VHDL 개발자를 위한 Verilog RTL 입문

PPT 제작용 슬라이드 가이드

VHDL 설계 경험을 기반으로 Verilog RTL을 읽고, 수정하고, 검토하기

구성: 1부 필요성 → 2부 구조 대응 → 3부 데이터 표현 → 4부 조합논리 → 5부 순차논리 → 6부 FSM/계층 → 7부 오류 → 8부 Testbench → 9부 정리 → Appendix

이 가이드 사용 방법

각 페이지는 PPT 한 장을 만들기 위한 설계안입니다. 왼쪽은 실제 슬라이드에 넣을 내용, 오른쪽은 발표자가 말할 설명 흐름입니다. “필수” 슬라이드를 중심으로 구성하고, “선택” 슬라이드는 시간과 청중 수준에 따라 Appendix 또는 후속 교육으로 이동하십시오.

표시	의미	권장 사용
필수	세미나 핵심 흐름	본문에 포함
선택	보조 통계·장기 방향·치트시트	Appendix 또는 질의응답
화면에 넣을 내용	실제 PPT 문구와 코드	슬라이드 밀도에 맞춰 축약
발표 멘트	설명 순서와 강조점	발표자 노트로 사용
디자인/주의	오해 방지와 시각 구성	최종 편집 시 반영

전체 구성

구분	슬라이드	핵심 질문
1부	01-07	왜 VHDL 개발자에게 Verilog 역량이 필요한가?
2부	08-14	VHDL의 익숙한 구조가 Verilog에서는 어떻게 보이는가?
3부	15-22	숫자, 벡터, 폭, signed를 어떻게 읽는가?
4부	23-31	Latch 없이 조합논리를 어떻게 작성하는가?
5부	32-41	Clock, Reset, Nonblocking을 어떻게 이해하는가?
6부	42-50	FSM과 계층 구조를 어떻게 읽고 구성하는가?
7부	51-57	실무에서 자주 발생하는 오류를 어떻게 찾는가?
8부	58-65	기본 Testbench를 어떻게 구성하고 판정하는가?
9부	66-68	외부 RTL을 조직의 설계 자산으로 어떻게 관리하는가?
Appendix	A1-A5	통계, AMD IP 사례, SystemVerilog 방향, 치트시트

1부

1부. 왜 Verilog인가

Verilog 학습을 언어 전환이 아니라 설계 자산 통제 역량으로 정의한다.

필수

VHDL 개발자를 위한 Verilog RTL 입문

이 슬라이드의 목적 세미나의 대상과 방향을 첫 장에서 분명하게 설정한다.

화면에 넣을 내용

기존 VHDL 설계 경험을 기반으로 Verilog RTL을 읽고, 수정하고, 검토하는 방법

- 대상: VHDL 기반 FPGA 개발자
- 범위: 합성 가능한 Verilog RTL 중심
- 목표: 외부 IP와 RTL 설계 자산의 독립적 검토 및 유지보수

발표 멘트 / 설명 흐름

첫 멘트에서 “VHDL을 버리고 Verilog로 전환하는 교육이 아니다”라는 방어적 표현을 길게 쓰지 않는다. 대신 기존 경험을 활용한다는 방향을 긍정적으로 제시한다.

발표자는 문법 강사가 아니라 실무 FPGA 엔지니어의 관점에서 설명한다.

디자인/주의

표지는 문구를 최소화한다. 회사 로고나 발표자 정보는 하단에 작게 배치한다.

필수

익숙한 VHDL에서 Verilog로

이 슬라이드의 목적 청중이 이미 보유한 RTL 역량과 이번 세미나에서 연결할 표현을 대응시킨다.

화면에 넣을 내용

이미 익숙한 설계 개념	이번에 연결할 표현
Entity / Architecture	Module / Port
Signal / Process	Wire / Reg / Always
조합·순차논리	Assign / Clocked Always
Generic / 계층 설계	Parameter / Instantiation
Simulation / Synthesis	Verilog Testbench 기초

새로운 하드웨어 설계 방법이 아니라, 이미 알고 있는 RTL을 다른 문법으로 표현하는 방법

발표 멘트 / 설명 흐름

청중은 이미 디지털 회로와 FPGA 개발을 알고 있다. 세미나의 진입점은 “새로운 설계 이론”이 아니라 “기존 개념의 문법 연결”이다.

VHDL과 Verilog를 우열 관계로 설명하지 않는다.

디자인/주의

2열 대응표만 사용하고 설명 문장은 한 줄로 끝낸다.

필수

FPGA 세대가 바뀌면 IP 주변 설계 자산도 달라진다

이 슬라이드의 목적 FPGA 세대 변경 시 IP 자체뿐 아니라 예제, 래퍼, 테스트벤치의 제공 언어가 달라질 수 있음을 설명한다.

화면에 넣을 내용

- 7 Series에서 사용하던 IP와 후속 세대 IP는 이름과 구조가 달라질 수 있음
- 동일 기능군이라도 Example Design / Test Bench의 제공 언어가 유지된다고 가정할 수 없음
- IP 생성 버튼만 누르는 것과 제공 RTL을 조직의 설계 자산으로 관리하는 것은 다름

후속 FPGA 세대에서도 외부 설계 자산을 직접 검토할 수 있는 역량이 필요

발표 멘트 / 설명 흐름

AMD가 VHDL 지원을 중단한다고 확대 해석하지 않는다. 정확한 표현은 “일부 후속 세대 IP에서는 VHDL 예제와 테스트벤치가 제공되지 않는 사례가 있다”이다.

다음 장에서 일반적인 Memory Interface IP 사례를 보여준다.

디자인/주의

세대 변화 화살표를 단순하게 배치한다: 7 Series → UltraScale / UltraScale+.

필수

Memory Interface IP의 제공 언어 비교

이 슬라이드의 목적 방산 여부와 무관하게 널리 쓰는 메모리 IP에서 세대별 제공 언어 차이를 공식 문서로 보여준다.

화면에 넣을 내용

항목	7 Series Memory Interface	UltraScale Memory IP
Design Files	Verilog, VHDL (Top-level only)	RTL
Example Design	Verilog, VHDL (Top-level only)	Verilog
Test Bench	N/A	Verilog
실무 영향	VHDL 상위 예제를 기준으로 접근 가능	통합·검증 예제를 읽으려면 Verilog 필요

IP 사용 가능 여부보다 Example Design과 Test Bench를 이해할 수 있는지가 통합·디버깅 단계에서 중요

발표 멘트 / 설명 흐름

이 비교는 “같은 IP 버전에서 VHDL을 삭제했다”는 주장이 아니다. 동일 기능을 담당하는 후속 세대의 IP 제공 방식 비교다.

Example Design에는 Clock/Reset, 초기화 완료, 사용자 인터페이스, Traffic Generator 연결이 포함되므로 실제 통합의 출발점이 된다.

디자인/주의

공식 IP Facts 표 형태를 그대로 재현하면 신뢰도가 높다.

근거/출처 AMD DS176 v4.2 (2024-11-13) IP Facts; AMD PG150 v1.4 (2025-12-03) IP Facts.

필수

Verilog 자료만 제공될 때 무엇이 달라지는가

이 슬라이드의 목적 학습 목표를 단순한 “예제 읽기”가 아니라 조직의 기술적 통제 역량으로 정의한다.

화면에 넣을 내용

오해	실제 의미
전체 프로젝트를 Verilog로 바꿔야 한다	기존 VHDL 설계 자산은 유지 가능
IP 내부를 다시 개발해야 한다	제공된 IP와 Example Design을 필요한 범위에서 활용
Verilog 전문 개발자로 전환해야 한다	Verilog 기반 설계 자산을 검토·수정·유지보수할 역량 확보
IP가 생성되면 충분하다	통합, 변경, 디버깅, 장기 유지보수까지 사내에서 수행

IP를 사용할 수 있는 것과 해당 IP를 조직의 설계 자산으로 관리할 수 있는 것은 다르다

발표 멘트 / 설명 흐름

방산 제품은 장기 운용과 형상 관리가 중요하다. 공급업체나 원개발자에게 계속 의존하지 않으려면 상위 RTL과 테스트 구조를 내부에서 검토할 수 있어야 한다.

“예제와 오류를 읽는 능력”처럼 학원 실습 느낌의 표현은 피한다.

디자인/주의

오해/실제 의미 표는 최대 4행으로 제한한다.

필수

실무에서 Verilog 설계 자산을 접하는 상황

이 슬라이드의 목적 Verilog 학습이 실제 프로젝트의 어떤 순간에 필요한지 구체화한다.

화면에 넣을 내용

- AMD IP의 Example Design, Wrapper, Test Bench
- 외부 협력업체 또는 상용 IP에서 전달된 RTL
- 오픈소스 인터페이스 코어와 참조 설계
- 기존 제품의 유지보수 및 FPGA 세대 변경
- Simulation 오류, Width 경고, Reset/Clock 동작 분석

핵심은 신규 모듈 작성뿐 아니라 기존 RTL의 구조와 의도를 정확히 파악하는 것

발표 멘트 / 설명 흐름

청중이 실제로 경험한 사례를 한두 개 연결한다. 예: MIG example, AXI wrapper, GT reset helper.

“Verilog 코드를 많이 작성해야 한다”보다 “남이 작성한 코드를 책임지고 검토해야 한다”는 점을 강조한다.

디자인/주의

아이콘 5개 정도로 실무 상황을 시각화해도 좋다.

필수

이번 세미나의 범위와 도달 목표

이 슬라이드의 목적 본문과 Appendix의 범위를 분리하고 기대 수준을 명확히 한다.

