TMR and Spatial Separation for Higher Reliability in Microchip FPGAs Application Note

AN3782

Introduction (Ask a Question)

The purpose of this application note is to help FPGA designers to implement the Triple Module Redundancy (TMR) design technique on a VHDL design. This application note describes how to implement each logical register with a TMR register on different hierarchies of a VHDL/Verilog design. It also shows how to use the `syn_radhardlevel` synthesis attribute on the architecture and signal on different hierarchies.

These design example projects are intended for use with the RTPF500T-CG1509M device (Rad-Tolerant PolarFire® FPGA, featuring 500K Logic-Elements with High-Speed Serial Transceivers in a Ceramic Column Grid Array Package with 1509 solder columns).

Important: The design example in this application note is provided for RT PolarFire® FPGA; however, the TMR attribute is applicable to all Microchip FPGA families.

Features (Ask a Question)

Following are the TMR features supported in Libero® SoC:

- TMR attribute
- Perform spatial placement of registers in a TMR triplet

Table of Contents

1. Design Requirements (Ask a Question)

The following table lists the hardware and software requirements for this design.

Table 1-1. Design Requirements

Requirement	Version
Operating System	64-bit Windows® 10 and 11
Software	
Libero® SoC	To create this reference design, see the `readme.txt` file provided in the design files, for all the software versions needed.

Important: The Synplify® screenshots shown in this guide are for illustration purposes only. To see the latest updates, open the Synplify® Pro tool.

2. Design Prerequisites (Ask a Question)

Before you begin, ensure that the following components are in place:

- Download and install Libero® SoC v12.0 and later from Libero SoC Design Suite Versions.
- Download the reference design files from AN3782: TMR and Spatial Separation for Higher Reliability in Microchip FPGAs Application Note.

3. Description (Ask a Question)

To meet a radiation requirement, the Radiation-Survivability engineer should evaluate the Single Event Upset (SEU) rate of RT PolarFire D flip-flops (DFF) with a TMR implementation of a register:
If the SEU rate required by a specific design or mission is less than the native SEU rate, each register function can be implemented by decreasing the SEU rate of a register function with the TMR technique.

The SEU rate of a single native register in the fabric of a RT PolarFire device is listed in the radiation test report which are available at FPGA Radiation and Reliability Data.

For example, the FPGA design consists of three functional blocks:

- Block#1 is the top-level
- Block#2 is the video processor
- Block#3 is a navigation controller

If the SEU requirements of the entire FPGA logic design are less than the native SEU rate of each single DFF of a LE in RT PolarFire, implement all register function of the entire FPGA design as TMR-registers.

If the SEU requirements of the navigation block are less than the native SEU rate of a single DFF of a LE in RT PolarFire, implement:

- The register function of the Navigation block as TMR registers.
- The register function of the top-level and video processor with single native register cells.

Based on the requirements of a particular program, and of a specific function inside the FPGA, the radiation survivability engineer can tell the FPGA designer, which blocks can be implemented using:

- Single native DFF/register cells
- TMR registers

The Synplify Synthesis tools attribute called `syn_radhardlevel` is used for implementation of register functions as single native DFF/register cells or TMR registers.

4. TMR (Ask a Question)

The Triple Modular Redundancy (TMR) technique is a widely recognized design method that implements three registers for each register function. This approach involves voting on the outputs of these three registers to produce the final output signal for the register function. Each register function, implemented using TMR, uses three registers and one Look-Up Table (LUT) which can be verified in Synplify Pro tool. When three registers are used to implement each register function, this group of three registers are sometimes referred to as a triplet.

The basic concept of TMR is that all three registers always sample the same data input, and have the same values. If radiation disturbs the stored charge/state of one of these three registers of the TMR triplet, two of the registers still have the correct logical value, and the LUT voter can still provide the correct/intended state that was stored into this TMR triplet.

5. Attribute Declaration (Ask a Question)

The TMR attribute `syn_radhardlevel` triplicates the registers in the design and adds voter logic that looks at all three outputs to determine the final output based on the majority value. The value of the TMR attribute can be `tmr` or `none`. The `syn_radhardlevel` attribute is supported at Global, Module Instance, and Register View levels in the Register Transfer Level (RTL) or FPGA Design Constraint (FDC), as listed in the following table.

Table 5-1. TMR Attribute

Attribute Name	Value	Global Attribute	Object
`syn_radhardlevel`	`tmr`	Yes	FDC: Global
			Verilog: Top module
			VHDL: Top module architecture
	`none`	No	FDC: Register instance
			Verilog: Register
			VHDL: Signal

Table 5-2. Attribute Level

File	Level at which Attribute is Applied	Example
FDC through Synthesis Scope or Libero > Constraint Manage > Netlist Attribute Tab	Global	`define_global_attribute {syn_radhardlevel} {tmr}`
	Instance level	`define_attribute {v:work.t6} {syn_radhardlevel} {tmr}`
	Register level	`define_attribute {i:I_RDATA[7:0]} {syn_radhardlevel} {tmr}`
Verilog	Global	`module top (datain1, datain2, datain3, datain4, clk, out1) /*synthesis syn_radhardlevel = ”tmr”*/`
	Register level	`reg [1:0] datain2 /* synthesis syn_radhardlevel = “tmr”*/`
VHDL	Global	Architecture behavior of test is: attribute `syn_radhardlevel`: string; attribute `syn_radhardlevel` of behavior: architecture is “`tmr`”
	Register level	Signal dout1: std_logic_vector(7 down to 0); attribute `syn_radhardlevel` of dout1: signal is “`tmr`”

6. Spatial Placement (Ask a Question)

The **Place and Route** tool in the Libero SoC **Design Flow** tab includes a placement tool, which decides the best location on the die, to place each register. If a register function is implemented with the TMR technique, the placement tool picks the location on the die for each of these three registers in the TMR triplet.

Test data shows that when the three registers that comprise a TMR triplet are physically spaced apart from each other on the die, this improves (lowers) the SEU rate of the TMR register function. This is because the radiation event can upset two or more of the registers in the TMR triplet when these three registers are placed in the same logic cluster sharing the same cluster clock buffer.

When TMR is used, Libero SoC generates a TMR report that includes the total number of TMR flip-flops.

Important: The top module of this design is named `MY_TOP_LEVEL`, and the report file is named `MY_TOP_LEVEL_tmr.rpt` under synthesis reports.

If the TMR constraints are not implemented by the Placer, the following error message is generated and displayed.

```
Error: The design contains TMR flip-flops which could not be placed with minimum spacing.
```

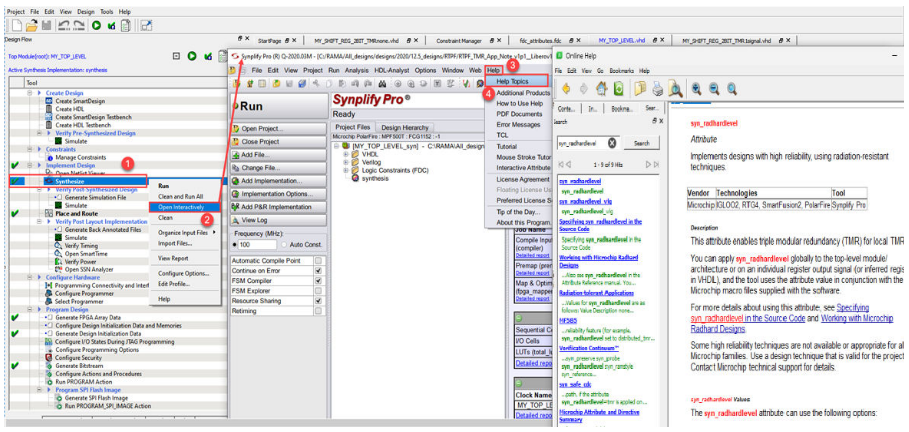
The preceding error might occur when there are very tight region constraints, or, when the TMR triplet is fixed. To resolve this, try relaxing the placement restrictions by checking if the violated cells are fixed, and then re-run with the **multi-seed layout** option enabled.

Important: If the TMR flip-flops are placed too close, the TMR implementation might not be able to correct errors as intended, and the design may be vulnerable to SEUs.

The spatial separation in placement of TMR triplets in Libero SoC actually places the triplets on three different cluster clocks. The spatial placement is constrained to achieve a balance of performance and SEU mitigation.

The following figure shows the `syn_radhardlevel` TMR attribute.

Figure 6-1. syn_radhardlevel TMR Attribute



syn_radhardlevel

Attribute

Implements designs with high reliability, using radiation-resistant techniques.

Vendor	Technologies	Tool
Microchip	IGLOO2, RTG4, SmartFusion2, PolarFire	Synplify Pro

Description

This attribute enables triple modular redundancy (TMR) for local TMR

You can apply syn_radhardlevel globally to the top-level module/ architecture or on an individual register output signal (or inferred regis in VHDL), and the tool uses the attribute value in conjunction with the Microchip macro files supplied with the software.

For more details about using this attribute, see Specifying syn_radhardlevel in the Source Code and Working with Microchip Radhard Designs.

Some high reliability techniques are not available or appropriate for al Microchip families. Use a design technique that is valid for the project Contact Microchip technical support for details.

syn_radhardlevel Values

The syn_radhardlevel attribute can use the following options:

7. Design Description (Ask a Question)

The FPGA design example describes how to implement TMR, which consists of six different logical paths. Each path uses a different technique or operates on a different level of the hierarchy, showing how to apply the `syn_radhardlevel` synthesis attribute.

The description of the six different paths are as follows:

- PATH1: Does not define any of the TMR attributes.
- PATH2: Defines that one specific register in the top-level is not TMR'd by synthesis.
- PATH3: Defines that one specific register in the top-level is TMR'd by synthesis.
- PATH4: Defines that all registers in a component, that is instantiated in the top-level, are not TMR'd by synthesis.
- PATH5: Defines that one specific register out of all the registers in the top-level is TMR'd by synthesis.
- PATH6: Defines that all registers in a component, that is instantiated in the top-level, are TMR'd by synthesis.

The following table lists the path number and quantity of registers that are synthesized by each path.

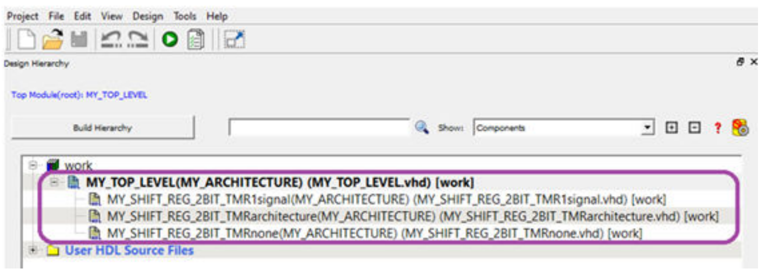
Table 7-1. PATH Information

Path Number	Registers in this Circuit Path	syn_radhardlevel is Defined on the Quantity of Registers	syn_radhardlevel is Defined Where	syn_radhardlevel Defined on this Object of the Design	syn_radhardlevel is Assigned this Value	Quantity of Registers Used
PATH1	1	None	MY_TOP_LEVEL	None Assigned	None Assigned	1
PATH2	1	0	MY_TOP_LEVEL	Signal	no	1
PATH3	1	1	MY_TOP_LEVEL	Signal	tmr	3
PATH4	2	0	Component	Architecture	None Assigned	2
PATH5	2	1	Component	1 signal	1 x tmr, 1 none	4
PATH6	2	2	Component	Architecture	tmr	6

Important: For Design Examples, analyze the Synplify `.srr` log file output listed in the FFs that are TMR’d.

The example design is a pure VHDL code and consists of four VHDL source files, as shown in the following figure.

