Gateway/SmartSpice에서 Verilog-A 모델을 실행하기 위한 새로운 구문

시뮬레이션에 Verilog-A 모델을 포함하면 시간이 절약되며, SmartSpice의 최근 업데이트에 의해 이를 회로에 추가하는 것이 더욱 편리해졌습니다. 이제 Verilog-A 모듈을 포함하기 위해, 인스턴스를 부회로로 칭하고, VERILOG 구문을 추가하면 Verilog-A 모델을 추가할 수 있습니다.

Verilog-A는 트랜지스터 수준에서 정확한 회로를 만드는데 사용할 수 있지만, 회로의 동작을 증명하기 위해 능동 소자 회로 대신 Behavioral 모델을 사용할 수 있습니다. 따라서 대규모 회로를 며칠이 아니라 몇 분 만에 실행할 수 있습니다. D Flip-Flop 예제 (그림 1)와 같은 간단한 회로도 단순한 수준의 트랜지스터 3개 대신 Verilog-A를 사용하여 약 10배 더 빠르게 동작합니다. 수준 높은 모델을 사용하는 복잡한 대규모 회로의 경우, SPICE 모델 대신 Verilog-A Behavioral 모델을 사용하면 100배 더 빨라질 수 있습니다.

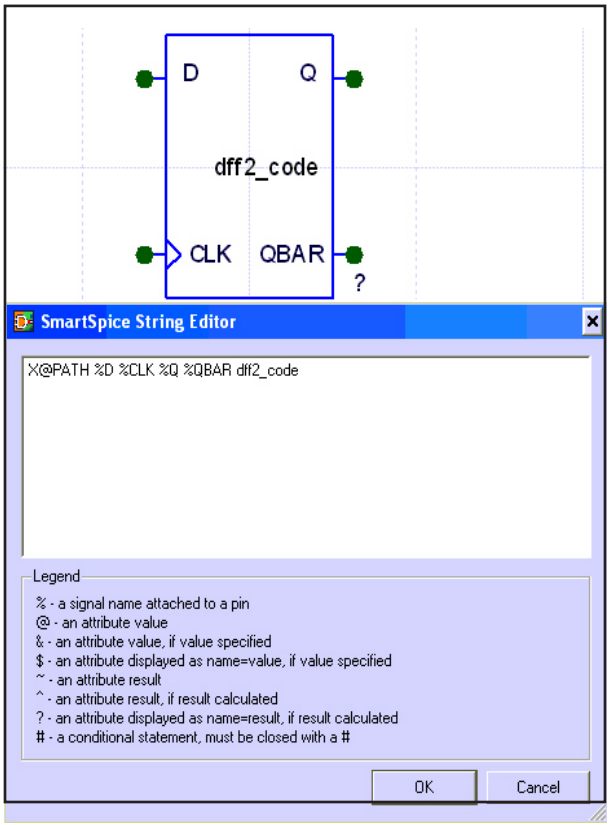

그림 1. SmartSpice 문자열

Gateway에 Verilog-A 소자를 추가하려면, 심볼을 만들어서 예제 1처럼 SmarttSpice 문자열 편집기에 부회로 호출을 추가합니다. 문자열을 X로 시작하면, 소자는 부회로로 지정됩니다. 핀은 Verilog-A 모듈과 동일한 순서가 되며, 모듈 이름이 열거됩니다. 컨트롤 카드에서 .VERILOG 호출을 VERILOG-A 모듈에 추가합니다. Verilog-A 모듈이 스키매틱과 동일한 디렉토리에 있을 필요는 없지만, 모듈이 동일한 디렉토리에 없다면 .VERILOG 호출에 전체 경로를 포함해야 합니다. 예제에 대한 Verilog-A 넷리스트를 참조하십시오.

