



4. Verilog 문법

논리회로실습

부경대 컴퓨터·인공지능공학부 최필주

- 하드웨어 설계 개요
- Verilog 문법
 - Module
 - 연산자 – bitwise logical operator
 - 수의 표현, 변수 선언과 자료형



하드웨어 설계 개요

비교) 소프트웨어 설계 과정

- 변환 및 실행

- C 등의 high-level language는 기계어로 변환됨
 - 기계어: 단순한 동작(산술/논리 연산, 데이터 이동, 흐름 제어)만 수행
- 기계어는 순차적으로 하나씩 실행 ➔ 순서도로 표현 가능



프로그래밍 언어



기계어

하드웨어 설계 과정

- 변환 및 구현 과정

- Hardware description language(HDL)은 논리소자로 변환됨
 - 조합논리소자: not, and, or, MUX, ...
 - 순서논리소자: latch, FF
- 논리소자는 layout되어 칩으로 구현됨 ➔ 블록 다이어그램
 - Layout: 논리 소자를 배치(place)하고 연결(route)하는 과정
 - 논리소자의 동작은 병렬로 이루어짐

소프트웨어 설계 vs. 하드웨어 설계

| | 소프트웨어 설계 | 하드웨어 설계 |
|--------------|---|--|
| 설계자 사용 언어 | High-level language (C, C++, Python, Java, ...) | HDL (Verilog, VHDL) Register transfer level (RTL) netlist |
| 변환 과정 | compile, interpret (= target processor의 기계어로 변환 + 최적화) | 합성(synthesis) (= translation + optimization + mapping) |
| 변환 후 | 기계어 | 논리소자 Gate level netlist |
| 실행/동작 | 기계어는 프로세서가 하나씩 읽혀서 순차적으로 실행됨 | 논리소자는 배치 및 연결되어 동시에 병렬로 동작함 |
| 설계 단위 | 함수 | 모듈 |

소프트웨어 설계 vs. 하드웨어 설계

- 예: if문

SW설계(C) & HW설계(Verilog)

```
if(a == b) dout = din0;  
else      dout = din1;
```

소프트웨어 설계 vs. 하드웨어 설계

- 예: if문

SW설계(C) & HW설계(Verilog)

```
if(a == b) dout = din0;  
else      dout = din1;
```

compile

```
lw $3,24($fp)    // $3 ← a  
lw $2,28($fp)    // $2 ← b  
bne $3, $2, $L2  // if(a==b)
```

```
lw $3,32($fp)    // $3 ← din0  
sw $3,8($fp)     // dout ← $3  
b $L3
```

```
$L2: lw $3,36($fp) // $3 ← din1  
     sw $3,8($fp)  // dout ← $3
```

```
$L3: ...
```

SW 설계(MIPS)

소프트웨어 설계 vs. 하드웨어 설계

- 예: if문

SW설계(C) & HW설계(Verilog)

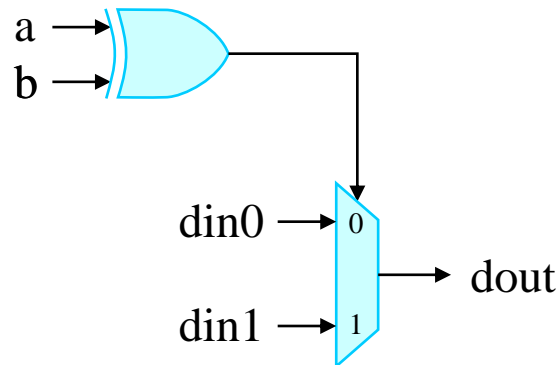
```
if(a == b) dout = din0;  
else      dout = din1;
```

compile


translate

```
lw $3,24($fp)    // $3 ← a  
lw $2,28($fp)    // $2 ← b  
bne $3, $2, $L2  // if(a==b)  
lw $3,32($fp)    // $3 ← din0  
sw $3,8($fp)     // dout ← $3  
b $L3  
$L2: lw $3,36($fp) // $3 ← din1  
     sw $3,8($fp)  // dout ← $3  
$L3: ...
```

SW 설계(MIPS)



HW 설계



Verilog - Module

Module 개요

- 하드웨어 설계 시 기본적인 설계 단위
 - 포함 내용: 입출력 + 입출력과 관련된 동작
- Module의 정의
 - module로 시작
 - endmodule로 끝남
 - `modular_name`
 - `interface`: 모듈의 입출력
 - `body`: 모듈의 내부 구성/연결 정의
 - 모듈 내에 모듈 재정의 불가능

```
module module_name (port_list)
```

```
port declarations
```

```
parameter declarations
```

```
variable declarations
```

```
assignments
```

```
lower-level module instantiation
```

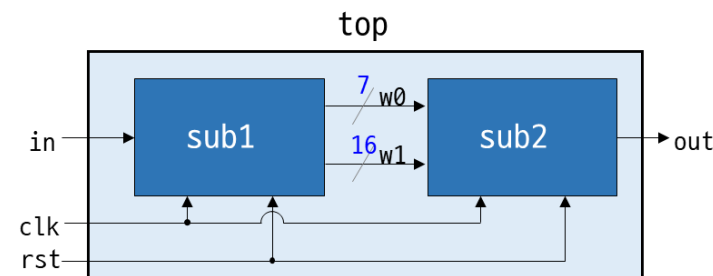
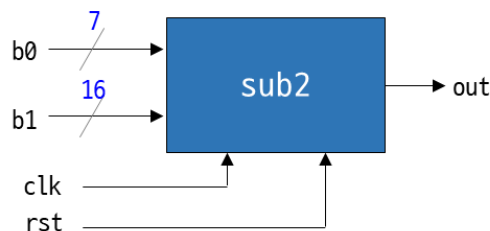
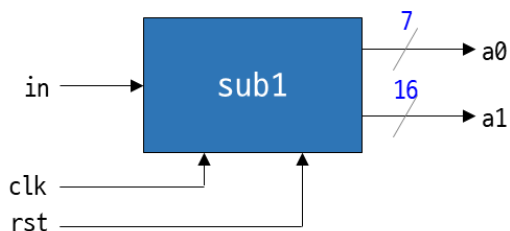
```
initial and always blocks
```

```
tasks and function
```

```
endmodule
```

Module을 이용한 구조적 모델링

- 하위 모듈을 포함(인스턴스화)하여 설계
 - 상위 모듈은 하위 모듈의 내부 설계에 관여 X
 - 입출력과 어떤 동작을 한다만 알고 사용



/* sub1 모듈의 Verilog 표현 */

```
module sub1 (clk, rst, in, a0, b1);
```

```
    input    clk, rst;
    input    in;
    output   [6:0] a0;
    output   [15:0] a1;
```

.....

회로 기능(sub1) 표현

.....

```
endmodule
```

/* sub2 모듈의 Verilog 표현 */

```
module sub2 (clk, rst, b0, b1, out);
```

```
    input    clk, rst;
    input    [6:0] b0;
    input    [15:0] b1;
    output   out;
```

.....

회로 기능(sub2) 표현

.....

```
endmodule
```

/* top 모듈의 Verilog 표현 */

```
module top (clk, rst, in, out);
```

```
    input    clk, rst;
    input    in;
    output   out;
    wire     [6:0] w0;
    wire     [15:0] w1;
```

.....

```
    sub1 U1 (clk, rst, in, w0, w1); // sub1 인스턴스화
    sub2 U2 (clk, rst, w0, w1, out); // sub2 인스턴스화
```

.....

```
endmodule
```



Verilog – bitwise/logical operator

- Verilog에서의 논리게이트의 표현

- AND, OR, NOT, XOR을 &, |, ~, ^로 표현하고 이를 조합하여 사용
- 논리게이트의 표현 예시

| 논리 게이트 | 예 |
|--------|--|
| AND | $c = a \& b$ |
| OR | $c = a b$ |
| NOT | $c = \sim a$ |
| XOR | $c = a \wedge b$ |
| NAND | $c = \sim(a \& b)$ |
| NOR | $c = \sim(a b)$ |
| XNOR | $c = a \wedge \sim b$ $c = a \sim \wedge b$ |

- Verilog에서의 논리게이트의 표현

- AND, OR, NOT, XOR을 &, |, ~, ^로 표현하고 이를 조합하여 사용
- 논리게이트의 표현 예시

| 논리 게이트 | 예 |
|--------|--|
| AND | $c = a \& b$ |
| OR | $c = a b$ |
| NOT | $c = \sim a$ |
| XOR | $c = a \wedge b$ |
| NAND | $c = \sim(a \& b)$ |
| NOR | $c = \sim(a b)$ |
| XNOR | $c = a \wedge \sim b$ $c = a \sim \wedge b$ |