Figure 7-1. Libero SoC Design Hierarchy



Top Module(root): MY_TOP_LEVEL

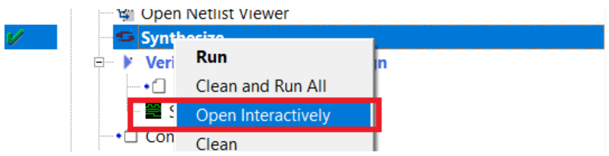
```
work
  MY_TOP_LEVEL(MY_ARCHITECTURE) (MY_TOP_LEVEL.vhd) [work]
    MY_SHIFT_REG_2BIT_TMR1signal(MY_ARCHITECTURE) (MY_SHIFT_REG_2BIT_TMR1signal.vhd) [work]
    MY_SHIFT_REG_2BIT_TMRarchitecture(MY_ARCHITECTURE) (MY_SHIFT_REG_2BIT_TMRarchitecture.vhd) [work]
    MY_SHIFT_REG_2BIT_TMRnone(MY_ARCHITECTURE) (MY_SHIFT_REG_2BIT_TMRnone.vhd) [work]
User HDL Source Files
```

7.1. PATH1 Description (Ask a Question)

PATH1 defines that one specific register in the top-level has no TMR attribute assigned to it. PATH1 is a VHDL RTL source code, describing a single DFF register, and the TMR attribute `syn_radhardlevel` is not defined on this path. The Synthesis tool has no explicit guidance.

After synthesis completes, right-click **Synthesis** > **Open Interactively**, as shown in the following figure.

Figure 7-2. Synthesis



To analyze the Synplify for design implementation, perform the following steps:

1. Open **HDL-Analyst** > **Technology** > **Flattened View**, click **Primitives** and then select the input and output ports to see the gate implementation, as shown in the following figures.

   Figure 7-3. HDL Flattened View

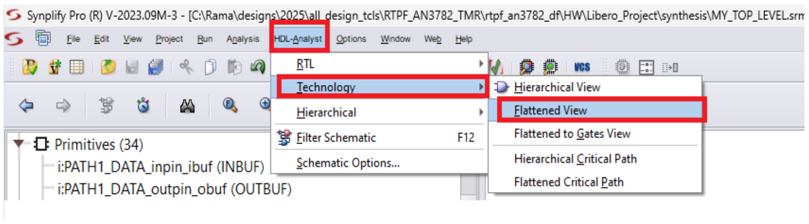
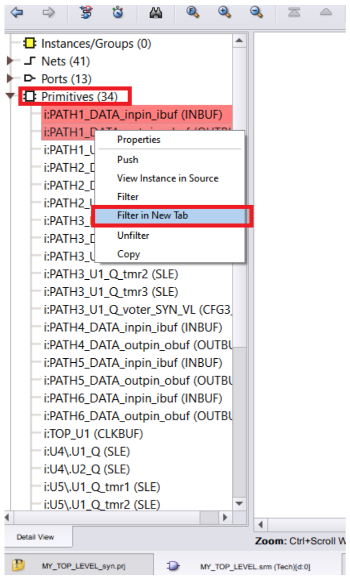

Synplify Pro (R) V-2023.09M-3 - [C:\Rama\designs\2025\all_design_tcls\RTPF_AN3782_TMR\rtpf_an3782_df\HW\Libero_Project\synthesis\MY_TOP_LEVEL.srm
File Edit View Project Run Analysis HDL-Analyst Options Window Web Help
RTL
Technology
Hierarchical
Filter Schematic F12
Schematic Options...
Hierarchical View
Flattened View
Flattened to Gates View
Hierarchical Critical Path
Flattened Critical Path
Primitives (34)
i:PATH1_DATA_inpin_ibuf (INBUF)
i:PATH1_DATA_outpin_obuf (OUTBUF)

   Figure 7-4. PATH1 View



2. Click **Isolate Paths** twice consecutively, as shown in the following figures.

Figure 7-5. PATH1 Isolate Paths-1

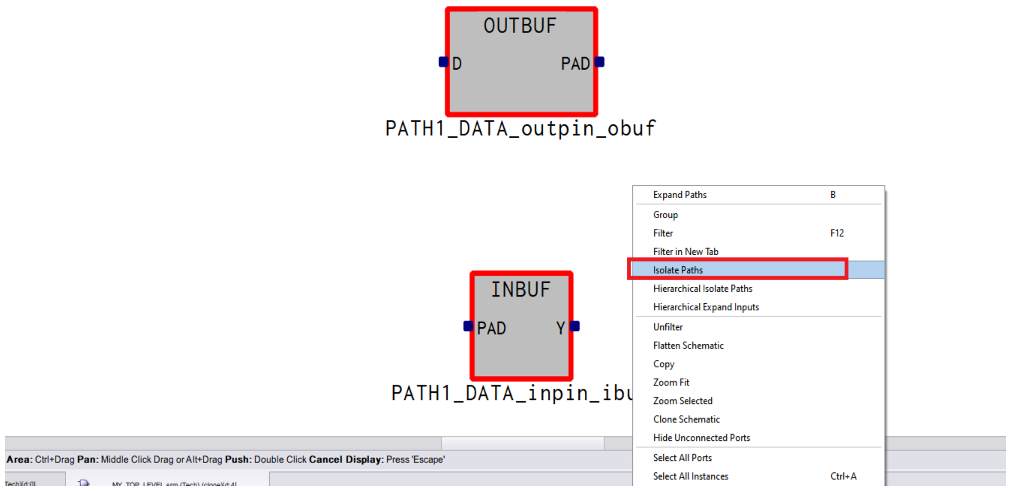
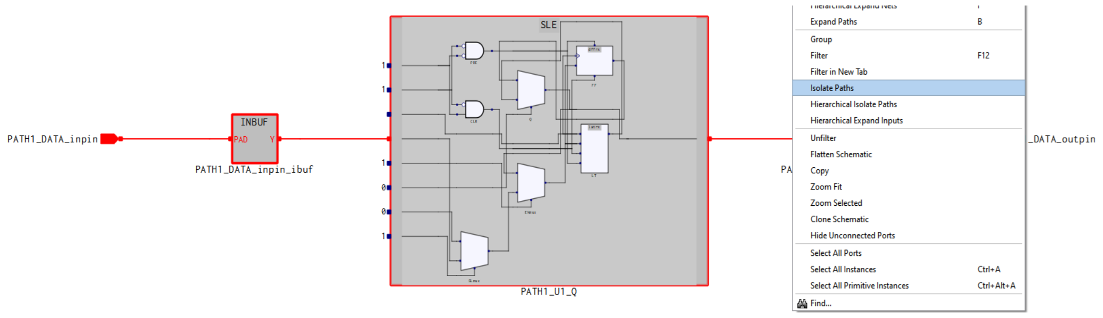
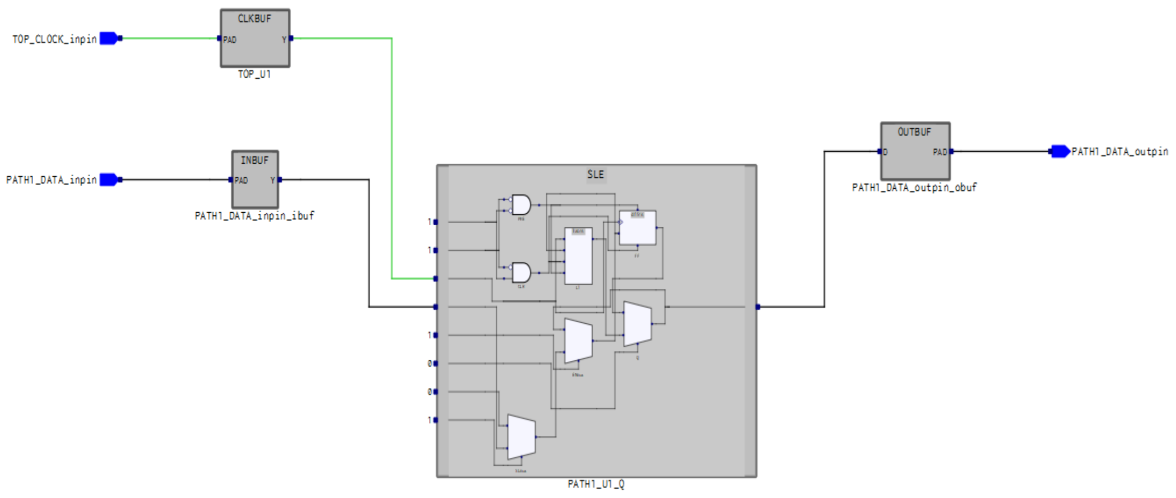


Figure 7-6. PATH1 Isolate Paths-2



This shows that PATH1 consumes one SLE macro, which is implemented by one DFF in one LE.

Figure 7-7. PATH1 Gate Implementation

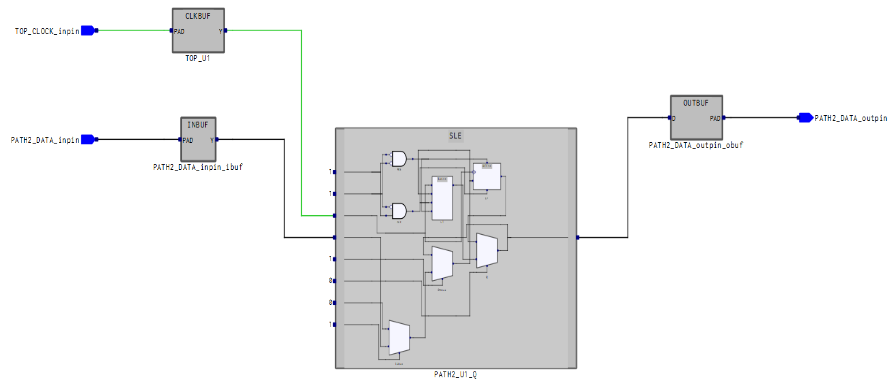


7.2. PATH2 Description (Ask a Question)

PATH2 defines that one specific register in the top-level is not TMR'd by synthesis. PATH2 is a VHDL RTL source code, describing a single DFF register, and the TMR attribute `syn_radhardlevel` is assigned a value `none` on the signal that is being driven by the register, to explicitly instruct the Synthesis tool not to implement TMR on the register.

After synthesis, perform the same procedure to analyze the Synplify for design implementation, as shown in the PATH1 Description section. This shows that PATH2 consumes one SLE macro, which is implemented by one DFF in one LE. The gate implementation is shown in the following figure.

Figure 7-8. PATH2 Gate Implementation

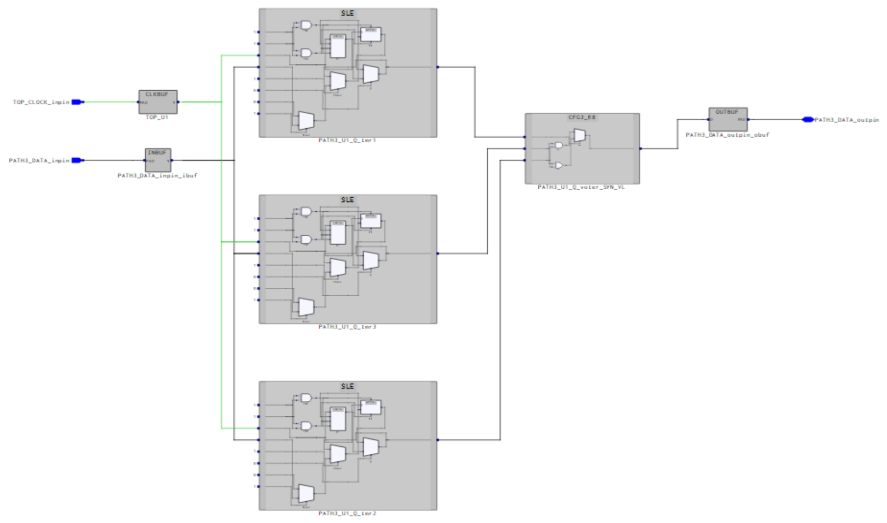


7.3. PATH3 Description (Ask a Question)

PATH3 defines that one specific register in the top-level is TMR'd by synthesis. PATH3 is a VHDL RTL source code, describing a single DFF register, and the TMR attribute `syn_radhardlevel` is assigned a value `tmr` on the signal that is being driven by the register, to explicitly instruct the Synthesis tool to implement TMR on the register.

