빠른 시작

`slang` 실행 파일은 주어진 SystemVerilog 소스 파일을 전체 컴파일합니다. 최종적으로는 시뮬레이션 실행 파일을 출력할 수 있지만, 현재는 소스 코드의 문법, 타입, 언어 규칙 등을 검사하는 용도로 사용됩니다.

하나 이상의 유효한 SystemVerilog 파일을 실행하면 최상위 모듈을 출력하고 종료 코드 0을 반환합니다:

```
// test1.sv
module m;
  struct { logic a; } s;
  int i = s.a + 1;
  initial $display("%d", i);
endmodule
```

```
> slang test1.sv
Top level design units:
    m

Build succeeded: 0 errors, 0 warnings

> echo $?
0
```

오류가 있는 파일을 실행하면 깔끔하게 포맷된 진단 메시지가 출력됩니다:

```
// test2.sv
module m;
  struct { logic a; } s;
  int i = s + 1;
  initial $display("%d", i);
endmodule
```

```
Top level design units:
    m

../test2.sv:4:12: error: invalid operands to binary expression ('<unnamed
unpacked struct>' and 'int')
        int i = s + 1;
              ~ ^ ~

Build failed: 1 error, 0 warnings
```

파일 패턴

`slang`에 파일 경로를 넘길 때는 와일드카드 패턴도 사용할 수 있습니다. 지원되는 와일드카드는 다음과 같습니다:

- `?` : 임의의 단일 문자
- `*` : 0개 이상의 문자
- `...` : 임의의 깊이의 하위 디렉토리 재귀 매칭
- `..` : 상위 디렉토리
- `.` : 현재 디렉토리
- `/`로 끝나는 경로는 그 디렉토리의 모든 파일을 포함합니다 (즉 `/*`와 동일).

참고: 커맨드라인에서 지정한 파일 패턴은 먼저 셸에서 해석됩니다. 이로 인해 `slang`이 보기 전에 이미 확장되어 의도와 다른 결과가 나올 수 있습니다.

컴파일 유닛

컴파일 유닛은 하나 이상의 소스 파일로 이루어진 독립적인 단위입니다. 매크로 확장 등 전처리는 파싱 중 수행되므로, 그 효과는 해당 유닛에 국한됩니다. 유닛들은 독립적이므로 병렬 파싱이 가능합니다. 모든 유닛이 파싱되면 정교화(elaboration)를 통해 최종 설계를 구성합니다.

기본적으로 slang은 모든 입력 파일을 개별 컴파일 유닛으로 취급합니다. 이는 파일 순서에 구애받지 않고, 내부적으로 병렬 처리를 가능하게 하므로 바람직한 방식입니다.

단일 컴파일 유닛으로 묶으려면 `--single-unit` 플래그를 사용합니다. 이는 대부분의 다른 SystemVerilog 툴들의 기본 동작과 동일합니다.

좀 더 세밀한 제어를 위해:

- `-v` 플래그로 전달된 라이브러리 유닛
- `--libmap`으로 지정된 라이브러리 매핑 파일
- 컴파일 유닛 나열 파일(`-C` 플래그로 지정) 등이 모두 개별 컴파일 유닛으로 취급됩니다.

소스 라이브러리

SystemVerilog는 설계에 포함될 수 있는 이름 있는 파일 모음인 "소스 라이브러리" 개념을 가집니다. 명시적으로 이름이 주어지지 않으면 기본 라이브러리("work")에 포함됩니다.

예:

```
> slang top_module.sv -v some/lib1.sv -v "my_lib=some/other/lib2.sv"
```

- `top_module.sv`: 비라이브러리 파일이지만 기본 라이브러리에 포함됨
- `lib1.sv`: 기본(work) 라이브러리에 포함된 라이브러리 파일
- `lib2.sv`: my_lib이라는 사용자 정의 라이브러리에 포함된 파일

라이브러리 파일은 자동 인스턴스화되지 않으며, 미사용 모듈에 대해서는 오류 검사를 하지 않습니다. 심지어 `--single-unit`을 사용하더라도 이들 파일은 서로 독립적으로 취급됩니다.

매크로 상속이 필요할 경우 `--libraries-inherit-macros` 플래그를 사용할 수 있습니다.

라이브러리 파일 검색

`--libdir` 플래그를 사용하여 slang이 라이브러리 파일을 자동으로 검색하도록 할 수 있습니다. 검색 시 사용할 확장자는 `--libext` 플래그로 지정하며 기본값은 `.v`, `.sv` 입니다.

