Synplicity

Synplify Pro에 서 Synthesis후 에 ModelSim에 서 Simulation하 기 위 한 절 차 및 방 법

(Post-Synthesis Simulation using Synplify Pro and ModelSim)

PLDWorld.com

- Inner Secret Devices -

This page is Blank…

Written by

Kyoung-mo KIM

(Sr. F.A.E. / Synplicity Korea / kmkim@synplicity.com)

Decorated by

Chang-woo YANG

(Webmaster / PLDWorld / podongii@PLDWorld.com)

Revision 2.0

Jan. 04, 2004

PLDWorld.com

- Inner Secret Devices -

This page is Blank...

Synplify Pro에서 Synthesis 후에 ModelSim에서 Simulation 하기 위한 절차 및 방법

FPGA를 이용해서 Project를 진행하면서 합성을 한 후에 실시하는 Post-synthesis Simulation은 Timing이 아닌 Function simulation이라고 봐야 한다. 그 이유는 합성 툴에서 만들어주는 Netlist는 Timing data가 없는 Function에 대한 부분만, 즉, Logic에 대한 기능적인 부분만을 포함하는 것이 대부분이기 때문이다.

그렇지만 합성 후에 Simulation을 한다는 것은 RTL source code를 합성 툴이 맞게 합성을 했는지를 검증해 보는 것이다. 하지만 종종 합성 후에는 내가 보고자 하는 신호들이 Optimize 되어서 Netlist에는 안 나타나는 경우가 있다. 그래서 합성 후에 Simulation 단계에서 내가 체크 해보고자 하는 신호들이 Optimize 되어서 다른 이름으로 바뀌거나 보이지 않게 되는 것을 합성 툴에서 막아줄 필요가 있다.

Synplify Pro에서는 이러한 목적이나 기타 다른 목적 등을 위해서 Optimize 되는 것을 막기 위한 Attribute들이 있는데 이것들을 살펴보고, 사용 방법들을 알아보도록 하자.

1. Attributes

(1) syn_keep : 내부 Net에 대한 Optimize를 막아준다.
(2) syn_preserve : Flip-flop과 같은 Sequential 소자들의 Optimize를 막아준다.
(3) syn_noprune : Designer가 어떤 목적 하에 Instantiation 해준 Component들의 Optimize를 막아준다.
(4) syn_probe : Debugging 작업이나 보드 테스트 등의 목적을 위해서 Design 내부의 신호나 Port등을 칩의 Output 핀으로 만들어 준다.
(5) syn_hier : Hierarchy design에서 특정 Hierarchy의 Interface의 대한 유지나 Optimize를 결정한다. 여기에 사용할 수 있는 값들로는 Hard, Soft, Flatten, Macro등이 있고 각각에 따라 Interface에 대한 유지의 정도가 결정된다.

2. Syntax

(1) Verilog HDL :

Object /* synthesis attribute_name = <value> */;

(2) VHDL :

Attribute attribute_name : <type>;
Attribute attribute_name of object : object_type is <value>;

혹은 library를 사용 할 수 있다. 예를 들면,

library synplify;
use synplify.attributes.all;

3. Examples

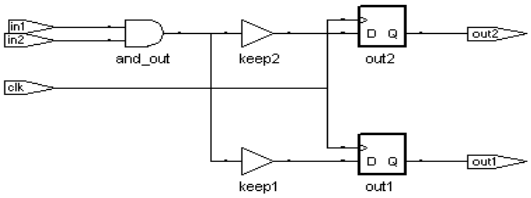
syn_keep Example

```
module syn_keep (out1, out2,clk, in1, in2);

output out1, out2;
input clk;
input in1, in2;

wire and_out;
wire keep1 /* synthesis syn_keep=1 */;
wire keep2 /* synthesis syn_keep=1 */;
reg out1, out2;

assign and_out = in1&in2;
assign keep1 = and_out;
assign keep2 = and_out;

always @(posedge clk)begin
   out1 <= keep1;
   out2 <= keep2;
end

endmodule
```

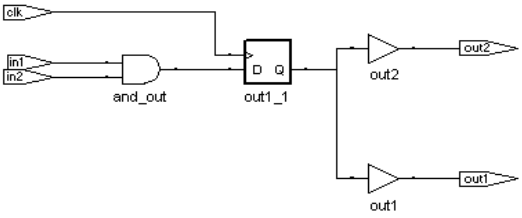
```
entity syn_keep is
  port(in1, in2 : in bit;
       clk : in bit;
       out1, out2 : out bit);
end syn_keep;

architecture rt1 of syn_keep is
  attribute syn_keep : boolean;
  signal and_out, keep1, keep2: bit;
  attribute syn_keep of keep1, keep2 : signal is
true;
  begin

    and_out <= in1 and in2;
    keep1 <= and_out;
    keep2 <= and_out;

    process(clk)
      begin
      if (clk'event and clk = '1') then
        out1 <= keep1;
        out2 <= keep2;
      end if;
    end process;
end rt1;
```

syn_keep Example

Use *syn_keep* at the input of the registers to get registered outputs for out1 and out2



Without *syn_keep* out1 and out2 optimize into one register.



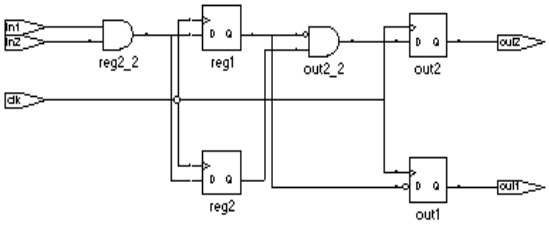
syn_preserve Example

```
module syn_preserve (out1, out2, clk, in1, in2)
/* synthesis syn_preserve=1 */;

output out1, out2;
input clk;
input in1, in2;

reg out1;
reg out2;
reg reg1;
reg reg2 ;

always@ (posedge clk)begin
   reg1 <= in1 &in2;
   reg2 <= in1&in2;
   out1 <= !reg1;
   out2 <= !reg1 & reg2;
end
endmodule
```