After synthesis, perform the same procedure to analyze the Synplify for design implementation as shown in the PATH1 Description section. The gate implementation is shown in the following figure. This register is named as PATH3_U1_Q_tmr1 by VHDL code. Since this register has TMR attribute, the Synthesis tool is created for two additional registers, named PATH3_U1_Q_tmr2 and PATH3_U1_Q_tmr3. This shows that PATH3 consumes three SLE macros and one CFG3 macro, which is implemented by three DFF in three LE's and one CFG3.

Figure 7-9. PATH3 Gate Implementation



7.3.1. Spatial Placement of the R-cells in a TMR Implemented Register Function (Ask a Question)

Libero SoC has an enhanced feature that improves (lowers) the SEU rate of a TMR register. The enhanced Libero SoC placement tool physically places the three registers of the TMR triplet into distinct logic clusters on the die, preventing the sharing of cluster clock buffers. This improves (lowers) the SEU rate of the TMR register because it dramatically reduces the probability of a clock upset in the cluster clock, affecting more than one of the three registers in the TMR triplet.

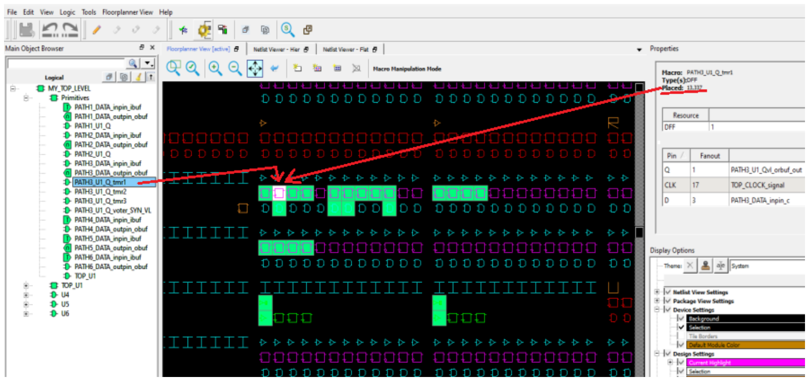
When the TMR attribute is assigned, the three registers are physically spaced apart with the Chip Planner tool placement of TMR. They are not directly next to each other on the die. It improves (reduces) the SEU rate compared to the SEU rate observed if these three registers were placed on the same cluster clock branch.

The location of the three registers in the Chip Planner is mapped as follows:

- PATH3_U1_Q_tmr1 is located at coordinates 13,337 on the die.
- PATH3_U1_Q_tmr2 is located at coordinates 24,337 on the die.
- PATH3_U1_Q_tmr3 is located at coordinates 12,334 on the die.

The following figures show the three registers in the Chip Planner.

Figure 7-10. PATH3_U1_Q_tmr1 Register

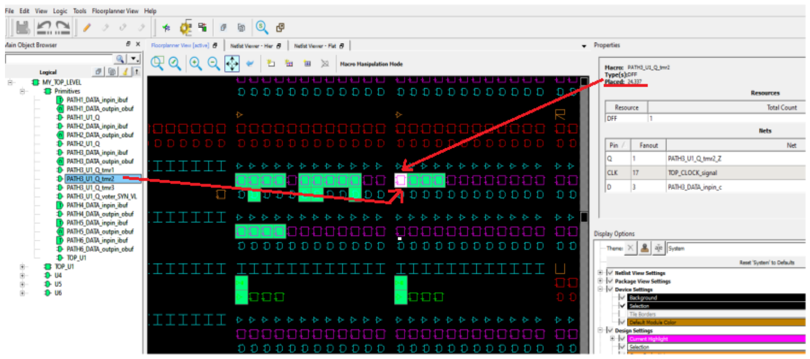


Macro: PATH3_U1_Q_tmr1
Type(s):DFF
Placed: 13,337

Resource	
DFF	1

Pin	Fanout	
Q	1	PATH3_U1_Qvl_orbuf_out
CLK	17	TOP_CLOCK_signal
D	3	PATH3_DATA_inpin_c

Figure 7-11. PATH3_U1_Q_tmr2 Register



Macro: PATH3_U1_Q_tmr2
Type(s):DFF
Placed: 24,337

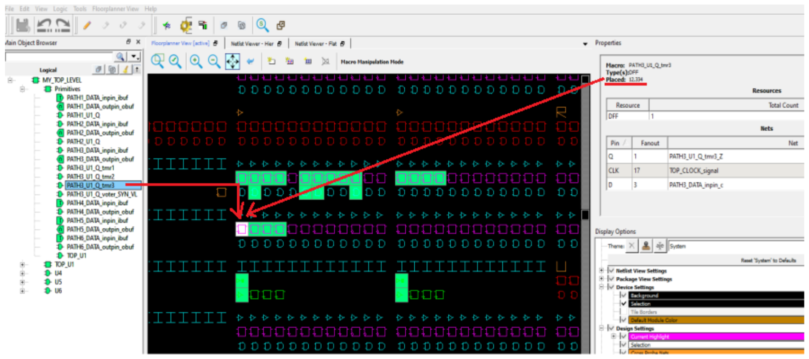
Resources

Resource	Total Count
DFF	1

Nets

Pin	Fanout	Net
Q	1	PATH3_U1_Q_tmr2_Z
CLK	17	TOP_CLOCK_signal
D	3	PATH3_DATA_inpin_c

Figure 7-12. PATH3_U1_Q_tmr3 Register



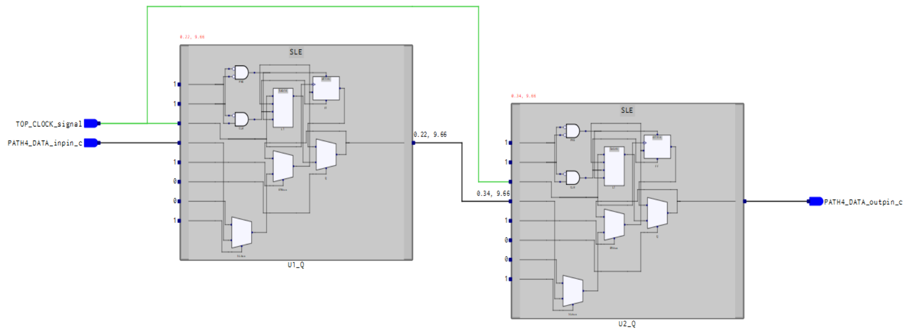
7.4. PATH4 Description (Ask a Question)

PATH4 defines that all registers in a component that is instantiated in the top-level are not TMR'd by synthesis. Its logic is defined in an instantiated component, named 'MY_SHIFT_REG_2BIT_TMRnone', and the TMR attribute `syn_radhardlevel` is not defined on this component in the top-level.

The 'MY_SHIFT_REG_2BIT_TMRnone' component is the VHDL RTL source code, describing two DFF registers connected in series (a 2-bit shift register). The TMR attribute `syn_radhardlevel` is not defined in this VHDL code.

After synthesis, perform the same procedure to analyze the Synplify for design implementation as shown in the PATH1 Description section. The gate implementation is shown in the following figure. This shows that PATH4 consumes two SLE macros, which is implemented by two DFF in two LE's.

Figure 7-13. PATH4 Gate Implementation



7.5. PATH5 Description (Ask a Question)

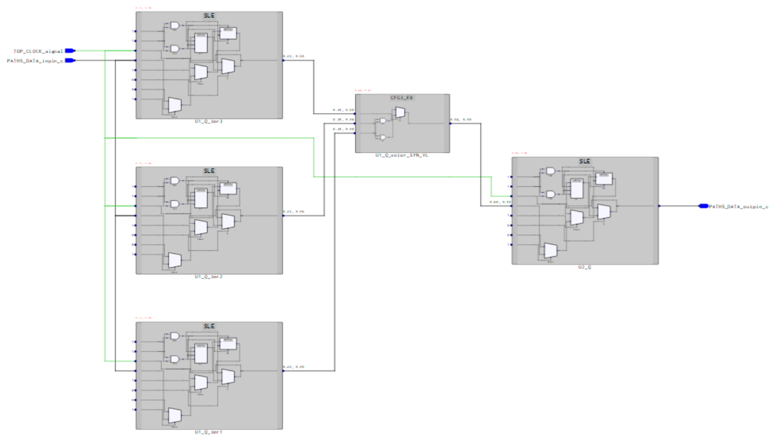
PATH5 defines that one of the two registers in the instantiated component is to be TMR'd, and one of the two registers in the same instantiated component is not to be TMR'd by Synthesis. Its logic is defined in an instantiated component, named 'MY_SHIFT_REG_2BIT_TMR1signal', and the TMR attribute `syn_radhardlevel` is not defined on this component in the top-level.

The 'MY_SHIFT_REG_2BIT_TMR1signal' component is the VHDL RTL source code, describing two DFF registers connected in series (a 2-bit shift register), and the TMR attribute `syn_radhardlevel` is defined as `tmr` on one signal driven by one register and is defined as `none` on one signal driven by a second register.

After synthesis, perform the same procedure to analyze the Synplify for design implementation as shown in the PATH1 Description section. The gate implementation is shown in the following figure. This shows that PATH5 logic consumptions are:

- The first register (that is TMR'd) consumes three SLE macros and one CFG3, which is implemented by three DFF in three LE's, and one CFG3.
- The second register (that is not TMR'd) consumes one SLE macro in one LE.

Figure 7-14. PATH5 Gate Implementation



7.6. PATH6 Description (Ask a Question)

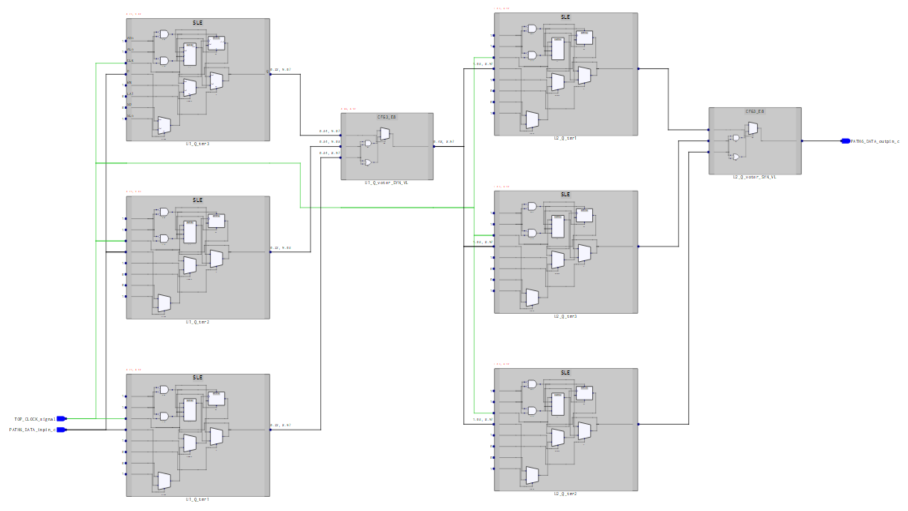
PATH6 defines that one of the two registers in the instantiated component are TMR'd by Synthesis. Its logic is defined in an instantiated component, named 'MY_SHIFT_REG_2BIT_TMRarchitecture' and the TMR attribute `syn_radhardlevel` is not defined on this component in the top-level.