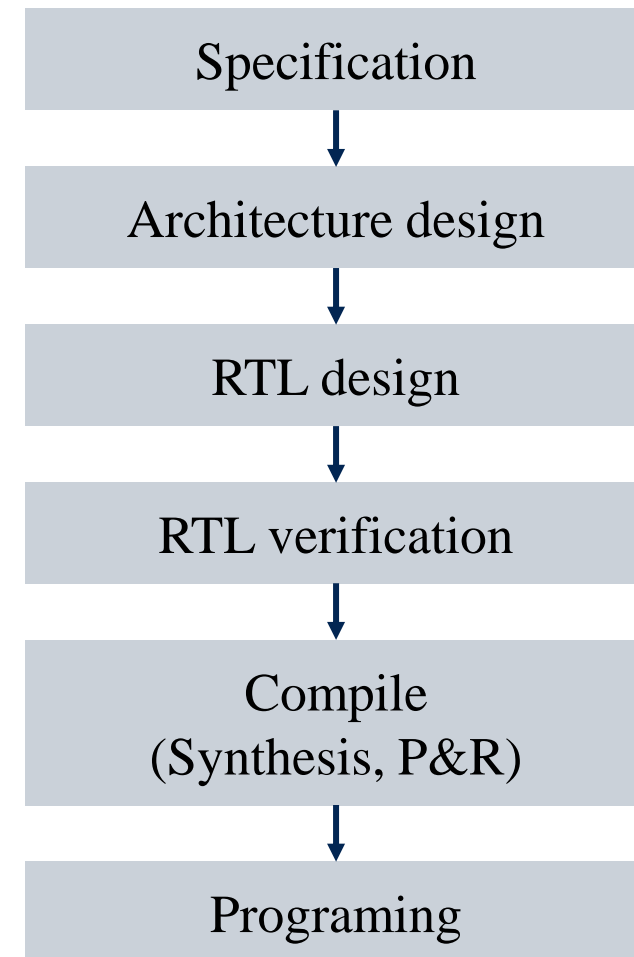
※ 값이 할당(정의)되는 신호들은 자료형을 미리 선언해야 함

- net: 소자간 물리적 연결
- reg: 값을 임시 저장할 수 있는 변수
- parameter: 상수로 선언되는 데이터

➔ 자세한 건 추후 설명 예정

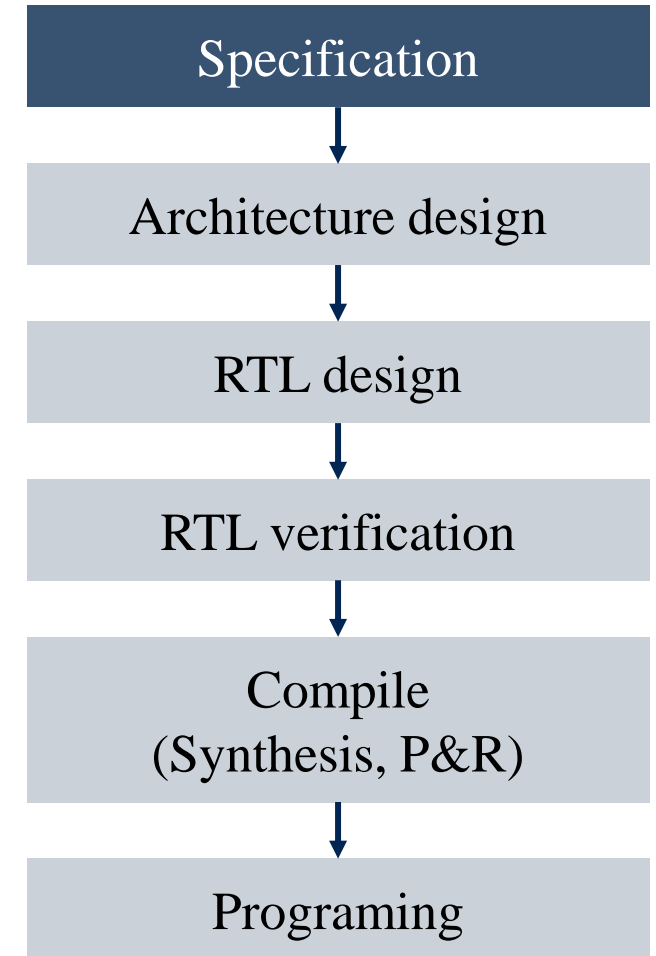
- 이러한 연산자들을 bitwise logical operator라고 부름

- 수행하려는 기능
 - 두 비트를 각각 AND, OR, NOT, XOR

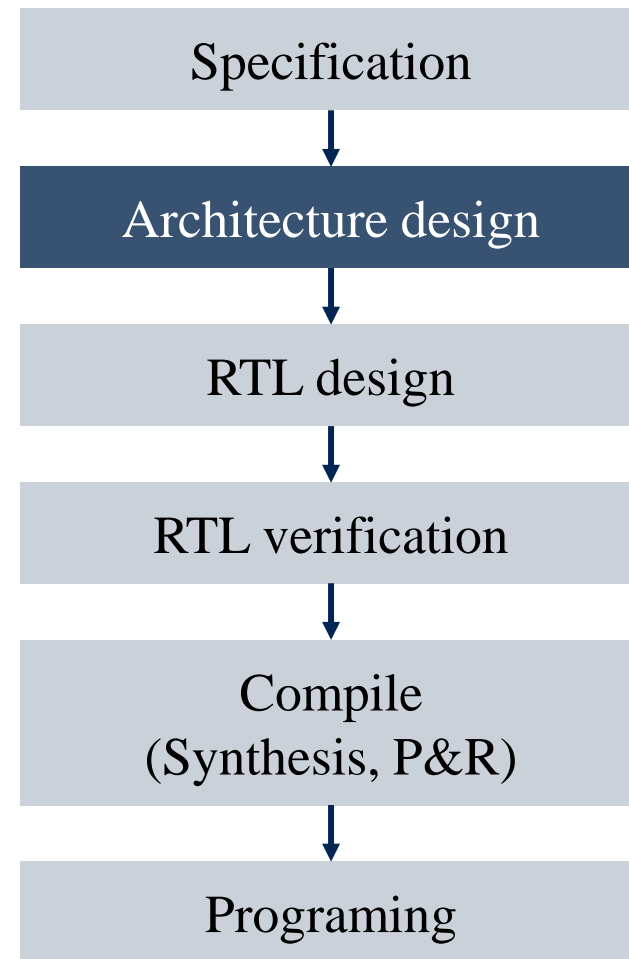
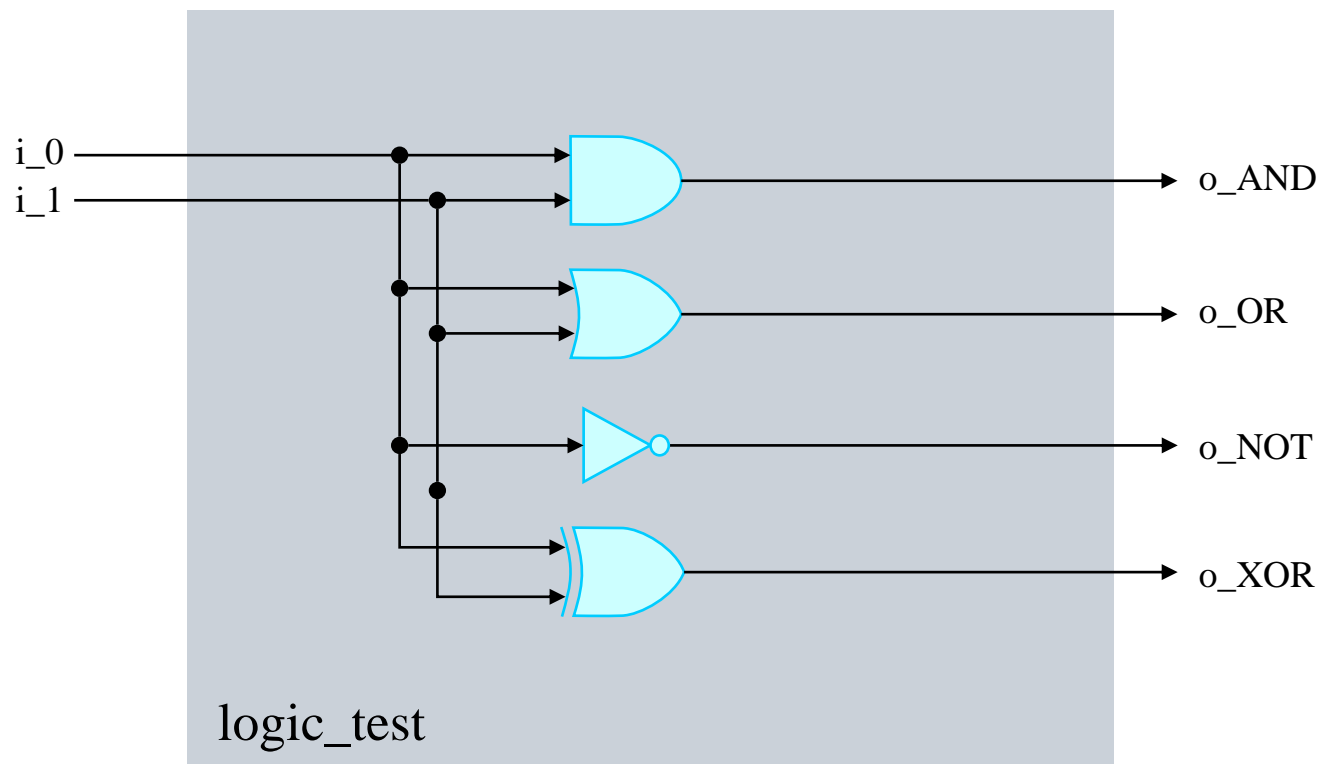


● 명세

- 입력: 1비트 입력 x 2 ➔ SW
- 출력: 1비트 출력 x 4 ➔ LED
- 수행 기능
 - 두 입력을 각각 AND, OR, NOT, XOR하여 출력
- 모듈명: logic_test