화면에 넣을 내용

본문	Appendix / 후속 학습
Verilog Module과 데이터 표현	FPGA 설계 언어 통계
조합·순차논리	SystemVerilog의 장기 방향
FSM과 계층 설계	고급 검증 방법론(UVM/SVA)
자주 발생하는 오류	혼합 언어 세부 제약
기본 Testbench	고급 Verification IP

도달 목표: 외부 Verilog RTL을 구조적으로 읽고, 제한된 수정과 코드 리뷰를 수행할 수 있는 수준

발표 멘트 / 설명 흐름

SystemVerilog와 UVM은 본문을 방해하지 않도록 Appendix에 둔다.
세미나 종료 후 청중이 할 수 있어야 하는 작업을 명확히 말한다.

디자인/주의
1부의 마지막 장이므로 전체 목차를 간단히 보여주는 역할도 겸한다.

2부

2부. VHDL 관점에서 보는 Verilog RTL

익숙한 VHDL 구조를 기준으로 Verilog의 module, wire/reg, assign/always를 연결한다.

필수

같은 회로, 다른 표현

이 슬라이드의 목적 VHDL 요소와 Verilog 요소의 큰 대응 관계를 먼저 제시한다.

화면에 넣을 내용

VHDL	Verilog
entity + architecture	module
signal	wire 또는 reg
동시 할당문	assign
process	always
generic	parameter
port map	module instantiation

완전한 1:1 변환표가 아니라 코드를 읽기 위한 출발점

발표 멘트 / 설명 흐름

특히 signal ↔ wire/reg는 상황에 따라 달라진다고 바로 설명한다.
문법은 달라도 합성하려는 회로 개념은 동일하다.

디자인/주의
이 대응표는 이후 각 장에서 반복 참조할 수 있도록 단순하게 만든다.

필수

Entity와 Architecture는 Module 하나로 표현된다

이 슬라이드의 목적 가장 단순한 조합회로를 양쪽 언어로 비교해 module의 경계를 익힌다.

화면에 넣을 내용

VHDL	Verilog
entity and_gate is port(a,b : in std_logic; y : out std_logic); end; architecture rtl of and_gate is begin y <= a and b; end;	module and_gate (input wire a, input wire b, output wire y); assign y = a & b; endmodule

- module 선언부: 포트 인터페이스
- module 본문: 내부 신호, assign, always, 하위 모듈

발표 멘트 / 설명 흐름

VHDL은 인터페이스와 구현을 entity/architecture로 분리한다. Verilog는 module 안에서 함께 표현한다.

endmodule까지가 하나의 설계 단위다.

디자인/주의

코드 글자 크기를 무리하게 줄이지 말고 핵심 줄만 보여준다.

필수

Verilog도 동시에 동작하는 하드웨어를 기술한다

이 슬라이드의 목적 C 언어처럼 위에서 아래로 한 번 실행되는 코드라는 오해를 제거한다.

화면에 넣을 내용

```
assign y1 = a & b;

always @(*) begin
    y2 = c | d;
end

sub_module u_sub (
    .clk  (clk),
    .data (data)
);
```

요소	의미
assign	지속적으로 평가되는 동시 할당
always	이벤트에 의해 실행되는 절차 블록
module instance	독립적인 하위 회로

블록끼리는 동시에 동작하고, 한 블록 내부 문장은 순서대로 평가된다

발표 멘트 / 설명 흐름

이 문장이 Blocking/Nonblocking 설명의 기반이 된다.

Verilog의 procedural block은 프로그램처럼 보이지만 합성 결과는 회로다.

디자인/주의

세 개의 블록을 병렬 박스로 그려도 좋다.

필수

wire와 reg는 무엇인가

이 슬라이드의 목적 VHDL의 signal에 익숙한 청중이 가장 먼저 혼동하는 자료형을 정리한다.

화면에 넣을 내용

구분	wire	reg
주 용도	회로 간 연결	always/initial 내부 대입 대상
대입 위치	assign, module output	procedural block
저장 기능	자체 저장 기능 없음	이름만으로 저장소자 결정 안 됨
합성 회로	코드 구조에 따라 결정	코드 구조에 따라 결정

```
wire and_result;
reg  mux_result;

assign and_result = a & b;

always @(*) begin
    mux_result = sel ? a : b;
end
```

reg라고 선언했다고 Flip-Flop이 합성되는 것은 아니다

발표 멘트 / 설명 흐름

reg는 Verilog의 procedural variable 개념이다. 조합 always에서 reg를 써도 조합회로가 될 수 있다.

SystemVerilog의 logic은 Appendix에서 언급한다.

디자인/주의

reg라는 이름에 취소선을 긋는 식의 과한 연출보다 “문법적 변수형” 문구를 강조한다.

필수

조합논리 표현: assign과 always @(*)

이 슬라이드의 목적 동시 할당과 조합 process의 대응을 한 장에서 보여준다.

화면에 넣을 내용

단순 수식	조건문이 필요한 경우
assign y = sel ? a : b;	always @(*) begin if (sel) y = a; else y = b; end
VHDL: y <= a when sel='1' else b;	VHDL: process(all) + if/else

- assign 좌변은 일반적으로 wire
- always 내부 대입 대상은 reg
- 조합논리는 모든 경로에서 출력값을 결정

발표 멘트 / 설명 흐름

두 표현은 정상적으로 작성하면 같은 MUX로 합성될 수 있다.
Latch 원리는 4부에서 VHDL과 비교해 다시 설명한다.

디자인/주의
좌우 비교 레이아웃이 가장 직관적이다.

필수

순차논리 표현: Clock Event를 지정한 always

이 슬라이드의 목적 rising_edge와 posedge, 그리고 순차논리의 기본 틀을 연결한다.

화면에 넣을 내용

VHDL	Verilog
process(clk) begin if rising_edge(clk) then if rst='1' then q<='0'; else q<=d; end if; end if; end process;	always @(posedge clk) begin if (rst) q <= 1'b0; else q <= d; end

- rising_edge(clk) → posedge clk
- falling_edge(clk) → negedge clk
- 순차논리에서는 일반적으로 <= 사용

발표 멘트 / 설명 흐름

VHDL의 <=와 모양은 같지만, Verilog에서는 Nonblocking Assignment라는 별도 의미가 있다.

5부에서 이벤트 큐와 레지스터 병렬 업데이트를 설명한다.

디자인/주의

이 장에서는 Reset 종류를 깊게 다루지 않는다.

필수

Parameter와 Module Instantiation

이 슬라이드의 목적 generic과 port map의 대응을 익히고, 외부 RTL을 읽는 순서를 제시한다.

화면에 넣을 내용

```
module counter #(
    parameter WIDTH = 8
)(
    input  wire clk,
    input  wire rst,
    output reg  [WIDTH-1:0] count
);
...
endmodule

counter #(
    .WIDTH(16)
) u_counter (
    .clk   (clk),
    .rst   (rst),
    .count (count)
);
```

VHDL	Verilog
generic	parameter
generic map	#(...)
port map	(...)
clk => clk	.clk(clk)

읽는 순서: Port → Parameter → 내부 신호 → assign/always → 하위 모듈

발표 멘트 / 설명 흐름

점 앞은 하위 모듈 포트 이름, 괄호 안은 상위 신호 이름이다.

Named port connection을 기본으로 권장하고 positional connection의 위험은 6부에서 설명한다.

디자인/주의

코드 전체보다 #(...)와 .port(signal) 부분을 색으로 강조한다.

3부

3부. 기본 문법과 데이터 표현

한 줄의 Verilog 표현을 정확하게 해석하기 위한 폭, 벡터, 연산, signed 규칙을 익힌다.

필수

문장의 기본 규칙과 파일 확장자

이 슬라이드의 목적 Verilog 문법의 표면적 차이를 짧게 정리한다.

화면에 넣을 내용

- 대소문자를 구분: data, Data, DATA는 서로 다른 이름
- 대부분의 문장은 세미콜론 ; 으로 종료
- 한 줄 주석 //, 여러 줄 주석 /* ... */
- 여러 문장은 begin / end로 묶음
- .v는 일반적으로 Verilog, .sv는 SystemVerilog

```
wire [7:0] data;
assign data = 8'hA5;
```

발표 멘트 / 설명 흐름

외부 RTL을 수정할 때 대소문자 실수는 연결 오류나 implicit net 문제로 이어질 수 있다.
Vivado에서는 .v를 Verilog 2001, .sv를 SystemVerilog로 처리하는 것이 기본이다.

디자인/주의

문법 규칙을 한 장 이상 늘리지 않는다.

근거/출처 AMD UG901 Vivado Synthesis: .v and .sv file type behavior.

필수

숫자는 폭과 진법을 함께 표현한다

이 슬라이드의 목적 Verilog literal을 정확히 읽는 습관을 만든다.

화면에 넣을 내용

<비트 수>'<진법><값>

```
8'b1010_0101
8'hA5
8'd165
8'o245
1'b0
16'hFFFF
```

기호	진법
b	Binary
o	Octal
d	Decimal
h	Hexadecimal

- _ 는 가독성을 위한 구분자
- 상수의 값뿐 아니라 비트 폭이 연산 결과에 영향

발표 멘트 / 설명 흐름

VHDL의 x"A5", to_unsigned(165,8)와 비교한다.

Unsized constant는 폭과 signed 규칙 때문에 예상과 다른 결과를 만들 수 있으므로 코드 리뷰에서 확인한다.

디자인/주의

숫자 예시는 실제 ADC, 카운터 폭과 연결하면 좋다.

필수

Scalar, Vector, Bit Select, Part Select

이 슬라이드의 목적 벡터 선언과 슬라이스 문법을 한 장에서 익힌다.