The 'MY_SHIFT_REG_2BIT_TMRarchitecture' component is VHDL RTL source code, describing two DFF registers connected in series (a 2-bit shift register), and the TMR attribute `syn_radhardlevel` is defined as `tmr` on the architecture, so all registers in this file are to be TMR'd by Synthesis.

After synthesis, perform the same procedure to analyze the Synplify for design implementation as shown in the PATH1 Description section. The gate implementation is shown in the following figure. This shows that PATH6 logic consumptions are:

- The first register (that is TMR'd) consumes three SLE macros and one CFG3 macro, which is implemented by three DFF in three LE's and one CFG3.
- The second register (that is TMR'd) consumes three SLE macros and one CFG3 macro, which is implemented by three DFF in three LE's and one CFG3.

Figure 7-15. PATH6 Gate Implementation



Important:

- When applying the `syn_radhardlevel` on a signal, you must first define the signal in the VHDL code, and then the signal is declared. You can define the `syn_radhardlevel` applied to that signal.

  Example: If the user declares a signal on line#100 of the VHDL/Verilog design, you must apply the `syn_radhardlevel` on line# 101 or later. Otherwise, Synplify Synthesis fails with a synthesis error.
- When a component is instantiated in a VHDL/Verilog block, the user cannot guide synthesis to implement that instantiated block with the TMR, by applying the `syn_radhardlevel` attribute onto the signal that the instantiated block is driving in the upper-level block.
- Do not apply `syn_radhardlevel` to the signal that are instantiated as modules. If you apply the `syn_radhardlevel` to the output signal of an instantiated block, Synthesis passes without an error or Synthesis does not implement it as TMR, and does not report any note, warning, or error.

8. Appendix: Running the TCL Script (Ask a Question)

TCL scripts are provided in the design files folder under the `HW` directory. If required, the design flow can be reproduced from the Design Implementation till generation of job file.

To run TCL, perform the following steps:

1. Launch the Libero software.
2. Select **Project** > **Execute Script....**
3. From the downloaded `HW` directory, click **Browse** and then select `script.tcl`.
4. Click **Run**.

After successful execution of the TCL script, a Libero project is created within `HW` directory.

For more information about TCL scripts, see `rtpf_an3782_df/HW/TCL_Script_readme.txt`. For more details on TCL commands, see Tcl Commands Reference Guide. For any queries encountered when running the TCL script, contact Technical Support.

9. Design Examples (Ask a Question)

Design examples related to TMR are explained in this chapter. All these examples are validated on the Libero SoC Synplify Pro tool.

Example 1: Basic Flop without Control Signal (TMR attribute globally applied through FDC file)

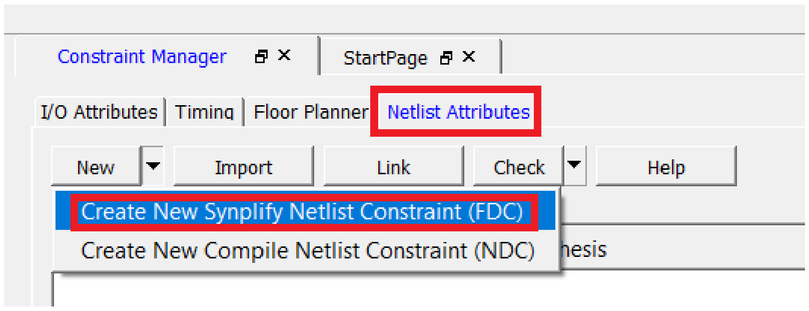
Synplify Pro triplicates each register and inserts majority voting logic at register outputs.

HDL:

```
module test1 (clk, din, dout);
input clk, din;
output reg dout;
always @(posedge clk)
         dout <= din;
endmodule
```

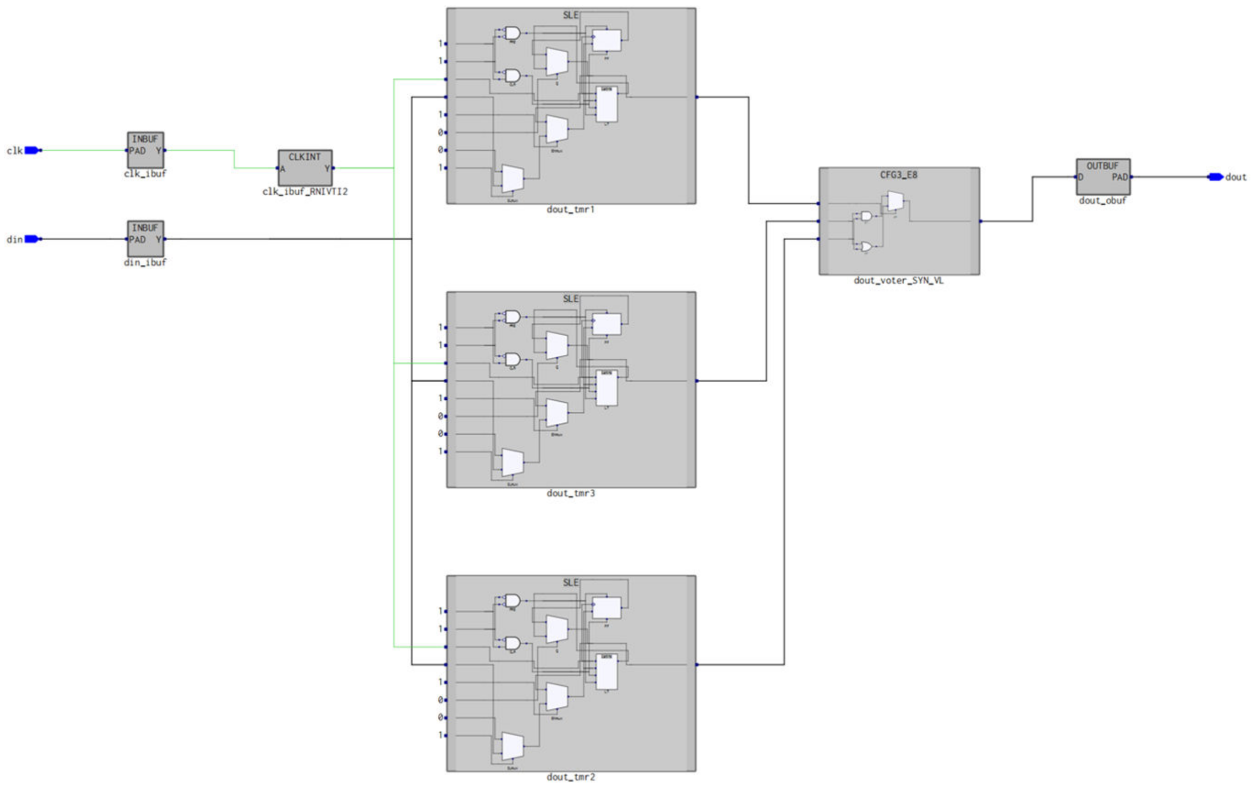
FDC: To add FDC constraint, click **Constraint Manager** > **Netlist Attributes** and select **Create New Synplify Netlist Constraint (FDC)**, as shown in the following figure.

Figure 9-1. FDC Creation



```
define_global_attribute {syn_radhardlevel} {tmr}
```

Figure 9-2. Basic Flop without Control Signal



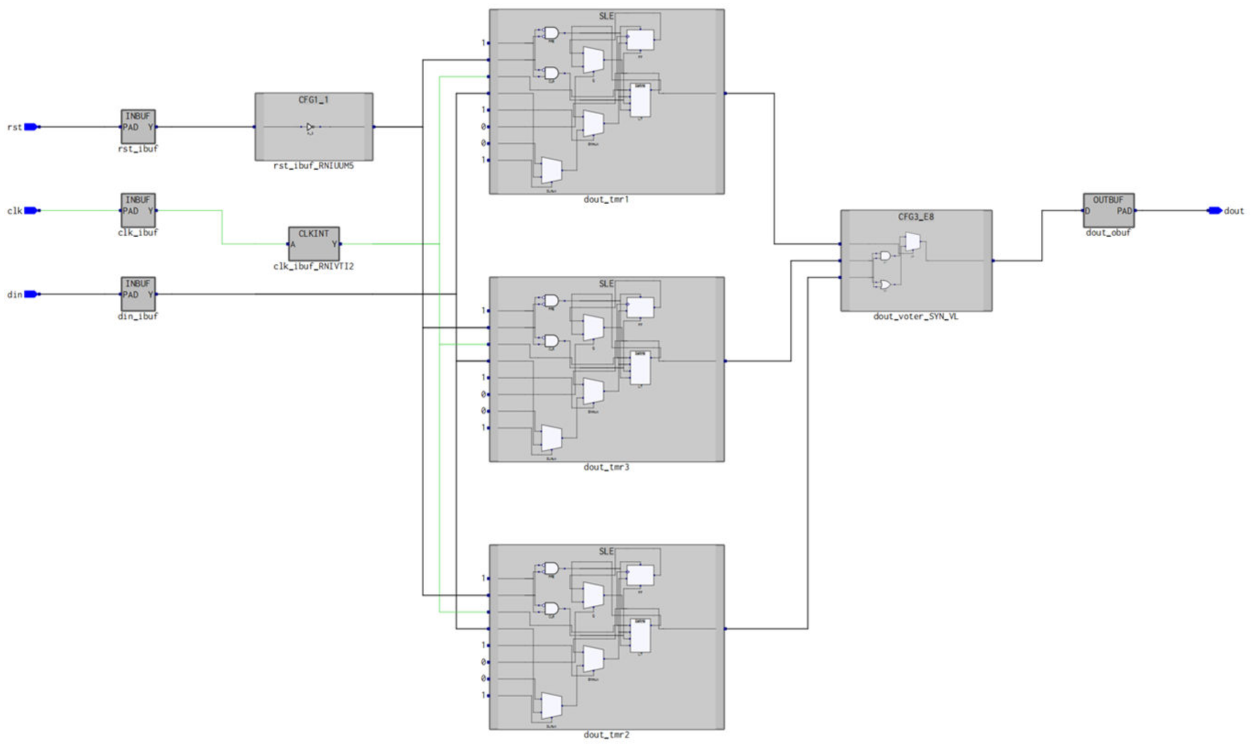
Example 2: Basic Flop with Asynchronous Reset Control Signal (TMR attribute is globally applied on the module in Verilog)

Synplify Pro triplicates each register and inserts majority voting logic at register outputs.

HDL:

```
module test2 (clk, rst, din, dout)/* synthesis syn_radhardlevel=tmr */;
input clk, rst, din;
output reg dout;
always @(posedge clk or posedge rst)
    if (rst)
        dout <= 1'b0;
    else
dout <= din;
 endmodule
```

Figure 9-3. Basic Flop with Asynchronous Reset Control Signal



Example 3: Basic Flop with Enable Control Signal (TMR attribute globally applied in Verilog)

Synplify Pro triplicates each register and inserts majority voting logic at register outputs. Enable logic implemented using voter logic output.

