Memorias ROM: FPGA-VHDL Cómo ???

Nota Técnica 03

Cristian Sisterna

Introducción

Básicamente hay dos formas de que una memoria ROM sea implementada en un FPGA a partir del código VHDL:

- Deduciendo la memoria desde el código VHDL. Comúnmente conocido como Inferir.
- Generando la memoria a partir de la herramienta de generación de IPs ofrecida por el vendedor del FPGA (CoreGen, MegaWizard, etc).

Por portabilidad y code-resuse el primer método, inferir, es el más aconsejado. Pero por supuesto también se puede recurrir a los IPs del FPGA (segundo caso).

A su vez dentro del método de inferir hay diferentes opciones para el modo de 'cargar' la ROM:

1- Declarando un tipo (type), luego una señal de ese tipo y asignarle a la señal valores constantes (datos de la ROM)
2- Declarando los datos de la ROM en un archivo (data file) y usar instrucciones de apertura y lectura de archivos para leer los datos.
3- Usando instrucción 'case'.

Analizando las distintas opciones en general podemos decir el caso 3 se usa cuando la memoria es pequeña, de lo contrario queda un *'case'* tremendamente largo y un poco anti-estético.

El caso 2 depende del entorno del proyecto, hay empresas en las que no les gusta o no están de acuerdo con el acceso a disco para leer/escribir datos, otras que directamente lo prohíben y otras que lo estimulan. Por lo dicho entonces, detallaré la metodología de la opción 1 que es la más comúnmente usada.

Descripción e Implementación de una Memoria ROM

El primer paso es la declaración del arreglo bi-dimensional que fija el tamaño de la memoria. Pero, antes de detallar esta declaración es conveniente declarar también dos constantes: una constante relacionada con el ancho del bus de direcciones de la memoria, y otra

constante relacionada con el ancho del bus de datos de la memoria. Conviene que estas declaraciones sean realizadas como *generics* dentro del componente, así es más fácil la modificación del tamaño de la misma.

De acuerdo a lo descrito, la primera parte del código de la memoria ROM quedaría:

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
4
5 entity sync_rom is
6  generic (data_width  : natural := 16;
7           addr_length : natural := 10);
8  port (
9   clk      :in  std_logic;
10  address :in  std_logic_vector(addr_length-1 downto 0);
11  data_out:out std_logic_vector(data_width-1  downto 0)
12     );
13 end sync_rom;
```

En la parte declarativa de la arquitectura se hacen las declaraciones de:

- constante del tamaño de la memoria basado en el *generic* que declara en ancho del bus de direcciones (línea 2).
- type que declara el tamaño del arreglo bidimensional usando la constante recién declarada y el ancho del bus de datos declarado como *generic* (líneas 3-4).
- constante que determina los distintos valores asignados a cada dirección de la memoria ROM (líneas 5-11).

Por último, en la parte de descripción funcional del componente, se describe el comportamiento de la memoria ROM. Sencillamente, a la señal dato de salida se le asigna el correspondiente valor de la memoria de acuerdo del valor del bus de direcciones. Dos aclaraciones con respecto a la descripción del comportamiento de la memoria:

1- en este caso se describe en un proceso sincrónico (líneas 13-18). Razón?... esto es debido a que por ejemplo para implementar la memoria ROM en un BRAM de los FPGAs de Xilinx, la memoria DEBE SER descripta de modo sincrónico; de lo contrario es implementada en LUTs. Lógicamente que nadie quiere que una gran memoria sea implementada en LUTs teniendo los BRAMs disponibles. En otros casos se puede

directamente hacer una descripción combinacional de la memoria, que directamente sería solo la instrucción de asignación detallada en línea 16.

2- la conversión de 'address' primero a *'unsigned'* usando *cast unsigned*, y luego de *unsigned* a entero (*integer*) usando la función *to_integer*. Con estas conversiones lo que se hace es encontrar el índice de valor entero del arreglo definido como constante. De este modo se accede al valor asignado a esa dirección y que a su vez es asignado a la señal en el RHS de la instrucción (Right Hand Side).

El código de la parte de la arquitectura de la memoria es la siguiente:

```
 1 architecture synth of sync_rom
 2 constant mem_size : natural := 2**addr_length;
 3 type mem_type is array (mem_size-1 downto 0) of
 4            std_logic_vector (data_width-1 downto 0);
 5 constant mem : mem_type :=
 6  (0=> x"abcd", 1=> x"beef", 2 => x"5555", 3 => x"1010",
 7   4=> x"5a6b", 5=> x"f0f0", 6 => x"1234", 7 => x"fabc",
 8   8=> x"2345", 9=> x"9876", 10=> x"5432", 11=> x"6666",
 9  12=> x"0101",
10  13=> std_logic_vector(to_unsigned (1234,16)), others => x"4247");
11

12 begin
13    rom : process (clk)
14    begin
15       if rising_edge(clk) then
16          data_out <=mem(to_integer(unsigned(address)));
17       end if;
18    end process rom;