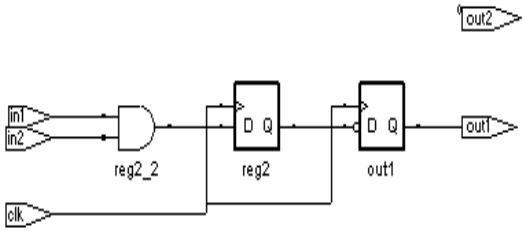
```
entity syn_preserve is
  port (out1    : out  std_logic;
        out2    : out std_logic;
        in1,in2,clk :  in std_logic);
end syn_preserve;

architecture behave of syn_preserve is
  attribute syn_preserve : boolean;
  attribute syn_preserve of behave: architecture
     is true;
  signal reg1 : std_logic;
  signal reg2 : std_logic;
  begin

    process(clk)
      begin
        if (clk'event and clk = '1') then
          reg1 <= in1 and in2;
          reg2 <= in1 and in2;
          out1 <= not ( reg1);
          out2 <= (not (reg1) and reg2) ;
        end if;
    end process;
end behave;
```

syn_preserve Example

Use *syn_preserve* to keep *reg2* and *out2* register from getting optimized



Without *syn_preserve reg2* and *out2* registers are optimized with a warning



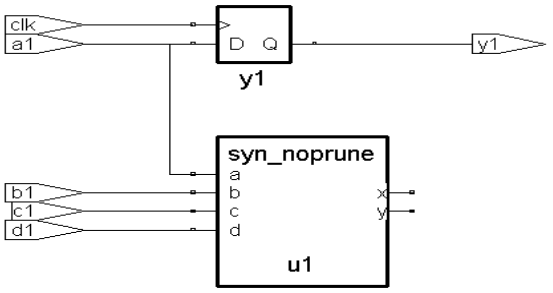
syn_noprune Example

```
module top (a1 ,b1 ,c1, d1, y1, clk);

output y1;
input a1, b1, c1, d1;
input clk;
wire x2, y2;
reg y1;
syn_noprune u1 (a1, b1, c1, d1,x2,y2)
/*synthesis syn_noprune=1 */;

always @(posedge clk)
  y1 <= a1;
endmodule

module syn_noprune (a, b, c, d, x, y);
output x, y;
input a, b, c, d;

assign x = a&b;
assign y = c&d;

endmodule
```

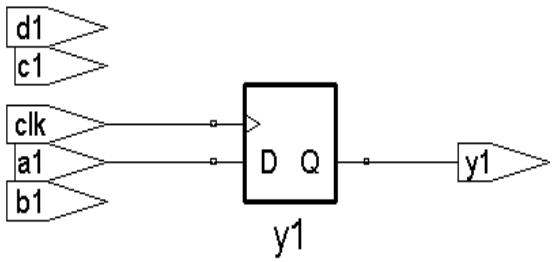
```
entity noprune is
  port (a, b, c,d  : in std_logic;
        x,y   : out std_logic);
end noprune;
architecture behave of noprune is
  begin
-- Empty architecture represents a black box.
end behave;

entity top is
  port (a1, b1, c1,d1,clk : in std_logic;
        y1   :out std_logic);
end top;
architecture behave of top is
  component noprune
    port (a, b, c, d : in std_logic;
          x,y    : out std_logic);
  end component;
  signal x2,y2 : std_logic;
  attribute syn_noprune : boolean;
  attribute syn_noprune of u1 : label is true;
  begin
    u1: noprune port map(a1, b1, c1, d1, x2, y2);
    process(clk)  begin
      if (clk'event and clk = '1') then
        y1 <= a1;
      end if;
    end process;
end behave;
```

syn_noprune Example

Set *syn_noprune=1* to prevent *U1* from being optimized away



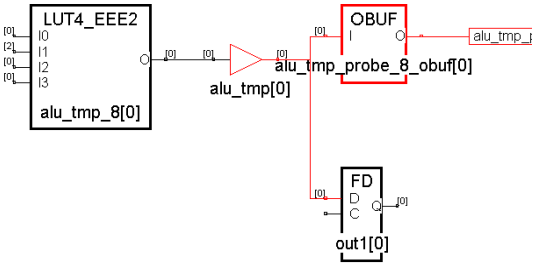
Without *syn_noprune, U1* is removed



syn_probe Example

```
module alu(out1, opcode, a, b, sel, clk);

output [7:0] out1;
input [2:0] opcode;
input [7:0] a, b;
input sel, clk;
reg [7:0] alu_tmp /* synthesis syn_probe=1 */;
reg [7:0] out1;

always @(opcode or a or b or sel)
begin
  case(opcode)
    3'b000 : alu_tmp <= a + b;
    3'b001 : alu_tmp <= a - b;
    3'b010 : alu_tmp <= a ^ b;
    3'b011 : alu_tmp <= sel ? a : b;
    default : alu_tmp <= a | b;
  endcase
end
always @(posedge clk)
  out1 <= alu_tmp;
endmodule
```

Output port created for each bit of alu_tmp



	Enabled	Object Type	Object	Attribute	Value	Val Type
1	☑	net	n:alu_tmp[7:0]	syn_probe	1	boolean
2	☑					
3	☑					

