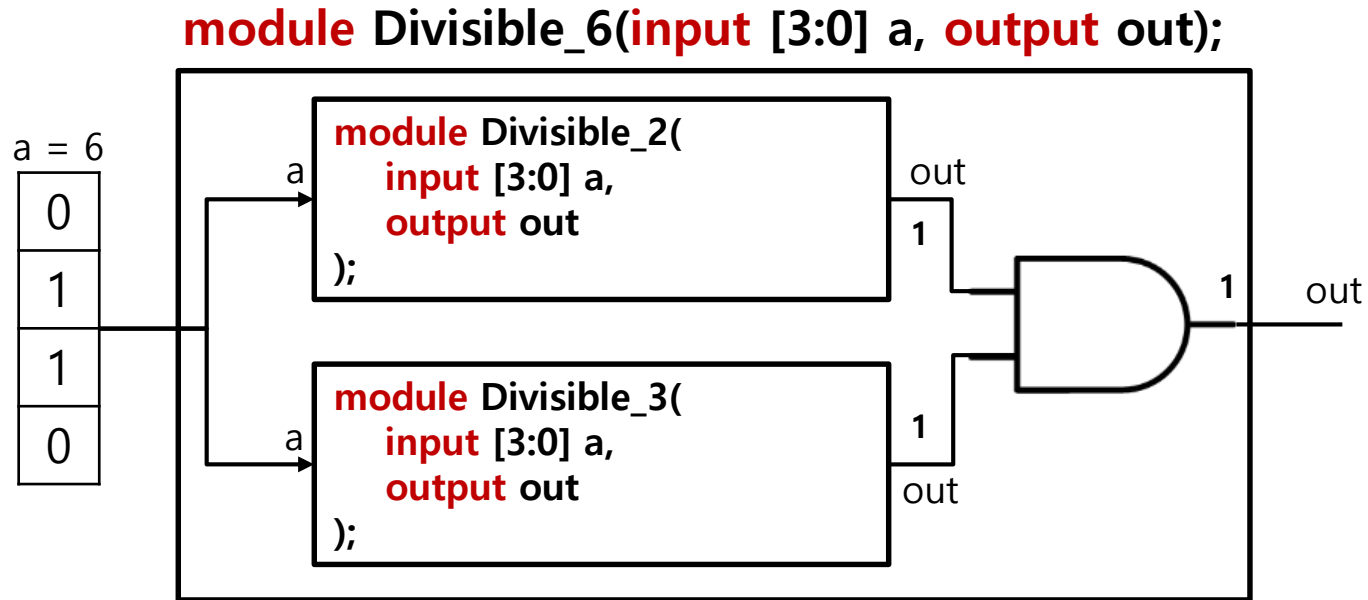
- 목표
 - 3-1. Divisible_6 구현
 - 3-2. Block Cipher 구현
- 실험 내용
 - 과제 세부 설명 참고
- 제출 사항
 - 과제 세부 설명 및 문제지 참고
- 과제는 개인 별 채점이므로 같은 팀원 끼리 공유 금지

과제 3-1 – Divisible_6

- 목표
 - 자기 자신의 수로 나누어 떨어지는 수를 판별하는 모듈 구현
 - **Divisible_6** 모듈 구현
- 실험 내용
 - **Divisible_2, Divisible_3** 모듈 구현 (각각 2, 3으로 나누어 떨어지면 1 출력)
 - 구현한 두 모듈을 이용하여 **Divisible_6** 모듈 구현
 - 시뮬레이션을 통해 입력1~15의 결과를 확인
- 제출 사항
 - 구현한 모든 Verilog 파일
 - 모든 입력 값의 시뮬레이션 결과 값 스크린샷