화면에 넣을 내용

```
wire        valid;
wire [13:0] adc_data;
wire [7:0]  upper_byte;

assign upper_byte = adc_data[13:6];
assign lsb        = adc_data[0];
```

표현	의미
data[7:0]	8비트, 7이 MSB
data[0:7]	8비트, 인덱스 방향 반대
data[3]	1비트 선택
data[7:4]	4비트 부분 선택

[7:0]과 [0:7]은 폭은 같지만 인덱스 방향이 다르다

발표 멘트 / 설명 흐름

VHDL의 std_logic_vector(7 downto 0), data(7 downto 4)와 직접 대응한다.
외부 IP에서 [0:N] 방향 벡터를 만나면 연결 순서를 반드시 확인한다.

디자인/주의
MSB/LSB 방향을 화살표로 표시한다.

필수

Concatenation과 Replication

이 슬라이드의 목적 여러 신호를 묶고 sign extension을 만드는 문법을 익힌다.

화면에 넣을 내용

```
assign data16 = {data_high, data_low};

assign zero_ext = {8'b0, data8};
assign sign_ext = {{8{data8[7]}}, data8};

wire [3:0] ones = {4{1'b1}};
```

- {a,b}: 왼쪽 항목이 상위 비트
- {N{pattern}}: 패턴을 N번 반복
- Sign extension, channel packing, header 생성에 자주 사용

발표 멘트 / 설명 흐름

중괄호는 C 언어의 block이 아니라 bit 연결 연산이다.

중첩된 {{...{...}}, ...} 표현은 처음 보면 어렵지만 sign bit 반복으로 해석하면 된다.

디자인/주의

16'h1234가 {8'h12,8'h34}로 만들어지는 그림을 넣으면 좋다.

필수

논리 연산과 비트 연산을 구분한다

이 슬라이드의 목적 비슷한 기호를 정확히 구분한다.

화면에 넣을 내용

구분	연산자	예
산술	+ - * / %	a + b
비교	== != < <= > >=	a == b
논리	&& || !	valid && ready
비트	& | ^ ~	a & mask
Shift	<< >>	data >> 2
조건	?:	sel ? a : b

a && b는 참/거짓 판단, a & b는 각 비트끼리 연산

발표 멘트 / 설명 흐름

1비트 신호에서는 결과가 같아 보일 수 있어 혼동이 더 크다.

!와 ~도 벡터에서 결과가 크게 다르다.

디자인/주의

연산자 표는 단순하게 유지하고 우선순위 전체 표는 Appendix로 보낸다.

필수

Reduction과 Equality

이 슬라이드의 목적 벡터 전체를 한 비트로 줄이는 연산과 X/Z 비교를 구분한다.

화면에 넣을 내용

```
assign all_one = &data;
assign any_one = |data;
assign parity  = ^data;

if (a == b)   ...
if (a === b)  ...   // 주로 Testbench
```

표현	의미
&data;	모든 비트 AND
|data	한 비트라도 1이면 1
^data	XOR parity
== / !=	일반 비교, X/Z 포함 시 결과 X 가능
=== / !==	X/Z까지 값으로 비교, 결과 0 또는 1

합성 RTL 일반 비교는 ==, X/Z 포함 검사는 주로 Testbench에서 ===

발표 멘트 / 설명 흐름

&data;와 a&b;를 구분한다. 피연산자가 한 개인지 두 개인지 보면 된다.

Case equality를 RTL에 남용하면 X 문제를 숨길 수 있다.

디자인/주의

Testbench 표시는 별도 색으로 처리한다.

필수

Signed와 Unsigned

이 슬라이드의 목적 같은 비트 패턴이 부호 속성에 따라 다른 값이 된다는 점을 강조한다.

화면에 넣을 내용

```
wire        [7:0] data_u;
wire signed [7:0] data_s;

// 8'b1111_1111
// unsigned = 255
// signed   = -1
```

VHDL	Verilog
unsigned(7 downto 0)	wire [7:0]
signed(7 downto 0)	wire signed [7:0]

- 기본 Vector는 unsigned
- 연산 피연산자 중 unsigned가 섞이면 해석이 달라질 수 있음
- 비교, Shift, 곱셈 결과 폭을 함께 확인

발표 멘트 / 설명 흐름

signed 선언만으로 모든 연산이 자동으로 안전해지는 것은 아니다.
상수의 signed 여부와 중간 연산 폭을 같이 봐야 한다.

디자인/주의
2의 보수 비트 패턴을 간단한 표로 보여준다.

필수

폭 확장, 절단, Parameter 기반 폭

이 슬라이드의 목적 자동 폭 변환에 의존하지 않고 의도를 코드에 표시하는 습관을 만든다.

화면에 넣을 내용

```
wire [7:0]  data8;
wire [15:0] data16;

assign data16 = {8'b0, data8};
assign data8  = data16[7:0];

module data_path #(
    parameter DATA_WIDTH = 14
)(
    input wire [DATA_WIDTH-1:0] data_in
);
```

- 큰 폭 → 작은 폭: 상위 비트 절단
- 작은 폭 → 큰 폭: zero/sign extension 규칙 확인
- 폭이 parameter면 [WIDTH-1:0]을 실제 숫자로 환산해서 읽기

자동 변환보다 확장·절단 의도를 코드에 명시

발표 멘트 / 설명 흐름

폭 불일치는 컴파일 오류가 아니라 경고만 발생하고 합성될 수 있다.
3부 마무리에서 “폭, 방향, signed” 세 가지를 항상 확인하도록 정리한다.

디자인/주의
마지막에 짧은 읽기 문제를 넣어도 좋다: {{2{adc[13]}},adc}.

4부

4부. 조합논리

assign과 always @(*)를 구분하고 Latch가 발생하지 않는 완전한 조합논리를 작성한다.

필수

조합논리의 조건

이 슬라이드의 목적 Verilog 문법보다 먼저 조합논리의 하드웨어 조건을 재확인한다.

화면에 넣을 내용

현재 입력만으로 출력이 결정되어야 한다

- Clock을 사용하지 않음
- 이전 출력값을 기억하지 않음
- 입력이 변하면 출력이 다시 결정됨
- 모든 입력 조건에서 출력값이 정의되어야 함
- 대표 회로: MUX, Decoder, Comparator, Adder, Priority Encoder

발표 멘트 / 설명 흐름

일부 조건에서 출력값을 결정하지 않으면 “값 유지”가 필요하고, 그 결과 Latch가 합성될 수 있다.

Latch 문제는 언어가 아니라 회로 기술의 완전성 문제다.

디자인/주의

Y=f(A,B,C)의 간단한 블록 그림을 사용한다.

필수

단순한 식은 assign과 조건 연산자

이 슬라이드의 목적 연속 할당과 간단한 MUX 표현을 익힌다.

화면에 넣을 내용

```
assign y        = a & b;
assign sum      = a + b;
assign mux_out  = sel ? b : a;
assign masked   = data & mask;
```

- assign은 항상 활성화된 동시 할당
- 우변 입력이 바뀌면 좌변이 다시 계산
- 좌변은 일반적으로 wire
- 단순 수식과 짧은 MUX에 적합

발표 멘트 / 설명 흐름

VHDL 동시 할당과 가장 가까운 표현이다.

?: 중첩은 우선순위 구조와 가독성 문제를 만들 수 있으므로 복잡해지면 always로 옮긴다.

디자인/주의

assign 문 3~4개를 한 화면에 배치하고 회로 아이콘을 옆에 둔다.

필수

복잡한 조건은 always @(*)

이 슬라이드의 목적 if/case가 필요한 조합논리의 기본 틀을 익힌다.

화면에 넣을 내용

```
reg [7:0] y;

always @(*) begin
    if (sel)
        y = b;
    else
        y = a;
end
```

- @(*)가 블록에서 참조하는 입력을 감지
- 출력은 reg로 선언
- 조합논리 대입은 일반적으로 = 사용
- 모든 실행 경로에서 출력값을 결정

발표 멘트 / 설명 흐름

reg 선언이 Latch나 Flip-Flop을 의미하지 않는다. 코드 경로가 회로를 결정한다.
SystemVerilog의 always_comb은 Appendix에서만 소개한다.

디자인/주의
VHDL process(all)과 나란히 비교한다.

필수

조합논리에서는 Blocking Assignment

이 슬라이드의 목적 블록 내부 계산 순서를 명확히 하기 위해 =를 사용하는 이유를 설명한다.

화면에 넣을 내용

```
always @(*) begin
    temp   = a + b;
    result = temp >> 1;
end
```

조합 always에는 =, 순차 always에는 <= 를 기본 규칙으로 사용

코드	평가
temp = a+b	즉시 temp 갱신
result = temp>>1	갱신된 temp 사용

발표 멘트 / 설명 흐름

Nonblocking을 조합논리에 사용하면 이벤트 순서 때문에 읽기와 시뮬레이션이 어려워진다.
합성기가 같은 회로를 만들 수 있는 경우가 있어도 코딩 규칙은 일관되게 유지한다.

디자인/주의
순서 화살표를 넣어 즉시 갱신을 시각화한다.

필수

if-else는 우선순위를 만들 수 있다

이 슬라이드의 목적 조건문 순서가 Priority Encoder 구조로 이어지는 것을 보여준다.

화면에 넣을 내용

```
always @(*) begin
    if (req0)
        grant = 2'd0;
    else if (req1)
        grant = 2'd1;
    else if (req2)
        grant = 2'd2;
    else
        grant = 2'd3;
end
```

동시에 여러 조건이 참이면 가장 먼저 나온 조건이 우선

- req0 > req1 > req2 > default
- Priority가 의도인지 우연인지 코드 리뷰에서 확인

발표 멘트 / 설명 흐름

if는 복합 조건과 우선순위에 적합하다.