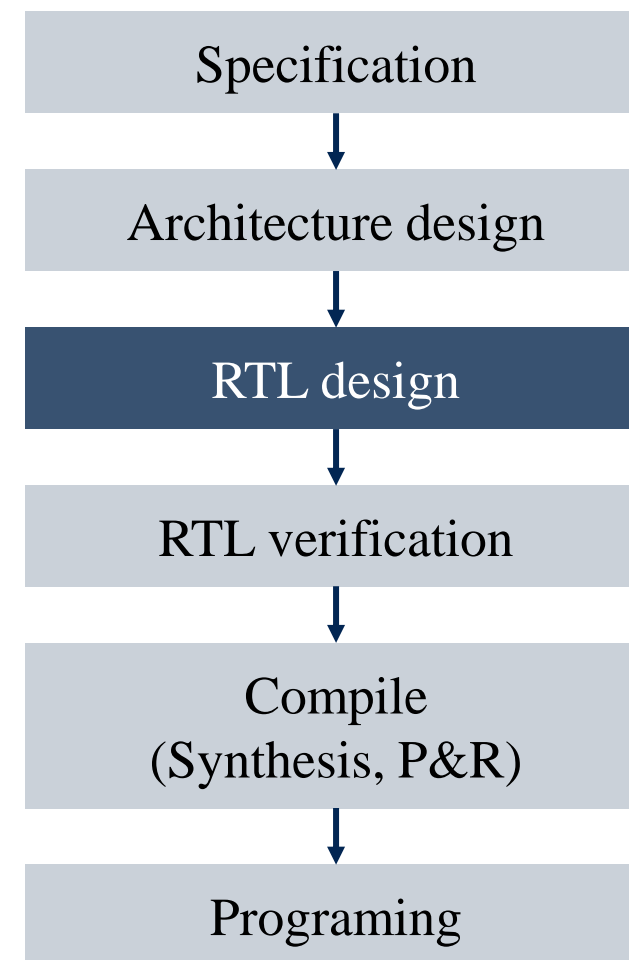
● 구조 설계



● RTL 설계

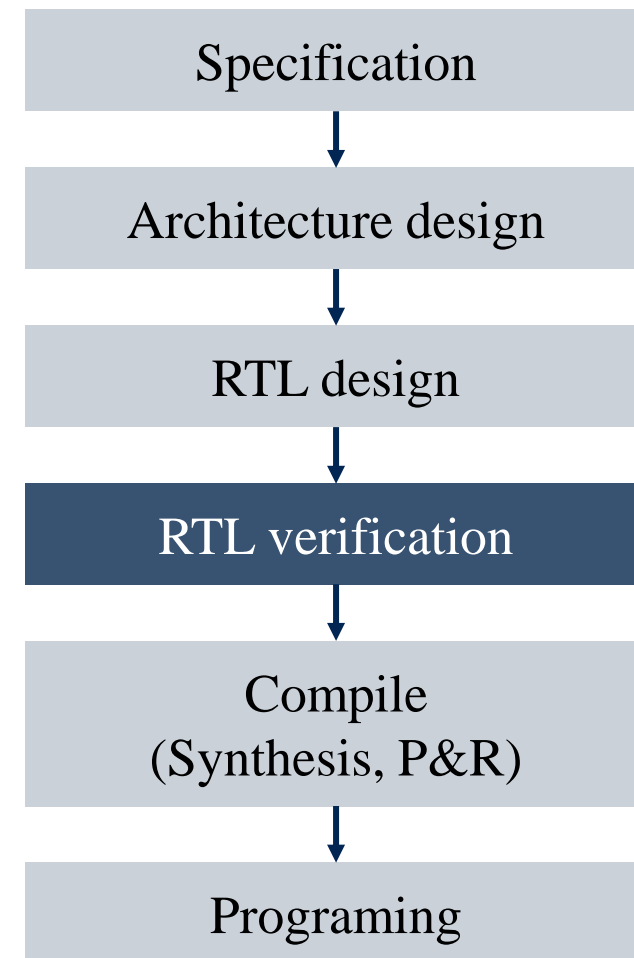
■ logic_test.v

```
module logic_test(i_0, i_1, o_AND, o_OR, o_NOT, o_XOR);  
input i_0;  
input i_1;  
output o_AND;  
output o_OR;  
output o_NOT;  
output o_XOR;  
  
assign o_AND = i_0 & i_1;  
assign o_OR = i_0 | i_1;  
assign o_NOT = ~i_0;  
assign o_XOR = i_0 ^ i_1;  
  
endmodule
```



● RTL 검증

- 입력에 따른 출력이 올바른지 확인하는 과정
- 입력을 어떻게 변화시킬까?
➔ 입력을 넣어줄 수 있는 외부 module 필요



- RTL 검증 - testbench module

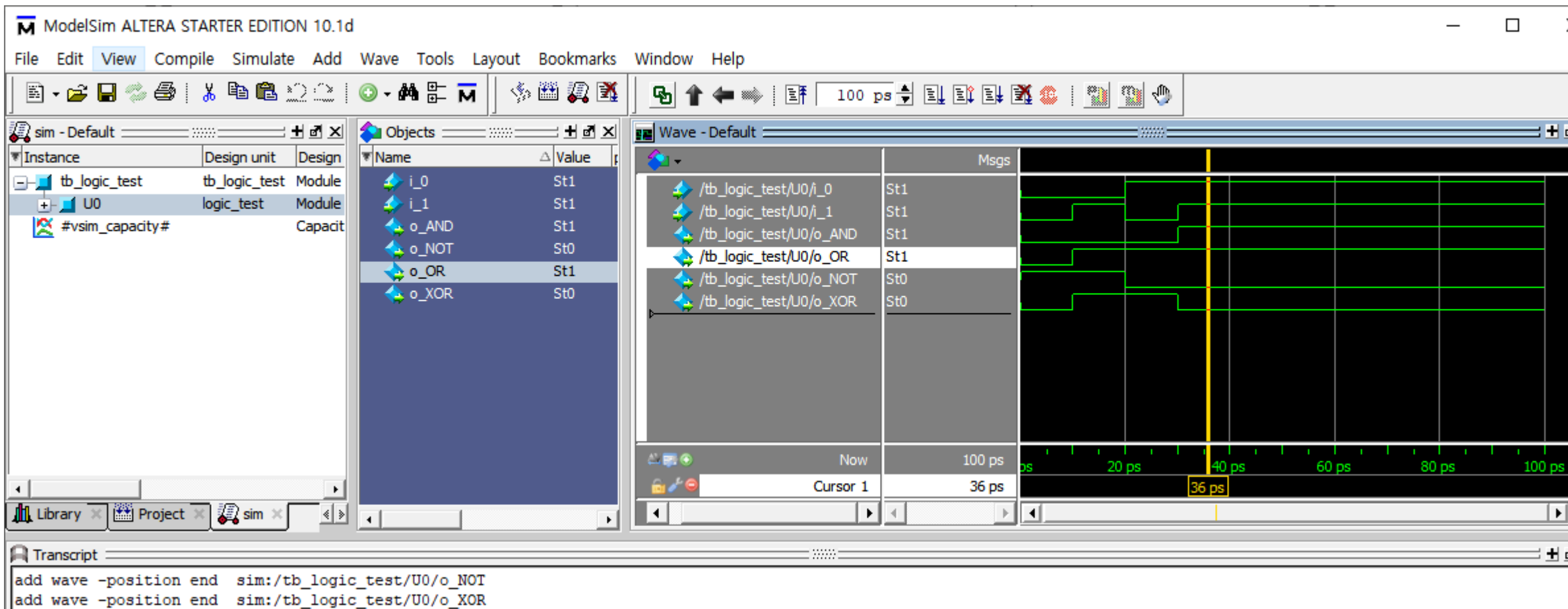
- tb_logic_test.v

```
module tb_logic_test;
reg LT_i_0;
reg LT_i_1;
wire LT_o_AND;
wire LT_o_OR;
wire LT_o_NOT;
wire LT_o_XOR;

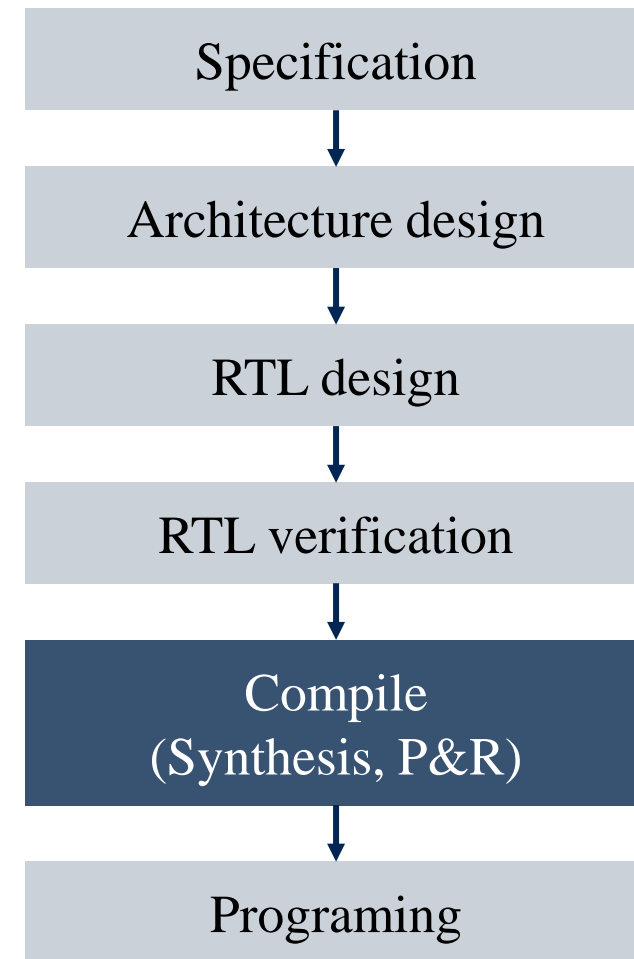
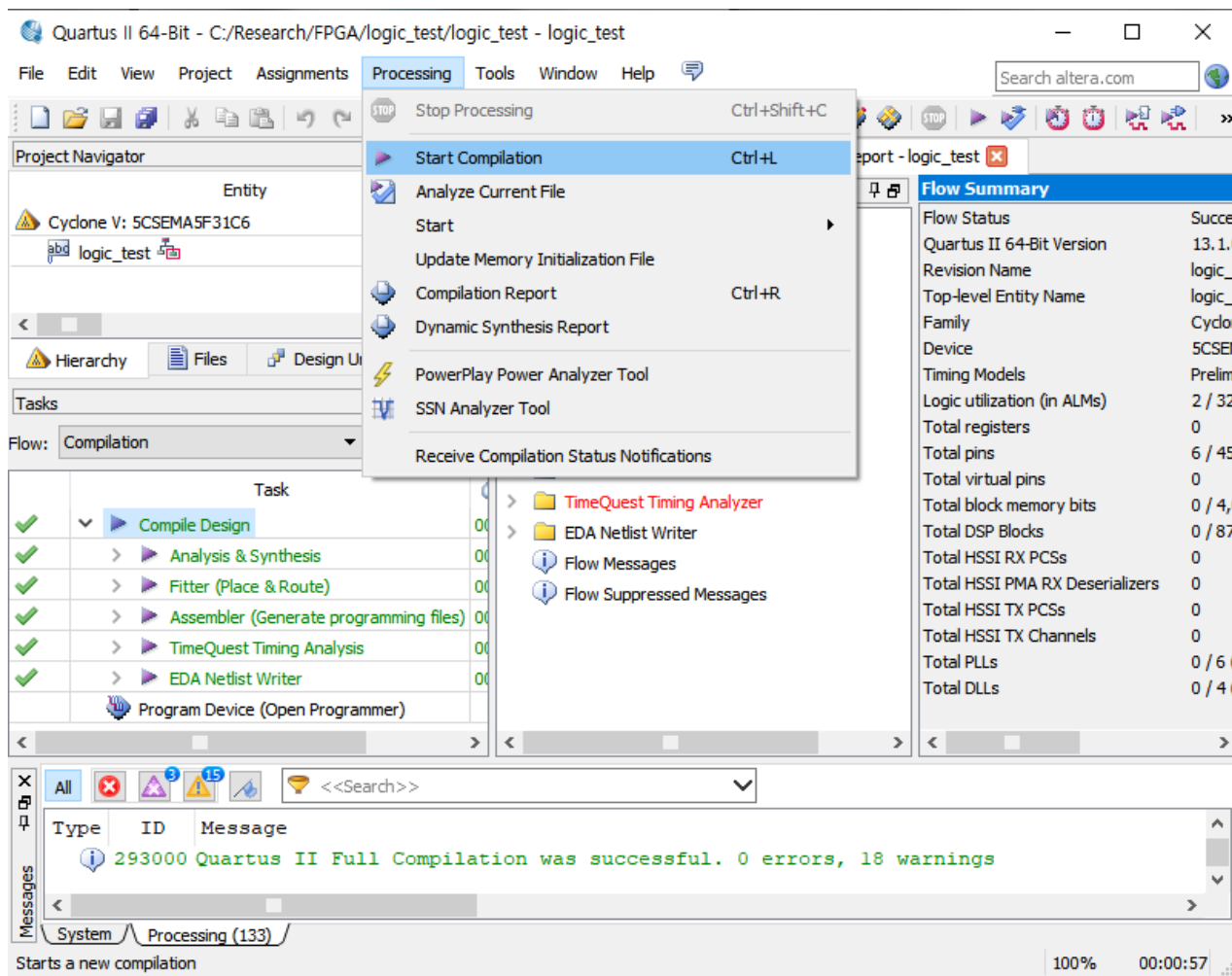
logic_test U0(LT_i_0, LT_i_1, LT_o_AND, LT_o_OR, LT_o_NOT, LT_o_XOR);

initial
begin
    LT_i_0 = 0; LT_i_1 = 0;
    #10 LT_i_0 = 0; LT_i_1 = 1;
    #10 LT_i_0 = 1; LT_i_1 = 0;
    #10 LT_i_0 = 1; LT_i_1 = 1;
end
endmodule
```