과제 3-1 – Divisible_6

- Divisible_6의 구조



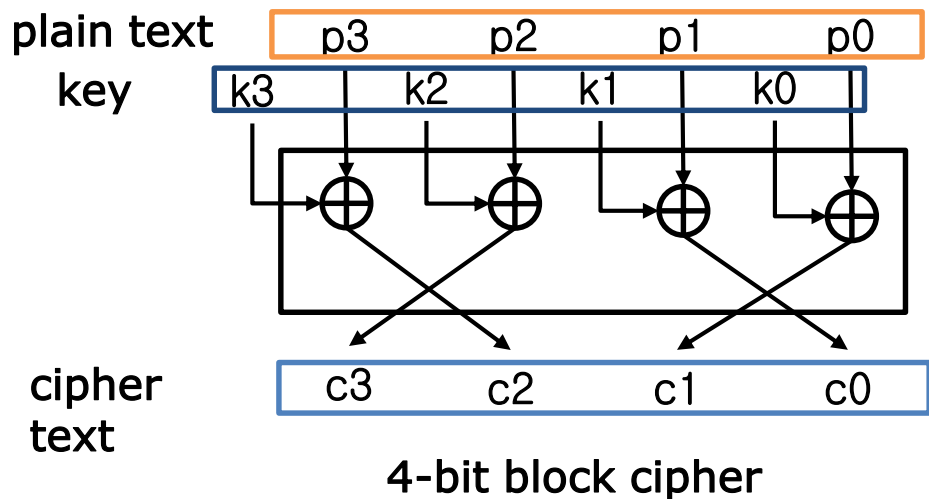
- 모든 가능한 입력(1~15)에 대한 시뮬레이션 결과를 확인하여라.

과제 3-2 – Block Cipher

- 목표
 - BlockCipherCBC, BlockDecryptorCBC 구현
- 실험 내용
 - 주어진 코드와 구조 이미지를 이용해 **BlockCipher4bit** 구현
 - **BlockCipher4bit**을 이용하여 **BlockCipherCBC, BlockDecryptorCBC** 구현
- 제출 사항
 - 구현한 모든 Verilog 파일
 - 문제지에서 요구하는 입력 값의 시뮬레이션 결과 스크린샷

과제 3-2 – Block Cipher

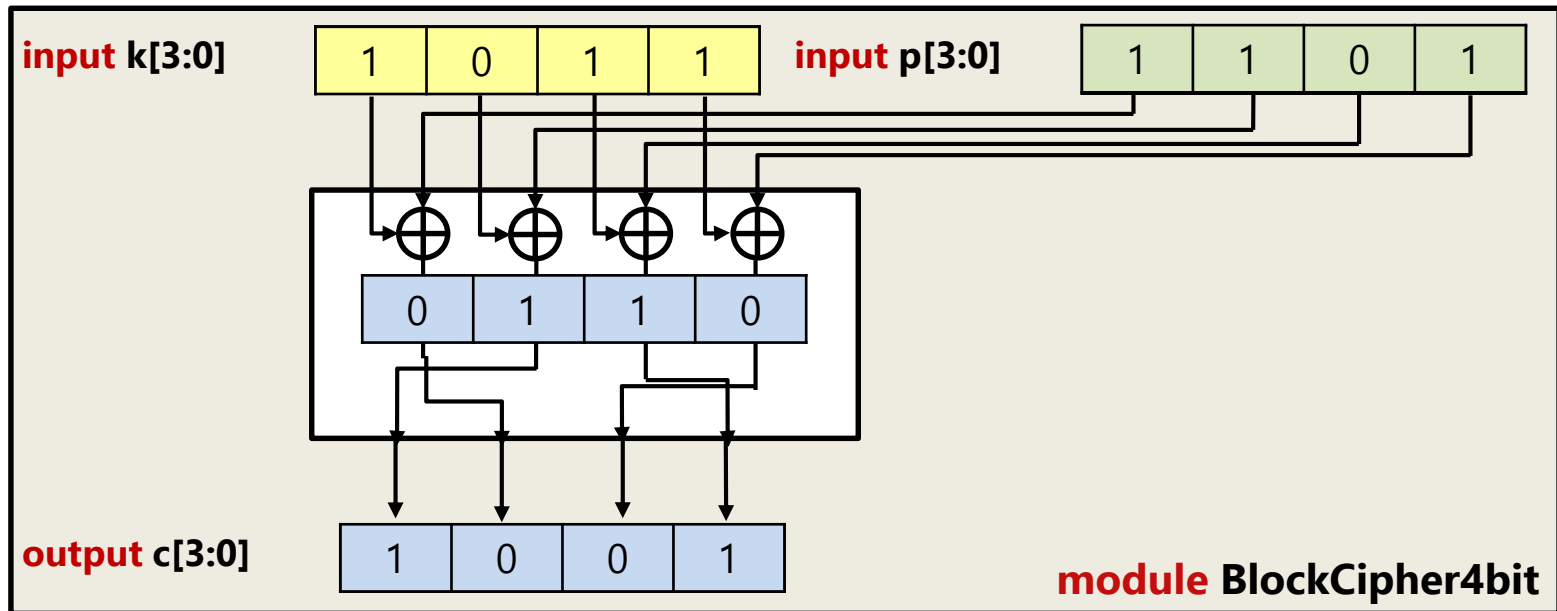
- XOR 연산의 경우, 2회 반복 수행 시 원래대로 돌아오는 특징을 가진다.
- 즉 $(p \text{ xor } k) \text{ xor } k = p$ 가 성립하며, 이러한 특징은 암호화에 활용할 수 있다. 본 실습은 이를 통하여 데이터를 암호화/복호화 하는 8-bit block cipher/decryptor 를 구현한다.
- 먼저, 4-bit을 암호화하는 BlockCipher4bit을 구현한다. BlockCipher4bit은 암호화할 4-bit p와 비밀키 4-bit k를 입력 받아 동일 자리끼리 XOR 연산 후 자리를 바꾸어 암호화된 4-bit c를 출력한다.



```
module BlockCipher4bit(  
  input [3:0] p, // plain text  
  input [3:0] k, // key  
  output [3:0] c // cipher text  
);
```

