Verilog-A 모델에서 최고의 성능 구현

소개

Verilog-A, 특히 트랜지스터 모델로 작성된 모델은 즉시 시뮬레이션할 수 있지만, SmartSpice에서 더 나은 성능을 구현하려면 다음 정책을 준수해야 합니다.

이러한 정책을 따르는 새로운 모델도 더 나은 성능을 발휘할 것입니다.

개요

1. 파라미터 값의 계산
2. 내부 노드의 수
3. 소음 차단
4. 함수 대신 매크로
5. GMIN!

정책

1. 파라미터 값은 한 번만 계산:

```
parameter length = 2;
parameter width = 3;
real    area;

analog begin
   area = length * width;
   ….
end
```

여기서 면적은 각각의 되풀이 구간에서 반복적으로 계산됩니다. 그러나, 첫 번째 이후의 계산은 불필요합니다.

면적은 파라미터로 선언한 길이와 너비에서 얻을 수 있으므로 실행 중에 값을 수정하는 것은 합당치 않으며 (LRM 2.3.1 Sec 3.4), 변수 값은 그 다음 반복에서 유지되므로 (LRM 2.3.1 Sec 5.6.1.3) 시뮬레이션을 시작할 때 한 번만 면적을 계산하는 것이 가능하기 때문입니다.

이러한 변경으로 원하는 결과를 얻습니다:

```
parameter length = 2;
parameter width = 3;
real    area;

analog begin
   @(initial_step) begin
    area = length * width;
    ….
    end
    ….
end
```

initial_step 이벤트에 의해 트리거된 이벤트 블록은 시뮬레이션을 시작할 때 한 번만 실행됩니다 (LRM 3.2.1 Sec 5.10.2). Verilog-A는 *initial_step*을 인식하고 할당된 변수를 다르게 분류하여 상당한 이점을 제공합니다.

ADMS를 직접 지원하는 일부 모델 (PSP, BSIMCMG)은 *initial_step*을 간단하게 추가할 수 있습니다. 이러한 모델은 "define"을 사용하여, ADMS 외부에서 유연하게 사용할 수 있습니다:

```
`ifdef ADMS
   `define INITIAL_MODEL@(initial_model)
   `define INITIAL_INSTANCE
                    @(initial_instance)
`else
   `define INITIAL_MODEL
   `define INITIAL_INSTANCE
`endif

analog begin
   `INITIAL_MODEL
   begin
     area = length * width;
     …
   end
```

```
end
```

ADMS를 대상으로 하지 않을 때, "'INITIAL_MODEL"과 "INTIAL_INSTANCE"는 공백으로 대체되는데, 이는 원하는 것이 아닙니다. 다음과 같이 간단하게 해결할 수 있습니다.

```
`ifdef ADMS
   `define INITIAL_MODEL @(initial_model)
   `define               INITIAL_INSTANCE
                       @(initial_instance)
`else
   `define INITIAL_MODEL @(initial_step)
   `define               INITIAL_INSTANCE
                         @(initial_step)
`endif
```

2. 내부 노드의 수를 최소화

모델의 기생 성분은 시뮬레이션 속도를 저하시키므로, 모델은 일반적으로 다른 수준으로 구현됩니다. 기본 수준은 최상위 수준보다 기생 성분이 적고, 시뮬레이션에 내부 노드를 적게 사용하므로 시뮬레이션 속도가 향상됩니다. Verilog-A LRM은 SPICE의 수준 파라미터와 비견할 만한 기능이 없으므로, 다른 수단을 통해 유사한 결과를 얻었습니다. 하나의 간단한 방법은 각 수준에 대해 상이한 모델 이름 (모듈 이름)을 사용하는 것입니다. 보다 복잡한 방식을 사용하려면 제공된 파라미터 값에 기초하여 Verilog-A 파서가 적절한 수준을 자동으로 추론해야 합니다. 이처럼 상이한 Verilog-A 코드 스타일은 하나의 Verilog-A 소스 모델에서 서로 다른 수준의 결과를 얻을 수 있습니다. 아래에서, *조건부 컴파일*, *노드 축소*의 두 가지 유형에 대해 설명합니다.

조건부 컴파일 (예: EKV3 모델)은 "ifdef"를 지엽적으로 사용하여, 각 수준에 대해 고유 모델 (모듈) 이름을 제공하고 내부 노드의 수를 정밀하게 제어합니다:

```
`ifdef DC
        module xyz       (d,g,s,b);
`endif
`ifdef RF
        module xyz_rf    (d,g,s,b);
`endif
….
`ifdef RF
        electrical           internal_d,
        internal_s, internal_b;
`endif
….
`ifdef RF
        I(internal_d,internal_s) <+ ….;
`endif
        endmodule
```

"RF"를 정의하여 모듈을 컴파일하면 고유 모듈의 이름은 "xyz_rf"가 되고, 정확하게 3개의 내부 노드가 모델에 추가됩니다. 이러한 기능은 "RF"가 정의될 때 *조건부*로 발생하므로 *조건부 컴파일*이라 합니다. *조건부 컴파일* 유형은 컴파일 단계에서 적용됩니다.

