

Relationally Placed Macros

By: Paul Glover and Steve Elzinga

NOTE: The material contained within this document is out of date, but it is provided as a historical reference.

The Xilinx Implementation Tools offer designers the flexibility and control over their design to enable quick time to market and increased clock speed. One feature of the software is Relationally Placed Macros (RPMs).

RPMs provide order and structure to related design elements without requiring you to specify their absolute placement location on the FPGA die. This gives the implementation tools more flexibility to meet timing whereas floorplanning requires absolute placement of the logic. Without reserving an entire area of the FPGA die, RLOC constraints allow you to increase speed and use die resources efficiently by placing logic blocks relative to each other. The implementation tools can also optimize and merge other logic within the RPM.

^{© 2002-2008} Xilinx, Inc. All rights reserved. All Xilinx trademarks, registered trademarks, patents, and further disclaimers are as listed at http://www.xilinx.com/legal.htm. All other trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Xilinx is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Xilinx makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Xilinx expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.



Description of RPMs

An RPM is a collection of elements (FFS, LUT, CY4, RAM, etc.) put into a group called a "set". The placement of each element in the group is the key to creating RPMs. The location of elements within the set, relative to other elements within the set, is controlled by Relative Location Constraints (RLOCs). The SET names (U_SET, H_SET, and HU_SET) determine which elements, with RLOC constraints, are associated.

A **Hierarchical Set** (H_SET) is implicitly defined by the design hierarchy; a designer cannot define this set. All elements, with RLOC constraints, at one level of hierarchy are implicitly members of an H_SET for that level of hierarchy. By default, all RLOCs on the same level of hierarchy are grouped into one set unless otherwise specified.

User-defined Set (U_SET) constraints allow the designer to group elements regardless of the design hierarchy. All elements, with RLOC constraints, with the same U_SET name are included in the same RPM.

A **Hierarchical User-defined Set** (HU_SET), defines a User-defined subset of the design hierarchy. HU_SET is the same as an H_SET, except it allows the user to define the set name. NGDBuild defines the name for an H_SET.

For detailed information regarding H_SET, HU_SET, or U_SET constraints, please refer to the Constraints Guide RLOC section.

The following symbols (primitives) accept RLOCs:

- Registers
- FMAP, HMAP, F5MAP
- CY4, CY_MUX
- ROM, RAM, RAMS, RAMD
- BUFT
- WAND primitives that do not have a DECODE constraint attached
- LUTs, F5MUX, F6MUX, MUXCY, XORCY, MULT_AND, SRL16, SRL16E, F7MUX, F8MUX

For VirtexTM-II/Pro, the general syntax for assigning elements to relative locations is RLOC=XmYn, where

- "m" and "n" are the relative X-axis (left/right) value and the relative Y-axis (up/down) value, respectively, and
- the X and Y numbers can be any positive or negative integer, including zero.

Because the X and Y numbers in RLOC constraints define only the order and relationship between design elements and not their absolute die locations, their numbering can include negative numbers. Even though you can use any integer for RLOC constraints, it is recommended that you use small integers for clarity and ease of use.

It is not the absolute values of the X and Y numbers that are important in RLOC specifications, but their relative values or differences. For example, if design element A has an RLOC=X3Y4 constraint and design element B has an RLOC=X6Y7 constraint, the absolute values of the X numbers (3 and 6) are not important in themselves. However, the difference between them is important; in this case, 3 (6-3) specifies that the location of design element B is three slices away from the location of design element A. To capture this information, a normalization process is used at some point in the design implementation. In the example just given, normalization would reduce the RLOC on design element A to X0Y0, and the RLOC on design element B to X3Y3.



Grid Systems

With the introduction of the ISE^{TM} 4, an additional grid-based system was introduced. This grid system utilizes the RPM_GRID option for the Virtex-II/Pro device families. In order to utilize the grid-based system, a RPM_GRID constraint must be applied to one member of the set.