그림 2에서 두 소자의 출력은 비슷하지만 그림 3에서 high-to-low 전환을 자세히 살펴보면, Verilog-A 모델은 단지 Behavioral 모델일 뿐 트랜지스터 수준에서 정확히 일치하지는 않는다는 것을 알 수 있습니다. 그러나 컨셉을 입증하거나 또는 파형보다 전환이 더 중요한 블록을 대체하기 위한 목적이라면, 그 결과는 유사성에 있어 충분합니다.

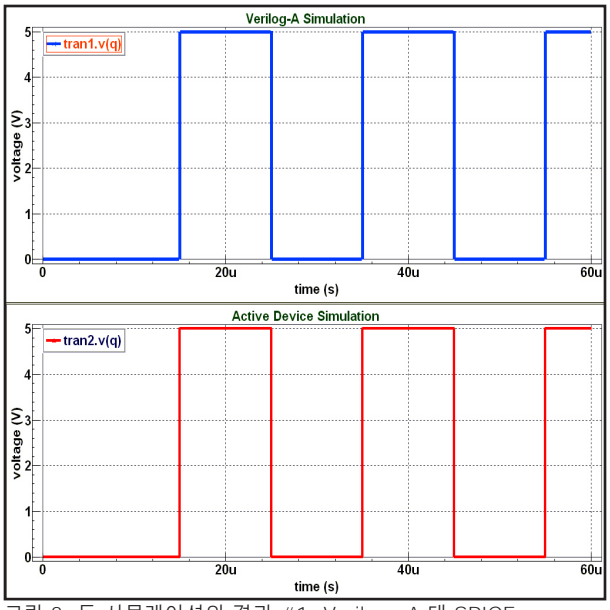

그림 2. 두 시뮬레이션의 결과 #1: Verilog-A 대 SPICE

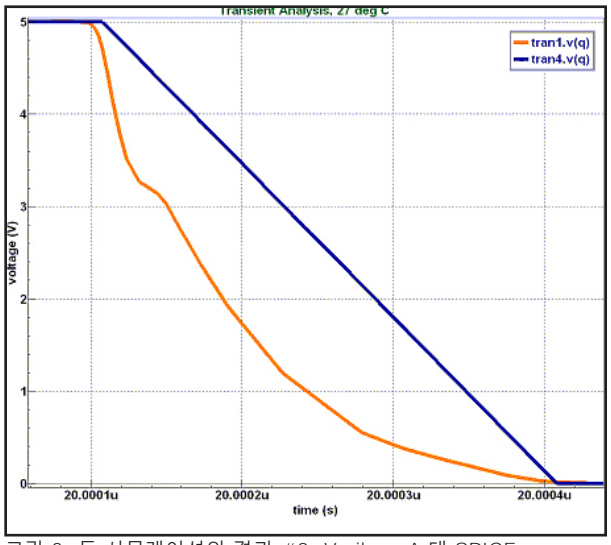

그림 3. 두 시뮬레이션의 결과 #2: Verilog-A 대 SPICE

Verilog Nelist

```
*dff_sim
* Gateway 2.6.7.R Spice Netlist
Generator
* Simulation timestamp: 27-Sep-2007
10:57:26
**
* Schematic name: dff_sim
*
V2 clk GND PULSE(0 5 5u 0.1n 0.1n 5u
10u)
V3 VDD GND DC 5
V4 d GND PULSE(0 5 10u 0.1n 0.1n 10u
20u)

** Verilog-A instance
X1 d clk q qbar dff2_code
*
* Global Nodes Declarations
.GLOBAL  clk GND VDD
* End of the netlist
*
* Markers to save
.SAVE ALL(V) V(q)

.VERILOG “dff2_code.va”

.tran 1n 60u

.END
```