모든 명시적 파일이 파싱된 후 slang은 모듈 인스턴스, 패키지 import, 인터페이스 포트를 확인하여 누락된 항목 이름을 수집하고 지정된 디렉토리들에서 검색합니다. 발견되면 해당 파일은 라이브러리 파일로 포함됩니다.

명령 파일

명령 파일은 커맨드라인 옵션을 독립적인 파일로 구성할 수 있게 해줍니다. `-F`, `-f` 플래그를 사용해 파일을 지정하며 둘은 상대 경로 기준이 다릅니다:

- `-F`: 경로가 명령 파일 기준
- `-f`: 경로가 현재 작업 디렉토리 기준

형식:

- 공백으로 구분된 인자
- `#`, `//`, `/* */` 주석 지원
- 따옴표로 묶인 문자열
- 백슬래시로 특수 문자 이스케이프
- `$VAR`, `$(VAR)`, `${VAR}` 형태의 환경 변수 지원
- 다른 명령 파일 포함 가능
- 커맨드라인에서 지정된 인자가 우선 적용됨

컴파일 유닛 나열 파일

`-C` 플래그로 지정된 파일로 컴파일 유닛을 정의할 수 있습니다. 명령 파일과 유사하지만 제한된 플래그만 지원되며, 항상 해당 파일 기준으로 경로를 해석합니다.

지원 플래그:

- 파일 경로
- `-I`, `+incdir+` : include 디렉토리
- `-D`, `+define+` : 매크로 정의
- `--library` : 포함될 라이브러리 명 지정

내장 매크로

이름	값
`**slang**	1
`**slang_major**	slang 메이저 버전
`**slang_minor**	slang 마이너 버전

비표준 내장 함수

`$static_assert(condition[, message])`

정교화 시점에서 상수 표현식 조건을 검사하고 실패하면 오류를 발생시킵니다. `$error`와 유사하지만 패키지 등 더 많은 문맥에서 사용할 수 있습니다.

예:

```
module m;
  localparam int foo = 12;
  struct packed { logic [4:1] a, b; } bar;
  $static_assert(foo < $bits(bar));
endmodule
```

출력:

```
test.sv:5:5: error: static assertion failed
    $static_assert(foo < $bits(bar));
   ^
test.sv:5:24: note: comparison reduces to (12 < 8)
    $static_assert(foo < $bits(bar));
                   ~~~~^~~~~~~~~~~~
```

드라이버 루프 전개 (Driver loop unrolling)

SystemVerilog의 longest static prefix 규칙(11.5.3절)에 따라 루프 내 변수 인덱싱은 복잡한 오류를 유발할 수 있습니다. 대부분의 도구(slanguages 포함)는 `for` 루프를 전개하여 문제를 회피합니다. 이를 비활성화하려면 `--max-loop-analysis-steps=0`으로 설정합니다.

제한사항:

- `for` 루프만 지원
- 반복 변수 및 조건 명시 필수
- 루프 반복 수는 설정된 한계값보다 작아야 함

프라그마 (Pragmas)

`pragma once`

`include` 된 파일을 한 번만 포함합니다. C++의 #pragma once`와 유사합니다.

`pragma diagnostic`

- `push` / `pop`: 경고 상태 저장/복원
- `ignore`, `warn`, `error`, `fatal`: 특정 경고 무시/활성화/에러/치명적 에러 처리

예:

```
module m;
  ; // warn
`pragma diagnostic ignore="-Wempty-member"
  ; // hidden
`pragma diagnostic push
  ; // also hidden
`pragma diagnostic error="-Wempty-member"
  ; // error
`pragma diagnostic warn="-Wempty-member"
  ; // warn
`pragma diagnostic pop
  ; // hidden again
endmodule
```

주석 지시어

프라그마 외에도 주석으로 경고를 제어할 수 있습니다.

```
module m;
  ; // warn

  // slang lint_off empty-member
  ; // hidden
  // slang lint_save
  /* slang lint_on empty-member */
  ; // warn
  /* slang lint_restore */
  ; // hidden
endmodule
```

지시어:

- `lint_off`, `lint_on`
- `lint_save`, `lint_restore`

또한 `translate_off` 스타일의 블록 무시 주석도 지원합니다 (`--translate-off-format` 참조).