Using Synplify to Design in Actel Radiation-Hardened FPGAs

Introduction

Actel's RadHard and RadTolerant FPGAs offer advantages for applications in commercial and military satellites, deep space probes, and all types of military and high reliability equipment.

Synplify version 5.31 and later provides designers of radiation-hardened FPGAs an automatic means of steering synthesis away from standard commercial sequential elements (flip-flops). Synplify automatically infers either C-C, TMR, or TMR_CC implementations in place of the normal flip-flops, instead of post-processing the netlist for flip-flop substitution.

This application note is intended to help designers understand the design flow required when using Synplify to design in Actel's radiation-hardened FPGAs.

Actel Register Implementation Techniques

Actel recommends three techniques for implementing the logic of the sequential elements in radiation-hardened FPGAs (C-C, TMR, and TMR_CC).

The C-C technique uses combinatorial cells with feedback (instead of flip-flop or latch primitives) to implement storage. For example, a DFP1 (comprised of two combinational modules) would be used in place of a DF1.

TMR is an acronym for triple-module-redundancy (or triple voting). It is a register implementation technique; each register is implemented by three flip-flops (or latches) that "vote" to determine the state of the register.

TMR_CC is also a triple-module-redundancy technique. Each voting register is composed of combinatorial cells with feedback (instead of flip-flop or latch primitives).

Some techniques are not available or appropriate for all Actel families. Please contact Actel technical support for more information.

Synplify Attribute syn_radhardlevel

Synplify provides an attribute called "syn_radhardlevel" to specify the register implementation technique for designs that use Actel's radiation-hardened FPGAs. You can apply this attribute to a module, architecture, or a register output signal (inferred register in VHDL). If necessary, you can apply it globally to the top-level module or architecture of your design and then selectively override it for different portions. You can also control the design technique you apply on a register by register basis.

The values for "syn_radhardlevel" are:

- "none" - Use standard design techniques
- "cc" - Use C-C implementation
- "tmr" - Use TMR implementation
- "tmr_cc" Use TMR_CC implementation

SEU Resistant Design Techniques

You can influence a device's resistance to SEU (single event upset) effects by using certain logic design techniques. The default technique, using S-FFs, produces designs that are the most susceptible to SEU effects. Because ACT 1 and 40 MX devices do not have S-modules, S-FFs cannot be implemented in these devices.

There are two SEU resistant design techniques (in addition to the default) that can be used in Actel devices with Synplicity. The techniques are, in order of increasing resistance to SEU effects, CC-FFs and triple voting. Synplicity also enables custom implementations. A single design may incorporate any or all of these design techniques.

Using CC-FFs

CC-FFs produce designs that are more resistant to SEU effects than designs that use S-FFs. ACT 1 and 40MX devices use CC-FFs by default. CC-FFs cannot be implemented in 54SX devices at this time. CC-FFs typically use twice the area resources of S-FFs.

Using Triple Voting

Triple voting, or triple module redundancy (TMR), produces designs that are most resistant to SEU effects. Instead of a single flip-flop, triple voting uses three flip-flops leading to a majority gate voting circuit. This way, if one flip-flop is flipped to the wrong state, the other two override it and the correct value is propagated to the rest of the circuit. Because of the cost (three to four times the area and two times the delay required for S-FF implementations), triple voting is usually implemented using S-FFs. However, you can implement triple voting using only CC-FFs in Synplicity.

Figure 1 displays some examples of the register implementation described above after using the "syn_radhardlevel" attribute.