노드 축소 (예: PSP, BSIMCMG)는 특별한 Verilog-A if-then-else 구조를 채택하여, 시뮬레이션 설정 단계에서 Verilog-A 파서가 해당 구조를 인식하고 작업을 수행하여 모델을 특정 수준으로 구분합니다:

```
parameter rg;

if(rg != 0)
  I(g, internal_g) <+ V(g, internal_g)/rg;
else
   V(g, internal_g) <+ 0;
```

이 코드에서, 기생 성분 "rg"가 0이 아닐 때 값이 "rg"인 저항이 노드 "internal_g"와 "g" 사이에 배치됩니다. 기생 성분이 없을 때 즉, "rg"가 0 일 때, 0V 전압원에 의해 노드 사이에 단락이 발생합니다. if-then-else 구조는 상수 파라미터 "rg"에 의해 설정된 스위치를 형성하며, Verilog-A 파서는 설정 중에 "rg"가 0일 때 스위치가 항상 "on"이고 노드 "g"와 "internal_g"는 단락되어, 결과적으로 두 노드가 하나로 축소되므로 노드 수가 줄어들 수 있음을 인지합니다. 따라서, 이를 *노드 축소*라 합니다.

0V 전압원이 단자 노드를 단락시켜 *노드 축소*로 이어지는 것은 사실이지만, Verilog-A LRM은 다른 목적으로 0V 전압원의 사용을 특별히 예약합니다. 특히 전류 프로브가 필요할 때 0V 전압원을 사용합니다 (LRM 2.3.1 Sec 5.4.2.1). Verilog-A는 *노드 축소*를 구현하지 않고 0V 전압원을 LRM에 따라 해석합니다.

Verilog-A는 if-then-else 구조를 정확하게 시뮬레이션하지만 "rg"의 0 또는 0이 아닌 값은 내부 노드 수를 개선하지 않습니다. "internal_g"를 위한 하나의 노드가 항상 존재하며, 불행하게도 내부 분기 노드는 전압원을 통과하는 전류용으로 예약되어 있습니다. "rg"가 0인 경우에도 노드는 항상 3개이며, 이 경우 노드가 하나만 필요합니다.

if-then-else 구조에 대한 Verilog-A 모델을 검색하여 *조건부 컴파일* 유형으로 대체하고, 해당되는 경우 내부 노드의 수를 줄이는 것이 좋습니다.

3. 노이즈 분석을 하지 않는 경우, 노이즈 블록 실행 불가

노이즈 분석을 요청하지 않는 경우, 노이즈 분석을 대신하는 계산값은 0이 됩니다. 다행히 Verilog-A 모델에서 노이즈 분석 구문이 함께 있는 경우가 많습니다.

```
I(…)  <+ white_noise(…);
I(…)  <+ white_noise(…);
….
```

또한, 간단한 이 솔루션은 필요할 때까지 계산을 하지 않아도 됩니다.

```
if (analysis("noise")) begin
   I(…)  <+ white_noise(…);
   I(…)  <+ white_noise(…);
   ….
end
```

4. 적절한 경우, 함수 호출 대신 매크로 사용

함수 호출은 실행하는 것은 대가가 따릅니다. 함수를 자주 호출한다면, 이를 매크로로 바꿉니다.

```
analog function real myexp;
   input x;
   real x;
   begin
      myexp = exp(x);
   end
endfunction

analog begin
   y = myexp(z);
end
```

이는 매크로 확장으로 쉽게 변경할 수 있습니다.

```
`define myexp(_x) (exp(_x_))

analog begin
   y = `myexp(z);
end
```

5. GMIN! 주의 으

"gmin"에 대한 값이 필요한 모델은 "gmin" 값을 모델에 넣을 수도 있습니다.

```
`define GMIN 1.0e-12
```

or

```
parameter real GMIN = 0.0;
```

이 경우, 회로 "gmin"을 변경하기 위한 SmartSpice 옵션은 모델에 이르지 못합니다. GMIN에 대한 참조를 다음으로 대체하여, 모델이 SmartSpice "gmin" 옵션에 응답하도록 합니다.

```
$simparam("gmin")
```

결론

여기서 설명한 다섯 가지 정책을 Verilog-A 모델에 적용하면 SmartSpice에서 최상의 시뮬레이션 성능을 얻을 수 있는 최적의 코드가 생성됩니다. Verilog-A 언어에 익숙하다면 30분 내에 기존 모델에 정책을 적용할 수 있습니다. 이는 최적화된 모델을 통해 시뮬레이션 시간을 절약함으로써 쉽게 복구할 수 있습니다.