과제 3-2 – BlockCipher4bit 모듈의 암호화 예

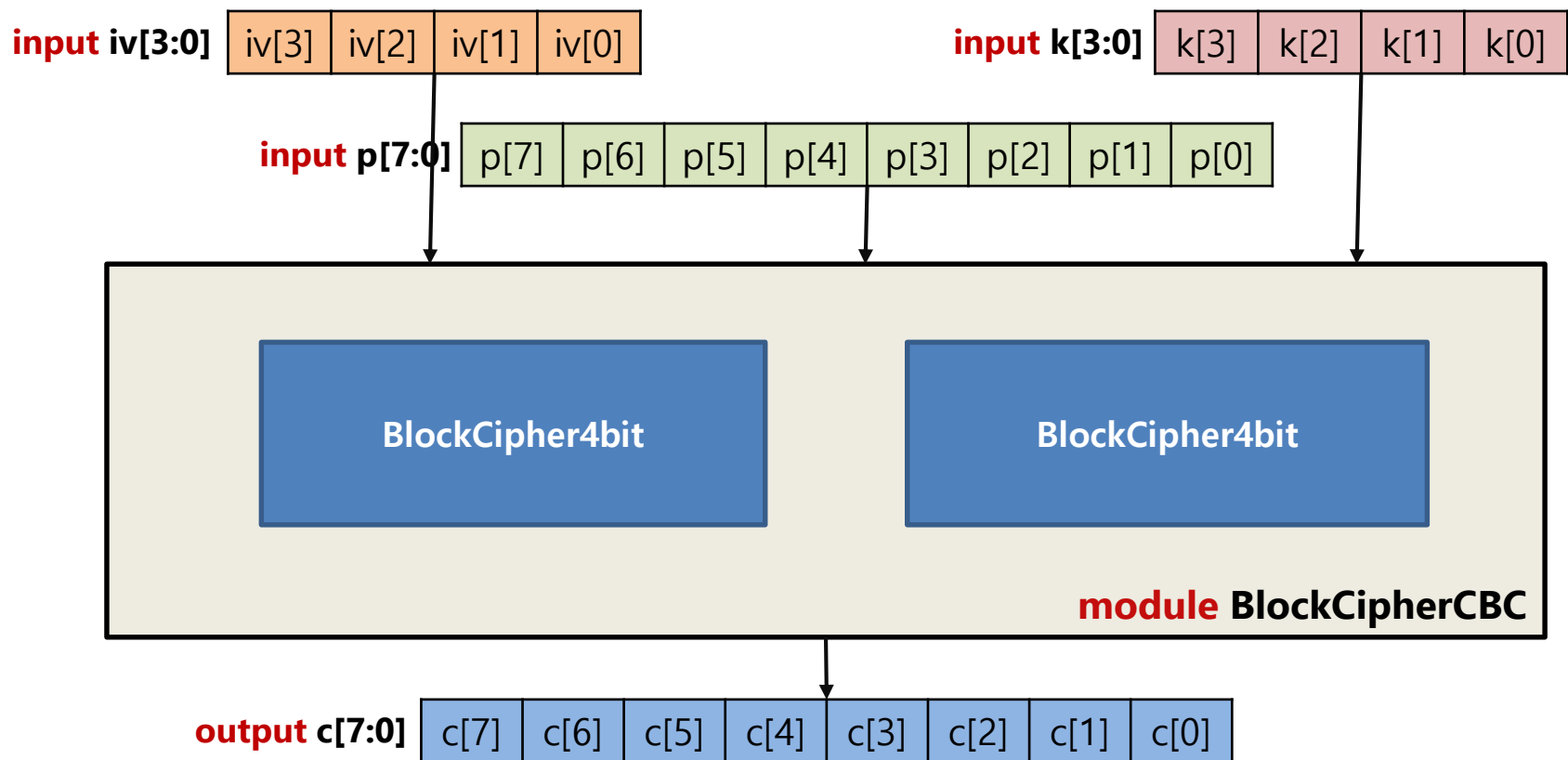
- $k = 4'b1011$, $p = 4'b1000$



```
module BlockCipher4bit(  
  input [3:0] p,  
  input [3:0] k,  
  output [3:0] c  
);
```