- RTL 검증 - Functional simulation
 - ModelSim을 이용하여 동작 검증



● FPGA 구현



● FPGA 구현

■ Pin 설정



Table 3-6 Pin Assignments for Slide Switches

| Signal Name | FPGA Pin No. | Description | I/O Standard |
|-------------|--------------|-----------------|--------------|
| SW[0] | PIN_AB12 | Slide Switch[0] | 3.3V |
| SW[1] | PIN_AC12 | Slide Switch[1] | 3.3V |
| SW[2] | PIN_AF9 | Slide Switch[2] | 3.3V |
| SW[3] | PIN_AF10 | Slide Switch[3] | 3.3V |
| SW[4] | PIN_AD11 | Slide Switch[4] | 3.3V |
| SW[5] | PIN_AD12 | Slide Switch[5] | 3.3V |
| SW[6] | PIN_AE11 | Slide Switch[6] | 3.3V |
| SW[7] | PIN_AC9 | Slide Switch[7] | 3.3V |
| SW[8] | PIN_AD10 | Slide Switch[8] | 3.3V |
| SW[9] | PIN_AE12 | Slide Switch[9] | 3.3V |

Table 3-8 Pin Assignments for LEDs

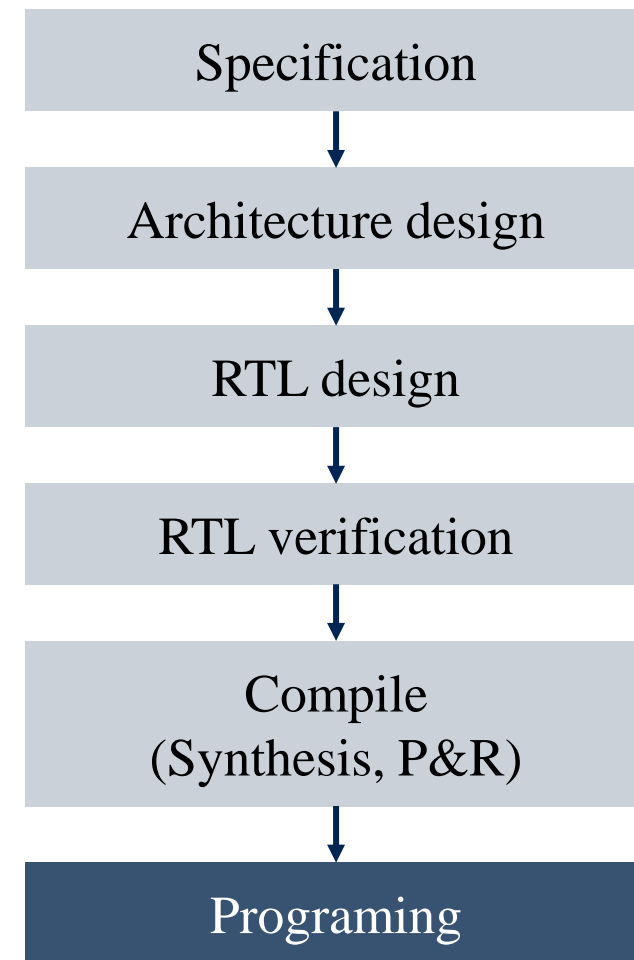
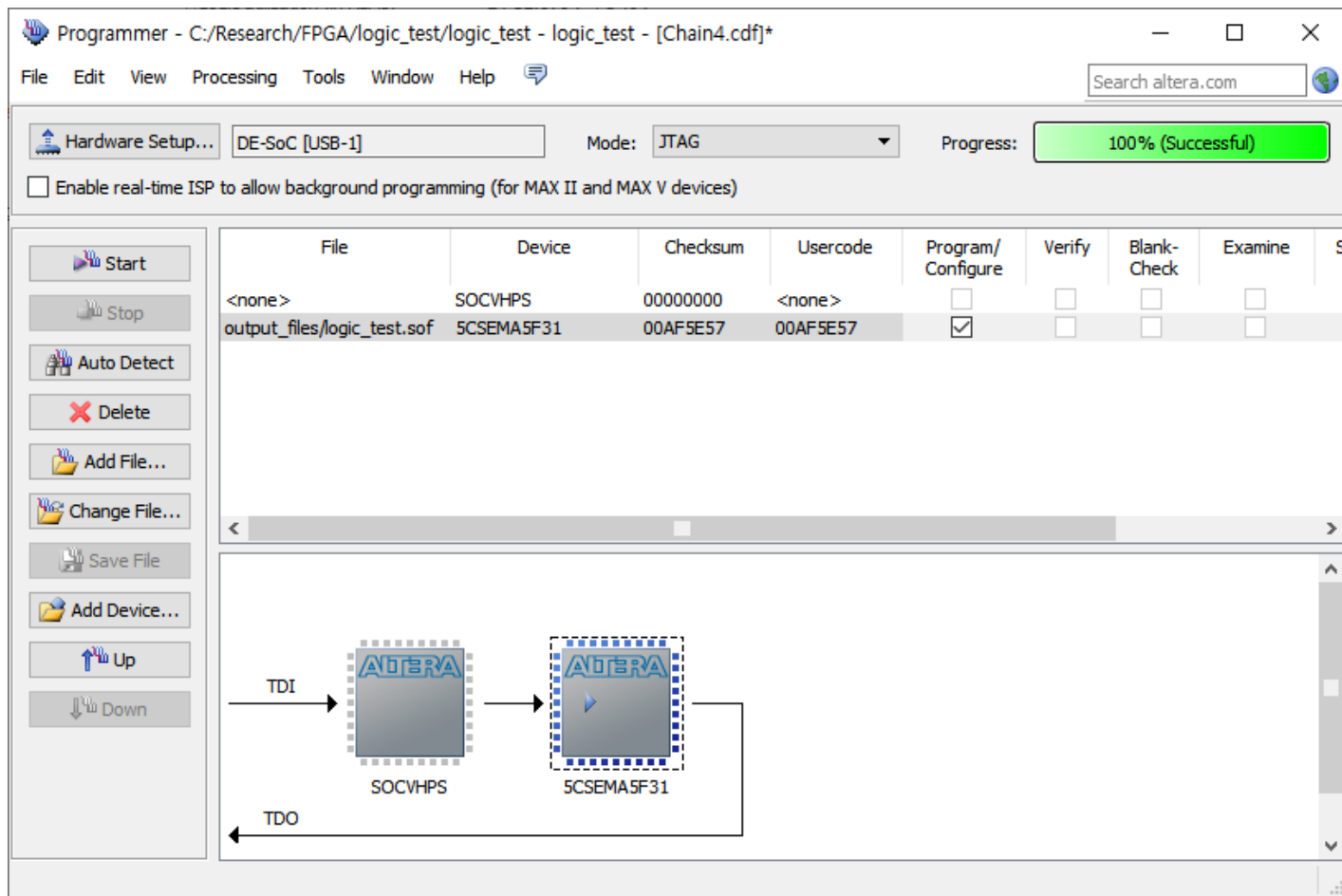
| Signal Name | FPGA Pin No. | Description | I/O Standard |
|-------------|--------------|-------------|--------------|
| LEDR[0] | PIN_V16 | LED [0] | 3.3V |
| LEDR[1] | PIN_W16 | LED [1] | 3.3V |
| LEDR[2] | PIN_V17 | LED [2] | 3.3V |
| LEDR[3] | PIN_V18 | LED [3] | 3.3V |
| LEDR[4] | PIN_W17 | LED [4] | 3.3V |
| LEDR[5] | PIN_W19 | LED [5] | 3.3V |
| LEDR[6] | PIN_Y19 | LED [6] | 3.3V |
| LEDR[7] | PIN_W20 | LED [7] | 3.3V |
| LEDR[8] | PIN_W21 | LED [8] | 3.3V |
| LEDR[9] | PIN_Y21 | LED [9] | 3.3V |



| Named: * | | Edit:   | | Filter: Pins: all | | |
|-----------|-----------|---|----------|-------------------|-----------------|-----------------|
| Node Name | Direction | Location | I/O Bank | VREF Group | Fitter Location | I/O Standard |
| in i_0 | Input | PIN_AB12 | 3A | B3A_N0 | PIN_AK28 | 2.5 V (default) |
| in i_1 | Input | PIN_AC12 | 3A | B3A_N0 | PIN_AG25 | 2.5 V (default) |
| out o_AND | Output | PIN_V16 | 4A | B4A_N0 | PIN_AJ27 | 2.5 V (default) |
| out o_NOT | Output | PIN_W16 | 4A | B4A_N0 | PIN_AD21 | 2.5 V (default) |
| out o_OR | Output | PIN_V17 | 4A | B4A_N0 | PIN_AH25 | 2.5 V (default) |
| out o_XOR | Output | PIN_V18 | 4A | B4A_N0 | PIN_V18 | 2.5 V (default) |