조건이 상호 배타적이면 case가 구조를 더 명확하게 보여줄 수 있다.

디자인/주의

Priority MUX 그림을 함께 넣는다.

필수

case와 default

이 슬라이드의 목적 선택값 기반 분기와 완전한 할당을 설명한다.

화면에 넣을 내용

```
always @(*) begin
    case (sel)
        2'b00: y = a;
        2'b01: y = b;
        2'b10: y = c;
        default: y = d;
    endcase
end
```

상황	권장
우선순위 조건	if-else
하나의 선택값 비교	case
MUX, Decoder, FSM next-state	case
범위·복합 조건	if

case의 누락 조건은 default 또는 블록 시작 기본값으로 처리

발표 멘트 / 설명 흐름

VHDL case는 모든 경우 처리를 강하게 요구하지만 Verilog case는 불완전 분기를 문법적으로 허용한다.

이 차이 때문에 Verilog에서 Latch를 더 쉽게 놓칠 수 있다.

디자인/주의

default 행만 색으로 강조한다.

필수

Latch: VHDL과 Verilog의 원리는 같다

이 슬라이드의 목적 청중의 VHDL 경험과 연결해 불완전 할당의 의미를 정확히 설명한다.

화면에 넣을 내용

VHDL 조합 Process	Verilog 조합 Always
process(all) begin if en='1' then y <= data; end if; end process;	always @(*) begin if (en) y = data; end

en=0에서 y가 대입되지 않음 → 이전 값 유지 → Latch 필요

위치	else 생략 결과
조합 process / always @(*)	Latch 가능
clocked process / always @(posedge clk)	Flip-Flop 값 유지, Clock Enable

발표 멘트 / 설명 흐름

VHDL에서 else 없이 괜찮았던 경험은 대개 clocked process이거나 process 시작에서 기본값을 넣은 경우다.

Sensitivity list가 완전해도 출력 할당이 불완전하면 Latch는 그대로 발생한다.

디자인/주의

이 장은 4부의 핵심 차이점처럼 보이지만 결론은 “언어 차이보다 작성 완전성 차이”다.

필수

안전한 패턴: 기본값을 먼저 대입한다

이 슬라이드의 목적 else와 default 누락을 줄이는 실무 패턴을 제시한다.

화면에 넣을 내용

```
always @(*) begin
    next_state = state;
    count_en   = 1'b0;
    error      = 1'b0;

    case (state)
        IDLE: if (start) begin
            next_state = RUN;
            count_en   = 1'b1;
        end
        RUN: begin
            count_en = 1'b1;
            if (done) next_state = IDLE;
        end
        default: begin
            next_state = IDLE;
            error = 1'b1;
        end
    endcase
end
```

- 1. 모든 출력에 기본값
- 2. 조건에 따라 필요한 값만 덮어쓰기
- 3. case에는 default 추가

발표 멘트 / 설명 흐름

FSM next-state, enable 생성, output decode에 특히 유용하다.

기본값이 자기 자신(next_state=state)인지 0인지 회로 의도에 따라 선택한다.

디자인/주의

코드가 길어 보이면 기본값 부분과 override 부분만 색으로 구분한다.

필수

조합논리 코드 리뷰 체크리스트

이 슬라이드의 목적 Sensitivity, Driver, X 처리까지 한 장에서 실무 점검 항목으로 정리한다.

화면에 넣을 내용

- always @(*)를 사용했는가?
- 모든 출력에 기본값 또는 완전한 분기가 있는가?
- 조합 always에서 =를 사용했는가?
- if 순서가 의도한 우선순위인가?
- case default가 있는가?
- 하나의 신호를 여러 assign/always가 구동하지 않는가?
- casex로 X 오류를 숨기지 않는가?
- 폭과 signed 속성을 함께 확인했는가?

Latch 경고는 “툴이 까다로운 것”이 아니라 코드가 값 유지를 요구한다는 신호

발표 멘트 / 설명 흐름

과거형 sensitivity list에서 입력 누락은 Simulation과 Synthesis 불일치를 만들 수 있다.
casex는 합성 RTL에서 가급적 피하고 casez도 제한적으로 사용한다.

디자인/주의
체크박스 형태로 구성해 코드 리뷰 장표로 재사용한다.

5부

5부. 순차논리

Clock Edge, Reset, Clock Enable, Nonblocking Assignment를 레지스터 동작과 연결한다.

필수

순차논리는 Clock Edge에서 상태를 저장한다

이 슬라이드의 목적 조합논리와 순차논리의 하드웨어 차이를 명확하게 구분한다.

화면에 넣을 내용

Q(next) = f(Q(current), Input) at Clock Edge

- Clock Edge에서만 값이 갱신
- 이전 상태를 저장하는 Flip-Flop/Register 사용
- Reset, Set, Clock Enable과 결합 가능
- Counter, Shift Register, Pipeline, FSM state register에 사용

발표 멘트 / 설명 흐름

Verilog always 문법보다 먼저 “언제 값이 바뀌는가”를 묻는다.
순차논리 블록에서 대입이 없는 경로는 Latch가 아니라 레지스터의 값 유지가 된다.

디자인/주의
Clock 파형과 Q 업데이트 지점을 간단히 그린다.

필수

posedge와 negedge

이 슬라이드의 목적 Clock 및 비동기 제어 신호를 sensitivity list에서 읽는 법을 익힌다.

화면에 넣을 내용

```
always @(posedge clk) begin
    q <= d;
end

always @(negedge clk) begin
    q_n <= d;
end
```

표현	의미
posedge clk	0→1 상승 Edge
negedge clk	1→0 하강 Edge
posedge rst	Reset 상승 시 즉시 반응
negedge rst_n	Active-low Reset 하강 시 즉시 반응

발표 멘트 / 설명 흐름

Clock Edge 하나만 있으면 기본 레지스터다.

Clock과 Reset이 sensitivity list에 함께 있으면 비동기 Reset 여부를 확인해야 한다.

디자인/주의

Sensitivity list를 코드에서 강조한다.

필수

Nonblocking Assignment의 의미

이 슬라이드의 목적 여러 레지스터가 같은 Clock Edge에서 동시에 갱신되는 동작을 이해시킨다.

화면에 넣을 내용

```
always @(posedge clk) begin
    q1 <= d;
    q2 <= q1;
    q3 <= q2;
end
```

오른쪽 값은 Clock Edge 직전 값을 읽고, 왼쪽 갱신은 블록 평가 후 반영

- q1, q2, q3는 같은 Edge에서 병렬로 갱신
- 3단 Pipeline / Shift Register로 합성

발표 멘트 / 설명 흐름

코드가 위에서 아래로 있으므로 q2가 새 q1을 받을 것처럼 보이지만 실제로는 이전 q1을 받는다.

이 특성이 레지스터 병렬 동작을 자연스럽게 표현한다.

디자인/주의

Clock 전후 값 표를 함께 보여준다.

필수

Blocking과 Nonblocking을 바꾸면 무엇이 달라지는가

이 슬라이드의 목적 잘못된 = 사용이 시뮬레이션에서 Pipeline을 붕괴시키는 사례를 보여준다.

화면에 넣을 내용

권장 코드	위험한 코드
always @(posedge clk) begin q1 <= d; q2 <= q1; end	always @(posedge clk) begin q1 = d; q2 = q1; end
q2는 이전 q1	q2가 갱신된 q1을 즉시 사용

순차논리에는 <= 를 기본 규칙으로 사용

발표 멘트 / 설명 흐름

일부 합성 결과가 예상과 비슷해도 simulation race와 코드 의도 문제가 생긴다.
한 블록 안에서 임시 계산용 변수를 쓰는 고급 패턴은 본 세미나 범위 밖으로 둔다.

디자인/주의
좌우 코드와 2단 Pipeline 그림을 나란히 배치한다.

필수

Synchronous Reset

이 슬라이드의 목적 Clock Edge에서만 적용되는 Reset 구조를 설명한다.

화면에 넣을 내용

```
always @(posedge clk) begin
    if (rst)
        q <= 1'b0;
    else
        q <= d;
end
```

- rst가 변해도 Clock Edge 전에는 q가 변하지 않음
- Clock Enable/데이터 MUX 구조와 함께 합성
- Reset deassert timing 관리가 상대적으로 단순

발표 멘트 / 설명 흐름

Reset이 sensitivity list에 없고 블록 내부 if에만 있으면 synchronous reset이다.
프로젝트 정책과 소자 자원에 따라 reset 방식이 결정되므로 언어 문법만으로 우열을 말하지 않는다.

디자인/주의
Clock 파형과 reset assertion 시점의 q 변화를 보여준다.

필수

Asynchronous Reset

이 슬라이드의 목적 Clock과 무관하게 즉시 동작하는 Reset 구조를 설명한다.