For example, the UCF file syntax is:

The difference between the slice-based and grid-based RLOC locations is that the grid-based locations are global rather than CLB or slice-based. The four slices that make up a Virtex-II CLB are aligned in a single column in grid notation. Figure 1 contains an example of the grid-based system:

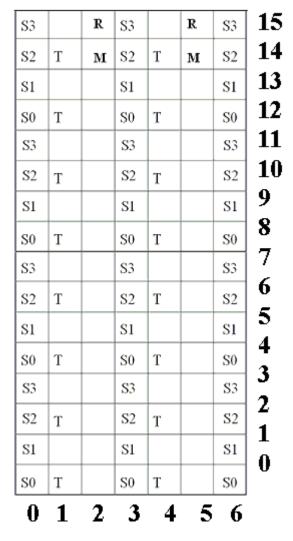


Figure 1: New Grid System



The symbols in Figure 1 represent the following CLB components:

- Slice (S0, S1, S2, S3)
- Three-state buffer (T)
- Block RAM (R)
- Multiplier (M)

For additional information related to the grid-based system, please refer to Xilinx Application Note 416: "RPM Grid Usage".

VHDL Test Case

For an RPM test case, we have chosen a FIR filter example located on www.opencores.org. The FIR filter makes use of, but is not limited to, an adder, a multiplier, and block RAM. Figure 2 illustrates the portion of the FIR filter design that we will focus upon; you may wish to replace key components of the design with constrained instantiated primitives.

The synthesis tools can already infer efficient adders and multipliers for Xilinx FPGAs; therefore, the example is used to demonstrate the passing of the RPM type attributes through VHDL code. By instantiating primitives, you can pass the constraint attributes onto the primitives to "lock" (the relative location - RLOC constraint) a group of primitives together.

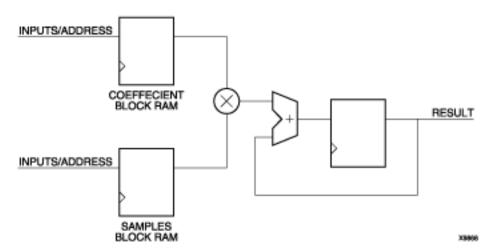


Figure 2: Small Portion of the FIR Filter

Modifying the Test Case:

1. Remove the "+" (add) VHDL operator and replace it with an instantiation of an adder ("mac_rpm_step1.vhd").

The basic full adder is created by instantiating and connecting the adder building blocks from Virtex-II primitives ("adder_prim_step1.vhd").

Figure 3 is an example of a 1-bit adder created from Virtex-II primitives. Two 1-bit adders will fit into one Virtex-II slice. The VHDL "Generate" function will be used to lay out the structure for the instantiated adder primitives.

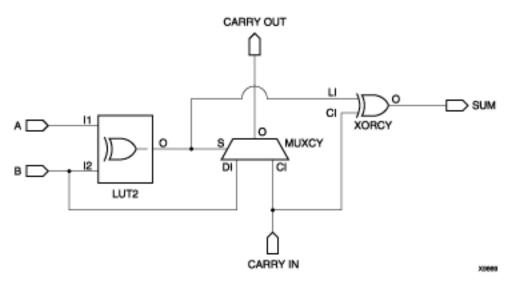


Figure 3: Virtex 1-bit Adder

- 2. Remove the "*" (multiply) VHDL operator and replace it with an instantiation of the Virtex-II multiplier primitive MULT18X18S ("mac_rpm_step2.vhd").
- 3. The design included instantiations of Virtex block RAM primitives, so the next step is to add the constraint attributes in the VHDL code.