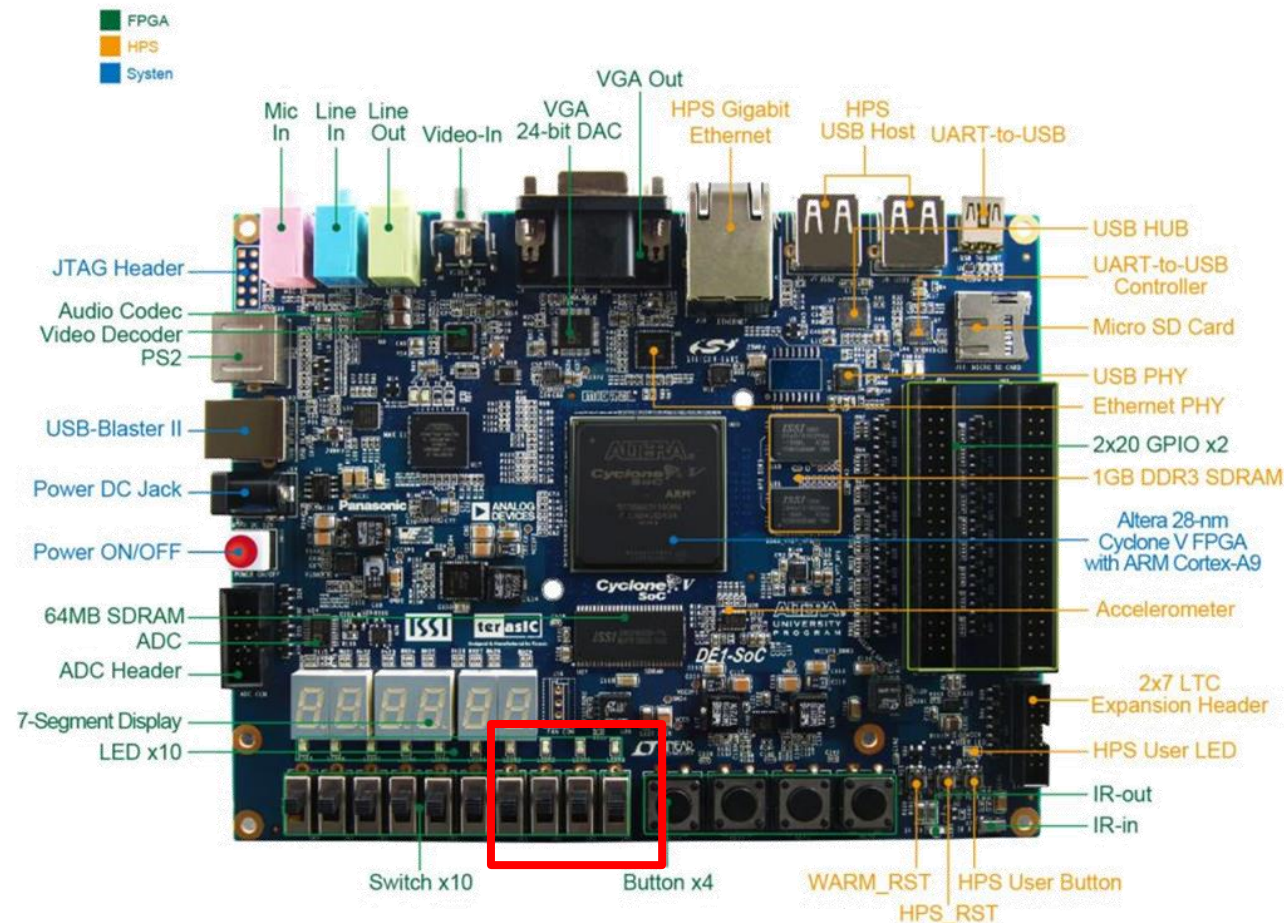
● FPGA 구현

■ Program



- FPGA 동작 확인

- SW0, SW1 변경에 따른 LED0 ~ LED3의 변화 확인



추가 연습

- 앞서 배운 연산자

- Bitwise operator: AND(&), OR(|), NOT(~), XOR(^)

- 비슷한 연산자

- Logical operator: AND(&&), OR(||), NOT(!)

➔ 앞서 연습한 코드에서 &, |, ~을 각각 &&, ||, !으로 바꾸고 시뮬레이션 결과 확인해보기



Verilog – 수의 표현, 변수 선언과 자료형

수의 표현

- 표현 방법: <비트 폭>'<진수><값>
 - 비트폭 생략: 32비트
 - 진수 생략: 10진수
 - 숫자 구분을 위해 언더 바(_) 사용 가능
- 사용 예

| 표현 | 비트 폭 | 진수 | 2진수 표현 | 비고 |
|-----------|------|------|-----------------|-----------------------|
| 10 | 32 | 10진수 | 00 ... 01010 | 설계, 시뮬레이션 모두 사용 가능 |
| 1'b1 | 1 | 2진수 | 1 | |
| 8'haa | 8 | 16진수 | 10101010 | |
| 8'o3_77 | 8 | 8진수 | 11111111 | |
| 'hff | 32 | 16진수 | 00...0111111111 | |
| 4'd5 | 4 | 10진수 | 0101 | |
| -5'b00001 | 5 | 2진수 | 11111 | |
| 32e-4 | - | 실수 | 0.0032 | 시뮬레이션만 사용 가능 |
| 4.1E3 | - | 실수 | 4100 | |

- 논리값

| 신호 | 의미 | 발생하는 상황 | ModelSim 내 표시 |
|----|-------------------------------|---------------|---------------|
| 0 | logic zero 또는 false condition | - | 검은색 |
| 1 | logic one 또는 true condition | - | 검은색 |
| X | Unknown logic value | 값 충돌 등의 계산 불가 | 붉은색 |
| Z | High impedance | 값 정의 X | 파란색 |

변수 선언과 자료형

- Verilog의 자료형

| | 용도 | 기본값 | 실 사용 |
|-----------|---|-----|--------------------------------------|
| net | 소자간 물리적 연결 | Z | 연속 할당문 (assign), 하위 모듈의 output |
| reg | 값을 임시로 저장할 수 있는 변수 (FF 등의 값 저장에 사용 가능) | X | 절차적 할당문(always, initial 등) 내에서 사용 |
| parameter | 상수로 선언되는 데이터 | | |

- 미리 선언되지 않은 신호는 net(1 bit wire)으로 간주됨

변수 선언과 자료형

● Verilog의 자료형 – net

| | 의미 |
|------|----------------------------------|
| wire | 논리적 행동(회로 동작)이나 기능 없이 단순한 연결에 사용 |
| wand | 여러 디바이스 출력을 선으로 연결하여 and |
| wor | 여러 디바이스 출력을 선으로 연결하여 or |

- 이외에도 supply0, supply1, tri 등이 있음

■ 선언 예

- wire x, y; // 1비트 크기의 네트 x와 y를 wire로 선언
- wire [3:0] x, y; // 4비트 크기의 네트 x와 y를 wire로 선언
- wire [1:0] x, [3:0] y; // 오류, 각 줄마다 하나의 크기만으로 선언 가능
- wire enable = 1'b0; // 선언과 동시에 초기값 0 할당 (권장 X)

변수 선언과 자료형

● Verilog의 자료형 – reg

| | 의미 |
|---------|---|
| reg | unsinged, 크기를 갖는 정수 |
| integer | signed, 32비트 정수, for 문 등의 반복 횟수 계산 등에 주로 사용 |
| real | signed 실수, 시뮬레이션에서만 사용 |
| time | unsigned, 64비트 정수, 시뮬레이션에서만 사용 |

■ 선언 예

- reg q, q_; // 1비트 크기의 레지스터 q와 q_를 선언
- reg [3:0] x, y; // 4비트 크기의 레지스터 x와 y를 선언
- reg [1:0] x, [3:0] y; // 오류, 각 줄마다 하나의 크기만으로 선언 가능
- reg [7:0] mem [0:255]; // 8-bit 레지스터 256개로 구성된 배열

- Verilog의 자료형 – parameter

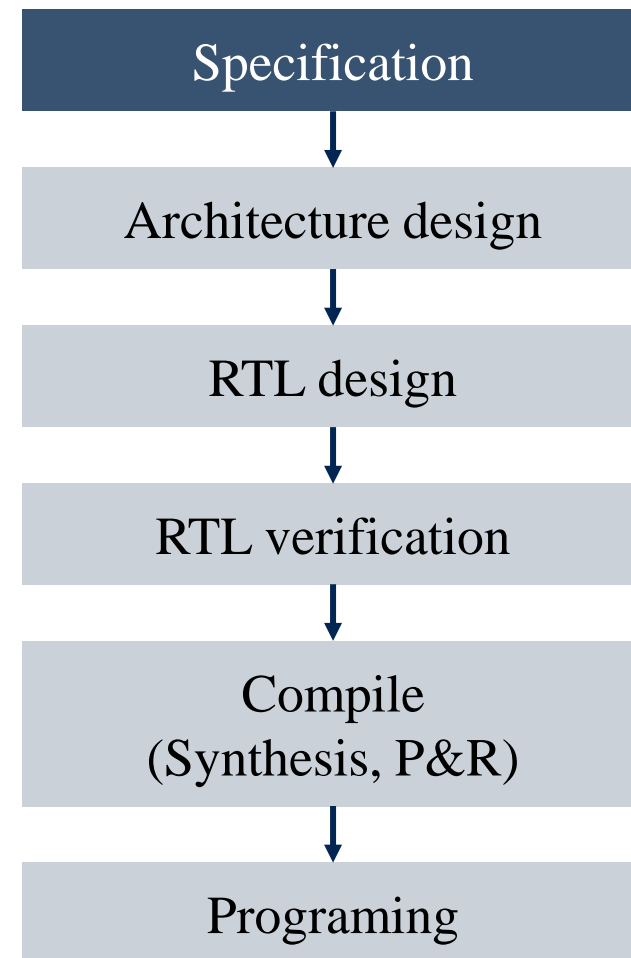
- C의 #define과 비슷
- 보통 대문자로 선언
- 선언 및 정의 예

- parameter X = 4'h5, Y = 4'h7; // X와 Y의 값을 각각 16진수 5와 7로 정의
- parameter AVG = (X + Y) / 2; // AVG를 X와 Y의 식에 의한 상수 결과로 정의
- parameter WIDTH = 32, DEPTH = 1024;
- Parameter SIZE = 12; // SIZE의 값을 상수 12로 정의
- parameter A = SIZE'b1010; // 문법 오류, 신호값의 비트 수 정의 불가

