How to use NetlistGen.exe

In this manual we focus on using NetlistGen for generating Verilog netlist useful for fault simulation, power estimation, fault tolerant design, and etc.

For using netlist generator, you should first install Xilinx ISE web pack edition. The installation process is listed below:

- Download windows based Xilinx ISE web pack edition from: http://xilinx.com
- Click on "*setup.exe*" and follow the setup process.
- Choose web pack edition for installation.
- Leave the default options of checked boxes unchanged.
- Click on the install button to start installation.
- After installing Xilinx ISE, for generating a Verilog netlist from a behavioral Verilog code, the *NetlistGen.exe* file should be copied in the location as the behavioral description of your Verilog code file. A partial Verilog code showing the required input/output format and a part of the RT level behavioral code is shown in Figure 1.

```
module Controller ( reset, clk, op_code, rd_mem, wr_mem, ir_on_adr,  pc_on_adr
                , dbus_on_data, data_on_dbus, ld_ir, ld_ac, ld_pc, inc_pc, clr_pc, pass_add, alu_on_dbus );
  input reset;
  input clk;
  //------------------------
  input[1:0] op_code ;
  output rd_mem;
  output wr_mem;
  output ir_on_adr;
  …
  reg rd_mem, wr_mem, ir_on_adr, …

  `define Reset 2'b00
  `define Fetch 2'b01
  `define WaitState 2'b10
  `define Execute 2'b11
  reg [1:0] present_state, next_state;

  always @( posedge clk )
  begin
  …
  end
  endmodule
```

Figure 1. An example design format for NetlistGen

- As shown, port declaration format for behavioral top module should list input output declarations after the module header. *NetlistGen* cannot translate input or output in the port identifier list. Same applies to port reg declaration.
- When the RT level input is prepared above, the following steps will result in the generation of a netlist.
- Run *NetlistGen.exe*.
- When prompted, choose a name for your project. This name will become the project name of the synthesis process of ISE.
- When prompted for the name of module, enter the name of the top module of your project. The name of the Verilog file of the top module should be the same as the name of the module. After that, *NetlistGen* reads the behavioral Verilog file. If you want to add any other Verilog

files to your design, you can add them in the next step. Added files should be available in the root directory.

- Now you should synthesize your design with Xilinx ISE. For that, choose "y" to synthesize your design. The NetlistGen uses the intermediate EDIF format, that is converted from Xilinx Synthesis Technology (*xst.exe*) NGC output, for generating netlist.
- NetlistGen runs *xst.exe* in TCL shell environment. The synthesis process will be done automatically if the location of the *xtclsh.exe* file is specified correctly. If you installed the ISE in a different location, or you use other versions of ISE, you should enter the correct path of the installation. For example if you use Xilinx_ISE_DS_Win_12.2 version, you should enter C:\Xilinx\12.2\ISE_DS\ISE\bin\nt, which is the path of *xtclsh.exe* file for the 32-bit Windows NT OS, in the installation path request prompt.
- After selecting the location of the *xtclsh.exe* file, the synthesis process will start. The process will be completed successfully if there are no compilation and synthesis errors.
- NetlisGen will generate the netlist in the *netlist_MODULENAME_V1.v* file. Part of what is automatically generated for example of Figure 1 is shown in Figure 2.

```
module Controller_net(reset,
clk,
op_code,
...);
input reset;
input clk;
input [0:1]op_code;
output rd_mem;
output wr_mem;
output ir_on_adr;
…
wire
wire_1,
wire_2,
wire_3,
wire_4,
wire_5,
wire_6,
...
pin #(4) pin_0 ({reset, clk, op_code[0...
pout #(13) pout_0 ({rd_mem_net_0,...

fanout_n #(8, 0, 0) FANOUT_1 (wire_2, {wire_2_0, wire_2_1, wire_2_2, wire_2_3, wire_2_4,
wire_2_5, wire_2_6, wire_2_7});
fanout_n #(7, 0, 0) FANOUT_2 (wire_16, {wire_16_0, wire_16_1, wire_16_2, wire_16_3,
wire_16_4, wire_16_5, wire_16_6});
fanout_n #(2, 0, 0) FANOUT_3 (wire_12, {wire_12_0, wire_12_1});
fanout_n #(3, 0, 0) FANOUT_4 (wire_10, {wire_10_0, wire_10_1, wire_10_2});
fanout_n #(2, 0, 0) FANOUT_5 (wire_13, {wire_13_0, wire_13_1});
bufg #(0, 0) BUF_1 (alu_on_dbus_net_0, wire_1);
and_n #(2, 0, 0) AND_1 (wire_1, {wire_2_0, wire_3});
bufg #(0, 0) BUF_2 (wire_4, clk_net_0);
bufg #(0, 0) BUF_3 (clr_pc_net_0, wire_11);
bufg #(0, 0) BUF_4 (data_on_dbus_net_0, wire_12_0);
bufg #(0, 0) BUF_5 (dbus_on_data_net_0, wire_13_0);
bufg #(0, 0) BUF_6 (inc_pc_net_0, wire_9);
...
dff DFF_1  (wire_3, wire_5, wire_4, 1'b0, 1'b0, 1'b1, NbarT, Si, 1'b0);
dff DFF_2  (wire_7, wire_6, wire_4, 1'b0, 1'b0, 1'b1, NbarT, Si, 1'b0);
dff DFF_3  (wire_9, wire_8, wire_4, 1'b0, 1'b0, 1'b1, NbarT, Si, 1'b0);
dff DFF_4  (wire_11, wire_10, wire_4, 1'b0, 1'b0, 1'b1, NbarT, Si, 1'b0);

endmodule
```

Figure 2. Netlist format of Controller module

Using custom cell library

If in the netlist generation process an EDIF primitive is needed that is not available in the internal library of *NetlistGen,* the following error will be issued.