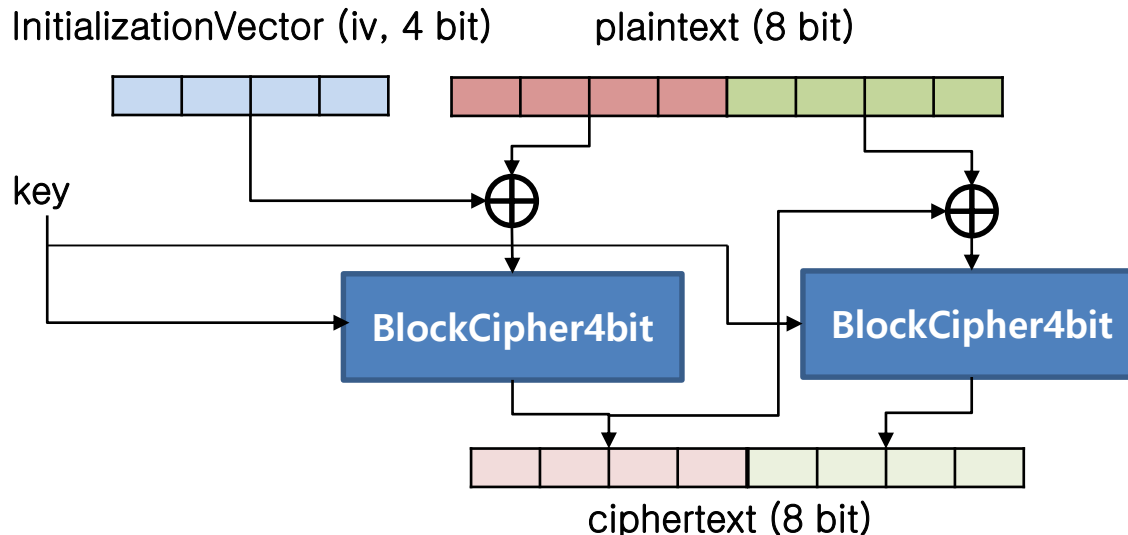
과제 3-2 – BlockCipherCBC

- 앞서 구현한 BlockCipher4bit 모듈 2개를 사용하여, 8-bit p 를 암호화하는 BlockCipherCBC 모듈을 구현한다.