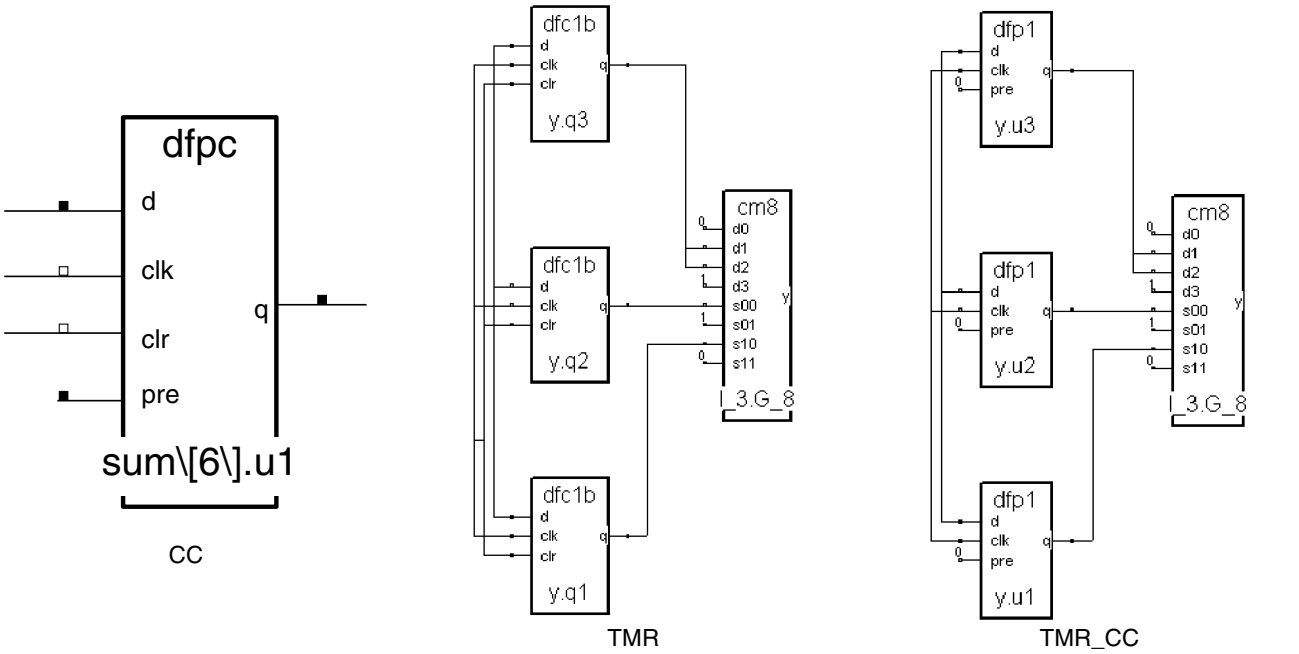

Figure 1 • Logic Implementations of Radiation-Hardened Register

The attribute is only effective if the corresponding Actel Verilog (*.v) or VHDL (*.vhd) macro file(s) for the design technique(s) you use is included in the Source Files list of your Synplify Project. The first Actel file specified in the list determines the default (global) design technique. Then you can use "syn_radhardlevel" to override your defaults on a register by register basis or at the sub-module level.

Using Attributes

You can use the "syn_radhardlevel" attribute in different ways. The following "syn_radhardlevel" examples describe its use in a design constraint, Verilog, and VHDL file.

Constraints File

```
define_attribute {dataout[3:0]}
syn_radhardlevel {"cc"}
```

Verilog

```
module top (clk, dataout, a, b);
input clk;
input a;
input b;
output dataout [3:0];
reg [3:0] dataout
     /* synthesis syn_radhardlevel="tmr" */ ;
/* Other coding */
```

VHDL

```
library synplify;
use synplify.attributes.all;
architecture top of top isattribute
syn_radhardlevel of top: architecture is
"tmr_cc";
-- Other coding
```

Design Example

The design example is not an actual design. It illustrates and example flow for Actel radiation-hardened design and the use of the attribute "syn_radhardlevel." The design is written in Verilog. All source code files are listed in the Appendix. The design has two levels of hierarchies, as shown in Figure 2 on page 3.

The design requirements for the radiation hardened example design are as follows:

1. Default (global) implementation for the registers must be "tmr".
2. Register "b1_int" in "top" module must be implemented as "tmr_cc".
3. All registers in "module_b" module must be implemented as "cc".

4. All registers in "module_d" module must be implemented as "tmr_cc"

.

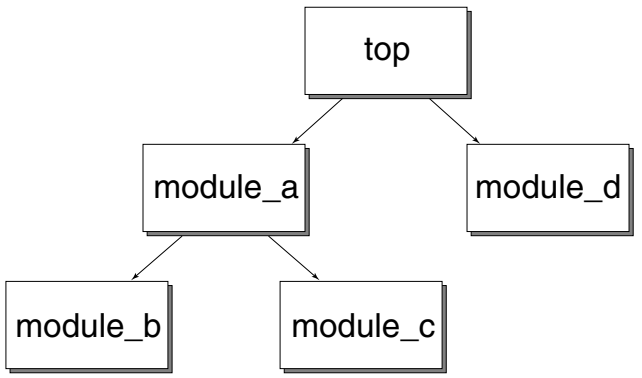

Figure 2 • Example Hierarchy of Design

Use the following steps to complete the design and satisfy the requirements:

1. Bring "top.v" to your favorite editor and make the following edit:

```
reg [15:0] a1_int, b1_int /* synthesis
syn_radhardlevel="tmr_cc" */;
```

Note: *Step 1 is for design requirement B.*

2. Edit "module_b.v" to the following:

```
module module_b (a, b, sub, clk, rst) /*syn-
thesis syn_radhardlevel="cc" */;
```

Note: *Step 2 is for design requirement C.*

3. Edit "module_d.v" to the following:

```
module module_d (a, b, sum, clk, rst) /*syn-
thesis syn_radhardlevel="tmr_cc" */;
```

Note: *Step 3 is for design requirement D.*

4. Bring up Synplify and create a new project.
5. In the Synplify "Set Device Options" window, select an Actel device that is RadHard or RadTolerant.
6. Add the Actel Verilog macro files ("cc.v," "tmr.v," and "tmr_cc.v") to the project, with "tmr.v" listed as the first file.