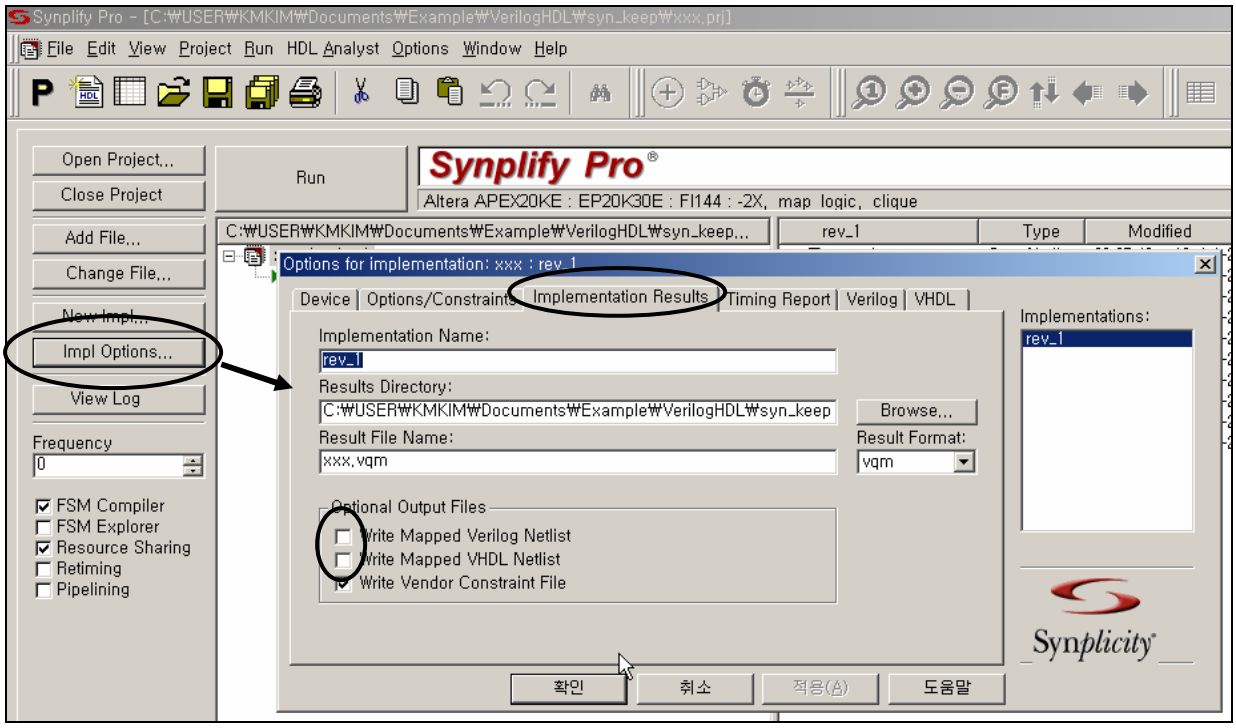
Inputs/Outputs | Registers | Multi-Cycle Paths | False Paths | Attributes | Ot

4. Prepare the Netlist for Post-Synthesis Simulation

Synplify Pro에서는 합성에 대한 결과로 Netlist와 P&R용 Constraint file을 만들어 주는데, Default로는 P&R에서 사용하기 위한 형태의 EDIF type이나 Verilog type의 Netlist를 만들어 준다. 그러나 보통의 Simulator에서는 Post-synthesis simulation을 하기 위해서는 합성 후에 만들어 주는 VHDL 혹은 Verilog HDL type의 Netlist가 필요하다.

이것을 위해 Synplify Pro에서는 약간의 작업이 필요한데, 다음의 그림에서 보듯이 Synplify Pro의 왼쪽 메뉴 버튼 중에서 "Impl Options..."라는 버튼을 누르면 Pop-up window가 하나 나타나게 되고, 거기에는 RTL source code type에 따라서 5~6개의 Tap이 있는데, 그 중에서 "Implementation Results"라는 탭을 선택해서, 아래에 위치한 "Optional Output Files" 라는 부분에서 자신에게 필요한 Type의 Netlist 출력을 위한 Option을 선택하면, 그에 해당하는 .vhm(VHDL type) file이나 .vm(verilog HDL type) file을 얻을 수 있다. 이것을 이용해서 Post-Simulation을 수행할 수 있다.

이 때 VHDL type의 경우에는 VHDL에 자체에 대한 Version이 있는데 Synplify Pro에서는 기본적으로 93 version으로 만들어 주고, Library등에 사용, 지원되는 VHDL file들은 모두 93 version을 지원한다. 이것은 Simulator에서, 특히 ModelSim에서는 Source들을 Compile 할 때 반드시 확인해 줘야 만 Compile error를 막아 줄 수 있다. 이것에 대한 그림은 실제 툴 사용법에 대한 내용 부분에서 Compile에 대한 부분에서 확인 할 수 있다.



5. Post-Synthesis Simulation using ModelSim

ModelSim에는 Simulation에 대한 두 가지의 라이센스가 존재한다. 하나는 VHDL type에 대한 것이고, 다른 하나는 Verilog HDL에 대한 것이다. 사용하는 방법이나 Command들도 각각에 따라서 약간의 차이가 있는데, 가장 큰 차이점은 VHDL의 Library 때문이다.

Synplify Pro에서는 두 가지에 대한 출력을 모두 만들 수 있기 때문에 자신이 가지고 있는 Simulator에 대한 라이센스에 맞는 출력을 선택하거나, 라이센스에 대한 제약이 없다면 Testbench file type이나 자신의 편리에 의해서 출력 File의 Type을 결정해서 만들어 주면 된다.

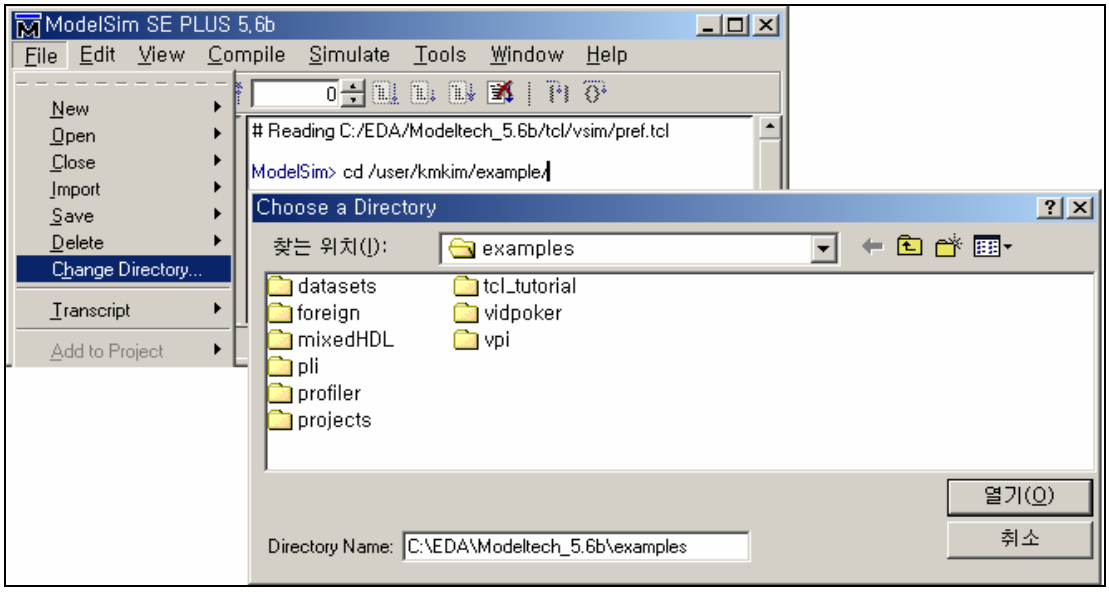
아래의 설명에서 각 단락의 위에 있는 명령어들은 Batch job을 위한 Tcl command들 이므로 Batch file을 만들고, 이를 이용해서 작업을 하고자 할 때 사용하면 편리하다.