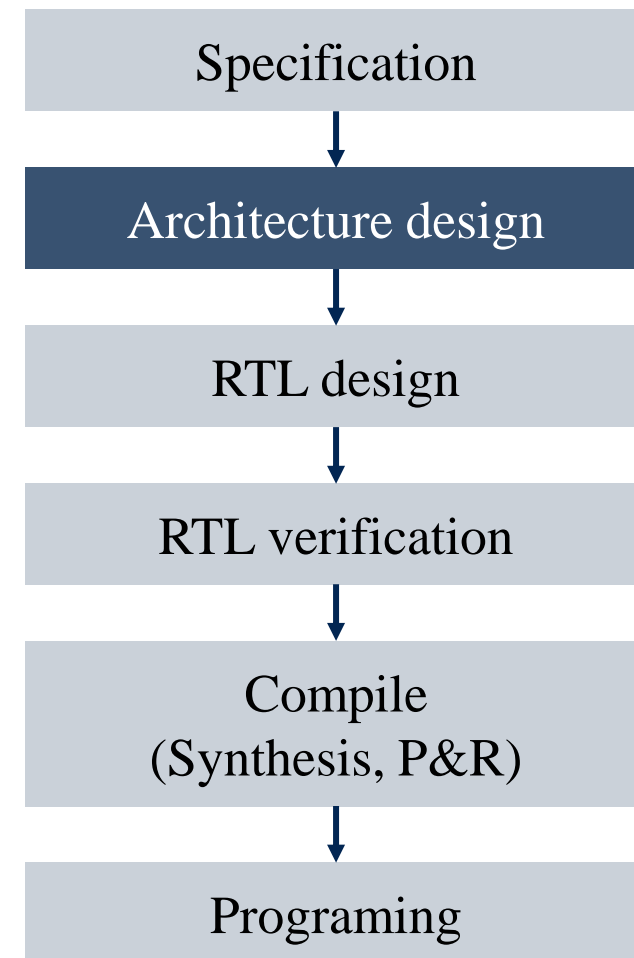
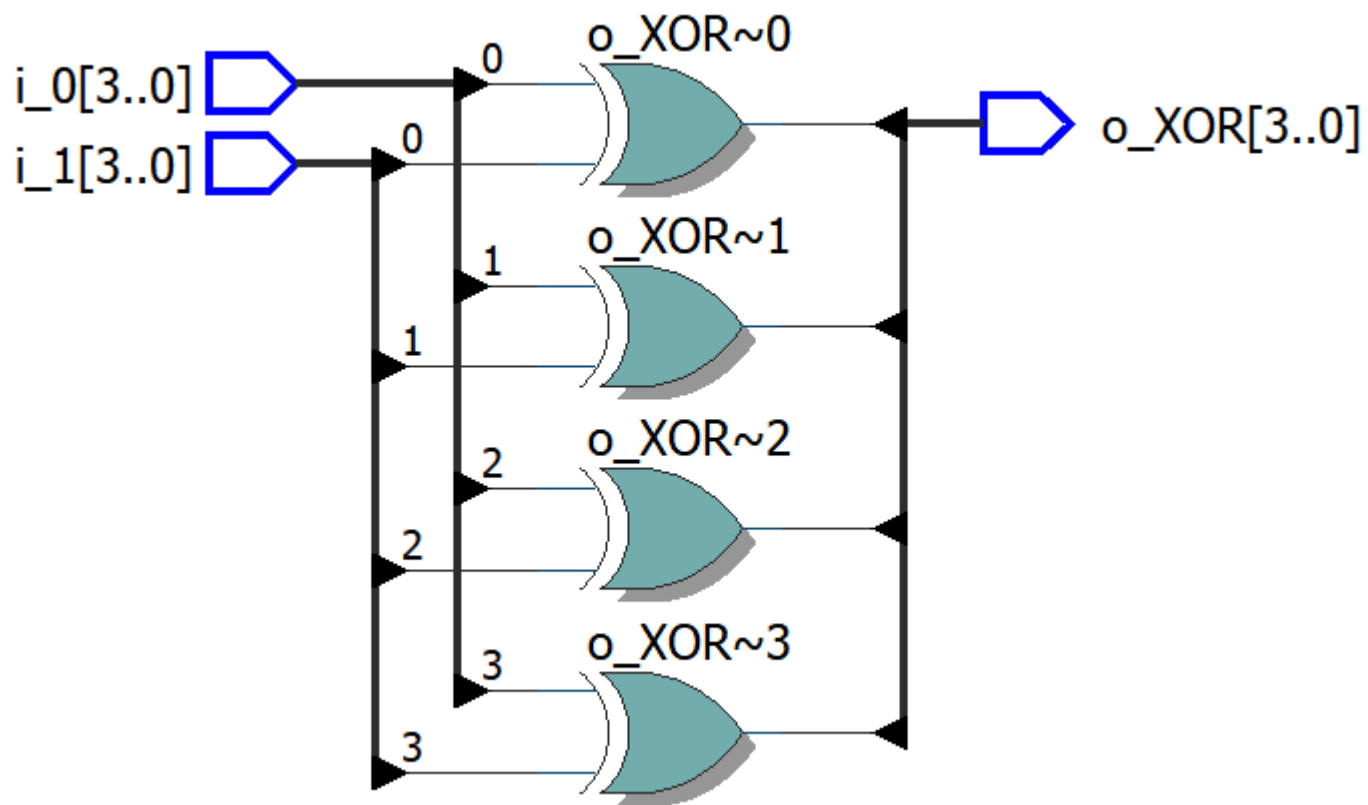
- 수행하려는 기능
 - 4비트 값 두 개를 XOR

● 명세

- 입력: 4비트 입력 x 2 → SW
- 출력: 4비트 출력 x 1 → LED
- 수행 기능
 - 두 입력을 각각 XOR하여 출력
- 모듈명: xor_test



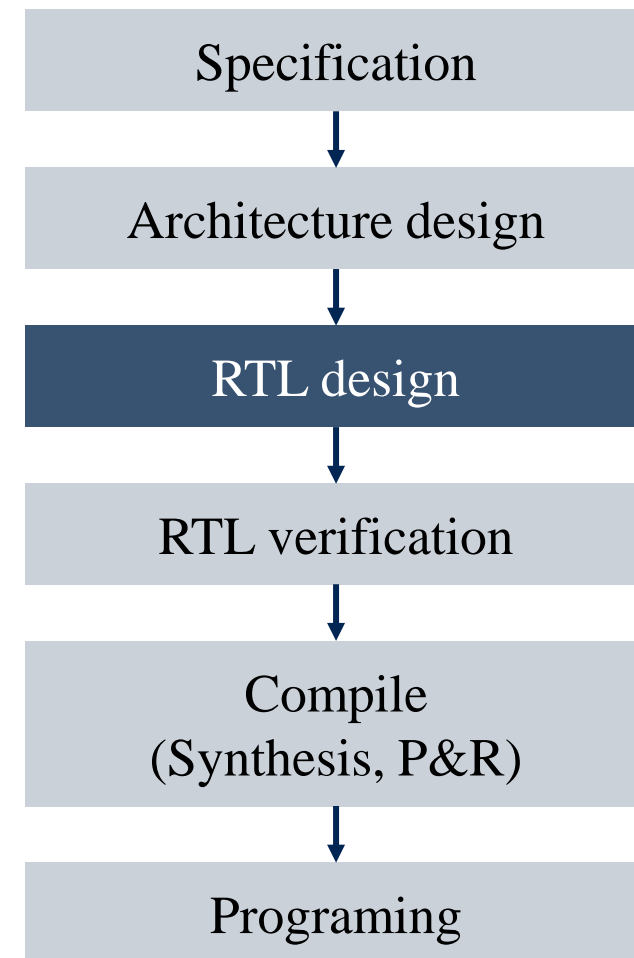
- 구조 설계



● RTL 설계

■ xor_test.v

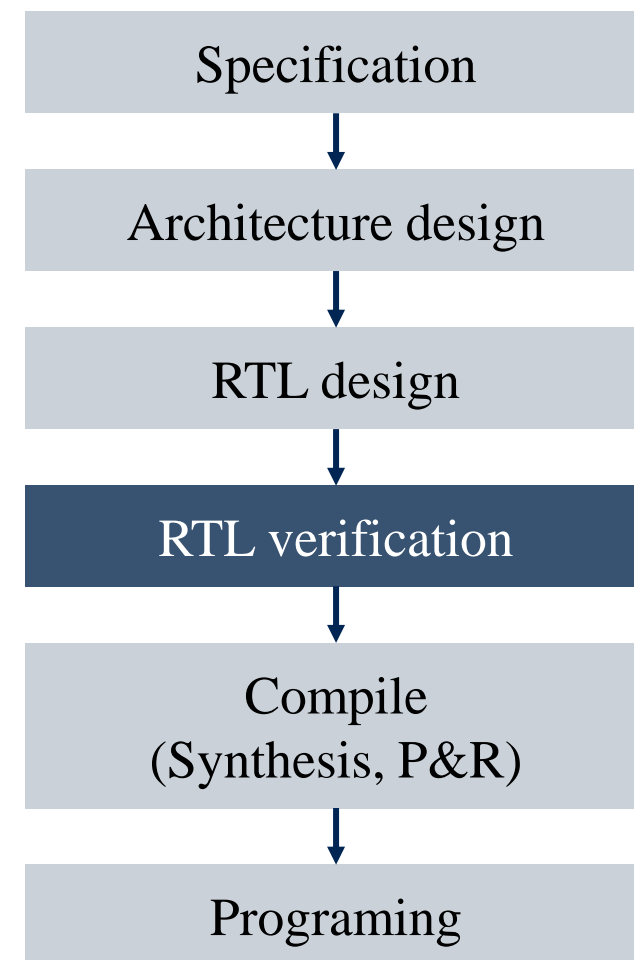
```
module xor_test(i_0, i_1, o_XOR);  
input [3:0] i_0, i_1;  
output wire [3:0] o_XOR;  
  
assign o_XOR = i_0 ^ i_1;  
  
endmodule
```



- RTL 검증 - testbench module

- tb_xor_test.v

```
module tb_xor_test;
reg      [3:0]    XOR_i_0;
reg      [3:0]    XOR_i_1;
wire[3:0] XOR_o;
xor_test U0(XOR_i_0, XOR_i_1, XOR_o);
initial
begin
    XOR_i_0 = 4'b1010; XOR_i_1 = 4'b1100;
    #10 XOR_i_0 = 4'b0101; XOR_i_1 = 4'b0111;
    #10 XOR_i_0 = 4'b1111; XOR_i_1 = 4'b1010;
end
endmodule
```



- RTL 검증 - testbench module

- xor_test.v와 tb_xor_test.v

```
module xor_test(i_0, i_1, o_XOR);  
input [3:0] i_0, i_1;  
output wire [3:0] o_XOR;  
  
assign o_XOR = i_0 ^ i_1;  
  
endmodule
```

```
module tb_xor_test;  
reg [3:0] XOR_i_0;  
reg [3:0] XOR_i_1;  
wire[3:0] XOR_o;  
xor_test U0(XOR_i_0, XOR_i_1, XOR_o);  
initial  
begin  
    XOR_i_0 = 4'b1010; XOR_i_1 = 4'b1100;  
    #10 XOR_i_0 = 4'b0101; XOR_i_1 = 4'b0111;  
    #10 XOR_i_0 = 4'b1111; XOR_i_1 = 4'b1010;  
  
end  
endmodule
```