Active Device Netlist

```
*dff_sim
* Gateway 2.6.7.R Spice Netlist
Generator
* Simulation timestamp: 27-Sep-2007
10:59:38
* Schematic name: dff_sim
V2 clk GND PULSE(0 5 5u 0.1n 0.1n 5u
10u)
V3 VDD GND DC 5
V4 d GND PULSE(0 5 10u 0.1n 0.1n 10u
20u)
X1 clk d q qbar dff2
*
* Schematic name: dff2
.SUBCKT dff2 CLK D Q Q_BAR
X1 NET4 NET1 NET2 nand2
X2 NET2 CLK NET1 nand2
X4 NET3 D NET4 nand2
X5 NET1 Q_BAR Q nand2
X6 Q NET3 Q_BAR nand2
X7 NET1 CLK NET4 NET3 nand3
M1 GND Q GND GND nmos w=5u l=5u M=1
M2 GND Q_BAR GND GND nmos w=5u l=5u
M=1
.ENDS dff2
*
* Schematic name: nand2
.SUBCKT nand2 a b c
M1 NET3 a GND GND nmos w=0.5u l=0.35u
M=1
M2 NET1 b NET3 GND nmos w=0.5u
l=0.35u M=1
M3 NET1 a VDD VDD pmos w=2u l=0.35u
m=1
M4 NET1 b VDD VDD pmos w=2u l=0.35u
m=1
M5 NET2 NET1 VDD VDD pmos w=2u
l=0.35u m=1
M6 NET2 NET1 GND GND nmos w=0.5u
l=0.35u
+ M=1
M7 c NET2 GND GND nmos w=0.5u l=0.35u
M=1
M8 c NET2 VDD VDD pmos w=2u l=0.35u
m=1
.ENDS nand2
*
* Schematic name: nand3
```

```
.SUBCKT nand3 a b c d
M3 NET1 a VDD VDD pmos w=2u l=0.35u
m=1
M4 NET1 b VDD VDD pmos w=2u l=0.35u
m=1
M5 NET2 NET1 VDD VDD pmos w=2u
l=0.35u m=1
M7 d NET2 GND GND nmos w=0.5u l=0.35u
M=1
M8 d NET2 VDD VDD pmos w=2u l=0.35u
m=1
M10 NET1 c VDD VDD pmos w=2u l=0.35u
m=1
M11 NET2 NET1 GND GND nmos w=0.5u
l=0.35u
+ M=1
M12 NET1 b NET3 GND nmos w=0.5u
l=0.35u M=1
M13 NET3 a NET4 GND nmos w=0.5u
l=0.35u M=1
M14 NET4 c GND GND nmos w=0.5u
l=0.35u M=1
.ENDS nand3
*
* Global Nodes Declarations
.GLOBAL  clk GND VDD
* End of the netlist
*
* Markers to save
.SAVE ALL(V) V(q)
.model nmos nmos level=3
.model pmos pmos level=3
.tran 1n 60u
.END
```

Verilog-A Module

```
//D-Flip Flop
`include “discipline.h”

module dff2_code (d, clk, q, qbar);
input clk, d;
output q, qbar;
electrical d, clk, q, qbar;
// first and second stage states
electrical A1,A2;
parameter real high=5,low=0;
parameter real vtrans=(high+low)/2;
parameter real tdel=0n, trise=0.3n;
parameter real tfall=0.3n;
parameter real dt=trise/100;
// current state
integer LogicClk,LogicVin;
integer LogicA1,LogicA2;
// states for track-and-hold
integer Ka1,Ka2;

analog begin
// Convert signals to logic format:
LogicClk = (V(clk)>vtrans);
LogicVin = (V(d)>vtrans);
LogicA1 = (V(A1)>0.5);
LogicA2 = (V(A2)>0.5);

// measure input at crossing:
@(cross( V(clk)-vtrans,+1 ))
LogicClk=0;

// compute new logic states desired:
Ka1 = LogicClk? LogicA1:LogicVin;
Ka2 = LogicClk? LogicA1:LogicA2;

// remove AC component of signal
I(A1) <+ ddt(dt*V(A1));
I(A2) <+ ddt(dt*V(A2));
I(A1) <+ (V(A1)-Ka1);
I(A2) <+ (V(A2)-Ka2);

// output using transition statement
// inverse of output
V(q) <+ transition(Ka2?high:low,tdel,
trise,tfall);
V(qbar) <+ high+low - V(q);

end
endmodule
```