(1) VHDL

여기 예제에서는 Altera사의 APEX20KE device를 사용하는 것을 다루고 있다. Xilinx device를 사용하는 경우는 Target library에 대한 부분이 달라지고 나머지는 유사하거나 같다.

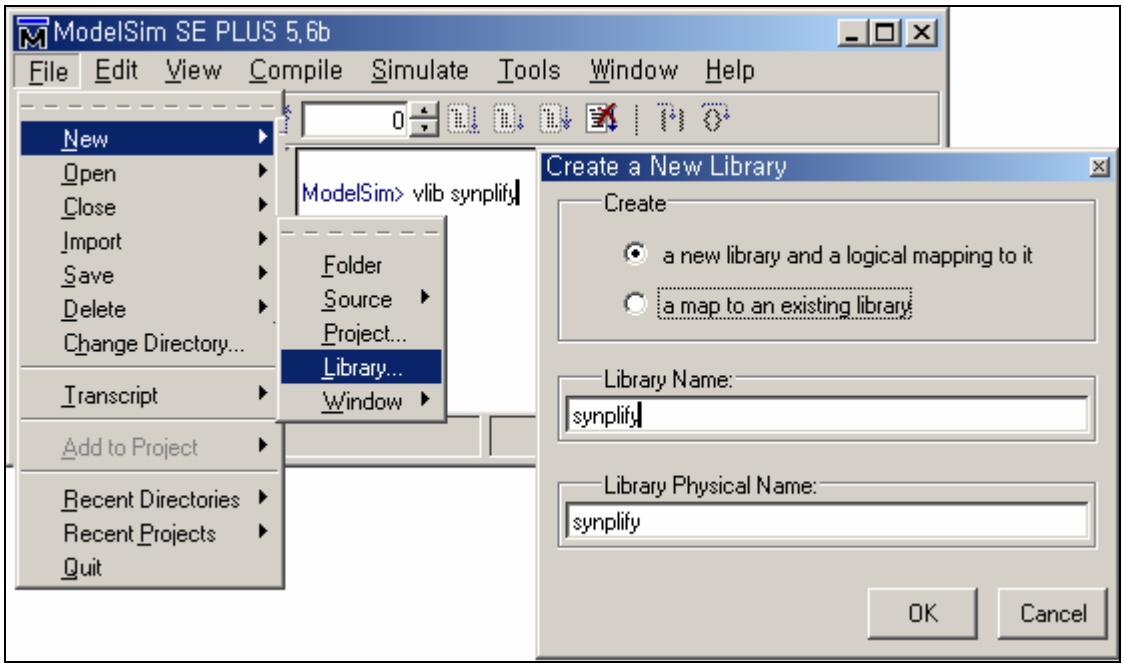
- ***cd <working directory>***

우선은 맨 먼저 툴을 띄운 후에 상단에 있는 여러 Menu 중에서 File을 선택한 후, 아래 항목 중에서 Change Directory를 선택한다. 그러면 Pop-up window가 나타나고 이것을 이용해서 해당하는 Directory로 이동한다. 이러한 진행들은 Main window 상의 message 창에서 Tcl command로 나타나기 때문에 이것들은 이용하면 Batch file을 쉽게 만들 수 있다.



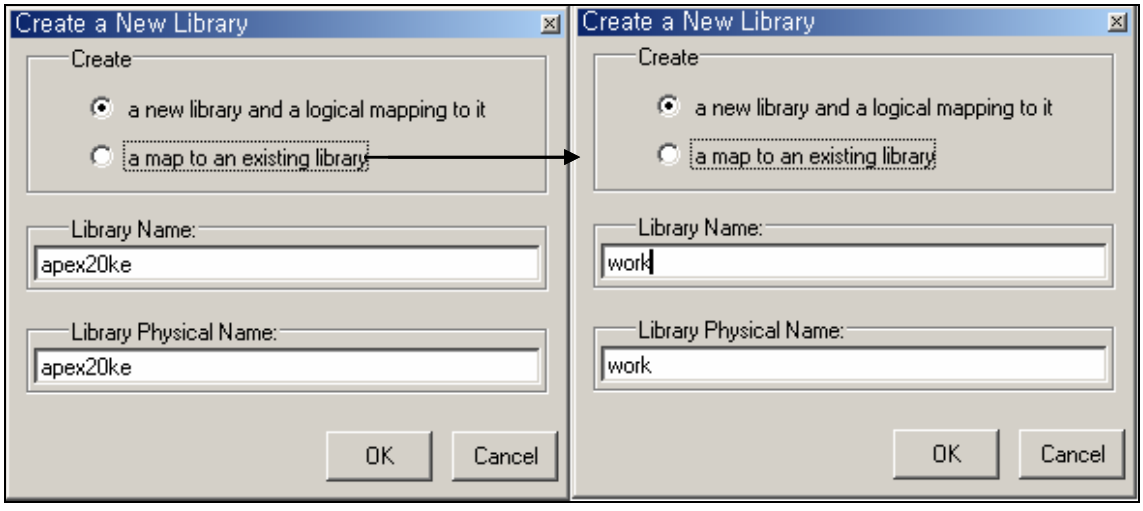
- ***vlib synplify***

그리고 다시 File Menu에 가서 New 항목에서 Library를 선택하면, 다시 Pop-up window가 하나 나타나고, 여기에 Library 이름을 써 준다. 이때 Library를 새로 만들어 주는 것인지, 아니면 이미 존재하는 Library를 이용하기 위해 단지 Library를 Mapping 시켜주는지를 선택해 준다.