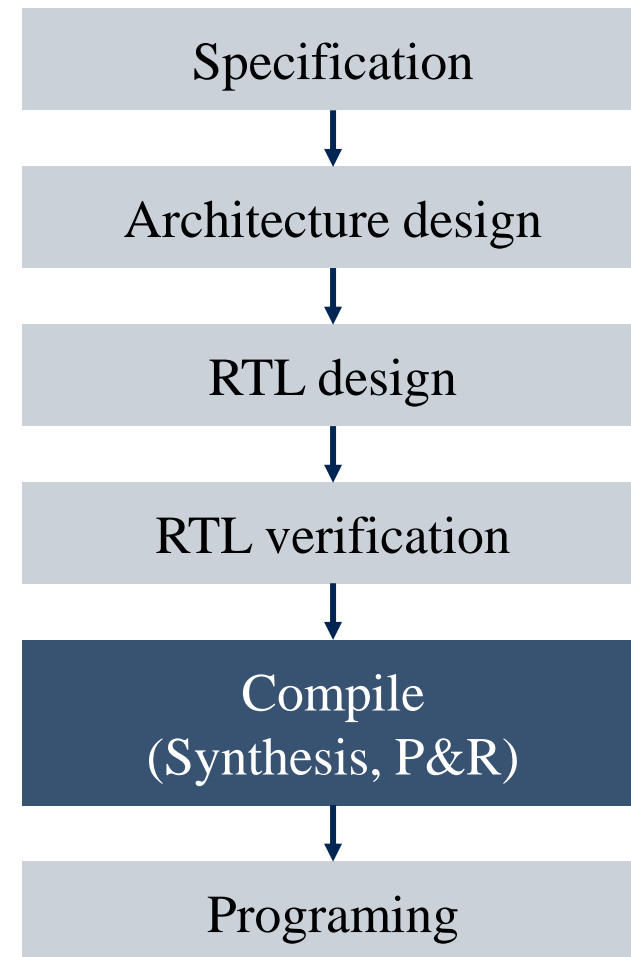

● FPGA 구현 – pin 설정

Table 3-6 Pin Assignments for Slide Switches

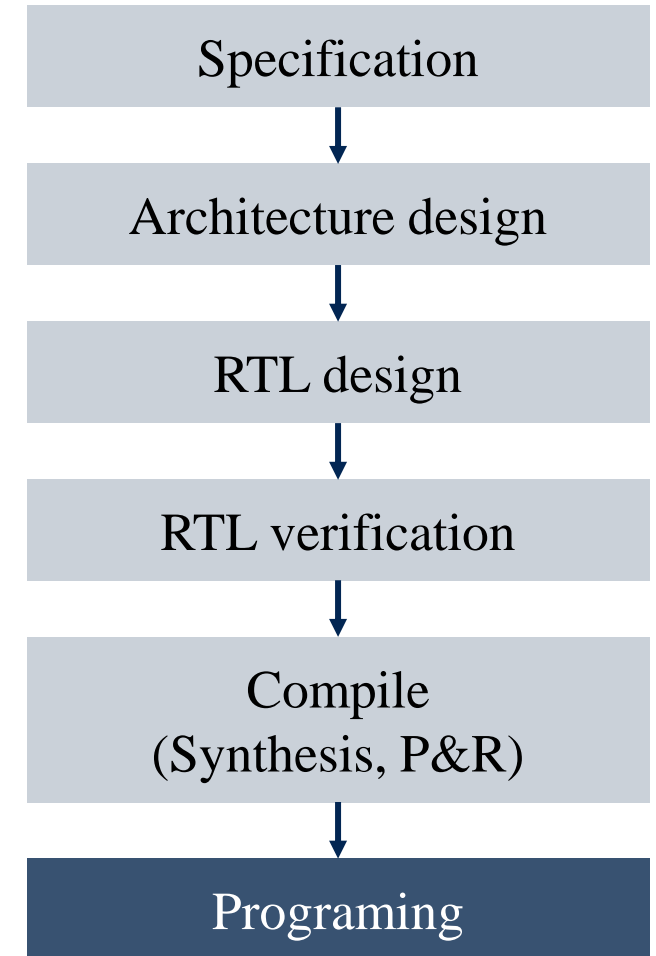
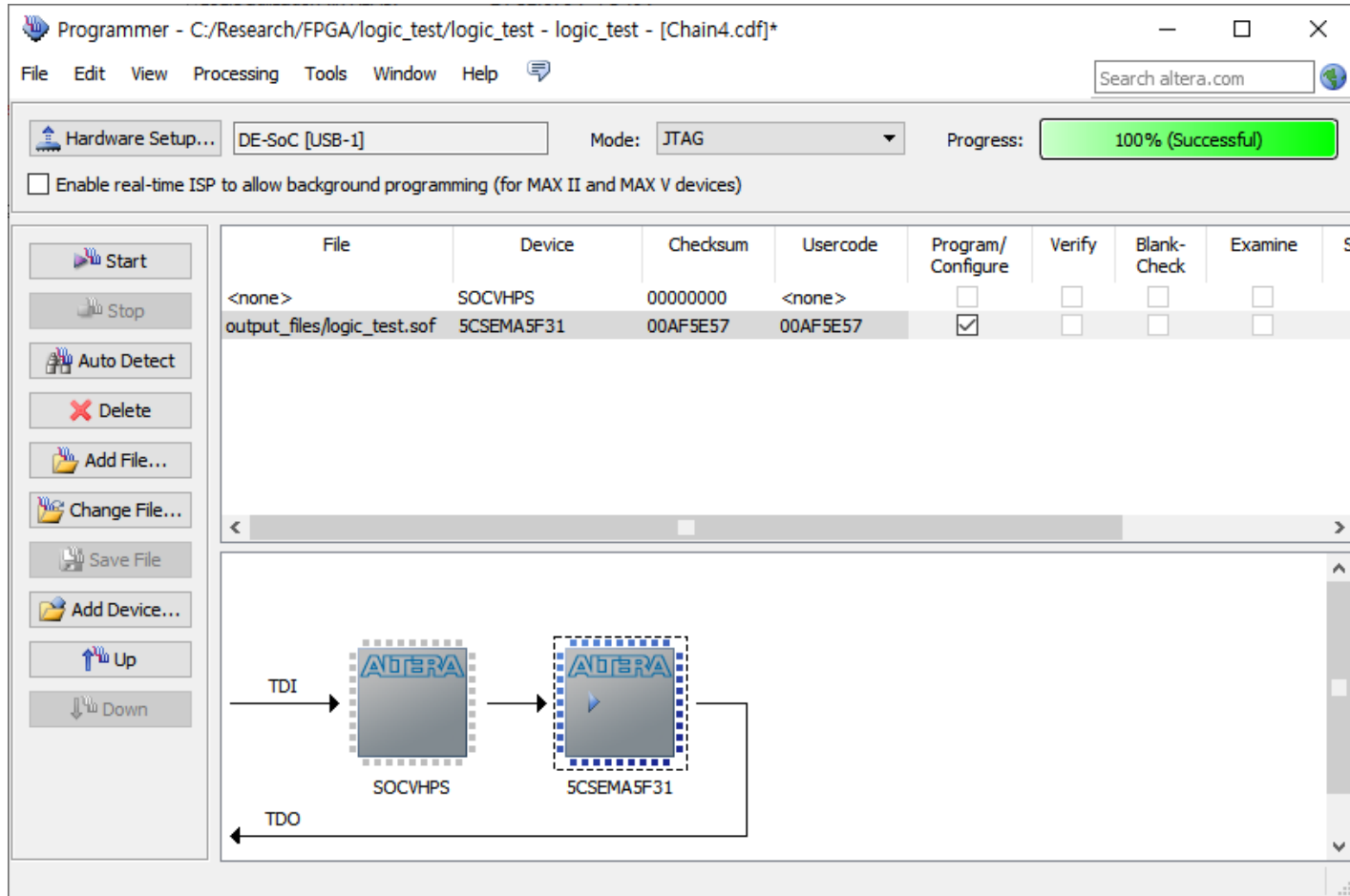
| Signal Name | FPGA Pin No. | Description | I/O Standard |
|-------------|--------------|-----------------|--------------|
| SW[0] | PIN_AB12 | Slide Switch[0] | 3.3V |
| SW[1] | PIN_AC12 | Slide Switch[1] | 3.3V |
| SW[2] | PIN_AF9 | Slide Switch[2] | 3.3V |
| SW[3] | PIN_AF10 | Slide Switch[3] | 3.3V |
| SW[4] | PIN_AD11 | Slide Switch[4] | 3.3V |
| SW[5] | PIN_AD12 | Slide Switch[5] | 3.3V |
| SW[6] | PIN_AE11 | Slide Switch[6] | 3.3V |
| SW[7] | PIN_AC9 | Slide Switch[7] | 3.3V |
| SW[8] | PIN_AD10 | Slide Switch[8] | 3.3V |
| SW[9] | PIN_AE12 | Slide Switch[9] | 3.3V |

Table 3-8 Pin Assignments for LEDs

| Signal Name | FPGA Pin No. | Description | I/O Standard |
|-------------|--------------|-------------|--------------|
| LEDR[0] | PIN_V16 | LED [0] | 3.3V |
| LEDR[1] | PIN_W16 | LED [1] | 3.3V |
| LEDR[2] | PIN_V17 | LED [2] | 3.3V |
| LEDR[3] | PIN_V18 | LED [3] | 3.3V |
| LEDR[4] | PIN_W17 | LED [4] | 3.3V |
| LEDR[5] | PIN_W19 | LED [5] | 3.3V |
| LEDR[6] | PIN_Y19 | LED [6] | 3.3V |
| LEDR[7] | PIN_W20 | LED [7] | 3.3V |
| LEDR[8] | PIN_W21 | LED [8] | 3.3V |
| LEDR[9] | PIN_Y21 | LED [9] | 3.3V |