The following lines will be added in the declarative region of the adder_arch architecture ("adder_prim_step2.vhd").

attribute RLOC : string; attribute U_SET : string; attribute RPM GRID : string;

Attributes Used:

RLOC:

The RLOC is used to pass the location information onto the instantiated primitives. The primitives in the adder consist of Virtex-II "carry logic". The "carry logic" is laid out vertically on the Virtex FPGA, so the RLOC values will only vary on the "Y" coordinate from the bottom of the slice column to the top of the slice column. The "Y" values for the RLOC constraints are created by passing the indices from the generate statement into the value of the RLOC attribute. The indices are first converted from an integer data type to a string data type by using the 'image VHDL attribute. Although there is a pattern as to how the "Y" value changes, extra manipulation is required in order to obtain the proper value for the new RPM grid system:

"... 'image(integer(index/2) + integer(index/4) + integer(index/4));"

By dividing the index value and converting the real number to an integer the real number will always round down to the closest integer. The "Y" coordinate now increments to the proper slice location for the primitives in the "carry logic" chain. Two sets of RLOC values will be generated for every slice since two 1-bit adders can fit into a single slice (Figure 4).



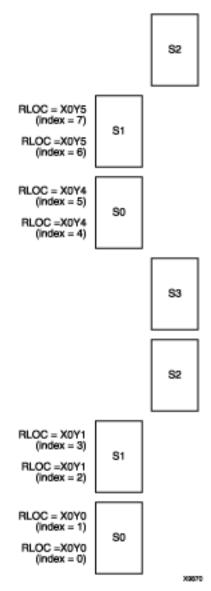


Figure 4: Sample RLOC Values with new RPM_GRID

U SET:

The U_SET attribute is used to group RLOC'ed instances into unique RPMs. In this design, the FIR filter may be cascaded multiple times, which creates the need for unique U_SET names for each instantiation of a FIR filter. If the U_SET attribute is not used, multiple RPM instantiations will result in the implementation tools attempting to pack the same primitive instantiation into the same slice. The U_SET attribute obtains its values from the "generate indices" in the top-level entity, passing them down to the "FIR filter entity" through the VHDL generic map statement. The index value from the generate statements are also passed to the adder instances and block RAM primitives; this creates unique U_SET set names attached to every primitive instantiation in the FIR filter with its associated block RAM.

RPM_GRID:

The RPM_GRID attribute alerts the implementation tools that the new grid notation will be used. The "RPM_GRID attribute" must be passed on to at least one component in a set. In the



example files, provided the RPM_GRID is passed onto every LUT2 component in the set only for convenience.

Synthesis Tool Support

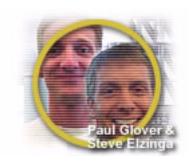
To varying degrees, FPGA ExpressTM, LeonardoSpectrumTM, Synplify®, and XST all support the passing of implementation constraints through VHDL attributes. LeonardoSpectrum, Synplify, and XST support the methods of passing implementation constraints outlined in this article.

However, Synplify requires that all MULT18X18S pins be connected, which is not the case in the FIR filter example. XST will pass the RPM_GRID constraint only on the instantiation of the component and not on the component itself.

<u>Download</u> the completed VHDL files (zipped).

About the Authors

Paul Glover and Steve Elzinga: Product Applications Engineers for Xilinx Colorado



After Steven received a BSEE from the University of Utah with an emphasis in RF and microwave antennas, he began working in the programmable logic industry. He currently serves the Xilinx support organization as an expert in the Synthesis tools utilized by Xilinx. Steve enjoys teaching customer education classes when the opportunity arises. He is currently a Masters candidate student at the University of Colorado and is the proud father of 3 beautiful children. Paul graduated with a BSEE in 1997. He began working in the Xilinx support organization soon after, serving as a Customer Applications Engineer for one year. He has worked as a Product Applications Engineer since 1998. Paul's areas of expertise include the Virtex-II Pro software, DATA2BRAM, Platform Generator, and MicroBlazeTM.