- ***vlib apex20ke, vlib work***

같은 방법으로 필요한 Library 들을 만들어 주거나 Mapping 시켜 준다. 예제처럼 Altera사의 사용하는 Device인 APEX20KE에 대한 Library를 만들어 주고, 최종적으로 Work library를 만들어 준다.



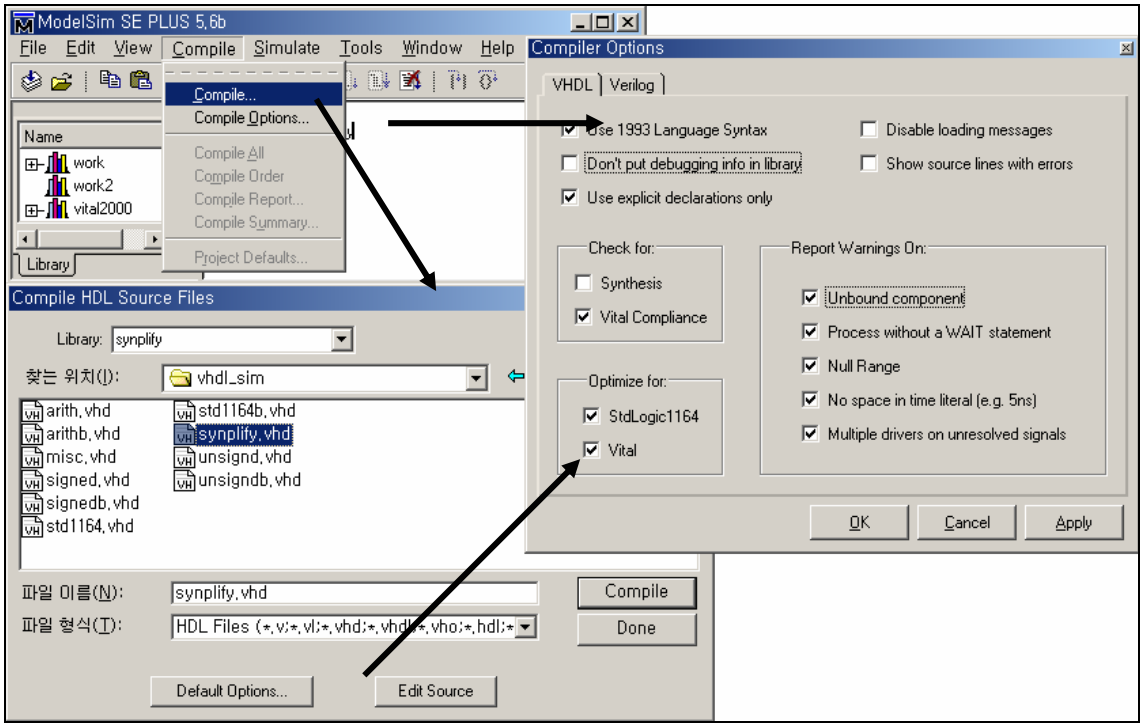
- ***vcom -93 -explicit -work synplify <Synplify Pro installed drive & directory>/lib/vhdl_sim/synplify.vhd***

이제는 Compile을 해야 할 차례이다. 그래서 위의 메뉴 중에서 "Compile"을 선택하고, 아래 List에서

다시 "Compile"을 선택하면 "Compile HDL Source Files"라는 Pop-up window가 나타나고, 순서대로 Compile을 수행하면 된다. 여기서 순서는 결국 Design의 Hierarchy와 관계가 있는데, 우선 사용하는 Library에 대해서 Compile을 해주어야 한다. 그래서 우선 Pop-up window에서 "Library" 항에서 Synplify를 선택해 준 다음, Synplify Pro가 설치되어 있는 Directory에서 lib/vhdl_sim directory에서 synplify.vhd file을 선택해서 "Compile" 해주면 된다. 이 때 우선 Compiler Option에서 VHDL version에 대한 Option이 Enable되어 있는지를 체크하고, 만일 Disable되어 있다면 반드시 Enable 해주고 나서 Compile 해야 Compile Error를 막을 수 있다. Compiler Option을 확인하기 위해서 Pop-up window를 띄우려면, 다음 그림처럼 "Compile HDL Source Files" pop-up window의 아래 부분에 있는 "Default Options..."라는 버튼을 누르거나 "Compile" menu에서 "Compiler Options..."을 선택하면 "Compiler Options"라는 Pop-up window가 나타난다. 여기에서 해당 Option들을 확인해서 사용할 수 있다.

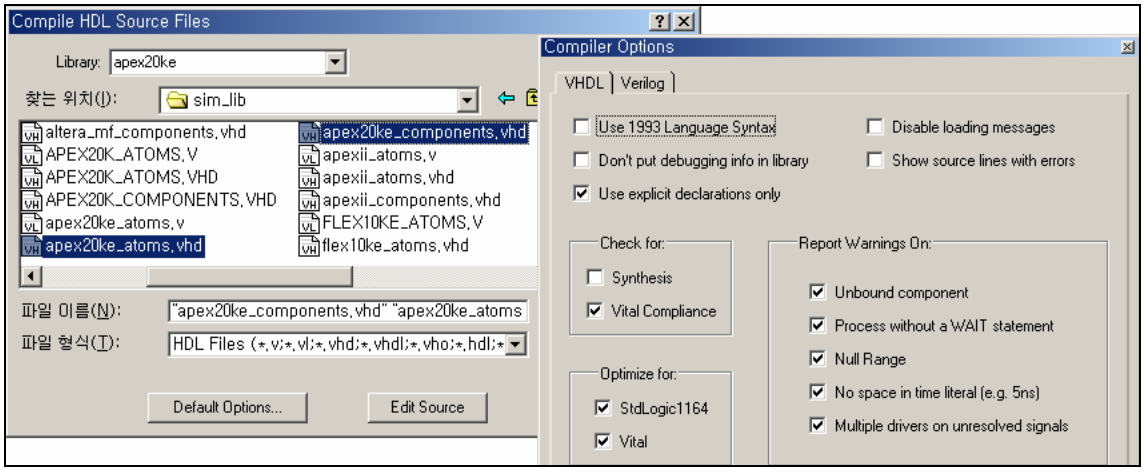
이것은 나중에 Synplify Pro에서 만들어낸 Simulation용 VHDL type의 Netlist를 Compile 할 때에도 마찬가지이다. 반면, Altera사에서 제공하거나 만들어내는 VHDL type의 출력들은 87 version을 Default로 사용하기 때문에 Compile 할 때에는 이러한 Vendor 들에서 지원하는 VHDL의 version을 반드시 알고 그에 따른 Option을 사용해야 불필요한 Compile Error들을 막을 수 있다.