화면에 넣을 내용

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        q <= 1'b0;
    else
        q <= d;
end
```

- rst_n이 0이 되면 Clock과 무관하게 즉시 q Reset
- assert는 비동기, deassert는 동기화가 중요
- Active-low 신호는 이름에 _n을 붙이는 규칙 권장

발표 멘트 / 설명 흐름

비동기 Reset의 release가 Clock Edge 근처에서 발생하면 recovery/removal 문제가 생길 수 있다.

실무에서는 “asynchronous assert, synchronous deassert” 구조를 자주 고려한다.

디자인/주의

Reset assertion/release 화살표를 구분한다.

필수

Clock Enable: else가 없어도 정상인 경우

이 슬라이드의 목적 4부의 Latch 논의와 연결해 clocked block에서의 값 유지를 설명한다.

화면에 넣을 내용

VHDL	Verilog
process(clk) begin if rising_edge(clk) then if en='1' then q<=d; end if; end if; end process;	always @(posedge clk) begin if (en) q <= d; end

en=0이면 기존 Flip-Flop 값을 유지 - 의도된 Clock Enable

- 조합 block의 미할당 → Latch
- clocked block의 미할당 → 기존 Register 유지

발표 멘트 / 설명 흐름

청중의 “VHDL에서는 else가 없어도 괜찮았다”는 경험을 정확히 설명하는 장이다.

Clock Enable을 Clock gating과 혼동하지 않도록 한다. FPGA에서는 전용 CE 핀을 활용하는 경우가 많다.

디자인/주의

4부 Slide 29와 동일한 레이아웃을 사용해 연결감을 준다.

필수

Counter와 Overflow

이 슬라이드의 목적 가장 기본적인 순차회로에서 폭과 리셋을 함께 점검한다.

화면에 넣을 내용

```
reg [7:0] count;

always @(posedge clk) begin
    if (rst)
        count <= 8'd0;
    else if (en)
        count <= count + 8'd1;
end
```

- 8비트 Counter는 255 다음 0으로 Wrap-around
- Carry가 필요하면 결과 폭을 9비트로 확장
- Unsized 1보다 폭을 명시한 8'd1 사용 권장
- Enable과 Reset 우선순위를 코드 순서로 확인

발표 멘트 / 설명 흐름

Counter 코드는 간단하지만 width, overflow, reset priority를 모두 보여준다.
Saturation counter인지 modulo counter인지 요구사항을 분명히 한다.

디자인/주의
0→255→0 타임라인을 간단히 표시한다.

필수

Shift Register와 Pipeline

이 슬라이드의 목적 Nonblocking Assignment가 실제 데이터 지연을 어떻게 표현하는지 보여준다.

화면에 넣을 내용

```
reg [13:0] d1, d2, d3;

always @(posedge clk) begin
    if (rst) begin
        d1 <= 14'd0;
        d2 <= 14'd0;
        d3 <= 14'd0;
    end else begin
        d1 <= adc_data;
        d2 <= d1;
        d3 <= d2;
    end
end
```

d3는 adc_data보다 3 Clock 늦은 데이터

- Timing closure를 위한 Pipeline
- Valid 신호도 동일한 지연만큼 함께 Pipeline

발표 멘트 / 설명 흐름

데이터만 지연하고 valid/enable을 지연하지 않으면 시스템 수준 버그가 된다.
파이프라인 단계 수가 latency 요구사항에 미치는 영향을 언급한다.

디자인/주의
레지스터 3개와 데이터 이동 화살표를 그린다.

필수

순차논리 코드 리뷰 체크리스트

이 슬라이드의 목적 Clock, Reset, Assignment, Enable을 실무 점검 항목으로 정리한다.

화면에 넣을 내용

- Clock Edge가 의도한 posedge/negedge인가?
- 순차 대입에 <=를 사용했는가?
- Reset이 synchronous인지 asynchronous인지 명확한가?
- Active-high/low 극성과 신호 이름이 일치하는가?
- Reset 우선순위와 Clock Enable 우선순위가 요구사항과 같은가?
- Pipeline 데이터와 valid/control의 지연이 일치하는가?
- Counter overflow 방식이 의도된 것인가?
- 한 Register가 하나의 always block에서만 구동되는가?

발표 멘트 / 설명 흐름

이 장을 실제 사내 코드 리뷰 체크리스트로 재사용할 수 있다.

CDC, Reset Domain Crossing은 별도 전문 주제이므로 여기서는 위험 신호만 언급한다.

디자인/주의

체크박스 형식으로 구성한다.

6부

6부. FSM과 계층 설계

FSM과 Module 계층을 구조적으로 읽고 안전한 상태 전이와 연결 방식을 익힌다.

필수

FSM은 상태 저장과 조합논리의 결합이다

이 슬라이드의 목적 FSM을 완전히 새로운 문법이 아니라 이미 배운 조합/순차 블록의 조합으로 이해시킨다.

화면에 넣을 내용

State Register + Next-State Logic + Output Logic

블록	역할
State Register	현재 상태 저장, clocked always
Next-State Logic	입력과 현재 상태로 다음 상태 계산, combinational always
Output Logic	상태/입력으로 출력 생성

- Moore: 출력이 상태에 주로 의존
- Mealy: 출력이 상태와 입력에 의존

발표 멘트 / 설명 흐름

VHDL의 2-process FSM과 구조적으로 같다.
첫 FSM 예제는 IDLE-RUN-DONE 정도의 단순한 제어 흐름으로 한다.

디자인/주의
세 블록 다이어그램을 중심에 둔다.

필수

상태 정의: parameter와 localparam

이 슬라이드의 목적 Verilog에서 상태 값을 정의하는 기본 방식을 설명한다.

화면에 넣을 내용

```
localparam [1:0] IDLE  = 2'b00;
localparam [1:0] RUN   = 2'b01;
localparam [1:0] DONE  = 2'b10;
localparam [1:0] ERROR = 2'b11;

reg [1:0] state;
reg [1:0] next_state;
```

- localparam은 외부에서 override하지 않는 내부 상수
- 상태 폭과 encoding을 명시
- SystemVerilog enum은 Appendix에서 소개

발표 멘트 / 설명 흐름

parameter를 상태에 써도 되지만 외부 override가 필요하지 않으므로 localparam이 더 명확하다.

Binary, one-hot encoding은 툴 옵션과 요구사항에 따라 달라질 수 있다.

디자인/주의

상태 이름과 비트값을 표로 보조한다.

필수

State Register

이 슬라이드의 목적 현재 상태를 저장하는 순차 블록을 분리한다.

화면에 넣을 내용

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        state <= IDLE;
    else
        state <= next_state;
end
```

- Reset 시 안전한 초기 상태로 진입
- 매 Clock마다 계산된 next_state 저장
- 순차 블록에는 상태 전이 조건을 복잡하게 넣지 않음

발표 멘트 / 설명 흐름

상태 저장 블록은 짧고 단순하게 유지한다.

비동기 reset을 사용한다면 release 동기화 정책도 프로젝트 차원에서 정의해야 한다.

디자인/주의

코드 오른쪽에 State Register 하나만 그린다.

필수

Next-State Logic

이 슬라이드의 목적 기본값과 case를 이용한 안전한 상태 전이 작성법을 보여준다.

화면에 넣을 내용

```
always @(*) begin
    next_state = state;

    case (state)
        IDLE:  if (start) next_state = RUN;
        RUN: begin
            if (fault)      next_state = ERROR;
            else if (done)  next_state = DONE;
        end
        DONE:  next_state = IDLE;
        ERROR: if (clear) next_state = IDLE;
        default: next_state = IDLE;
    endcase
end
```

- 기본값 next_state = state
- 필요한 전이 조건만 override
- default에서 안전 상태 복귀

발표 멘트 / 설명 흐름

조합논리의 기본값 패턴을 FSM에 그대로 적용한 것이다.

RUN 상태에서 fault와 done의 우선순위를 코드 순서로 확인한다.

디자인/주의

State diagram과 코드의 각 분기를 색으로 연결한다.

필수

Output Logic: Moore와 Mealy

이 슬라이드의 목적 출력 의존성에 따른 타이밍과 코드 구조 차이를 설명한다.

화면에 넣을 내용

구분	Moore	Mealy
출력 의존	상태	상태 + 입력
반응	상태 변경 후	입력 변화에 즉시 반응 가능
주의	한 Clock 지연 가능	Glitch/조합 경로 주의

```
always @(*) begin
    busy = 1'b0;
    done_pulse = 1'b0;
    case (state)
        RUN:  busy = 1'b1;
        DONE: done_pulse = 1'b1;
    endcase
end
```

발표 멘트 / 설명 흐름

출력이 외부 인터페이스로 나가거나 Timing에 민감하면 Registering을 고려한다.

Moore/Mealy를 암기보다 “출력이 무엇에 의존하는가”로 판단한다.

디자인/주의

표와 짧은 출력 코드만 사용한다.

필수

FSM 전체 코드를 읽는 순서

이 슬라이드의 목적 외부 FSM RTL을 빠르게 파악하는 실무 순서를 제시한다.

화면에 넣을 내용

- 1. 상태 정의와 state/next_state 폭 확인
- 2. Reset 시 진입 상태 확인
- 3. 각 상태의 전이 조건과 우선순위 확인
- 4. default 또는 illegal state recovery 확인
- 5. Moore/Mealy 출력 의존성 확인
- 6. pulse 출력이 한 Clock인지 level인지 확인
- 7. timeout, fault, abort 경로 확인

상태도와 코드가 일치하는지, 안전 상태 복귀가 가능한지 검토

발표 멘트 / 설명 흐름

방산 제어 로직에서는 정상 경로뿐 아니라 fault/timeout 복구 경로가 중요하다.
State encoding보다 전이 완전성과 출력 안전성을 먼저 본다.

디자인/주의
리뷰 순서를 번호가 큰 카드 형태로 배치한다.

필수

Module Instantiation: Named 연결을 기본으로

이 슬라이드의 목적 계층 설계에서 포트 연결 실수를 줄이는 규칙을 제시한다.

화면에 넣을 내용

Named Connection	Positional Connection
sub u_sub (.clk(clk), .rst(rst), .data_in(data));	sub u_sub (clk, rst, data);
포트 이름으로 연결	선언 순서에 의존

실무 RTL에서는 Named Port Connection을 기본으로 권장

- 포트 추가/순서 변경 시 영향 감소
- 코드 리뷰와 연결 추적이 쉬움

발표 멘트 / 설명 흐름

외부 IP wrapper는 포트가 많으므로 positional 방식은 특히 위험하다.
미연결 포트는 .port()로 의도를 명시할 수 있다.