19 end architecture synth;
```

Línea 10 ejemplifica otro modo de declarar el valor constante del dato correspondiente a una dirección (se puede usar cuando estamos lazzy (flojos) y queremos evitar el cálculo de entero a slv :)).

En caso que el tamaño de la ROM sea muy grande, el tamaño de la declaración de valores del arreglo bidimensional crecerá proporcionalmente y puede llegar a ser de varias hojas de valores, perdiéndose el código VHDL descriptivo de la memoria. Para estos casos lo mejor es declarar las constantes y el arreglo de la memoria en un paquete. Así, el código VHDL es más compacto y el paquete 'oculta' la longitud de la memoria.

Consideraciones de Síntesis

La implementación de la memoria ROM en el FPGA puede realizarse en:

- Lógica distribuida.

- Bloques dedicados de Memoria RAM.

En qué bloque se implementaría la memoria descrita anteriormente ? ? ? ? . . . La implementación depende de varios factores, entre ellos principalmente el tamaño de la memoria (1), pero depende también del 'seteo' de la herramienta de síntesis (2), por último si se usan o no atributos al respecto (3). Veamos:

1- Las herramientas de síntesis tienen un 'umbral de tamaño'. Si el tamaño de la memoria supera ese umbral la memoria implementará en BRAM, y sino supera el umbral se implementará en LUTs. Este umbral depende del tamaño del dispositivo FPGA que se esté usando, de la profundidad de la memoria y del número total de bits de memoria. Por ejemplo en la documentación de Xilinx se puede encontrar la siguiente tabla que muestra los valores de umbral mínimos para implementar memoria en los BRAM:

Rules for Small RAMs

In order to save block RAM resources, XST does not implement small memories on block RAM. The threshold can vary depending on:

- The targeted device family
- The number of addressable data words (memory depth)
- The total number of memory bits (number of addressable data words * data word width)

Inferred RAMs are implemented on block RAM resources when the criteria in the following table are met.

Criteria for Implementing Inferred RAMs on Block RAM Resources

Devices	Depth	Depth * Width
Spartan®-6	>= 127 words	> 512 bits
Virtex®-6	>= 127 words	> 512 bits

Use RAM Style (RAM_STYLE) to override these criteria and force implementation of small RAMs and ROMs on block resources.

Mapping Logic and Finite State Machine (FSM) Components to Block RAM

In addition to RAM inference capabilities, XST can be instructed to implement the following to block RAM resources:

- Finite State Machine (FSM) components

  For more information, see:

  Finite State Machine (FSM) Components
- General logic

  For more information, see:

  Mapping Logic to Block RAM

XST User Guide for Virtex-6, Spartan-6, and 7 Series Devices
www.xilinx.com UG687 (v 13.1) March 1, 2011

Sin embargo, y esto es importante!, se puede 'obligar' a la herramienta de síntesis a forzar la implementación ya sea en LUTs o Bloques dedicados de RAM, dejando sin efecto los valores detallados en la tabla anterior. Para ello se tienen las dos opciones siguientes:

2- Variar el 'seteo' de la herramienta de síntesis. Por defecto el seteo para la implementación de memorias es 'AUTO', esto quiere decir que dependiendo del tamaño será donde se implementará la memoria. Cambiando este valor a BRAM o DISTRIBUITED se logra forzar a la herramienta de síntesis a implementar la memoria donde uno desee.
3- Usar atributos en el código fuente VHDL o Verilog. Para ello, primero se debe declarar el atributo de síntesis respectivo como tipo 'string'. Para el caso de Xilinx XST, el atributo se denomina 'rom_style', para Altera Quartus 'romstyle', y para Synplify 'syn_romstyle'. Una vez declarado el atributo es necesario asociarlo con la señal o variable que representa la memoria ROM que se desea inferir e implementar. Al hacer la asociación se fija también el tipo de memoria que se desea usar para la implementacion de la misma. Este valor depende de la herramienta de sintesis, asi por ejemplo para Quartus los valores permitidos son: "logic", "M512", "M4K", M9K", M144K", "MLAB" o "M_RAM". Para el caso de XST, los valores permitidos son "auto", "block" o "distributed". Finalmente para Synplify (syn_romstyle), los valores permitidos son 'register' o 'block_rom'.

A continuación un ejemplo del uso de atributos de síntesis detallado en las siguientes líneas de código:

```
-- Xilinx XST
1 attribute rom_style : string;
2 attribute rom_style of data_out:signal is "block";

-- Altera Quartus

1 attribute romstyle : string;

2 attribute romstyle of data_out:signal is "M_RAM";

-- Synplify

1 attribute syn_romstyle : string;

2 attribute syn_romstyle of data_out:signal is "block_rom";
```

Estas líneas de código deberían ir a continuación de la línea 11 (del ejemplo anterior) en la parte declarativa de la arquitectura. Para el caso de XST en esta línea se asigna el atributo 'rom_style" a la señal 'data_out' (salida de la ROM) con el valor 'block'. De este modo la ROM se implementará en un BRAM sin importar su tamaño ni el seteo de XST. El otro valor que se le puede asigna al atributo 'rom_style' es 'distributed', el cual hará que la memoria se implemente en lógica distribuida, es decir en LUTs.

Lo mejor para tu próxima implementación!