Articles by Paul Glover and Steve Elzinga

Relationally Placed Macros 08/30/2002



Appendix A - Code

mac_rpm_step1.vhd

```
-----
-- Added the adder component to replace the
-- "+" VHDL operator.
component adder_prim is
generic (set_number : integer);
port (a : in std_logic_vector (COEFF_WIDTH+INPUT_WIDTH-1 downto 0);
    b: in std_logic_vector (COEFF_WIDTH+INPUT_WIDTH+TAPS-1 downto
     carry_out : out std_logic;
     sum : out std_logic_vector (b'high downto 0));
end component;
signal sum_b : std_logic_vector (COEFF_WIDTH+INPUT_WIDTH+TAPS-1
downto 0);
signal sum_temp : std_logic_vector (COEFF_WIDTH+INPUT_WIDTH+TAPS-1
downto 0);
-- End of component declaration and needed
-- signals.
begin
-- Instantiation of the adder component
u1 : adder_prim
generic map
(set_number => set_number)
port map
(a => product(COEFF_WIDTH+INPUT_WIDTH-1 downto 0),
b => sum_b,
 sum => sum_temp);
-- End of instantiation
_____
library ieee;
use ieee.std_logic_1164.all;
```

adder_prim_step1.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use work.config.all;
-- Add the following two lines for Synplify
--library virtex2;
--use virtex2.components.all;
entity adder_prim is
```



```
generic (set_number : integer);
port (a : in std_logic_vector (COEFF_WIDTH+INPUT_WIDTH-1 downto 0);
      b : in std_logic_vector (COEFF_WIDTH+INPUT_WIDTH+TAPS-1
downto 0);
      carry_out : out std_logic;
      sum : out std_logic_vector (b'high downto 0));
end entity;
architecture adder_prim_arch of adder_prim is
component MUXCY
port(LO : out std_logic;
     DI : in std_logic;
     CI : in std_logic;
     S : in std_logic);
end component;
component XORCY
port(LO : out std_logic;
      LI : in std_logic;
      CI : in std_logic);
end component;
component LUT2
generic (INIT : bit_vector := "0110");
port(LO : out std_logic;
     I0 : in std_logic;
     I1 : in std_logic);
end component;
signal slct, carryout : std_logic_vector(b'range);
signal carryin : std_logic_vector(b'high+1 downto 0);
signal a_extended : std_logic_vector(b'high downto 0);
begin
a_extended(a'range) <= a;</pre>
a_extended(b'high downto a'high+1) <= (others => a(a'high));
carryin(0) <= '1';
carry_out <= carryout(b'high);</pre>
generate_adder : for index in 0 to b'high generate
begin
gen_LUT2 : LUT2
port map (0 => slct(index),
          I0 => a_extended(index),
          I1 \Rightarrow b(index));
gen_MUXCY : component MUXCY
port map (0 => carryout(index),
          DI => a_extended(index),
          CI => carryin(index),
          S => slct(index));
```



```
gen_XORCY : XORCY
              port map (0 => sum(index),
                       LI => slct(index),
                       CI => carryin(index));
              carryin(index+1) <= carryout(index);</pre>
              end generate;
              end architecture;
mac_rpm_step2.vhd
              component MULT18X18S is
              port (a : in std_logic_vector (17 downto 0);
                    b : in std_logic_vector (17 downto 0);
                         c : in std_logic;
                         r : in std_logic;
                         p : out std_logic_vector (COEFF_WIDTH+INPUT_WIDTH-1
              downto 0));
              end component;
              signal sum_b : std_logic_vector (COEFF_WIDTH+INPUT_WIDTH+TAPS-1
              downto 0);
              signal sum_temp : std_logic_vector (COEFF_WIDTH+INPUT_WIDTH+TAPS-1
              downto 0);
              signal coeff_in_extended : std_logic_vector (17 downto 0);
              signal sample_in_extended : std_logic_vector (17 downto 0);
              ______
              -- End of component declarations and needed
              -- signals.
              _____
              begin
              -- The following was added
              coeff_in_extended(COEFF_WIDTH-1 downto 0) <= COEFF_IN;</pre>
              coeff_in_extended(17 downto COEFF_WIDTH) <= (others =>
              COEFF_IN(COEFF_WIDTH-1));
              sample_in_extended(INPUT_WIDTH-1 downto 0) <= SAMPLE_IN;</pre>
              sample_in_extended(17 downto INPUT_WIDTH) <= (others =>
              SAMPLE_IN(INPUT_WIDTH-1));
              _____
              -- Instantiation of the multiplier primitive
               ______
              mult_primitive : mult18x18s port map
              (a => coeff_in_extended,
```

b => sample_in_extended,

c => clk,
r => rst,
p => product);



adder_prim_step2.vhd