디자인/주의
Named 연결의 점(.)을 강조한다.

필수

Parameter Override와 Generate

이 슬라이드의 목적 재사용 가능한 모듈을 폭과 채널 수에 맞춰 확장하는 방법을 소개한다.

화면에 넣을 내용

```
filter #(
    .DATA_WIDTH(14),
    .TAP_NUM(8)
) u_filter ( ... );

genvar i;
generate
    for (i=0; i<4; i=i+1) begin : GEN_CH
        channel u_ch (
            .din  (din[i]),
            .dout (dout[i])
        );
    end
endgenerate
```

- Parameter: 폭, 깊이, 기능 옵션 설정
- Generate-for: 동일 구조 반복 생성
- Generate-if: 조건에 따른 하드웨어 구성 선택

발표 멘트 / 설명 흐름

generate는 실행 중 Loop가 아니라 elaboration 단계에서 하드웨어 인스턴스를 만드는 문법이다.

생성 블록 이름 GEN_CH가 hierarchy path에 포함될 수 있다.

디자인/주의

반복되는 4개 채널 블록을 시각화한다.

필수

계층 구조를 읽는 순서

이 슬라이드의 목적 대형 외부 RTL에서 top-down 분석 절차를 제시한다.

화면에 넣을 내용

- Top module의 Clock, Reset, 주요 I/O 식별
- Parameter와 주요 폭 확인
- 하위 모듈 인스턴스 목록 작성
- 데이터 경로와 제어 경로 분리
- Clock domain별 블록 구분
- Generated hierarchy 이름 확인
- Black-box/IP 경계와 사용자 수정 가능 영역 구분

한 줄씩 읽기보다 인터페이스와 계층을 먼저 지도처럼 만든다

발표 멘트 / 설명 흐름

외부 IP 예제를 볼 때 core 파일 자체보다 wrapper, reset helper, clock helper를 먼저 파악한다.

수정 가능한 example design과 수정하면 안 되는 generated core 영역을 구분한다.

디자인/주의

Top-down tree diagram을 사용한다.

7부

7부. 실무 오류와 코드 리뷰

느슨한 Verilog 규칙에서 발생하는 오류를 로그와 코드 리뷰로 조기에 발견한다.

필수

Implicit Net: 오타가 새로운 신호가 될 수 있다

이 슬라이드의 목적 Verilog의 느슨한 기본 규칙이 실무 오류로 이어지는 대표 사례를 설명한다.

화면에 넣을 내용

```
wire data_valid;
assign out_valid = data_vaild;  // 오타
```

- 일부 Verilog 설정에서는 선언하지 않은 이름이 암시적 wire로 생성될 수 있음
- 컴파일은 통과하지만 연결이 끊어진 기능 오류 가능

```
`default_nettype none

module top (...);
...
endmodule

`default_nettype wire
```

파일 또는 프로젝트 코딩 규칙에서 implicit net을 금지

발표 멘트 / 설명 흐름

VHDL은 미선언 식별자 오류가 명확해 이런 실수가 상대적으로 적다.

외부 RTL에 default_nettype none을 추가할 때는 다른 파일 영향 범위를 확인한다.

디자인/주의

오타 글자 한 글자를 빨간색으로 표시한다.

필수

Width와 Signed 경고를 무시하지 않는다

이 슬라이드의 목적 합성은 되지만 기능이 달라질 수 있는 경고를 우선 검토한다.

화면에 넣을 내용

```
wire [7:0] a, b;
wire [7:0] sum8;
wire [8:0] sum9;

assign sum8 = a + b;          // carry 절단
assign sum9 = {1'b0,a}+{1'b0,b};
```

- Port width mismatch
- Truncation / extension
- Signed/unsigned comparison
- Shift result width
- Parameter 계산 폭

Warning가 많을수록 정상 경고와 실제 결함을 구분하기 어려워진다

발표 멘트 / 설명 흐름

Warning zero가 목표라는 뜻은 아니지만, 의도된 경고는 근거와 waiver를 남긴다.
폭 관련 경고는 시뮬레이션 데이터 패턴에 따라 뒤늦게 나타날 수 있다.

디자인/주의
Vivado 메시지 예시는 실제 프로젝트 로그로 교체하면 좋다.

필수

Blocking / Nonblocking 혼용

이 슬라이드의 목적 대입 연산자 규칙이 코드 스타일이 아니라 시뮬레이션 의미와 연결됨을 재강조한다.

화면에 넣을 내용

블록	기본 대입
조합 always @(*)	=
순차 always @(posedge clk)	<=

- 순차 블록에서 = 사용 시 레지스터 간 즉시 전달처럼 보일 수 있음
- 조합 블록에서 <= 사용 시 Delta-cycle 의존성이 생김
- 같은 신호에 =와 <=를 혼용하지 않음
- Testbench stimulus의 blocking/nonblocking은 목적에 따라 구분

발표 멘트 / 설명 흐름

이 규칙은 모든 고급 코딩 패턴을 설명하는 절대 법칙은 아니지만, 입문과 사내 RTL 규칙으로 매우 효과적이다.

예외를 먼저 설명하면 오히려 혼란이 커진다.

디자인/주의

한 줄 규칙을 크게 배치한다.

필수

Multiple Driver와 Incomplete Assignment

이 슬라이드의 목적 하나의 출력이 여러 곳에서 구동되거나 일부 경로에서 미할당되는 문제를 구분한다.

화면에 넣을 내용

문제	대표 원인	결과
Multiple Driver	동일 reg/wire를 여러 assign/always가 구동	합성 오류 또는 충돌
Incomplete Assignment	조합 block 일부 경로에서 출력 미대입	Latch 합성 가능
Unconnected Port	인스턴스 포트 누락/이름 오타	기능 단절 또는 X
Combinational Loop	출력이 조합적으로 자기 입력으로 피드백	Timing/합성 문제

신호별 Driver를 한 곳에서 추적할 수 있어야 한다

발표 멘트 / 설명 흐름

“reg니까 한 block만 가능”이 아니라 일반적인 synthesizable RTL 관리 원칙으로 설명한다.
Tri-state I/O 같은 예외는 FPGA 내부 일반 신호와 구분한다.

디자인/주의
문제-원인-결과 표가 가장 효율적이다.

필수

X/Z를 숨기지 않는다

이 슬라이드의 목적 시뮬레이션의 Unknown을 결함 탐지 신호로 활용한다.

화면에 넣을 내용

- X의 주요 원인: 초기화 누락, 다중 Driver, 불완전 분기, 타이밍 위반 모델
- Z의 주요 원인: 미구동 net, 외부 Tri-state, 연결 누락

```
casez (data)
  4'b1???: y = 2'd3;
  ...
endcase

// casex는 X까지 don't-care 처리하므로 주의
```

- RTL 일반 비교는 ==
- Testbench에서 X/Z 포함 비교는 ===

X를 0이나 1로 간주해 넘기면 원인 분석 기회를 잃는다

발표 멘트 / 설명 흐름

casex는 초기화되지 않은 신호를 정상 조건으로 매칭할 수 있다.

합성 결과와 시뮬레이션 결과가 다르게 느껴질 때 X-optimism/X-pessimism 관점이 필요할 수 있다.

디자인/주의

X를 경고 아이콘으로 시각화한다.

필수

Simulation용 문법과 합성 RTL을 구분한다

이 슬라이드의 목적 동작 시뮬레이션에는 유용하지만 실제 회로로 합성되지 않는 구문을 구분한다.

화면에 넣을 내용

구문	일반적 용도	합성 RTL
initial	Testbench 초기화, 일부 FPGA 초기값	툴/대상에 따라 제한
#10 delay	Stimulus/Timing model	일반적으로 합성 불가
$display / $monitor	Simulation 출력	합성 불가
$finish	Simulation 종료	합성 불가
forever	Clock 생성 등 TB	일반 RTL에서 주의

시뮬레이션에서 동작한다고 FPGA 회로가 구현되는 것은 아니다

발표 멘트 / 설명 흐름

FPGA Register initialization은 소자와 툴에서 지원되는 경우가 있으므로 “initial은 항상 합성 불가”라고 단정하지 않는다.

본 세미나에서는 synthesizable RTL과 Testbench 파일을 분리한다.

디자인/주의

지원 여부는 프로젝트의 Vivado 버전과 소자 기준으로 확인한다.

필수

로그를 읽는 순서

이 슬라이드의 목적 문법 오류보다 먼저 발생한 근본 메시지를 찾는 습관을 만든다.

화면에 넣을 내용

- 1. 최초 Error 위치와 파일/라인 확인
- 2. 그 이전의 Warning 또는 Parser 메시지 확인
- 3. 괄호, begin/end, 세미콜론, 포트 쉼표 점검
- 4. 선언되지 않은 신호와 폭 확인
- 5. Elaboration 단계의 Parameter/Port 오류 확인
- 6. Synthesis Warning 중 Latch, Multiple Driver, Width를 우선 검토
- 7. 수정 후 전체 로그가 아니라 새로 발생한 메시지 비교

마지막 Error보다 첫 번째 원인 메시지가 더 중요할 때가 많다

발표 멘트 / 설명 흐름

Verilog parser는 실제 원인보다 몇 줄 뒤에서 오류를 보고할 수 있다.
에러 번호를 외우기보다 메시지의 단계(Parse, Elaborate, Synthesis)를 구분한다.

디자인/주의
실제 Vivado 로그 캡처를 넣을 경우 프로젝트 정보는 제거한다.