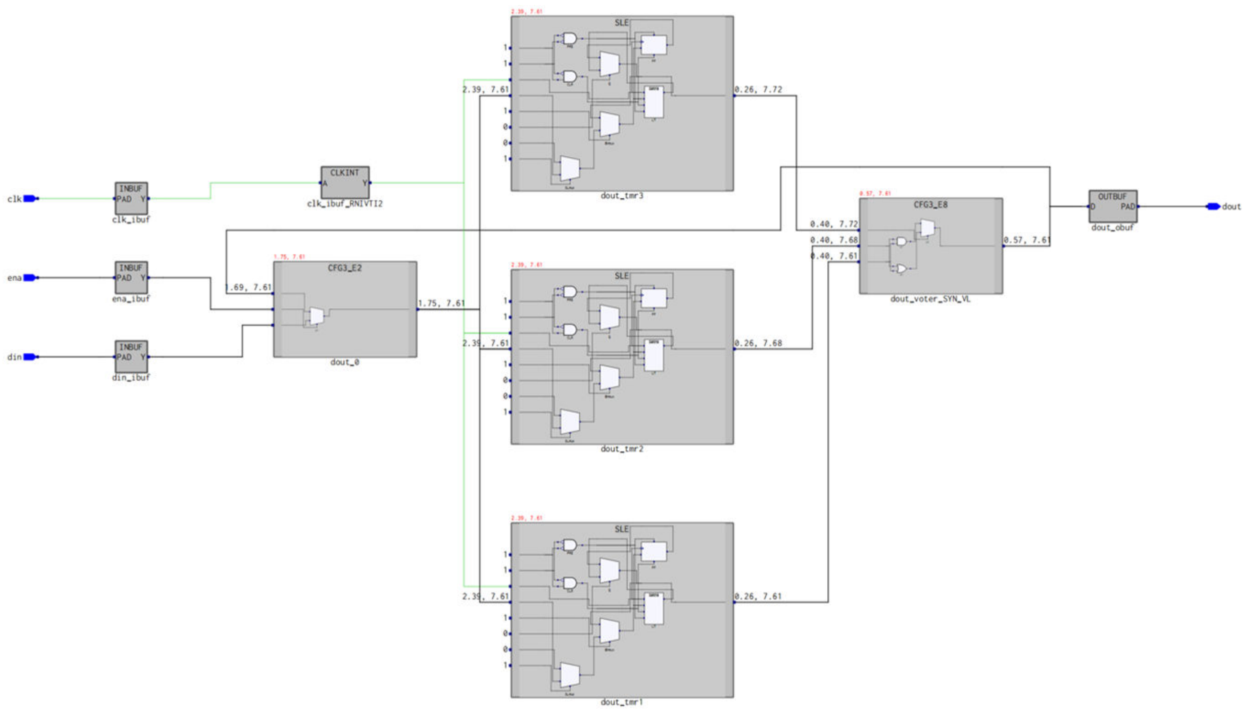
HDL:

```
module test3 (clk, ena, din, dout);
input clk, ena, din;
output reg dout;
always @(posedge clk)
    if (ena)
        dout <= din;
 endmodule
```

FDC:

```
define_global_attribute {syn_radhardlevel} {tmr}
```

Figure 9-4. Basic Flop with Enable Control Signal



Example 4: Flop with Asynchronous Reset and Set Control Signals (TMR attribute is globally applied in FDC)

Synplify Pro triplicates each register and inserts majority voting logic at register outputs. Asynchronous set logic is implemented using latch and flop, and triplicated.

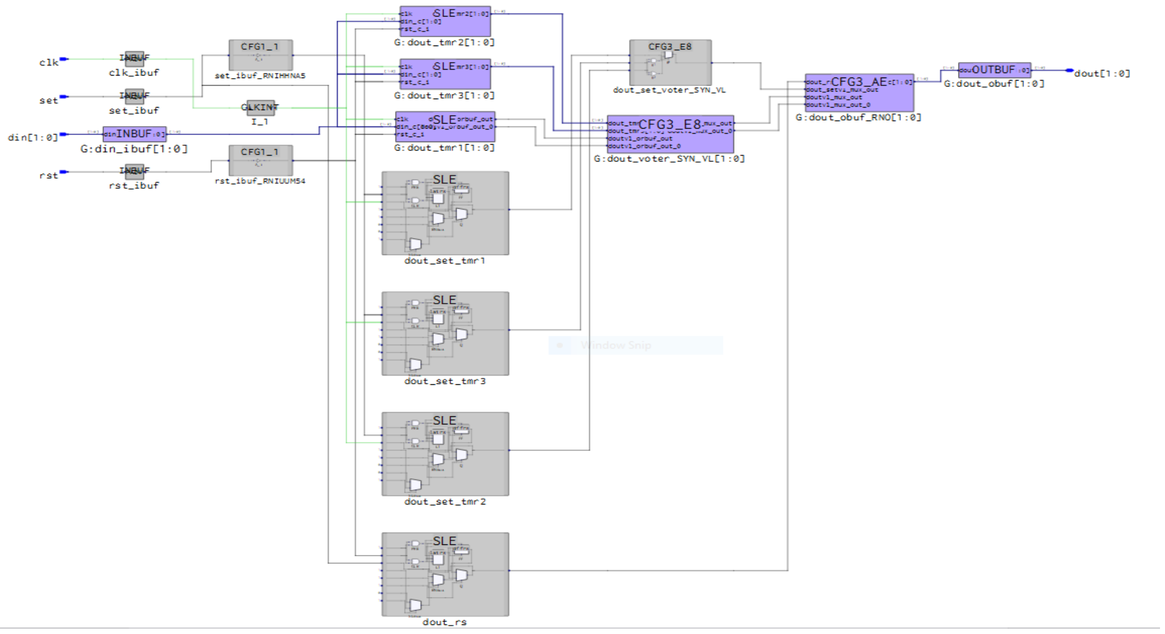
HDL:

```
module test4 (clk, rst, set, din, dout);
input clk, rst, set;
input [1:0] din;
output reg [1:0] dout;
always @(posedge clk or posedge rst or posedge set)
    if (rst)
        dout <= 2'b00;
    else if (set)
        dout <= 2'b11;
    else
        dout <= din;
endmodule
```

FDC:

```
define_global_attribute {syn_radhardlevel} {tmr}
```

Figure 9-5. Flop with Asynchronous Reset and Set Control Signals



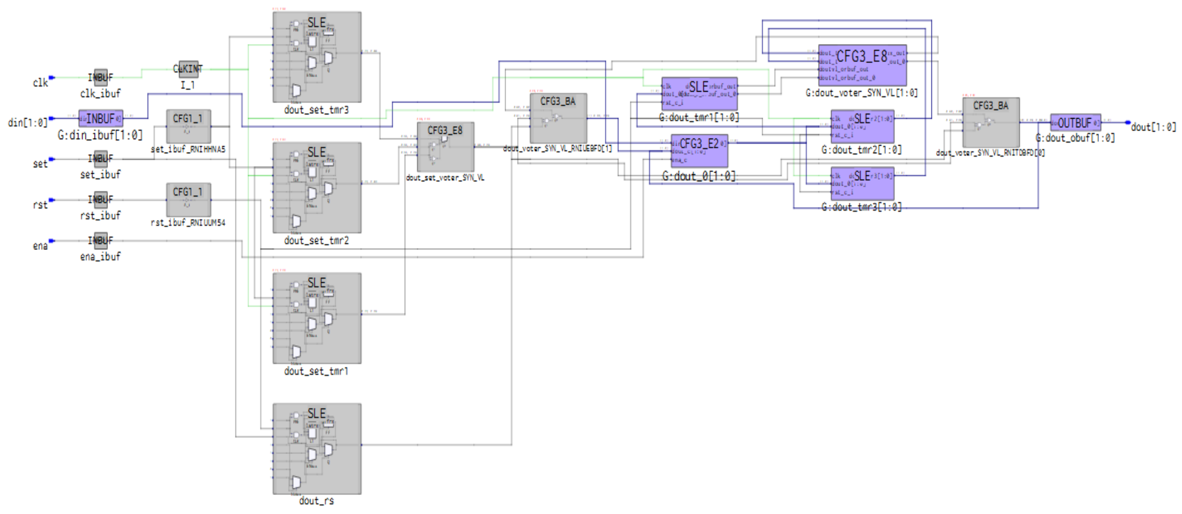
Example 5: Flop with Asynchronous Reset, Asynchronous Set, and Enable Control Signal (TMR attribute globally applied in Verilog)

Synplify Pro triplicates each register and inserts majority voting logic at register outputs. Asynchronous set logic is implemented using latch and flop, and triplicated. Enable logic implemented using voter logic output.

HDL:

```
module test5 (clk, rst, set, ena, din, dout)/* synthesis syn_radhardlevel=tmr */;
input clk, rst, ena, set;
input [1:0] din;
output reg [1:0] dout;
always @(posedge clk or posedge rst or posedge set)
    if (rst)
        dout <= 2'b00;
    else if (set)
        dout <= 2'b11;
    else
        if (ena)
            dout <= din;
 endmodule
```

Figure 9-6. Flop with Asynchronous Reset, Asynchronous Set, and Enable Control Signal Applied in Verilog



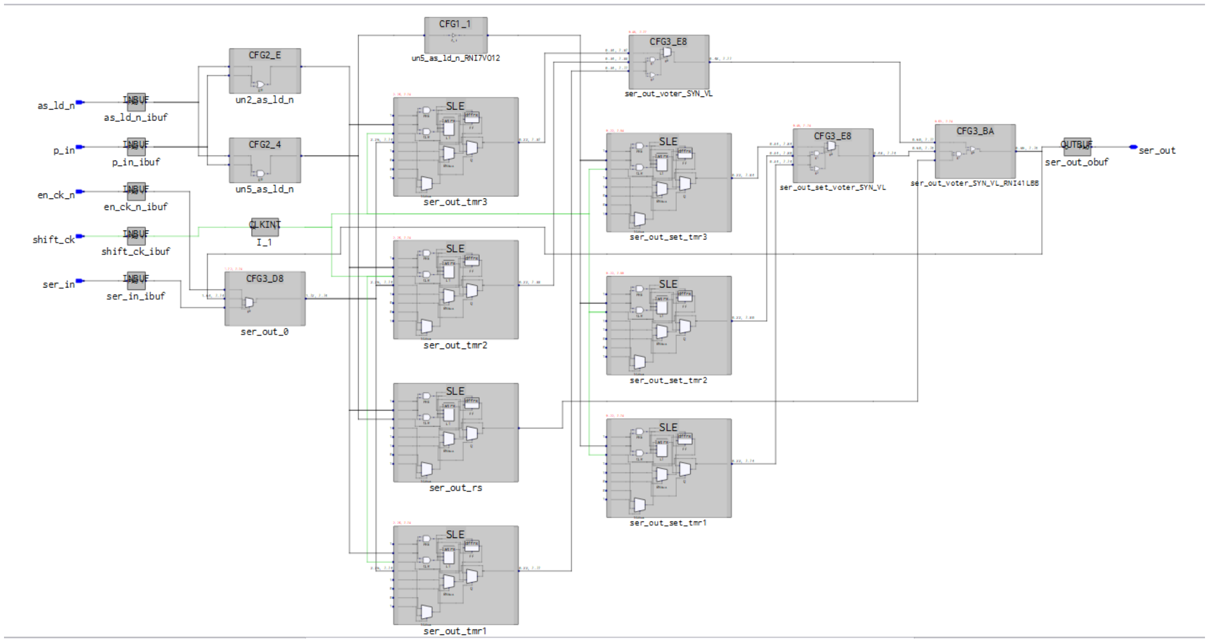
Example 6: Flop with Asynchronous Reset, Asynchronous Set, and Enable Control Signal (TMR attribute globally applied in VHDL)

Synplify Pro triplicates each register and inserts majority voting logic at register outputs. Asynchronous set logic is implemented using latch and flop, and triplicated. Enable logic is implemented using voter logic output.