과제 3-2 – BlockCipherCBC 구현

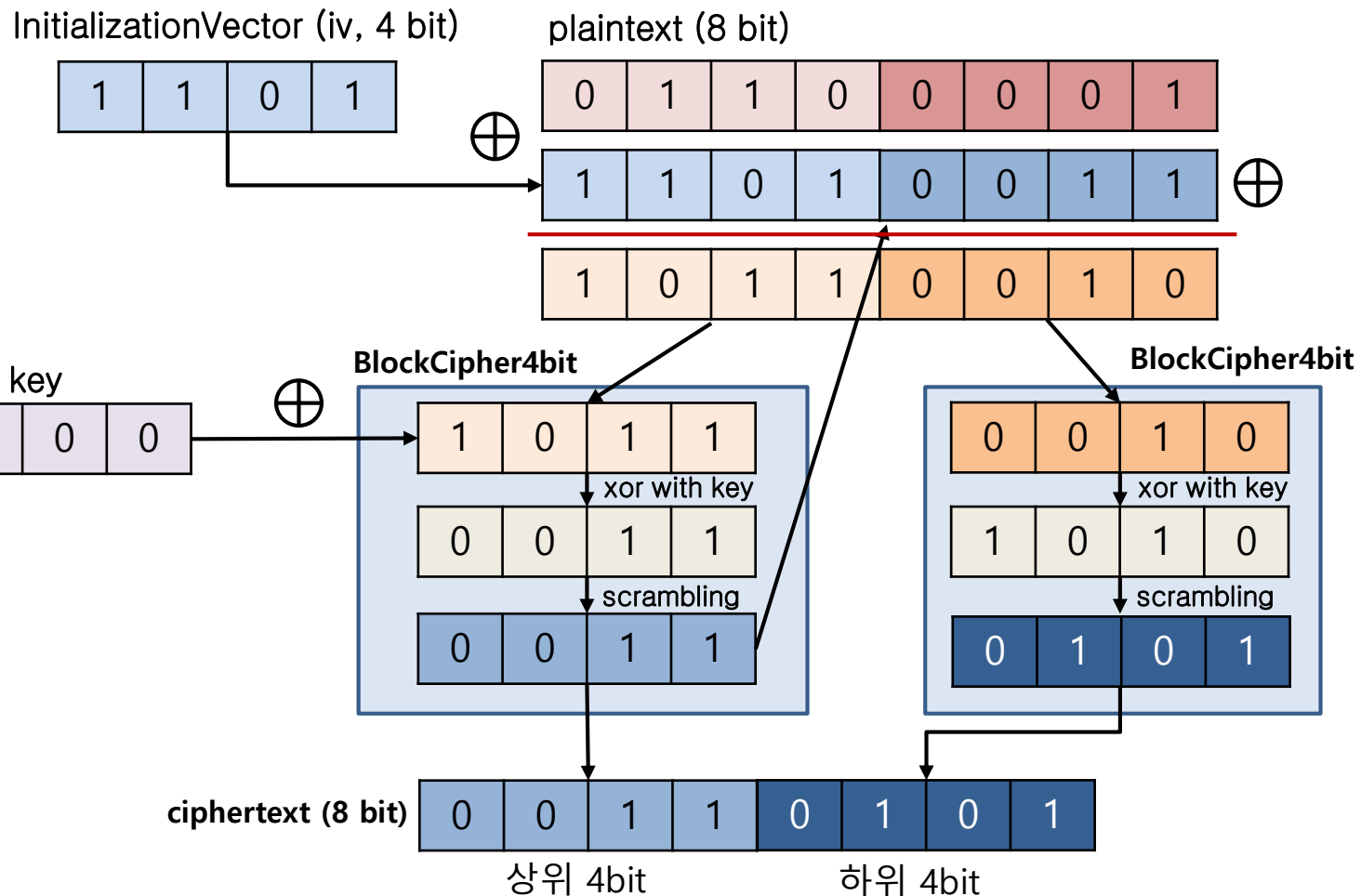
- BlockCipherCBC는 BlockCipher4bit을 사용하여 8-bit을 암호화하기 위한 모듈로, 8-bit p와 4-bit k 그리고 4-bit iv를 입력 받는다.
- Ciphertext의 상위 4-bit은 p의 상위 4-bit와 iv를 XOR 연산 후, BlockCipher4bit으로 암호화한 결과값이 된다.
- Ciphertext의 하위 4-bit은 앞서 암호화한 ciphertext의 상위 4-bit과 p의 하위 4-bit을 XOR 연산 후, BlockCipher4bit으로 암호화한 결과값이 된다.



```
module BlockCipherCBC(  
  input [7:0] p, // plain text  
  input [3:0] k, // key  
  input [3:0] iv, // init vector  
  output [7:0] c // cipher text  
);
```