8부

8부. 기본 Testbench

간단한 self-checking testbench를 만들고 외부 IP 테스트 구조를 읽는 기반을 만든다.

필수

Testbench의 기본 구조

이 슬라이드의 목적 합성 대상 RTL과 Simulation 전용 환경의 경계를 이해시킨다.

화면에 넣을 내용

```
`timescale 1ns/1ps

module tb_counter;

reg clk;
reg rst;
wire [7:0] count;

// DUT instance
// clock/reset generation
// stimulus
// checking

endmodule
```

- Testbench module은 일반적으로 외부 Port가 없음
- 입력 자극은 reg, DUT 출력 관찰은 wire
- Clock/Reset/Stimulus/Check를 분리

발표 멘트 / 설명 흐름

Testbench는 FPGA에 합성하는 회로가 아니라 DUT를 검증하기 위한 Simulation 환경이다.
처음부터 UVM이 아니라 단순 self-checking testbench를 목표로 한다.

디자인/주의
DUT를 중앙에 두고 TB가 둘러싼 그림을 사용한다.

필수

DUT Instantiation

이 슬라이드의 목적 설계 모듈을 테스트벤치에 연결하는 방법을 익힌다.

화면에 넣을 내용

```
counter #(
    .WIDTH(8)
) dut (
    .clk   (clk),
    .rst   (rst),
    .en    (en),
    .count (count)
);
```

- 인스턴스 이름은 dut 또는 u_dut 등 명확하게
- Named port connection 사용
- Parameter 값은 실제 검증 조건에 맞게 설정
- 미연결 Port와 폭 경고 확인

발표 멘트 / 설명 흐름

DUT의 input은 TB가 drive하고, output은 TB가 monitor한다.
DUT 인스턴스부터 정상 elaboration 되는지 먼저 확인하면 디버깅이 쉽다.

디자인/주의
Port 방향을 화살표로 표시한다.

필수

Clock과 Reset 생성

이 슬라이드의 목적 가장 기본적인 Simulation stimulus를 작성한다.

화면에 넣을 내용

```
initial begin
    clk = 1'b0;
    forever #5 clk = ~clk;   // 10 ns period
end

initial begin
    rst = 1'b1;
    en  = 1'b0;
    #20;
    rst = 1'b0;
end
```

- `timescale에 따라 #5의 실제 시간이 결정
- Reset 극성과 DUT 코드 일치 확인
- Reset 해제 시점을 Clock Edge와 의도적으로 배치

발표 멘트 / 설명 흐름

여기서 # delay는 Testbench 전용이며 합성하지 않는다.

비동기 reset release 검증은 더 정교한 stimulus가 필요할 수 있다.

디자인/주의

Clock period 계산을 파형으로 표시한다.

필수

Stimulus 작성

이 슬라이드의 목적 입력 시퀀스를 시간과 Clock 기준으로 구동하는 방법을 보여준다.

화면에 넣을 내용

```
initial begin
    en = 1'b0;
    @(negedge rst);

    repeat (2) @(posedge clk);
    en = 1'b1;

    repeat (5) @(posedge clk);
    en = 1'b0;
end
```

- # delay보다 @(posedge clk) 기반 stimulus가 DUT 동작과 정렬하기 쉬움
- 입력 변경 시 Setup 여유를 두기 위해 negedge에서 drive하는 방식도 가능
- repeat, for를 사용해 반복 시퀀스 생성

발표 멘트 / 설명 흐름

Stimulus가 DUT Clock Edge와 정확히 동시에 바뀌면 race가 발생할 수 있다.
입문에서는 입력 drive 시점을 한 규칙으로 통일한다.

디자인/주의
시간 기반과 Edge 기반 stimulus를 비교한다.

필수

관찰과 Self-Checking

이 슬라이드의 목적 Waveform 확인을 넘어 자동 판정을 수행하는 기본 방식을 소개한다.

화면에 넣을 내용

```
always @(posedge clk) begin
    if (!rst && en) begin
        expected = expected + 1;
        if (count !== expected) begin
            $display("ERROR t=%0t exp=%0d got=%0d",
                     $time, expected, count);
            error_count = error_count + 1;
        end
    end
end
```

- $display: 특정 시점 메시지
- $monitor: 값 변화 자동 출력
- !==를 사용해 X/Z도 불일치로 검출
- Expected model과 DUT 출력 비교

발표 멘트 / 설명 흐름

파형을 눈으로만 보는 검증은 반복성과 회귀성이 낮다.
작은 RTL도 PASS/FAIL 자동 판정 습관을 만드는 것이 좋다.

디자인/주의
PASS/FAIL 카운터를 오른쪽에 표시한다.

필수

Simulation 종료와 결과 요약

이 슬라이드의 목적 테스트가 끝났음을 명확히 하고 결과를 한 줄로 남긴다.

화면에 넣을 내용

```
initial begin
    ...
    repeat (20) @(posedge clk);

    if (error_count == 0)
        $display("TEST PASS");
    else
        $display("TEST FAIL: %0d errors", error_count);

    $finish;
end
```

- 무한 Clock만 남아 Simulation이 계속되지 않도록 종료
- Timeout 조건을 추가해 Hang 검출
- 로그의 최종 PASS/FAIL 형식을 프로젝트 표준으로 통일

발표 멘트 / 설명 흐름

테스트벤치가 끝나지 않으면 자동 회귀 테스트에 넣기 어렵다.

DUT completion signal을 기다릴 때도 timeout을 함께 둔다.

디자인/주의

최종 로그 한 줄이 눈에 띄도록 크게 표시한다.

필수

기본 Testbench 전체 흐름

이 슬라이드의 목적 앞 장의 요소들을 하나의 검증 절차로 연결한다.

화면에 넣을 내용

- 1. `timescale 및 TB module 선언
- 2. DUT input reg / output wire 선언
- 3. DUT instantiation
- 4. Clock/Reset generation
- 5. Stimulus sequence
- 6. Expected model / output checking
- 7. Timeout 및 PASS/FAIL
- 8. $finish

작은 기능 단위부터 자동 판정 가능한 Testbench를 만든다

발표 멘트 / 설명 흐름

이 장은 전체 코드보다 구조를 보여주는 목차 역할이다.

외부 IP Test Bench를 읽을 때도 Clock, Reset, Stimulus, Checker 순서로 분해한다.

디자인/주의

Flow chart 형태로 구성한다.

필수

Vivado/XSim에서 확인할 항목

이 슬라이드의 목적 툴 조작보다 Simulation 결과를 해석하는 체크포인트를 정리한다.

화면에 넣을 내용

- Simulation top이 Testbench로 설정됐는가?
- Design Source와 Simulation Source가 올바르게 분리됐는가?
- Compilation order와 include path가 맞는가?
- DUT parameter와 Port 연결 경고가 없는가?
- Reset 이후 X가 남는 신호가 없는가?
- Clock period와 `timescale이 의도와 같은가?
- 로그 PASS/FAIL과 Waveform 결과가 일치하는가?

먼저 Behavioral Simulation에서 RTL 기능을 확인하고, 필요 시 Timing Simulation으로 확장

발표 멘트 / 설명 흐름

Vivado 버전에 따라 UI 이름이 조금 달라질 수 있으므로 메뉴 캡처보다 개념 중심으로 설명한다.

이번 세미나의 목적은 XSim 조작법 전체가 아니라 Verilog TB 구조 이해다.

디자인/주의

Vivado Flow Navigator의 Simulation 영역을 작게 참고로 넣어도 좋다.

9부

9부. 정리와 실무 적용

세미나 내용을 외부 RTL 분석 절차와 사내 코드 관리 규칙으로 정리한다.

필수

외부 Verilog RTL을 읽는 실무 순서

이 슬라이드의 목적 세미나 전체 내용을 실제 코드 분석 절차로 압축한다.

화면에 넣을 내용

- 1. Module Port와 Parameter 확인
- 2. Clock/Reset Domain 식별
- 3. wire/reg와 주요 데이터 폭 확인
- 4. assign, combinational always, clocked always 분류
- 5. 하위 모듈과 hierarchy map 작성
- 6. FSM과 제어 우선순위 확인
- 7. Width, signed, latch, multiple driver 경고 확인
- 8. 제공 Testbench/Example Design으로 동작 검증

한 줄씩 번역하지 말고 구조 → 데이터 경로 → 제어 경로 순서로 읽는다

발표 멘트 / 설명 흐름

이 절차는 MIG, GT Wizard, 외부 IP wrapper 모두에 적용할 수 있다.
처음부터 모든 내부 모듈을 이해하려 하지 않고 시스템 경계를 먼저 잡는다.

디자인/주의
8단계 흐름도를 사용한다.

필수

사내 적용 체크리스트

이 슬라이드의 목적 세미나 이후 실제 프로젝트에 적용할 최소 규칙을 제안한다.

화면에 넣을 내용

- Named port connection 사용
- `default_nettype none 적용 검토
- 조합 = / 순차 <= 규칙
- 조합 출력 기본값과 case default
- 폭과 signed를 명시적으로 처리
- 한 신호 한 Driver 원칙
- Warning 원인과 waiver 기록
- 외부 IP example은 원본 보존 후 사용자 수정 영역 분리
- 작은 self-checking testbench 작성

Verilog를 주 개발 언어로 바꾸는 것이 아니라, 외부 설계 자산을 내부 품질 체계에 편입한다

발표 멘트 / 설명 흐름

회사 코딩 규칙이 이미 있다면 이 장의 항목을 해당 규칙에 맞게 수정한다.
방산 프로젝트에서는 변경 이력, 검토 기록, 도구 버전과 함께 관리한다.

디자인/주의
체크리스트 형태로 마무리한다.

필수

핵심 정리

이 슬라이드의 목적 세미나의 메시지를 세 문장으로 마무리한다.