그리고 만일 Design과 합성 단계에서 Attribute들을 사용하기 위해서 Attribute Library를 사용했다면 결과 Netlist에도 이것에 대한 부분이 존재하면서, synattr.vhd file에 대한 것을 요구 할 수 있다, 이 때에도 역시 위의 synplify.vhd와 똑같은 방법으로 Compile 해주면 된다.



- ***vcom -87 -explicit -work apex20ke <Quartus installed drive & directory>/eda/sim_lib/apex20ke_atoms.vhd, apex20ke_components.vhd***

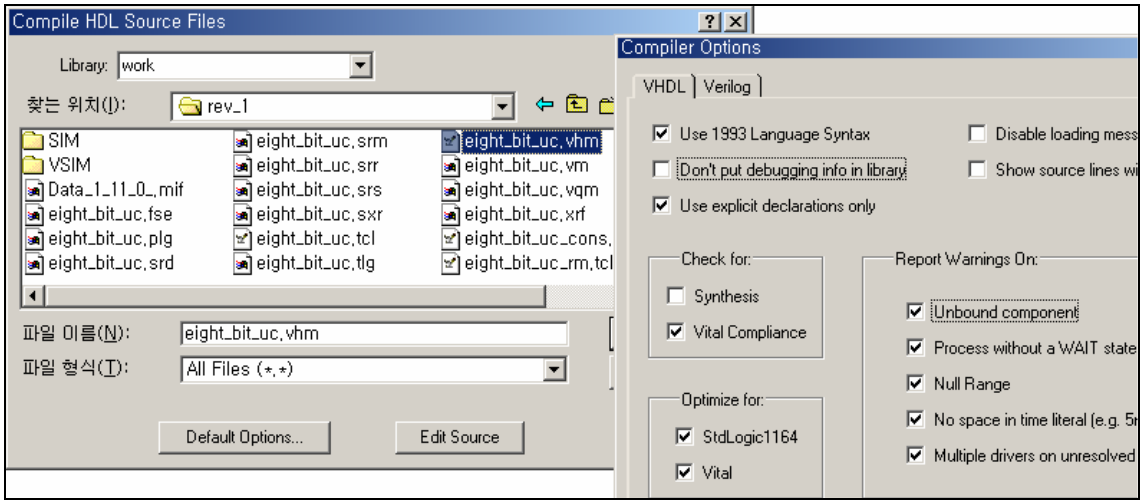
이제는 실제 사용하는 FPGA device에 대한 Library를 Compile 해 주어야 한다. 예제에서처럼 Altera사의 APEX20KE device를 사용했다면, 우선 Library를 apex20ke로 선택하고, 이것에 대한 Library용 VHDL file을 찾아서 Compile 하면 된다. 이 때 ModelSim에서 Altera사의 이러한 Device에 대한 Compile 되어있는 Library를 제공하고 있다면 그것을 Library Mapping해서 사용하면 되고, 그렇지 않다면 보통은 Quartus가 설치된 Directory에서 ~/eda/sim_lib directory에 가면 각 Device에 해당하는 VHDL 혹은 Verilog file들을 발견할 수 있다. 여기서 해당하는 VHDL file을 선택해서 Compile 해 주면 된다. 이 때 두 개 이상의 Library용 VHDL file을 사용하는 경우에는 순서에 맞춰서 Compile 해줘야 Error를 막아 줄 수 있다. Error 발생시에는 순서를 달리해서 다시 Compile 해 주면 된다. 이 때에도 위의 Synplify Pro에 대한 경우와 마찬가지로 Compiler Option에서 각 FPGA vendor들이 지원, 제공하는 VHDL version에 대한 Option을 확인해 주는 것이 불필요한 Compile Error를 막아 줄 수 있다.



- ***vcom -93 -explicit -work work <working directory>/<design_filename>.vhm***

Compile에서 최종적으로 가장 중요한 Netlist를 Compile해 주기 위한 단계이다. 물론, Testbench file을 이용해서 Simulation을 한다면 Testbench file을 마지막으로 Compile 해 줘야 한다. VHDL netlist를 Compile하기 위해서 위에서와 마찬가지로 우선, Library를 work로 선택해 주고, 해당하는 Design의 VHDL netlist를 선택, Compile 해 주면 된다. 참고로, Synplify Pro에서 만들어주는 VHDL type의 Netlist의 확장자는 .vhm을 사용하고, 이것은 VHDL Mapping netlist를 의미한다. Verilog의 경우에는 .vm의 확장자를 사용한다. 역시 마찬가지로 이 때에도 VHDL의 Version에 대한 Option을 확인하고 Compile 해준다. Synplify Pro에서는 93 version의 VHDL Netlist를 만들어 준다.