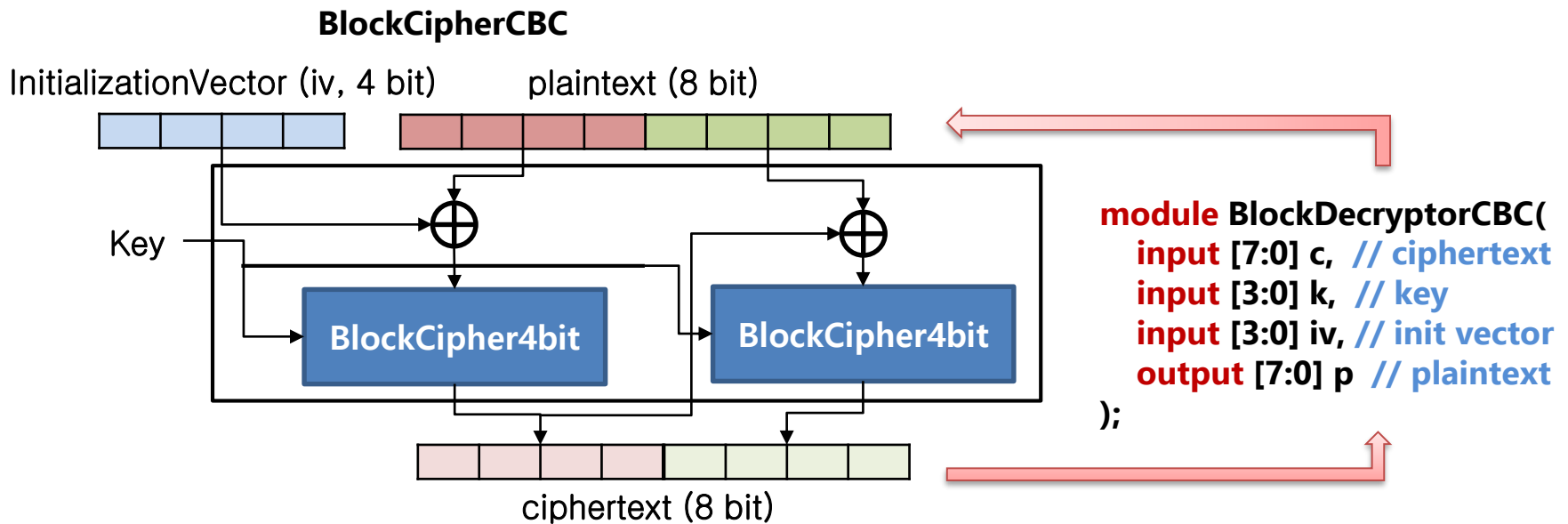
과제 3-2 – BlockCipherCBC 모듈의 암호화 방식

- iv = 4'b1101, key = 4'b1000, plaintext = 8'b01100001 ('a', 97)



과제 3-2 – BlockDecryptorCBC 구현

- 이 암호화가 올바르게 되었는지 확인하기 위해서는 복호화하기 위한 Decryptor 모듈이 필요하며, 복호화는 암호화할 때 수행한 연산을 반대로 수행하면 된다.
- 암호화 모듈과 같은 프로젝트 내에 구현할 것



과제 3-2 – BlockCipherCBC 결과값 예시

- $iv = 8$ (4'b1000), $k = 6$ (4'b0110)
 - $p\ 102$ (8'b01100110) \rightarrow $c\ 72$ (8'b01001000)
 - $p\ 85$ (8'b01010101) \rightarrow $c\ 120$ (8'b01111000)
- $iv = 10$ (4'b1010), $k = 2$ (4'b0010)
 - $P\ 64$ (8'b01000000) \rightarrow $c\ 205$ (8'b11001101)
 - $P\ 108$ (8'b01101100) \rightarrow $c\ 211$ (8'b11010011)
- BlockDecryptorCBC를 통해 복호화했을 때, 원래 P값이 나오는지 확인.
- 채점 시 위 case 외의 경우도 계산하여 점수에 반영함.

실험 및 과제 제출 안내

- 제출 사항
 - 실험은 실습지, 과제는 문제지 참고
- 제출 방법 및 기한
 - 실험
 - ETL 과제 게시판에 팀별로 제출
 - 일요일 오후 6시 전까지
 - 과제
 - ETL 과제 게시판에 개인별로 제출
 - 다음 주 수요일(10/24) 실습시간 전까지