High-Level Design Tips for Synopsys FPGA Express

by KIRK A. OWYANG ◆ Manager, Corporate Applications Engineering ◆ Synopsys, Inc.

The advantages of using high-level design languages (HDLs) for FPGA design have become readily apparent as device densities skyrocket to more than 500K gates. The benefits of using VHDL or Verilog HDL for design entry and logic synthesis are especially great if you are using the XC4000E/X, XC5200, Spartan, or Virtex FPGA families. When you design with HDLs, the key to obtaining the best performance using Synopsys FPGA Express technology is to adopt applicable high-level design practices and techniques, some of which are described in this article.

Specifying Design Constraints and Requirements

The most effective way to increase the performance of your design is to thoroughly specify your detailed design constraints and requirements in FPGA Express. Performance requirements include system clock speed, multi-cycle and other sequential path timing delays, and input/output timing. You can also synthesize for best area or speed performance, preserve or flatten design hierarchy, and control operator (resource) sharing. In addition, entering your design constraints improves more than the synthesis results; place and route results also improve because constraints are passed as TIMESPECs in the XNF (or NCF) files.

You can also enter other design specifications. In the FPGA Express interface, you can specify and control port/pad locations, global buffer (BUFG) allocation, slew rate, and I/O register insertion.

All of these constraints and requirements enable FPGA Express and the Xilinx software to extract the best performance from the devices.

Describing Finite State Machines

There are specific ways to describe finite state machines (FSMs) in VHDL and Verilog that result in one-hot or binary encoding. You can create any finite state machine using the following FSM templates, which give optimal synthesis results with FPGA Express synthesis technology.

VHDL Template

In VHDL, you can enumerate states symbolically or you can assign values to states with the ENUM_ENCODING attribute.

FPGA Express supports automatic FSM encoding for enumerated types in VHDL. To take advantage of automatic encoding:

1. Use the VHDL template without the ENUM_ENCODING attribute.
2. From ***Synthesis*** > ***Options*** > ***Project***, choose One Hot or Binary encoding. The default is One Hot.

Use the ENUM_ENCODING attribute when you want to control the state encoding.

The following example of a simple counter shows the VHDL template using an enumerated type for states (shown in bold text). Note that the WHEN OTHERS statement is not needed for this template, though it can be used if required.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity shift_enum is
port (CLK,RST : bit;
 I : std_logic_vector
 (2 downto 0);
 O : out std_logic_vector
 (2 downto 0));
end shift_enum;

architecture beh of shift_enum is

type state_type is (S0, S1, S2);
— Do not use the following 2 lines
for automatic FSM extraction:
attribute ENUM_ENCODING: STRING;
attribute ENUM_ENCODING of
state_type: type is “001 010 100”;

signal machine : state_type;

begin
 process (CLK,RST)
 begin
  if RST= ‘1’ then
   machine <= S0;
  elsif CLK= ‘1’ and CLK’event
then
   case machine is
   when S0 => machine <= S1;
   when S1 => machine <= S2;
   when S2 => machine <= S0;
   end case;
  end if;
 end process;

with machine select
 o <= “001” when S0,
 “010” when S1,
 “100” when S2;
end;
```

Verilog Template

Verilog does not support the enumerated type. However, you can use the Verilog template to design a finite state machine and fully control the state encoding.

In Verilog, state values are defined with the parameter statement. The conventions for this Verilog template are:

- Use the parameter statement to define state values.
- Use a CASE statement and the Synopsys directive //synopsys parallel_case full_case to describe the state machine.

The following example of the same simple counter described in the VHDL template shows the Verilog template using the parameter statement for state values.

```
module shift (clk, rst, in, out);
input clk, rst;
input [2:0] in;
output [2:0] out;
parameter [2:0]
  S0 = 3’d1,
  S1 = 3’d2,
  S2 = 3’d4;
reg [2:0]
  state, next_state ;

always @ (in or state)
begin
  case (state) // synopsys
  parallel_case full_case
  S0: next_state = S1;
  S1: next_state = S2;
  S2: next_state = S0;
  endcase
end

always @ (posedge clk or posedge
rst)
begin
  if (rst) state <= S0;
  else state <= next_state;
end

assign out = state;
endmodule
```

Conclusion

There is no question that the future of the FPGA industry depends upon very large and complex FPGAs such as those in the new Xilinx Virtex family. To reap the maximum benefit from these new devices, FPGA design engineers must adopt design methodologies (such as the use of HDLs) that closely mimic the traditional ASIC design flow. A couple of the key components of a successful HDL-based design flow for Xilinx customers, described above, are detailed design constraint capture and adherence to specific templates for FSM descriptions. Following simple design practices like these ensure the best performance for your Xilinx design.

For more information, see the online help in the FPGA Express software. ◆

“There is no question that the future of the FPGA industry depends upon very large and complex FPGAs such as those in the new Xilinx Virtex family.”