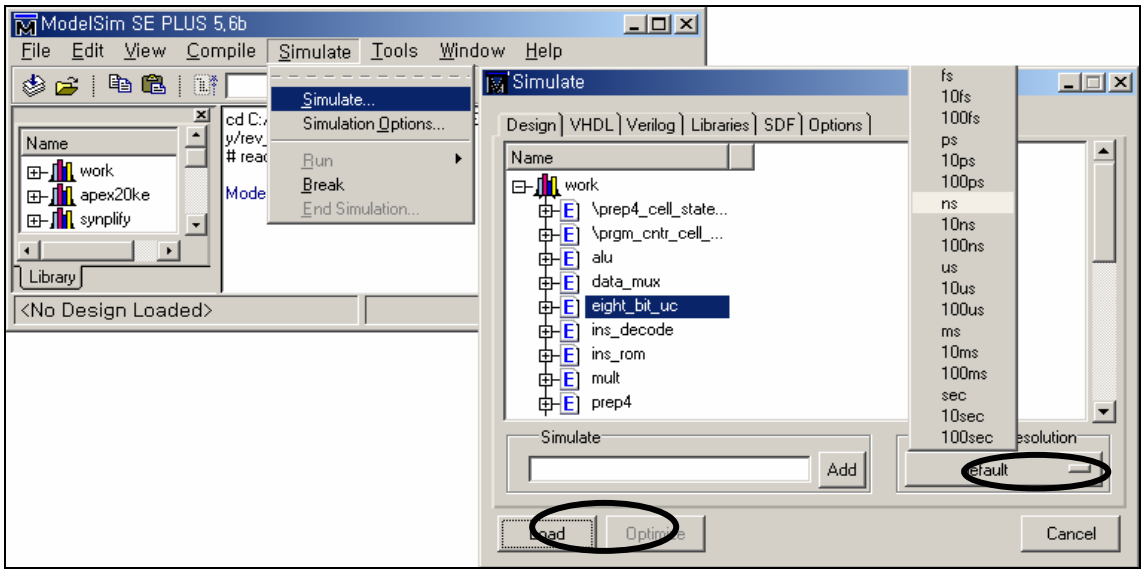
그리고 Testbench file이 있으면 맨 마지막에 Compile 해 준다. 만일 Testbench file이 없다면 ModelSim의 Command들을 사용해서 Interactive 하게 Simulation 할 수 있다. 이 때 사용하는 명령어들에 대한 정보는 ModelSim의 Main window에서 Help를 이용해서 찾아 사용할 수 있다. (help commands)



- ***vsim -t ns work.<top level name>***

Compile이 Error 없이 다 끝났으면 이제는 진짜 Simulation을 할 차례이다. 우선 Simulation을 하기 전에 Simulator의 Resolution을 자신의 Design에 맞게 조정해 줄 필요가 있다. 이것은, 위의 Menu 중에서 "Simulate"를 선택하면, "Simulate"라는 Pop-up window가 나타나고, 여기에서 오른쪽 아래 부분에 있는 "default"라는 버튼을 누르면 각 resolution에 대한 단위들이 나타나고, 여기에서 자신에게 맞는 Resolution을 선택하면 된다.

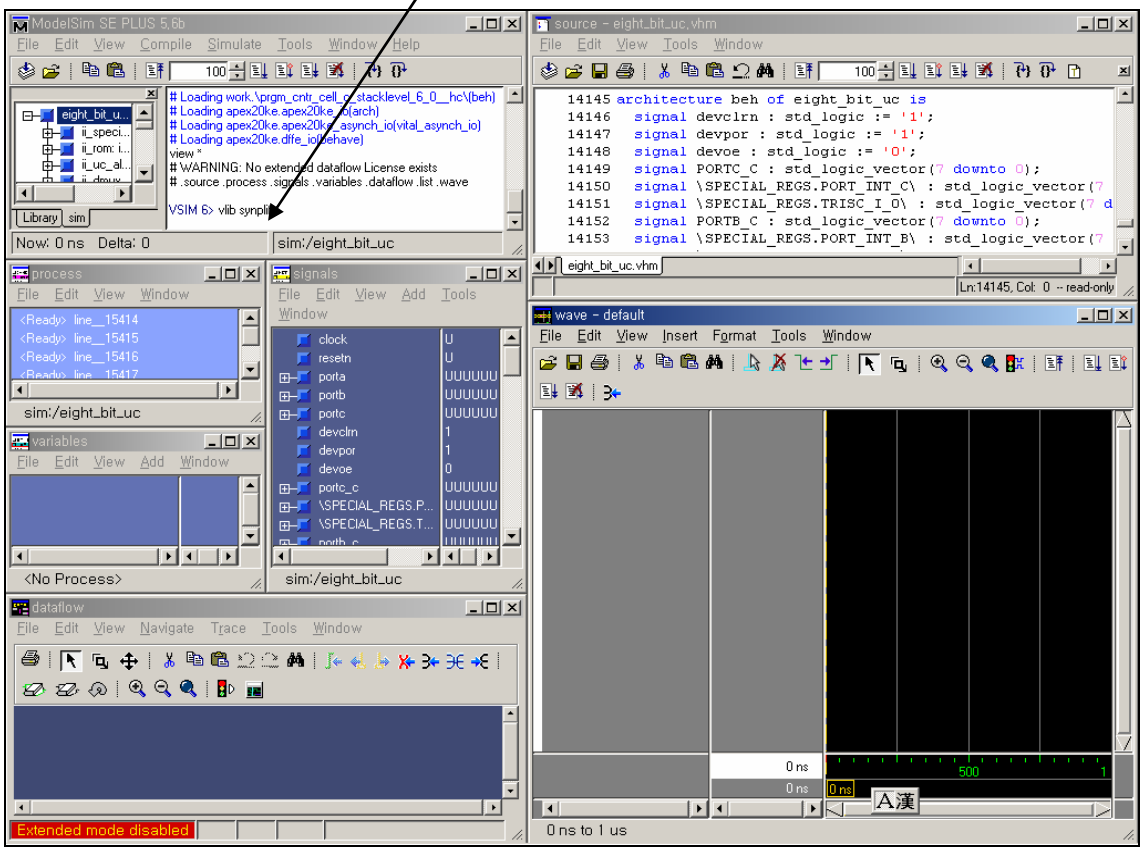
Resolution을 선택했으면 이제는 Top level의 Design을 선택해야 할 차례이다. 여기서 Testbench file이 있다면 Top level은 바로 Testbench file이 된다. 이 때 File 이름과 Testbench에서의 Entity 이름이 다르다면 Simulator에서는 Entity이름으로 나타나게 되기 때문에 혼동하지 말아야 한다. 그리고 나서 최종적으로 "Load" 버튼을 누르면 Main window상에는 Load하는데 따른Mmessage들이 나타나게 되고, 간혹 Library나 기타 이유로 Error가 나타나게 되는데, 이 때에는 Message를 보고 이것에 대한 이유를 분석해서 조치를 취해주면 된다. Netlist에서 사용한 Library와 Vendor에서 제공하는 Library간에 Version이 맞지 않아서 Error가 나오는 경우도 있기 때문에 아무 문제가 발견되지 않았다면 가능한 최신 버전들을 사용하는 것이 이러한 Error를 방지할 수 있다.



