Using Relative Location Constraints in Synplify For Improved Control of Timing and Placement

A short description of how and why to use RLOCs to control your design.

by Mala Sathyanarayanan, Senior Corporate Applications Engineer, Synplicity, Inc., mala@synplicity.com

In Synplify, you can use Relative Location Constraints, RLOCs, to specifically control how logic is implemented within Xilinx structures such as FMAP, HMAP, and registers. Additionally, Synplify allows you to group these related structures together.

You will ultimately have small sections of your designs where logic placement is crucial to meeting system performances. RLOCs provide a powerful method of controlling placement in performance critical sections. These tailored RLOCs can be made as efficient as Xilinx relatively placed macros. This article summarizes how to use RLOCs with Synplify.

Synplify allows you to specify relative location constraints, which may be placed either in the HDL code or through Synplify's graphical constraint editor, SCOPE (Synthesis Constraints Optimization Environment). These location constraints are passed to the Xilinx place and route tool through the EDIF or XNF netlist. The attributes that you use are:

xc_rloc, **xc_map**, and **xc_uset**.

The example below shows how to implement a 9-input XOR gate into a single CLB. The first step is to define a 4-input XOR function (fmap or 4-input LUT), and a 3-input XOR function (hmap or 3-input LUT). The next step is to instantiate two fmaps (or 4-input LUTs, for Virtex) to XOR eight inputs, and one hmap (or 3-input LUT, for Virtex) to XOR the outputs of the two fmaps (or 4-input LUTs) and the 9th input. The final step is to instantiate the whole CLB in the top level design.

Create modules and specify them as **fmap**, **hmap** or **lut** (for Virtex) as shown below. Make sure that the module mapped to **fmap/lut** has a maximum of four inputs, and the module mapped to **hmap** has a maximum of three inputs.

Verilog

XC4000, Spartan family:

```
module fmap_xor4(z, a, b, c, d)/* synthesis
 xc_map=fmap */; // –(1)
output z;
input a, b, c, d;
assign z = a ^ b ^ c ^ d;
endmodule

module hmap_xor3(z, a, b, c)/* synthesis
 xc_map=hmap */; // –(2)
output z;
input a, b, c;
assign z = a ^ b ^ c;
endmodule
```

Virtex family: Replace line (1) and (2) with:

```
module fmap_xor4(z, a, b, c, d)/* synthesis
 xc_map=lut */; // –(1)
module hmap_xor3(z, a, b, c)/* synthesis
 xc_map=lut */; // –(2)
```

VHDL:

XC4000, Spartan family:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity fmap_xor4 is
 port (a : in std_logic;
       b : in std_logic;
       c : in std_logic;
       d : in std_logic;
       z : out std_logic
    );
end fmap_xor4;

architecture rtl of fmap_xor4 is
attribute xc_map : STRING;
attribute xc_map of rtl : architecture is "fmap"; –(1)
begin
 z <= a xor b xor c xor d;
end rtl;

library IEEE;
use IEEE.std_logic_1164.all;
entity hmap_xor3 is
 port (a : in std_logic;
       b : in std_logic;
       c : in std_logic;
       z : out std_logic
    );
end hmap_xor3;
```

Continued on the following page

VHDL example for XC4000, Spartan family (continued)

```
architecture rtl of hmap_xor3 is
attribute xc_map : STRING;
attribute xc_map of rtl : architecture is “hmap”; —(2)
begin
 z <= a xor b xor c;
end rtl;
```

Virtex family:
Replace lines (1) and (2) above with:

```
attribute xc_map of rtl : architecture is “lut”;
```

Instantiate these modules at a higher level and specify the **xc_rloc** and **xc_uset** attributes on these instances. The **xc_uset** attribute is used to group the instances together, and the **xc_rloc** attribute specifies the relative locations of all the instances with the same value for **xc_uset**.

Verilog

XC4000, Spartan family:

```
module clb_xor9(z, a);
output z;
input [8:0] a;
wire z03, z47;
fmap_xor4 x03 /* synthesis xc_uset=”SET1"
 xc_rloc=”R0C0.f”*/
 (z03, a[0], a[1], a[2], a[3]);
fmap_xor4 x47 /* synthesis xc_uset=”SET1"
 xc_rloc=”R0C0.g” */
 (z47, a[4], a[5], a[6], a[7]);
hmap_xor3 zz /* synthesis xc_uset=”SET1"
 xc_rloc=”R0C0.h” */
 (z, z03, z47, a[8]);
endmodule
```

Virtex family:

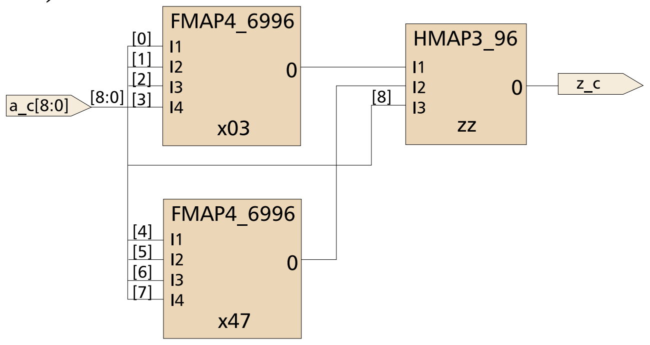
```
module clb_xor9(z, a);
output z;
input [8:0] a;
wire z03, z47;
fmap_xor4 x03 /* synthesis xc_uset=”SET1"
 xc_rloc=”R0C0.S0"*/
 (z03, a[0], a[1], a[2], a[3]);
fmap_xor4 x47 /* synthesis xc_uset=”SET1"
 xc_rloc=”R0C0.S0" */
 (z47, a[4], a[5], a[6], a[7]);
hmap_xor3 zz /* synthesis xc_uset=”SET1"
 xc_rloc=”R0C0.S1" */
 (z, z03, z47, a[8]);
endmodule
```

VHDL:

XC4000, Spartan family:

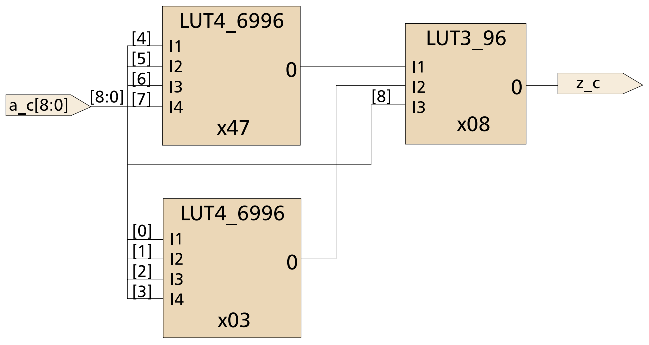
```
library IEEE;
use IEEE.std_logic_1164.all;
entity clb_xor9 is
 port (a : in std_logic_vector(8 downto 0);
       z : out std_logic
    );
end clb_xor9;

architecture rtl of clb_xor9 is
signal z03, z47 : std_logic;
component hmap_xor3
 port (a : in std_logic;
       b : in std_logic;
       c : in std_logic;
       z : out std_logic
    );
end component;

component fmap_xor4
 port (a : in std_logic;
       b : in std_logic;
       c : in std_logic;
       d : in std_logic;
       z : out std_logic
    );
end component;
attribute xc_uset : string;
attribute xc_rloc : string;
attribute xc_uset of x03 : label is “SET1”;
attribute xc_rloc of x03 : label is “R0C0.f”; —(1)
attribute xc_uset of x47 : label is “SET1”;
attribute xc_rloc of x47 : label is “R0C0.g”; —(2)
attribute xc_uset of zz : label is “SET1”;
attribute xc_rloc of zz : label is “R0C0.h”; —(3)
begin
 x03 : fmap_xor4 port map(a(0), a(1), a(2), a(3), z03);
 x47 : fmap_xor4 port map(a(4), a(5), a(6), a(7), z47);
 zz : hmap_xor3 port map(z03, z47, a(8), z);
end rtl;
```

This means all instances that have USET=SET1 should have relative locations R0C0 (meaning they should be in the same CLB).



Virtex family: Replace the xc_rloc value to include the slice also:

```
attribute xc_rloc of x03 : label is “R0C0.S0”; –(1)
attribute xc_rloc of x47 : label is “R0C0.S0”; –(2)
attribute xc_rloc of zz : label is “R0C0.S1”; –(3)
```



Make a top level design, instantiating the CLB:

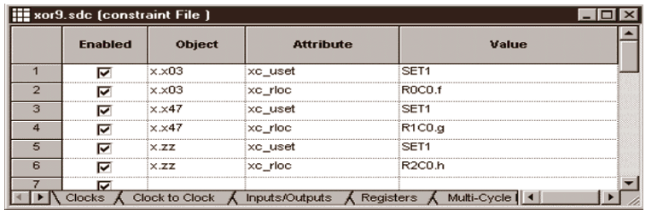
Verilog:

```
module xor9top(z, a);
output z;
input [8:0] a;

clb_xor9 x(z, a);
endmodule
```

This article shows how RLOCs can be embedded in RTL code to control the implementation and placement of logic elements in CLBs. RLOCs are a powerful tool for Synplify designers who need additional control to achieve timing requirements.

VHDL:

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

entity xor9top is
 port (z : out std_logic;
       a : in std_logic_vector(8 downto 0)
    );
end xor9top;

architecture rtl of xor9top is
component clb_xor9
 port (z : out std_logic;
       a : in std_logic_vector(8 downto 0)
    );
end component;
begin

x: clb_xor9 port map ( z, a );
end rtl;
```



	Enabled	Object	Attribute	Value
1	✓	x.x03	xc_uset	SET1
2	✓	x.x03	xc_rloc	R0C0.f
3	✓	x.x47	xc_uset	SET1
4	✓	x.x47	xc_rloc	R1C0.g
5	✓	x.zz	xc_uset	SET1
6	✓	x.zz	xc_rloc	R2C0.h
7	✓			

The RLOC attributes can also be applied through SCOPE: *The constraints are then passed to the Xilinx place and route tool through the EDIF or XNF netlist.*

Conclusion

This article shows how RLOCs can be embedded in RTL code to control the implementation and placement of logic elements in CLBs. RLOCs are a powerful tool for Synplify designers who need additional control to achieve timing requirements.

For more information, please refer to the Synplicity website at http://www.synplicity.com for the full application note. Σ