```
-- The RPM_GRID needs to be passed onto only one component in
                -- a macro. In this case the RPM_GRID is being passed onto
                -- all instantiations of the LUT2 primitive only for
                -- convenience sake.
                attribute RPM GRID : string;
                attribute U_SET : string;
                attribute RLOC : string;
                attribute RPM_GRID of LUT2 : component is "GRID";
                -- The XST black box constraint is being passed so that XST
                -- will not generate warnings for every instantiated primitive.
                attribute box type : string;
                attribute box_type of MUXCY : component is "black_box";
                attribute box_type of XORCY : component is "black_box";
                attribute box_type of LUT2 : component is "black_box";
... 'image(integer(index/2) + integer(index/4) + integer(index/4));
                generate_adder : for index in 0 to b'high generate
                -- U_SET attrbutes being passed onto the instantiated adder
                primitives
                -- The U_SET values are integer data types from the generic
                set_number.
                -- The integer data types are then converted to strings by use of
                -- 'image VHDL attribute.
                attribute U_SET of gen_LUT2: label is "set" &
                integer'image(set_number);
                attribute U_SET of gen_MUXCY: label is "set" &
                integer'image(set number);
                attribute U_SET of gen_XORCY: label is "set" &
                integer'image(set_number);
                -- RLOC attrbutes being passed onto the instantiated adder
                primitives
                -- The RLOC values are integer data types from the indices of the
                -- generate statement. The integer data types are then converted to
                -- strings by use of the 'image VHDL attribute.
                attribute RLOC of gen LUT2: label is "X3Y" &
                integer'image(integer(index/2) + integer(index/4) +
                integer(index/4));
                attribute RLOC of gen_MUXCY: label is "X3Y" &
                integer'image(integer(index/2) + integer(index/4) +
                integer(index/4));
                attribute RLOC of gen_XORCY: label is "X3Y" &
                integer'image(integer(index/2) + integer(index/4) +
                integer(index/4));
```



generate indices

```
cascaded_firs : for i in 0 to length-1 generate
begin

rpm : fir_rpm
generic map
(set_number => i)
port map
(clk => clk,
   rst => valid_rpm(i),
   loading_coeff => loading_coeff_rpm(i),
   output_valid => valid_rpm(i+1),
   input => results_rpm(i),
   output => results_rpm(i+1));
```

FIR filter entity

```
cascaded_firs : for i in 0 to length-1 generate
begin

rpm : fir_rpm
generic map
(set_number => i)
port map
(clk => clk,
   rst => valid_rpm(i),
   loading_coeff => loading_coeff_rpm(i),
   output_valid => valid_rpm(i+1),
   input => results_rpm(i),
   output => results_rpm(i+1));
```

RPM_GRID attribute

```
-- The RPM_GRID needs to be declared and set to "GRID" attribute RPM_GRID : string; attribute RLOC : string; attribute RPM_GRID of LUT2 : component is "GRID";
```



Revision History

The following table shows the revision history for this document.

Date	Version	Revision
8/30/02	1.0	Initial Xilinx release.
1/11/08	1.0.1	Converted format from HTML to PDF.

Notice of Disclaimer

The information disclosed to you hereunder (the "Information") is provided "AS-IS" with no warranty of any kind, express or implied. Xilinx does not assume any liability arising from your use of the Information. You are responsible for obtaining any rights you may require for your use of this Information. Xilinx reserves the right to make changes, at any time, to the Information without notice and at its sole discretion. Xilinx assumes no obligation to correct any errors contained in the Information or to advise you of any corrections or updates. Xilinx expressly disclaims any liability in connection with technical support or assistance that may be provided to you in connection with the Information. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE INFORMATION, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT OF THIRD-PARTY RIGHTS.