ERROR : The cell library file cannot be found. The Cell CELL referenced from xilinx synthesis tool is not supported by our default cells. For creating a custom cell library file refer to NetlistGen manual. Press any key to continue...

In this case, to continue the netlist generation process, you should make a custom cell library file for defining the specified cell. The name of the file should be "EDIF_Cell_Library.txt", and you should put this file in the root directory. For defining cells, the following directives should be used:

//cell cellName --Declares the name of the cell. The name should be the same as the name of the cell in the EDIF file

//inputs --Used for input declaration

//outputs --Used for output declaration

//equations --Used for defining function of the cell

//end cell --Used for defining the end of the cell declaration

dff --Used for defining D-Flip Flops

assign --Used for defining combinational part of the cell

symbol	function	example
!	Inverter	assign temp = !w; assign temp = !(A || B);
&&	AND operation	assign temp = A && B;
||	OR operation	assign temp = A || B;
^	XOR operation	assign temp = A ^ B;
( )	Giving priority to operations	assign temp = (A&&S)||(B&&!S)

Defining a combinational custom cell in EDIF_Cell_Library.txt

For defining a combinational cell you should only use **assign** statements. For defining the name of the cell you should use *//cell* keyword. The name of the cell should be the same as that of the specified name in the EDIF file.

Inputs of the cells should be listed after the *//inputs* keyword in a line. Inputs should be the same as those of in the UNISIM library of the EDIF file. Outputs of each cell should be listed after the *//outputs* keyword in a line. Like input definitions, output port names should be the same as those used in the UNISIM library of the EDIF file.

After declaring inputs and outputs, you should define the function of the cell after the *//equations* keyword. You can write equations using the *assign* keyword similar to Verilog use of **assign**. After each definition a semicolon is needed. After defining the cell, *//end cell* keyword in a separate line is needed. Figure 3 shows the definition of a simple OR2 gate that is already predefined in the internal cell library of NetlistGen.

Defining a sequential custom cell in EDIF_Cell_Library.txt

For defining a sequential cell for custom cell library you can add D-Flip-Flop using the *dff* keyword. Each *dff* has six interfaces and is declared as shown below :

dff (Q, D, C, CLR, PRE, CE);

```
//cell OR
//inputs
I0
I1
//outputs
O
//equations
assign O = I0 || I1;
//end cell
```

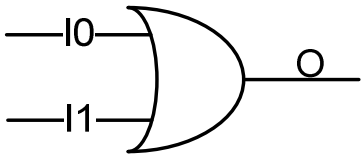

Figure 3. Defining an OR gate in custom cell library

In this line, Q is the output of the flip flop, D is the input, C is the clock signal, CLR is the reset signal, PRE is the pre set signal, and CE is the clock enable signal. Function of the *dff* is the same as the function of *dff* in the "component_library.v" file. You can disable each signal, if needed, by assigning 1'b0 or 1'b1 as in Verilog.

For designing a sequential cell, you should first change the design into a combinational part and memory part (Hoffman model), and after that you should use **assign** statement and *dff*(s) for modeling combinational and sequential parts. Figure 4 shows an example for a sequential cell.

```
//cell Seq1
//inputs
A
B
Clock
Reset
//outputs
Output
//equations
assign temp = A || B;
dff (Output, temp, Clock, Reset, 1'b0, 1'b1);
//end cell
```

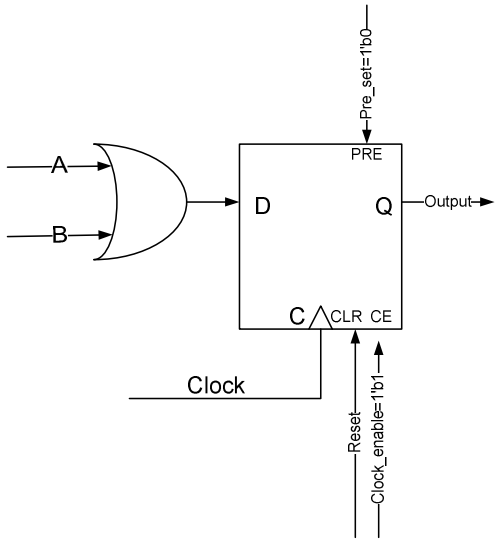

Figure 4. Defining a simple sequential cell in the custom cell library

Multiple cells can be defined in the "EDIF_Cell_Library.txt" file. NetlistGen will issue a note when it refers to the custom cell library.

M. Hashem Haghbayan
hashem@cad.ut.ac.ir