Inferring Read Only Memory in FPGA Compiler II and FPGA Express

by Alan Ma
Senior Corporate Applications Engineer, Synopsys, Inc.
alanma@synopsys.com

Prior to the recent advancements in FPGA technology, you had to rely on external RAM or ROM. Now, with the introduction of million-gate FPGAs such as the Xilinx Virtex devices, you have access to abundant on-chip memory resources. FPGA Compiler II/FPGA Express (FCII/FE) takes advantage of Virtex resources such as Look-Up Tables (LUTs) MUXF5s, MUXF6s, and on-chip block SelectRAM to provide the highest quality of results for ROM functions.

Coding Styles

Version 3.4 of FCII/FE recognizes a ROM description using CASE statements in both Verilog and VHDL, and by using constant arrays in VHDL.

Using CASE Statements

FCII/FE infers ROM when the inputs to the CASE statement are constant and all the states are specified. Figure 1 shows an example of an 8x4 ROM in Verilog. Figure 2 illustrates its VHDL equivalent.

Using Constant Arrays

You also have the option of using constant arrays in VHDL. Figure 3 describes the coding style for the same 8x4 ROM where CONV_INTEGER is a built-in function that converts std_logic_vector to integer.

General Implementations

FCII/FE generally implements ROM using LUTs, MUXF5s, and MUXF6s when targeting Xilinx Virtex devices. The actual resources used are closely related to the width of the address port. If the address has less than or equal to four bits, then the ROM will be implemented using LUTs. If the address has five bits, then MUXF5s, (which provide multiplexer functions in one half of a Virtex Configurable Logic Block), will be used in addition to the LUTs. If the address has between six to nine bits, the ROM will be implemented using LUTs, MUXF5s, and MUXF6s which provide multiplexer functions in a

```
module rom_8x4          (
                        address,
                        rom_out
                        );

input   [2:0]           address;
output  [3:0]           rom_out;

reg     [3:0]           rom_out;

always @(address)
        case (address)
                3'b000  : rom_out <= 4'b0001;
                3'b001  : rom_out <= 4'b0010;
                3'b010  : rom_out <= 4'b0100;
                3'b011  : rom_out <= 4'b1000;
                3'b100  : rom_out <= 4'b1000;
                3'b101  : rom_out <= 4'b0100;
                3'b110  : rom_out <= 4'b0010;
                3'b111  : rom_out <= 4'b0001;
        endcase
endmodule
```

Figure 1 - Using CASE statements for ROM in Verilog.

```
library ieee;
use ieee.std_logic_1164.all;

entity rom_8x4 is port   (
                         address: in std_logic_vector(2 downto 0);
                         output: out std_logic_vector(3 downto 0)
                         );
end rom_8x4;

architecture rtl of rom_8x4 is
begin
        process (address)
        begin
                case address is
                        when "000"      => output <= "0001";
                        when "001"      => output <= "0010";
                        when "010"      => output <= "0100";
                        when "011"      => output <= "1000";
                        when "100"      => output <= "1000";
                        when "101"      => output <= "0100";
                        when "110"      => output <= "0010";
                        when "111"      => output <= "0001";
                        when others     => output <= "0000";
                end case;
        end process;
end rtl;
```

Figure 2 - Using CASE statements for ROM in VHDL.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity rom_8x4 is port   (
                         address: in std_logic_vector(2 downto 0);
                         output: out std_logic_vector(3 downto 0)
                         );
end rom_8x4;

architecture rtl of rom_8x4 is

constant num_word        : integer := 8;
constant num_bit         : integer := 4;
signal int_add           : integer range 0 to 7;

type ROMARRAY is array (0 to (num_word-1)) of
                std_logic_vector((num_bit-1) downto 0);

constant ROM: ROMARRAY := (
        "0001",
        "0010",
        "01                         00",
                                "1000",
                                "1000",
                                "0100",
                                "0010",
                                "0001"
                                );
begin
        int_add <= CONV_INTEGER(address);
        output <= ROM (int_add);

end rtl;
```

Figure 3 - Using constant arrays for ROM in VHDL.

full Virtex CLB. When the address has ten or more bits, FCII/FE implements ROM using on-chip block SelectRAM resources if certain conditions are met.

Using Block SelectRAM

The Virtex series provides dedicated blocks of on-chip, dual port synchronous RAM, with 4096 memory cells (bits). These resources can also be used for ROM if certain conditions are met. Our research indicates that when the address of the ROM has ten bits and the data has more than or equal to three bits, quality of results can be improved by mapping the ROM to block SelectRAM if the output of the ROM is registered. However, if the data has less than or equal to two bits (when the address has ten bits), using LUTs, MUXF5s, and MUXF6s, as described in the previous section, yields better results. For ROM whose address has more than or equal to eleven bits, block SelectRAM will always be used if the output is registered. Table 1 summarizes the conditions for ROM inference.

To reserve block SelectRAM for user-defined functions, you can use the scripting command "set_chip_instantiated_blockram". For example, the following reserves 4096 bits of block SelectRAM for inferred ROM:

```
set_chip_instantiated_blockram 4096
```

While set_chip_instantiated_blockram applies to the current project, you can use the variable proj_user_instantiated_blockram to reserve block SelectRAM globally for all the subsequent projects. For example, the following reserves 4096 bits of block SelectRAM for all future projects:

```
proj_user_instantiated_blockram 4096
```

Conclusion

Inferring ROM is easy with FPGA Compiler II and FPGA Express, which take full advantage of the abundant on-chip memory resources of the Xilinx Virtex devices.

Visit the Synopsys FPGA website at www.synopsys.com/fpga for other information on the latest FPGA synthesis technologies.

		LUT	MUXF5	MUXF6	SelectRAM
ADD <= 4 Bits	Any Data Width	◆			
ADD = 5 Bits	Any Data Width	◆	◆		
6 Bits <= ADD <= 9 Bits	Any Data Width	◆	◆	◆	
ADD = 10 Bits	1 Bit <= DATA <= 2 Bits	◆	◆	◆	
ADD = 10 Bits*	DATA >= 3 Bits*				◆
ADD >= 11 Bits*	Any Data Width*				◆

Table 1 - LUTs, MUXF5s, MUXF6s, and block SelectRAM utilization.

* Note that LUTs, MUXF5s, and MUXF6s will be used if the output of the ROM is not registered.