화면에 넣을 내용

1. VHDL과 Verilog는 같은 RTL 회로를 다른 문법으로 표현한다
2. Verilog의 핵심 위험은 폭, signed, 대입 방식, 불완전 할당과 같은 언어 규칙이다
3. 목표는 Verilog 기반 IP와 RTL을 독립적으로 검토·수정·유지보수하는 역량이다

발표 멘트 / 설명 흐름

마지막에는 새로운 내용을 추가하지 않는다.

질의응답은 실제 VHDL 코드를 Verilog로 비교하거나 회사에서 사용하는 IP 예제로 연결한다.

디자인/주의

3개의 큰 문장만 배치한다.

Appendix

Appendix

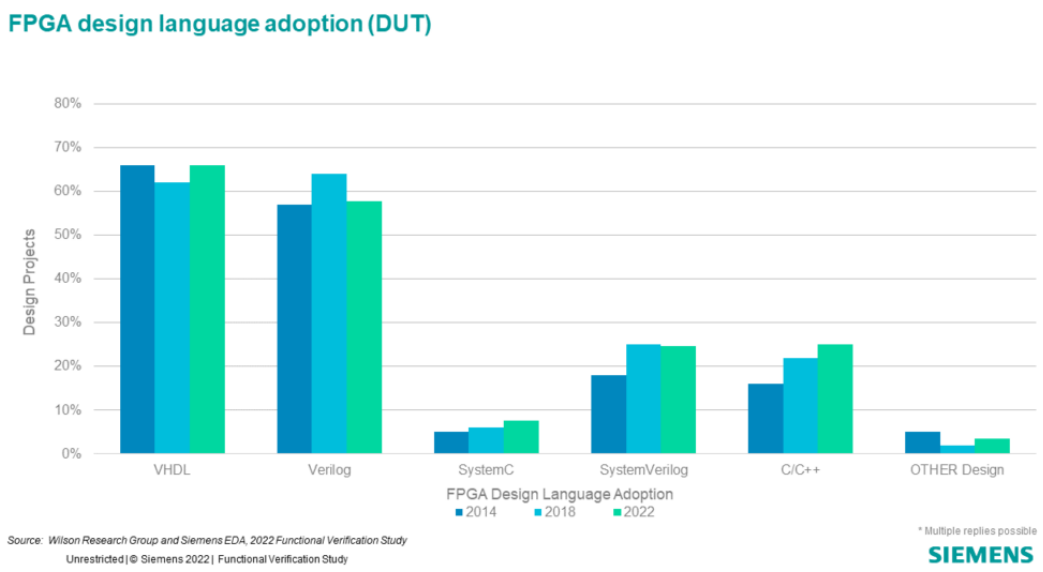
본문 흐름을 보조하는 통계, AMD 공식 사례, SystemVerilog 방향과 치트시트.

선택

FPGA Design Language Adoption (DUT)

이 슬라이드의 목적 본문의 필요성 논리를 보조하는 업계 통계 자료를 선택적으로 제공한다.

화면에 넣을 내용



- 전 세계·전체 산업군을 통합한 FPGA RTL 설계 언어 채택 결과
- 한 프로젝트가 복수 언어를 사용하므로 합계가 100%를 넘을 수 있음
- VHDL은 여전히 널리 사용되고 Verilog도 절반 이상의 프로젝트에서 사용

발표 멘트 / 설명 흐름

이 그래프를 “Verilog가 VHDL을 대체한다”는 근거로 사용하지 않는다.
정확한 메시지는 “FPGA RTL 생태계는 단일 HDL 환경이 아니다”이다.
Mil/Aero나 특정 지역만 필터링하면 비율이 달라질 수 있다.

디자인/주의

Appendix에서 질문이 나왔을 때만 설명한다. 원본 Figure 6-1 제목과 출처를 유지한다.

근거/출처 Siemens EDA / Wilson Research Group, 2022 Functional Verification Study, Part 6, Figure 6-1.

선택

AMD IP 제공 언어 사례: Memory와 GT Wizard

이 슬라이드의 목적 Memory 외에도 고속 직렬 IP에서 유사한 제공 언어 차이가 있음을 보조한다.

화면에 넣을 내용

기능군	7 Series	UltraScale / UltraScale+
Memory Interface	Example: Verilog, VHDL Top-level only	Example: Verilog Test Bench: Verilog
GT Wizard	Example/Test Bench: Verilog, VHDL	Example/Test Bench: Verilog

모든 최신 AMD IP가 Verilog-only인 것은 아니며 IP별 공식 IP Facts 확인이 필요

발표 멘트 / 설명 흐름

GT Wizard example은 PRBS generator/checker, reset helper, user clock, link status 등을 포함해 통합 출발점으로 사용된다.

이 장은 특정 인터페이스 홍보가 아니라 세대별 제공 방식이 달라질 수 있다는 보조 근거다.

디자인/주의

문서 버전과 날짜를 반드시 함께 적는다.

근거/출처 AMD DS176 v4.2; AMD PG150 v1.4; AMD PG182 v1.7 IP Facts.

선택

FPGA도 SystemVerilog로 넘어가는가

이 슬라이드의 목적 장기적인 언어 방향 질문에 과장 없이 답한다.

화면에 넣을 내용

영역	현실적인 방향
기존 VHDL RTL	장기 유지 가능성이 높음
기존 Verilog RTL	logic, always_comb/ff 등 SystemVerilog RTL 문법으로 점진 확장 가능
검증	SVA, UVM, Coverage 등 SystemVerilog 기반 기술 활용 확대
방산/항공	전면 전환보다 검증된 자산과 장기 공존

VHDL이 SystemVerilog로 대체된다기보다, Verilog 계열이 SystemVerilog로 확장되는 흐름에 가깝다

발표 멘트 / 설명 흐름

SystemVerilog는 ASIC 전용이 아니라 설계와 검증을 통합한 IEEE 표준 언어다.
방산에서는 도구 버전 고정, 검증 증적, 장기 유지보수 때문에 전환 속도가 느릴 수 있다.

디자인/주의
본문 학습 범위를 흔들지 않도록 Appendix에만 둔다.

근거/출처 IEEE 1800-2023; AMD UG901 SystemVerilog Support (2026.1).

선택

VHDL - Verilog 빠른 대응표

이 슬라이드의 목적 발표 후 참고할 수 있는 한 장짜리 치트시트를 제공한다.

화면에 넣을 내용

VHDL	Verilog
entity + architecture	module ... endmodule
std_logic / vector	wire/reg [N:0]
signal assignment <=	assign 또는 procedural assignment
process(all)	always @(*)
process(clk) + rising_edge	always @(posedge clk)
generic	parameter
constant	localparam
port map	module instantiation
downto slice	[MSB:LSB]
& concatenation	{a,b}
when others	default

발표 멘트 / 설명 흐름

완전한 자동 변환표가 아니라 코드를 읽는 빠른 참고표다.

VHDL <=와 Verilog <=는 문맥상 의미가 다르다는 주석을 남긴다.

디자인/주의

인쇄해서 한 장으로 배포하기 좋은 구성이다.

선택

Literal / Operator / Review Cheat Sheet

이 슬라이드의 목적 숫자와 연산자, 코드 리뷰 포인트를 마지막 참고 장에 압축한다.

화면에 넣을 내용

항목	예
Literal	8'hA5, 14'd0, 1'b1
Vector	wire [13:0] data
Select	data[13], data[13:6]
Concat / Repeat	{hi,lo}, {8{sign}}
Logical / Bitwise	&& || ! / & | ^ ~
Reduction	&data;, |data, ^data
Equality	== != / === !==
Combinational	assign, always @(*), =
Sequential	always @(posedge clk), <=

Review: Width - Signed - Driver - Assignment - Complete Path - Reset

발표 멘트 / 설명 흐름

마지막 줄 6개 단어를 코드 리뷰 기억법으로 사용한다.

상세 연산자 우선순위는 언어 레퍼런스를 참조하고, RTL에서는 괄호로 의도를 명확히 한다.

디자인/주의

한 장짜리 PDF 출력용으로 사용할 수 있다.

참고 문서 및 링크

기관	문서	URL
Siemens EDA / Wilson Research Group	Part 6: The 2022 Wilson Research Group Functional Verification Study	https://blogs.sw.siemens.com/verificationhorizons/2022/11/21/part-6-the-2022-wilson-research-group-functional-verification-study/
AMD	DS176 v4.2 - Zynq 7000 SoC and 7 Series Devices Memory Interface Solutions, IP Facts	https://docs.amd.com/r/en-US/ds176_7Series_MIS/IP-Facts
AMD	PG150 v1.4 - UltraScale Architecture-Based FPGAs Memory IP, IP Facts	https://docs.amd.com/r/en-US/pg150-ultrascale-memory-ip/IP-Facts
AMD	PG182 v1.7 - UltraScale FPGAs Transceivers Wizard, IP Facts	https://docs.amd.com/r/en-US/pg182-gtwizard-ultrascale/Features
AMD	UG901 Vivado Design Suite User Guide: Synthesis - Verilog/SystemVerilog Support	https://docs.amd.com/r/en-US/ug901-vivado-synthesis/SystemVerilog-Support
IEEE	IEEE 1800-2023 - SystemVerilog Unified Hardware Design, Specification, and Verification Language	https://standards.ieee.org/ieee/1800/7743/

주의: AMD IP의 제공 언어와 지원 범위는 Vivado 버전, IP 버전, 대상 Device Family에 따라 달라질 수 있습니다. 실제 프로젝트에서는 사용 중인 버전의 IP Facts와 생성된 Output Products를 기준으로 확인하십시오.