Note: *Since you use all three register implementations, all three Verilog macro files need to be included in the project. With "tmr.v" listed as the first file, it ensures that the global register implementation is "tmr" (requirement A).*

7. Add all design modules to the project.
8. Click RUN.
9. Click the "Technology View" button to confirm the implementations.

Summary

By using the Synplify attribute "syn_radhardlevel" in conjunction with Actel macro files, Synplify enables you to design in Actel's radiation-hardened FPGAs with little effort. However, it allows you precise control of the register implementation. You only need to focus on controlling designs, not on controlling the tool. The easy and clean flow helps you reduce design cycle and improve productivity.

Appendix

This appendix lists all five modules used in the Design Example section of the application note.

```
/********************************** top.v*********************************/
module top ( a1, b1, sel_byte0, clk, sum_out, sum_carry, sub_out, sub_carry, shft_out, rst);

input [15:0] a1, b1;
input clk,rst,sel_byte0;

output [7:0] sum_out, sub_out;
output sum_carry, sub_carry;

output [8:0] shft_out;

wire [7:0] sum_out, sub_out;
wire sum_carry, sub_carry;

wire [8:0] sum_out_int, sub_out_int, shft_out_int;

reg [15:0] a1_int, b1_int /* synthesis syn_radhardlevel="tmr_cc" */;
reg [7:0] a_byte, b_byte;
regsel_byte0_int;

always @ (posedge clk or posedge rst)
begin
  if (rst) begin
    a1_int <= 0;
    b1_int <= 0;
    sel_byte0_int <= 0;
  end
  else begin
    a1_int <= a1;
    b1_int <= b1;
    sel_byte0_int <= sel_byte0;
  end
end

always @ ( a1_int or b1_int or sel_byte0_int)
begin
  if (sel_byte0_int) begin
    a_byte <= a1_int [7:0];
    b_byte <= b1_int [7:0];
  end
  else begin
    a_byte <= a1_int [15:8];
    b_byte <= b1_int [15:8];
  end
end

module_a i1 (a_byte, b_byte, sub_out_int, shft_out, clk, rst);
module_d i2 (a_byte, b_byte, sum_out_int, clk, rst);


assign sum_out = sum_out_int[7:0];
assign sum_carry = sum_out_int[8];

assign sub_out = sub_out_int[7:0];
assign sub_carry = sub_out_int[8];

assign shft_out = shft_out_int;
endmodule
```

```
/********************************** module_a.v*********************************/
module module_a (a, b, sub, shft, clk, rst);

input [7:0] a, b;
input clk, rst;

output [8:0] sub, shft;

module_b i2 (a, b, sub, clk, rst);
module_c i3 (a, shft, clk, rst);

endmodule



/********************************** module_b.v*********************************/
module module_b (a, b, sub, clk, rst) /*synthesis syn_radhardlevel="cc" */;

input [7:0] a, b;
input clk, rst;

output [8:0] sub;
reg [8:0] sub;

reg [7:0] a_int, b_int;

always @ (posedge clk or posedge rst)

  if (rst) begin
    a_int <= 0;
    b_int <= 0;
    sub <= 0;
    end
  else begin
    a_int <= a;
    b_int <= b;
    sub <= a_int - b_int;
    end
endmodule



/********************************** module_c.v*********************************/
module module_c (a, shft, clk, rst);

input [7:0] a;
input clk, rst;

output [8:0] shft;
reg [8:0] shft;

reg [7:0] a_int;

always @ (posedge clk or posedge rst)

  if (rst) begin
    a_int <= 0;
    shft <= 0;
    end
  else begin
    a_int <= a;
    shft <= a_int >> 2;
    end
endmodule
```

```
/********************************* module_d.v*********************************/
module module_d (a, b, sum, clk, rst) /*synthesis syn_radhardlevel="tmr_cc" */;

input [7:0] a, b;
input clk, rst;

output [8:0] sum;
reg [8:0] sum;

reg [7:0] a_int, b_int;

always @ (posedge clk or posedge rst)

  if (rst) begin
    a_int <= 0;
    b_int <= 0;
    sum <= 0;
    end
  else begin
    a_int <= a;
    b_int <= b;
    sum <= a_int + b_int;
    end
endmodule
```

http://www.actel.com

Actel Europe Ltd.
Daneshill House, Lutyens Close
Basingstoke, Hampshire RG24 8AG
United Kingdom
Tel: +44-(0)125-630-5600
Fax: +44-(0)125-635-5420

Actel Corporation
955 East Arques Avenue
Sunnyvale, California 94086
USA
Tel: (408) 739-1010
Fax: (408) 739-1540

Actel Asia-Pacific
EXOS Ebisu Bldg. 4F
1-24-14 Ebisu Shibuya-ku
Tokyo 150 Japan
Tel: +81-(0)3-3445-7671
Fax: +81-(0)3-3445-7668

5192665-0/5.00