- FPGA 구현
 - Program



추가 연습

- 4비트 AND 연습

- 4bit 데이터 두 개를 AND하기로 바꾸기
- 연산자를 & 또는 &&로 각각해보고 두 개의 결과 비교해보기

- 4비트 OR 연습

- 4bit 데이터 두 개를 OR하기로 바꾸기
- 연산자를 | 또는 ||로 각각해보고 두 개의 결과 비교해보기

- 4비트 NOT 연습

- 4bit 데이터 한 개를 NOT하기로 바꾸기
- 연산자를 ~ 또는 !로 각각해보고 두 개의 결과 비교해보기



Verilog – 축약 연산자

축약(Reduction) 연산의 필요성

- 다음과 같은 사항은 어떻게 표현/계산할 수 있을까?
 - 3비트 counter의 값이 7일 때(모든 비트의 값이 1)
 - 4비트 counter의 값이 0일 때(모든 비트의 값이 0)
 - 8비트 data의 even parity bit

축약(Reduction) 연산의 필요성

- 기존 연산자를 사용하는 경우

| 연산자 | 기존 연산자 사용 |
|---------------------------------------|--|
| 3비트 counter의 값이 7일 때 (모든 비트의 값이 1) | $fCountIs7 = counter[0] \& counter[1] \& counter[2]$ |
| 4비트 counter의 값이 0일 때 (모든 비트의 값이 0) | $fCountIs0 = \sim count[0] \& \sim count[1] \& count[2] \& \sim count[3]$ or $fCountIs0 = \sim(count[0] count[1] count[2] count[3])$ |
| 8비트 data의 even parity bit | $parityBit = data[0] \wedge data[1] \wedge data[2] \wedge data[3] \wedge$ $data[4] \wedge data[5] \wedge data[6] \wedge data[7]$ |

➔ 표현 복잡, bit길이가 달라지면 표현도 달라짐

축약(Reduction) 연산의 필요성

- 축약 연산자를 사용하는 경우

| 연산자 | 기존 연산자 사용 | 축약 연산자 사용 |
|---------------------------------------|--|---------------------------|
| 3비트 counter의 값이 7일 때 (모든 비트의 값이 1) | $fCountIs7 = counter[0] \& counter[1] \& counter[2]$ | $fCountIs7 = \&counter$ |
| 4비트 counter의 값이 0일 때 (모든 비트의 값이 1) | $fCountIs0 = \sim(count[0] \mid count[1] \mid count[2] \mid count[3])$ | $fCountIs0 = \sim count$ |
| 8비트 data의 even parity bit | $parityBit = data[0] \wedge data[1] \wedge data[2] \wedge data[3] \wedge data[4] \wedge data[5] \wedge data[6] \wedge data[7]$ | $parityBit = \wedge data$ |

➔ 표현 간단, bit길이가 달라지더라도 표현 동일

축약 연산자

- 사용 방법

- 예: $a = \sim b$... 단항 연산

- 연산자 종류

| 연산자 | 의미 | 사용 예 |
|---------|-----------|---|
| & | 비트단항 and | &(0101) $\rightarrow (0 \cdot 1 \cdot 0 \cdot 1) = 0$ |
| | 비트단항 or | (0101) $\rightarrow (0 + 1 + 0 + 1) = 1$ |
| ~ & | 비트단항 nand | ~ &(0101) $\rightarrow \sim (0 \cdot 1 \cdot 0 \cdot 1) = 1$ |
| ~ | 비트단항 nor | ~ (0101) $\rightarrow \sim (0 + 1 + 0 + 1) = 0$ |
| ^ | 비트단항 xor | ^(0101) $\rightarrow (0 \oplus 1 \oplus 0 \oplus 1) = 0$ |
| ^~, ~ ^ | 비트단항 xnor | ~ ^ (0101) $\rightarrow (0 \odot 1 \odot 0 \odot 1) = 1$ |

- 수행하려는 기능
 - 8비트 입력 데이터에 대해 even parity bit을 생성
- 명세
 - 입력: i_Data(8비트) → SW
 - 출력: o_Data(8비트), o_Parity(1비트) → LED
 - 모듈명: GenEvenParity
- 시뮬레이션과 FPGA 검증까지 해보기

- 4비트 a , b 의 값이 같음을 어떻게 표현할 수 있을까?

추가 연습

- 4비트 a, b 의 값이 같음을 어떻게 표현할 수 있을까?
 - a 와 b 를 XOR하면 모든 값이 0이어야 함
 - ➔ $fA_eq_B = \sim|(A \wedge B)$



Verilog – 기타 연산자들

관계 연산의 필요성

- 4비트 a, b의 값이 같다
 - $fA_eq_B = \sim|(A \wedge B)$
→ 표현이 복잡하지는 않지만 직관적이지 않음

관계 연산자

- 사용 방법

- 예: `a = b == c` ... 2항 연산

- 연산자 종류

| 연산자 | 의미 | 사용 예 |
|--------------------|--------|-----------------------------------|
| <code>==</code> | 같다 | <code>fA_eq_B = a == b;</code> |
| <code>!=</code> | 같지 않다 | <code>nA_eq_B = a != b;</code> |
| <code><</code> | 작다 | <code>fA_lt_B = a < b;</code> |
| <code><=</code> | 같거나 작다 | <code>fA_le_B = a <= b;</code> |
| <code>></code> | 크다 | <code>fA_gt_B = a > b;</code> |
| <code>>=</code> | 크거나 같다 | <code>fA_ge_B = a >= b;</code> |

결합 연산의 필요성

- Parity bit 추가

- 기존) Parity bit을 생성하는 경우 데이터와 parity bit을 별도로 관리
- 결합 연산) 데이터와 parity bit을 하나의 변수로 표현 ➔ 관리 용이

- 사용법

- $a = \{b, c\}$

- 예

- parity bit 추가하기: $\text{newData} = \{\text{data}, \text{^data}\}$
- Shift (data, newData는 8비트 데이터)
 - 왼쪽으로 1비트 shift: $\text{newData} = \{\text{data}[6:0], 1'b0\}$
 - 오른쪽으로 2비트 shift: $\text{newData} = \{2'b0, \text{data}[7:2]\}$
 - 왼쪽으로 1비트 rotate: $\text{newData} = \{\text{data}[6:0], \text{data}[7]\}$
 - 오른쪽으로 2비트 rotate: $\text{newData} = \{\text{data}[1:0], \text{data}[7:2]\}$
- 주의: $\text{newData} = \{\text{data}, 1\}$

반복 연산의 필요성

- 8-bit 2진수 (signed)

- $1000\ 0000_{\text{two}} = -1 \times 2^7 = -128_{\text{ten}}$
- $1000\ 0001_{\text{two}} = -1 \times 2^7 + 1 = -127_{\text{ten}}$
- ...
- $1111\ 1111_{\text{two}} = -1 \times 2^7 + 127 = -1_{\text{ten}}$

- 8-bit 2진수 (unsigned)

- $1000\ 0000_{\text{two}} = 1 \times 2^7 = 128_{\text{ten}}$
- $1000\ 0001_{\text{two}} = 1 \times 2^7 + 1 = 129_{\text{ten}}$
- ...
- $1111\ 1111_{\text{two}} = 1 \times 2^7 + 127 = 255_{\text{ten}}$

} 16비트로 확장하는 경우는?

반복 연산의 필요성

- 예
 - 8비트 signed 변수인 data를 16비트 signed 변수인 newData로 확장
 - 결합 연산을 이용한 sign bit 확장
 - $\text{newData} = \{\text{data}[7], \text{data}[7], \text{data}[7], \text{data}[7], \text{data}[7], \text{data}[7], \text{data}[7], \text{data}[7], \text{data}\}$
- ➔ 반복되는 부분을 일일이 적어줘야 함

- 사용법

- $a = \{3\{b\}\}$

- 예

- 8비트 signed 변수인 data를 16비트 signed 변수인 newData로 확장
 - $\text{newData} = \{\{8\{\text{data}[7]\}\}, \text{data}\}$

Summary

● Module

- 하드웨어 설계 시 기본적인 설계 단위
- 정의
 - module
 - **modular_name**
 - **interface**
 - **body**
 - endmodule

```
module module_name (port_list)
```

```
port declarations
```

```
parameter declarations
```

```
variable declarations
```

```
assignments
```

```
lower-level module instantiation
```

```
initial and always blocks
```

```
tasks and function
```

```
endmodule
```

Summary

- 수의 표현 방법: <비트 폭>'<진수><값>
- 논리값

| 신호 | 의미 | 발생하는 상황 | ModelSim 내 표시 |
|----|-------------------------------|---------------|---------------|
| 0 | logic zero 또는 false condition | - | 검은색 |
| 1 | logic one 또는 true condition | - | 검은색 |
| X | Unknown logic value | 값 충돌 등의 계산 불가 | 붉은색 |
| Z | High impedance | 값 정의 X | 파란색 |

Summary

- Verilog의 자료형

| | 용도/실제 사용 | 크기 |
|-----------|-----------------------------------|-----------------|
| wire | 연속 할당문 (assign), 하위 모듈의 output | 지정 가능(기본 1 bit) |
| reg | 절차적 할당문(always, initial 등) 내에서 사용 | 지정 가능(기본 1 bit) |
| integer | for 문 등의 반복 횟수 계산 등에 주로 사용 | 32 bits |
| parameter | 상수로 선언되는 데이터 | 정의시 결정 |

- 미리 선언되지 않은 신호는 net(1 bit wire)으로 간주됨

Summary

| 종류 | 연산자 | 사용 예 |
|---------------------------|--------|------------|
| 논리연산자 (Logical) | && | a = b && c |
| | | a = b c |
| | ! | a = !b |
| 비트 단위 연산자 (Bitwise) | & | a = b & c |
| | | a = b c |
| | ~ | a = ~b |
| | ^ | a = b ^ c |
| | ^~, ~^ | a = b ^~ c |
| 축약연산자 (Reduction) | & | a = &b |
| | | a = b |
| | ~& | a = ~&b |
| | ~ | a = ~ b |
| | ^ | a = ^b |
| | ^~, ~^ | a = ~^b |

| 종류 | 연산자 | 의미 | 사용 예 |
|--------------------------|---------|--------|--|
| 관계연산자 (Relational) | == | 같다 | fA_eq_B = a == b; |
| | != | 같지 않다 | nA_eq_B = a != b; |
| | < | 작다 | fA_lt_B = a < b; |
| | <= | 같거나 작다 | fA_le_B = a <= b; |
| | > | 크다 | fA_gt_B = a > b; |
| 결합연산자 (Concatenation) | { } | 결합 | a = {1'b1, 1'b0, 2'b1} newData = {data[6:0], data[7]} |
| | { { } } | 반복 | a = {2{2'b01}} newData = {{8{data[7]}}, data} |

※ 논리, 비트단위, 관계 조건 연산자는 C와 연산자 사용법과 결과 동일