Load에서도 아무런 문제가 없다면 이제는 Simulation을 진행하면서 결과를 분석하고 확인해야 할 차례이다. 결과를 확인하는 방법에 따라서 자신이 원하는 방법을 선택하면 된다. Waveform을 통해서 확인, 분석한다면 "View" Menu에서 "Wave"를 선택하고, Table 형태로 보고 싶다면 "List"를 선택해서 확인하면 된다. 원활한 확인과 분석작업을 위해서 제공하는 모든 기능들을 보고 싶다면 "All windows"을 선택하면 모두 볼 수 있다. 그러나 너무 많은 Window들을 보고 있으면 혼란스러울 수 있으므로 최소한의 window를 이용하는 것이 여러 가지 측면에서 좋다.

자신이 원하는 기능들의 Window를 선택했으면 우선, Waveform이나 Table에서 자신이 확인하고자 하는 신호들이나 핀들을, "Structure"와 "Signals" window등을 이용해서 선택해준다. 이렇게 선택된 신호나 핀들은 Waveform viewer나 List에 나타나고, Simulation이 진행되는 동안 선택된 신호들에 대한 결과 값이 Waveform viewer나 List에서 Waveform이나 Table 형태로 나타난다.

Simulation을 진행하기 위해서는 "Simulate" menu의 Run 항목에서 자신이 진행하고 싶은 시간단위로 선택하거나, Main window의 Command prompt에서 명령을 직접 쳐서 실행시켜준다. 예를 들면, run 1000 ns, 혹은 run –all 등과 같이 실행해 준다. 그리고 나서 이제는 결과를 확인해주면 된다.



(2) Verilog

Verilog HDL은 Library들을 사용하지 않기 때문에 VHDL 보다는 훨씬 간단하다. 여기서는 간단하게 절차만을 설명하겠다.

- ***cd <working directory>***

"File" menu에서 Change Directory를 선택해서 해당 작업 Directory로 이동한다.

- ***vlib work***

그리고 Simulation을 위한 work library를 만들어 준다. 그러나 여기서는 VHDL처럼 여러 Library를 만들지 않아도 되기 때문에 편리하다. 절차는 File menu에서 New를 선택하고 Library 항목을 선택한 후에 나타나는 Pop-up window에서 work library를 만들면 된다.

- ***vlog -work work {<Quartus installed drive & directory>/eda/sim_lib/apex20ke_atoms.v}***

이제는 work library에서 해당하는 Verilog 파일들을 차례대로 Compile 해주면 된다. 우선, 사용한 FPGA

vendor의 Library cell에 대한 Verilog 파일을 Compile 해준다. 예제에서처럼 Altera사의 APEX20KE device를 사용했다면, Quartus tool이 설치된 Directory의 하위폴더(~/eda/sim_lib/)에서 apex20ke_atoms.v file을 찾아서 Compile 한다.

- ***vlog -work work {<working directory>/design_filename.vm}***
- ***vlog -work work {<working directory>/ testbench_filename.v}***

Library cell에 대한 Compile에서 아무 문제가 없었다면, 이제는 실제 Design에 대한 Netlist를 Compile 할 차례이다. Synplify Pro에서 만들어주는 Verilog HDL netlist에 대한 File 확장자는 .vm file 이다. 이것을 Compile하고 난 후에, Testbench file이 있으면 마지막으로 Compile 해준다.

- ***vsim -t ns work.<top level name>***

모든 Compile이 끝났다면, 이제는 본격적인 Simulation을 할 차례이다. 여기서는 VHDL과 똑같은 방법으로 진행하면 된다. 위의 Menu중에 Simulate에서 Simulate를 선택하면 Pop-up window가 나타나고, 여기에서 최종 Top-level의 이름을 선택한 후에, Simulation resolution을 확인 하고, Load 버튼을 눌러서 Design을 Load 해준다. 이 때 Testbench file이 있으면 Testbench가 Top이 되고, 그렇지 않으면 Design netlist가 Top이 된다. Testbench가 없으면 Command 형태로 Simulation을 진행할 수 있도록 준비된 .do file을 이용할 수도 있고, 만일 .do file이 없다면 Command들을 이용해서 순서대로 Simulation을 진행해준다. Command들을 잘 모를 경우에는 Help를 이용해서 해당 Command들에 대한 사용법이나 예제를 이용해서 작업하거나 이러한 일련의 작업들을 .do file로 저장해서 나중에 다시 이용할 수도 있다.

물론 Simulation을 위해서 View menu를 이용해서 필요한 View item들을 열어 놓고 작업한다는 것을 잊지 말자. 이러한 모든 작업들에 대한 Log는 Main window의 Message 창에 기록되기 때문에 이것들을 이용하면 쉽게 .do file를 작성하거나, Batch job file을 만들 수 있다.

6. 맺음말

미흡하나마 Synplify Pro를 사용해서 합성을 하시는 설계자 분들께 합성 후에 Simulation을 할 때 유익한 정보가 될 수 있기를 바라는 마음에서 나름대로 정리했습니다.

문서의 내용 중 의문이 가시거나, 이상한 부분, 추가했으면 하는 내용, 오탈자등이 있을 땐, 주저 말고 연락 바랍니다...

- **연락처 :**

Kyoung-mo KIM (김경모)

(Sr. F.A.E. / Synplicity Korea / kmkim@synplicity.com)

Revision History

- 2002-8-5 - Ver 1.0: Initialize Release…
- 2003-10-13 - Ver 1.0: Document 작업…
- 2003-11-3 – Ver 2.0: 2 차 문서 보완…
- 2004-1-4 – Ver 2.0: Document 작업…