HDL:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
ENTITY test39_sle_reseten_globtmr IS
PORT (
          shift_ck:in     STD_LOGIC;
          ser_in:in     STD_LOGIC;
          p_in:in     STD_LOGIC;
          as_ld_n:in     STD_LOGIC;
          en_ck_n:in     STD_LOGIC;
          ser_out:out    STD_LOGIC
);
END test39_sle_reseten_globtmr;
architecture behaviour of test39_sle_reseten_globtmr is
attribute syn_radhardlevel: string;
attribute syn_radhardlevel of behaviour: architecture is "tmr";
begin
process (as_ld_n,p_in,shift_ck)
begin
    IF (as_ld_n = '0' and p_in = '0') THEN
        ser_out <= '0';
    ELSIF (as_ld_n = '0' and p_in = '1') THEN
        ser_out <= '1';
    ELSIF rising_edge(shift_ck) THEN
        IF (en_ck_n = '0') THEN
            ser_out <= ser_in;
        END IF;
    END IF;
END process;
END behaviour;
```

Figure 9-7. Flop with Asynchronous Reset, Asynchronous Set, and Enable Control Signal Applied in VHDL



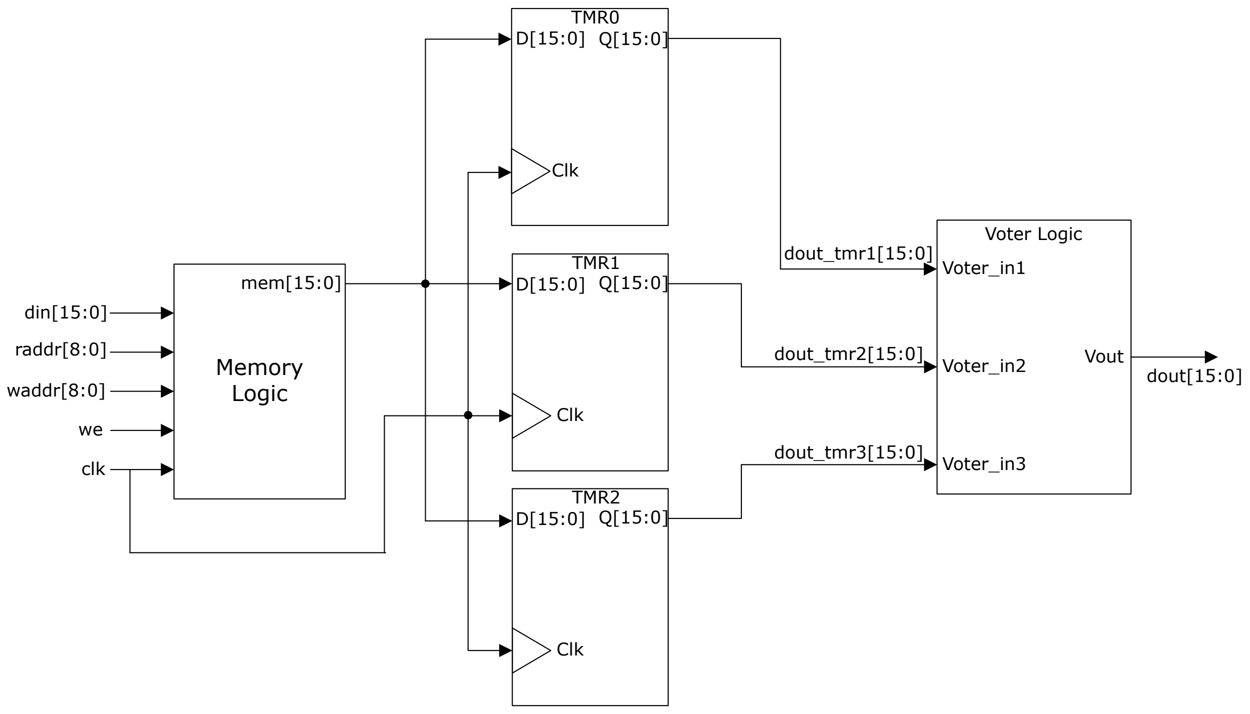
Example 7: `syn_preserve` attribute on register associated with RAM Block (TMR attribute globally applied in Verilog)

When the `syn_preserve` attribute is applied on the read address register and output register of the memory, Synplify Pro stops packing the registers inside the RAM block, triplicates each register, and inserts majority voting logic at register outputs.

HDL:

```
module test7 (dout, waddr, raddr, din, we, clk)/* synthesis syn_radhardlevel=tmr */;
output reg [15:0] dout/* synthesis syn_preserve=1 */;
input [15:0] din;
input [8:0] waddr;
input [8:0] raddr;
input we, clk;
reg [15:0] mem [511:0];
always @(posedge clk)
begin
dout <= mem[raddr];
        if(we)
             mem[waddr] <= din;
    end
endmodule
```

Figure 9-8. syn_preserve Attribute on Register Associated with RAM Block



Important: The preceding figure illustrates the high-level block diagram of RAM Block (TMR attribute globally applied in Verilog). Kindly review the TMR implementation of RTL in Synplify Pro.

Example 8: No `syn_preserve` attribute on register with asynchronous reset associated with RAM block (TMR attribute globally applied in FDC)

When the `syn_preserve` attribute is not applied to the output register of memory, Synplify Pro packs the registers inside the RAM block, triplicates the control logic, and inserts majority voting logic at register outputs.

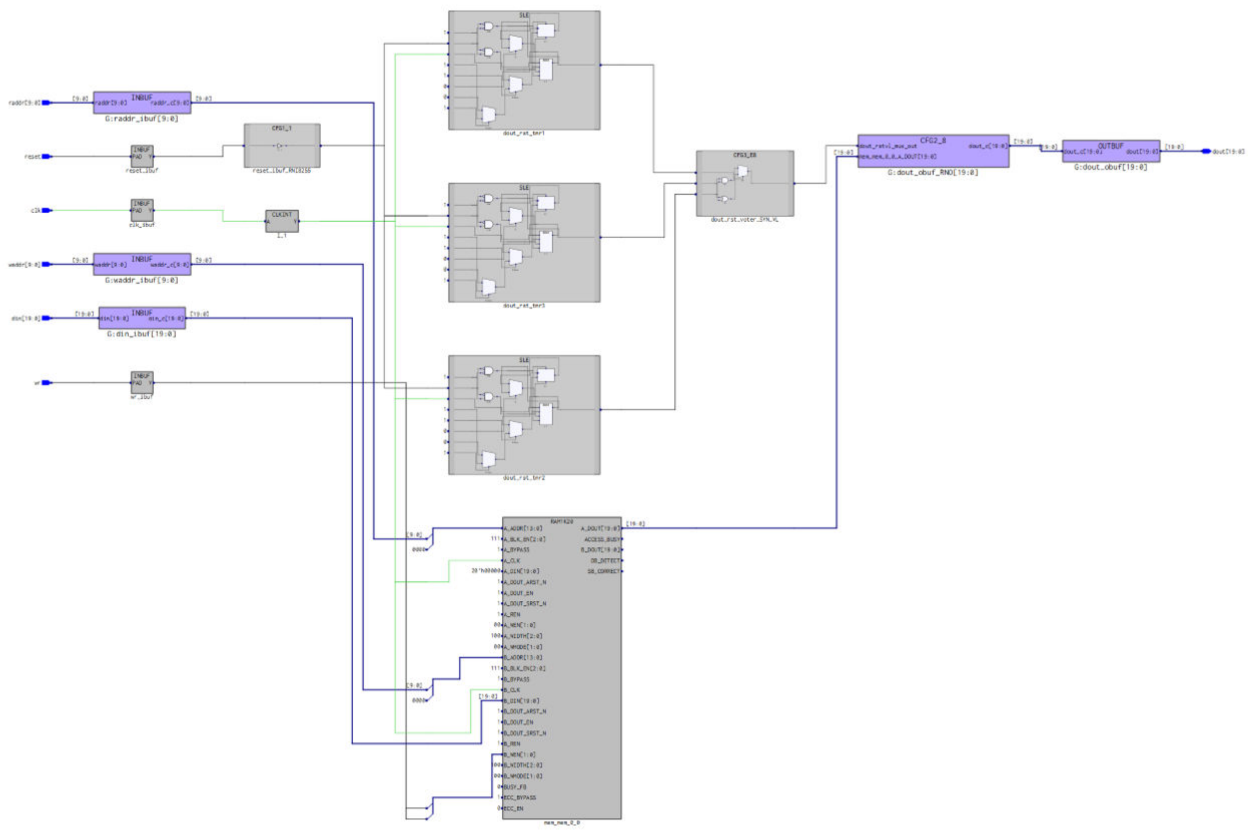
HDL:

```
module ram_2port_outreg_areset(clk,din,wr,reset,waddr,raddr,dout);
input clk;
input [19:0] din;
input wr,reset;
input [9:0] waddr,raddr;
output [19:0] dout;
reg [19:0] dout;
reg [19:0] mem [0:1023];
always@(posedge clk)
begin
if(wr)
      mem[waddr] <= din;
end
always@(posedge clk or posedge reset )
begin
if(reset)
dout <= 0;
else
      dout <= mem[raddr];
end
endmodule
```

FDC:

```
define_global_attribute {syn_radhardlevel} {tmr}
```

Figure 9-9. No syn_preserve Attribute on Register with Asynchronous Reset Associated with RAM Block



Example 9: The `syn_preserve` attribute on register associated with DSP logic (TMR attribute globally applied in FDC, `syn_preserve` in FDC)

When the `syn_preserve` attribute is applied to the input and output register of multiplier/MAC, Synplify Pro stops packing the register inside block DSP, triplicates each register, and inserts majority voting logic at register outputs.

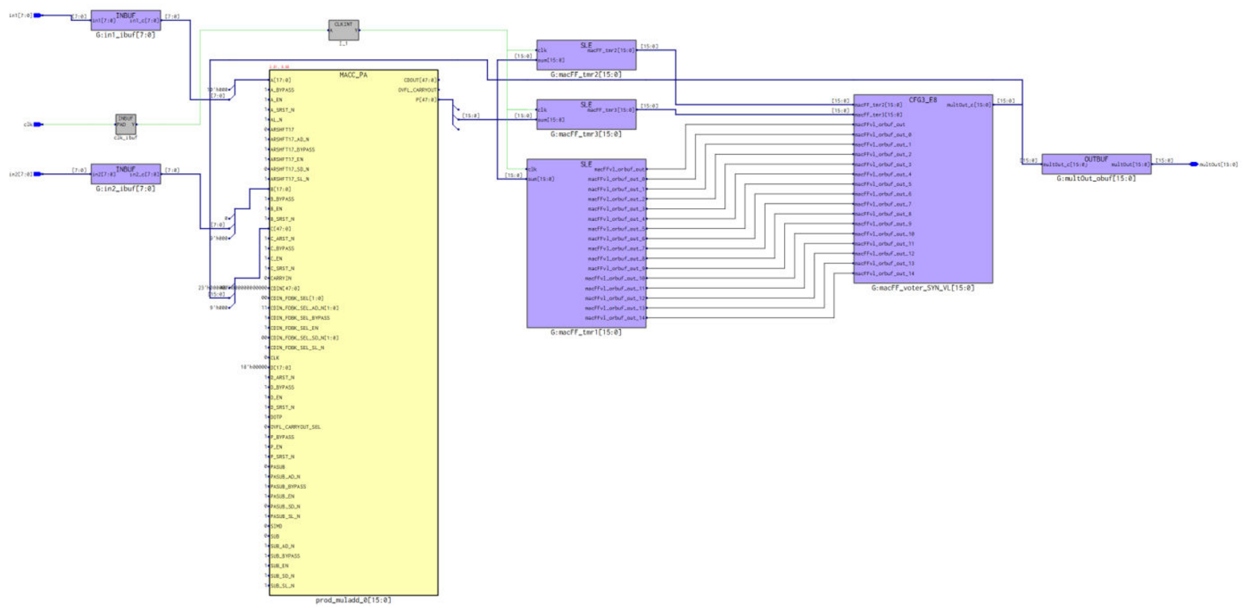
HDL:

```
module test9 (input clk , input [7:0] in1, input [7:0] in2, output [15:0] multOut);
wire [15:0] prod;
wire [16:0] sum;
reg [16:0] macFF;
assign prod = in1 * in2;
assign sum = prod + macFF;
always @ (posedge clk)
    begin
            macFF <= sum;
    end
assign multOut = macFF;
endmodule
```

FDC:

```
define_global_attribute {syn_radhardlevel} {tmr}
define_attribute {i:macFF[15:0]} {syn_preserve} {1}
```

Figure 9-10. syn_preserve Attribute on Register Associated with DSP Logic



Example 10: Stop TMR replication (TMR attribute set to none in the FDC file for instance)

Synplify Pro stops the triplication of registers inside the module when the `syn_radhardlevel` = `none` attribute is specified for the view.

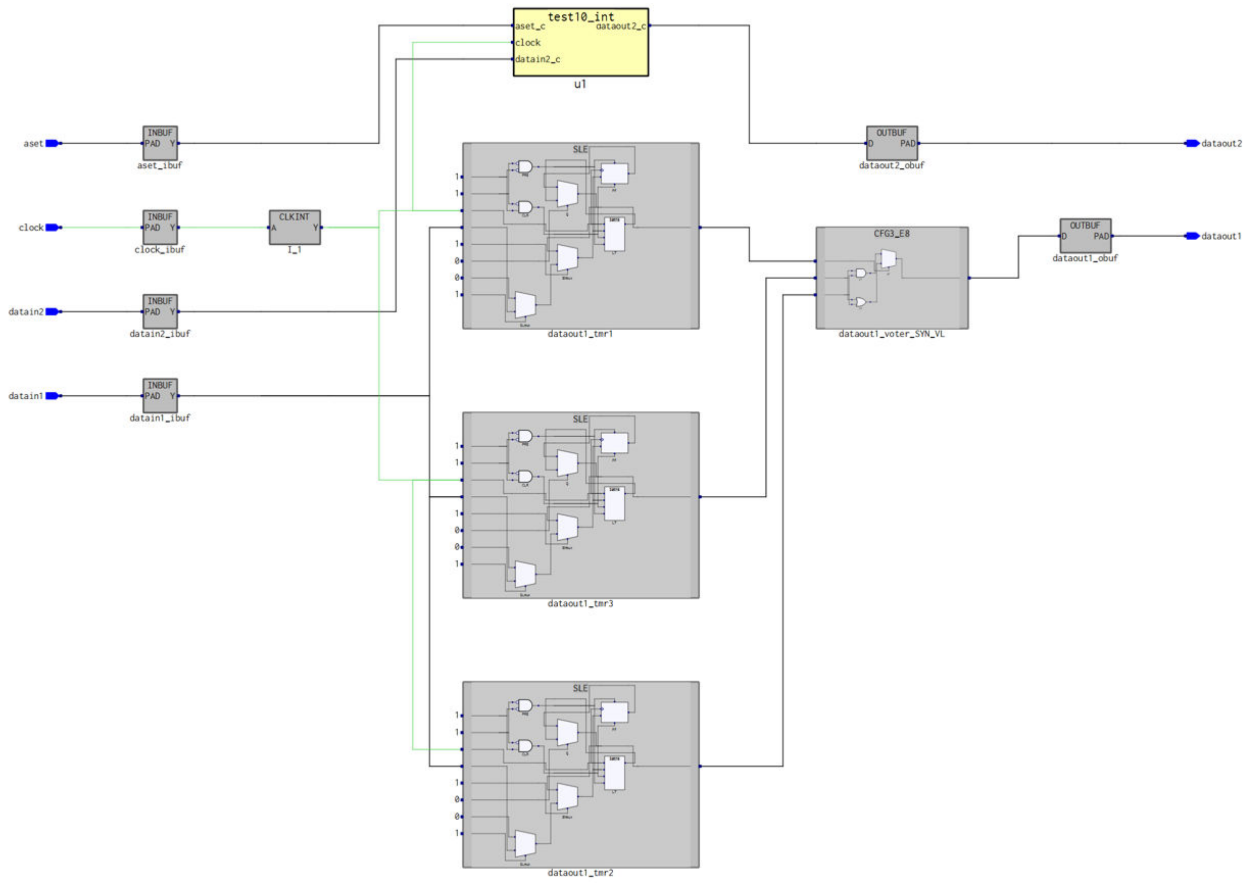
HDL:

```
module test10(clock,aset,datain1,datain2, dataout1, dataout2);
input clock, aset, datain1, datain2;
output reg dataout1;
output dataout2;
always @(posedge clock)
    dataout1 <= datain1;
test10_int u1 (.clock (clock),.aset (aset),.datain (datain2),.dataout (dataout2));
endmodule
module test10_int(clock,aset,datain,dataout);
input clock,aset;
input datain;
output reg dataout;
always @(posedge clock or posedge aset)
begin
    if (aset == 1'b1)
        dataout <= 1;
    else
        dataout <= datain;
end
endmodule
```

FDC:

```
define_global_attribute {syn_radhardlevel} {tmr}
define_attribute {i:u1.dataout} {syn_radhardlevel} {none}
```

Figure 9-11. Stop TMR Replication



Example 11: Safe CASE FSM (TMR attribute set globally in RTL)

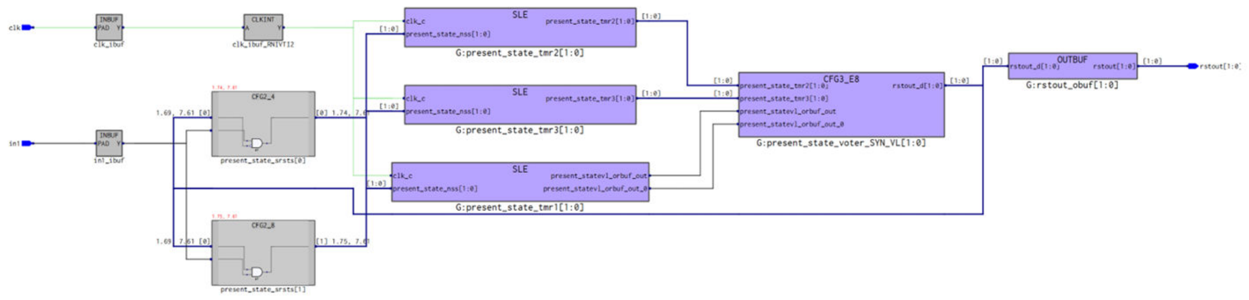
Synplify Pro triplicates all state registers and inserts voter logic at the output of the register. Illegal state transition logic is implemented using voter logic output.

HDL:

```
module state_test(clk, in1,rstout)/* synthesis syn_radhardlevel=tmr */;
input in1;
input clk;
parameter s0 = 2'b00;
parameter s1 = 2'b01;
parameter s2 = 2'b10;
reg [1:0] present_state,next_state;
wire [1:0] present_state_keep/* synthesis syn_preserve=1 */;
output [1:0] rstout;
assign present_state_keep = present_state;
assign rstout = present_state_keep;
always @ (posedge clk)
begin
    present_state <= next_state;
end
always @ (in1,present_state)
    case (present_state)
s0: begin
            if (in1) next_state <= s1;
               else next_state <= s0;
        end
        s1: begin
            if (in1) next_state <= s2;
               else next_state <= s0;
        end
        s2: begin
            if (in1) next_state <= s1;
```

```
                else next_state <= s0;
        end
        default: next_state <= s0;
endcase
endmodule
```

Figure 9-12. Safe CASE FSM



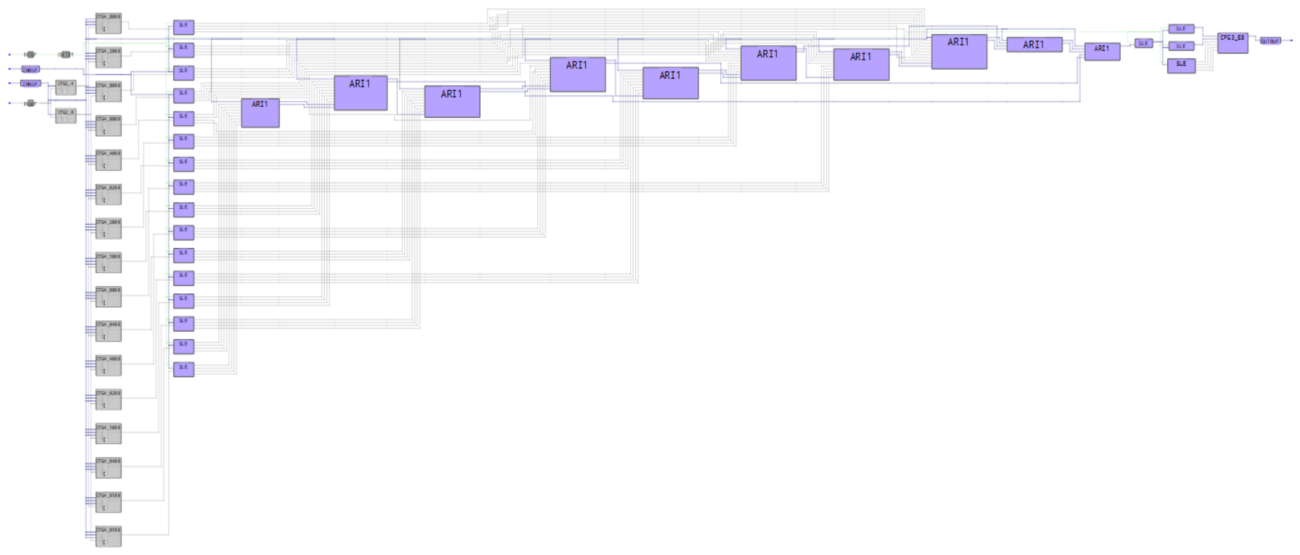
Example 12: Register array for single port pipeline register, use of `syn_ramstyle` = register attribute, TMR attribute applied on pipeline register.

Synplify Pro triplicates the pipeline register and none of the registers in the design.

HDL:

```
module test26_spregarray_pipe_globtmr(clk,we,addr,din,dout);
output reg [3:0] dout /* synthesis syn_radhardlevel=tmr */;
input [3:0] din;
input [3:0] addr;
input we, clk;
reg [3:0] mem [15:0]/* synthesis syn_ramstyle= "registers" */;
reg [3:0] dout1;
always @(posedge clk)
begin
dout1 <= mem[addr];
       if(we)
mem[addr] <= din;
end
always @(posedge clk)
begin
          dout <= dout1;
end
endmodule
```

Figure 9-13. Register Array for Single Port Pipeline Register



Example 13: Shift register implemented using URAM (TMR attribute is applied through FDC on the output register)

Synplify Pro triplicates the output register and inserts majority voting logic at register outputs.

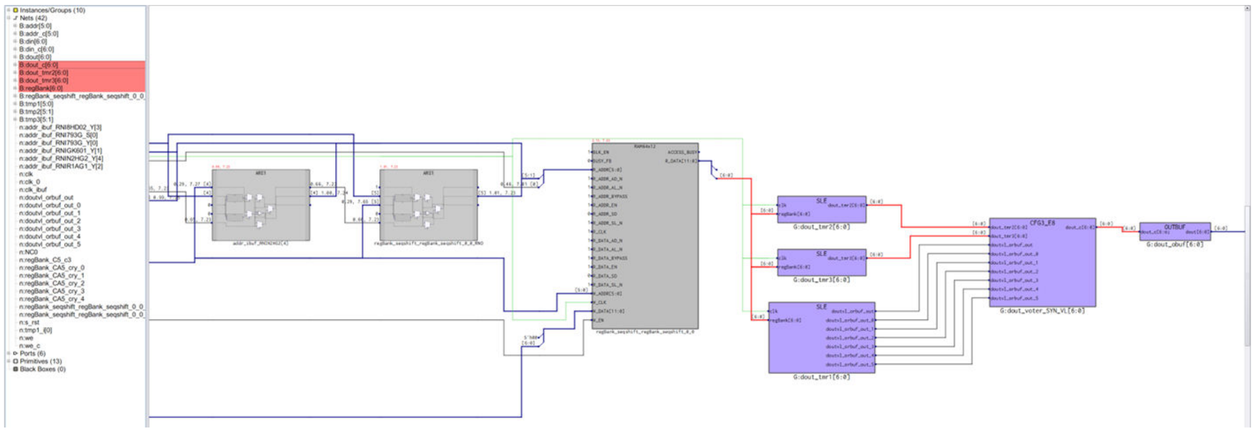
HDL:

```
module test33_shiftregINIT_fdctmr(clk,we,din,dout,s_rst,addr);
input clk, we, s_rst;
input [6:0] din;
input [5:0] addr;
output [6:0] dout;
reg [6:0] dout /* synthesis syn_preserve = 1 */;;
reg [6:0] regBank[36:0];
integer i;
always @(posedge clk) begin
 if (we) begin
    for (i=37; i>0; i=i-1) begin
     regBank[i] <= regBank[i-1];
    end
    regBank[0] <= din;
end
dout = regBank[addr];
end
endmodule
```

FDC:

```
define_attribute {i:dout} {syn_radhardlevel} {tmr}
```

Figure 9-14. Shift Register Implemented using URAM



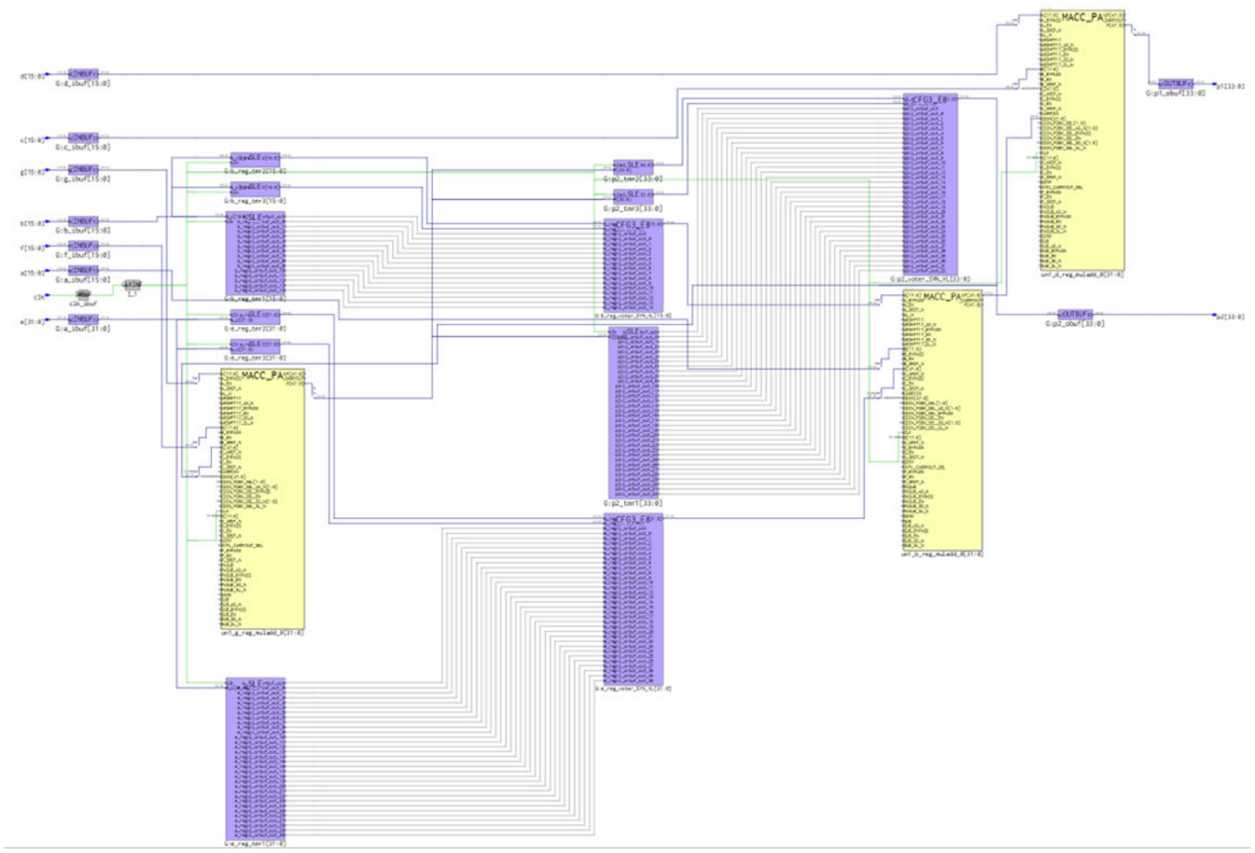
Example 14: Register triplication with `syn_preserve` attribute on few registers associated with MACC logic

Synplify Pro triplicates the register on which the `syn_preserve` attribute is applied and inserts majority voting logic at register outputs.

HDL:

```
module test49_maccvariations_tmr( clk,a,b,c,d,e,f,g,p1,p2 );
/* synthesis syn_radhardlevel=tmr */
parameter M = 16;
parameter N = 16;
input clk;
input signed[M-1:0] a, b;
input signed[N-1:0] c, d,f,g;
input signed[N*2-1:0] e;
output signed[M+N+1:0] p1,p2;
reg [M-1:0] a_reg;
reg [M-1:0] b_reg /* synthesis syn_preserve=1 */;
reg [N-1:0] c_reg, d_reg;
reg [N-1:0] f_reg, g_reg;
reg [N*2-1:0] e_reg /* synthesis syn_preserve=1 */;
reg [M+N+1:0] p1;
reg [M+N+1:0] p2 /* synthesis syn_preserve=1 */;
always@(posedge clk)
begin
 a_reg <= a;
 b_reg <= b;
c_reg <= c;
 d_reg <= d;
 e_reg <= e;
 f_reg <= f;
 g_reg <= g;
 p1  <= e_reg + (a_reg * b_reg) + (c_reg * d_reg);
 p2  <= p2 + (f_reg * g_reg);
 end
endmodule
```

Figure 9-15. Register Triplication with syn_preserve Attribute on Few Registers Associated with MACC Logic



Example 15: Global TMR attribute in FDC on three modules, but disable TMR attribute on one module in RTL

Synplify Pro triplicates the registers in t1 and t3 module, but it does not triplicate registers in the t2 module.

HDL:

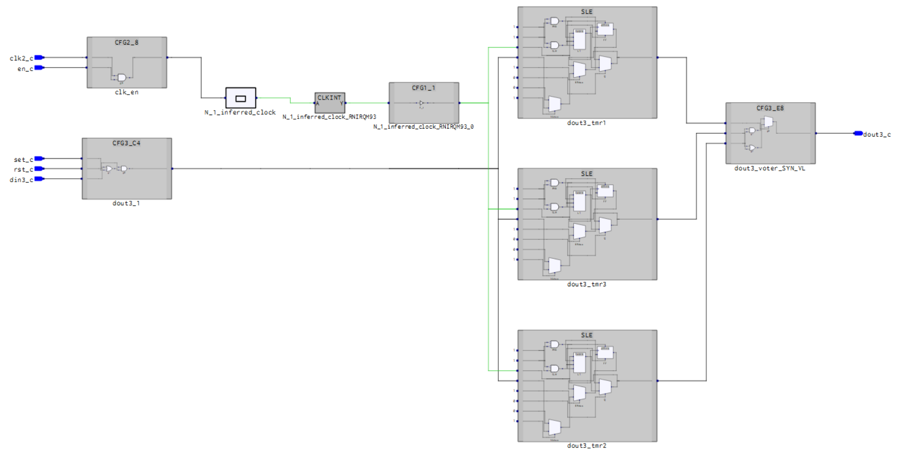
```
module test9_3modules_fdcglobtmr(clk0,clk1,clk2,rst,set,en,din1, dout1,din2,dout2,din3,dout3);
input clk0,clk1,clk2,rst,set,en,din1,din2,din3;
output dout1,dout2,dout3;
t1 t1_0 (clk0, rst, set, din1, dout1);
t2 t2_0 (clk1, rst, set, din2, dout2);
t3 t3_0 (clk2, rst, set,en,din3, dout3);
endmodule
module t1 (clk0, rst, set, din1, dout1);
input clk0, rst, set;
input din1;
output reg dout1;
always @(posedge clk0 or posedge rst or posedge set)
    if (rst)
        dout1 <= 0;
    else if (set)
        dout1 <= 1;
    else
        dout1 <= din1;
endmodule
module t2 (clk1, rst, set, din2, dout2)/* synthesis syn_radhardlevel = "none" */;
input clk1, rst, set;
input din2;
output reg dout2;
always @(posedge clk1)
    if (rst)
        dout2 <= 0;
    else if (set)
```

```
        dout2 <= 1;
    else
        dout2 <= din2;
endmodule
module t3 (clk2, rst, set, en, din3, dout3);
input clk2, rst, set,en;
input din3;
output reg dout3;
assign clk_en = clk2 && en;
always @(negedge clk_en)
    if (!rst)
        dout3 <= 0;
    else if (!set)
        dout3 <= 1;
    else
        dout3 <= din3;
endmodule
```

FDC:

```
define_global_attribute {syn_radhardlevel} {tmr}
```

Figure 9-16. Global TMR Attribute in FDC on Three Modules



Example 16: Register triplication with `syn_preserve` attribute on registers associated with MACC_PC_BA_ROM logic

Synplify Pro triplicates the output register on which the `syn_preserve` attribute is applied. MACC_PA_BC_ROM is inferred.

HDL:

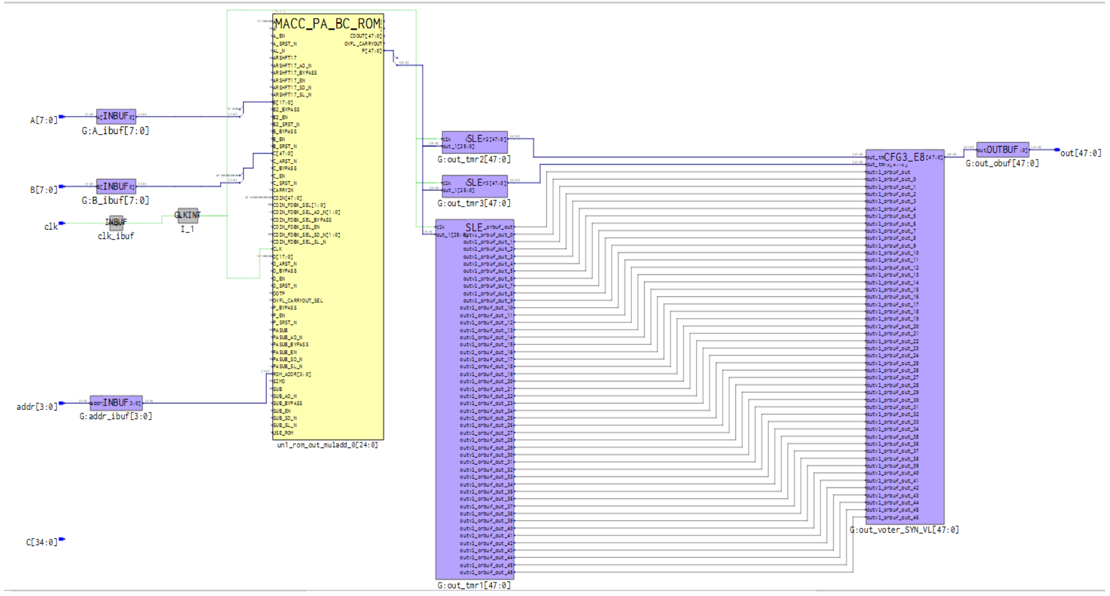
```
module test31_maccrom_synpreserve_globtmr(clk, A, B, C, out, addr) /* synthesis
syn_radhardlevel=tmr */;
 input clk;
 input [7:0] A;
 input [7:0] B;
 input [34:0] C; // Note: C is not used in the current logic
 input [3:0] addr;
 output reg [47:0] out /* synthesis syn_preserve=1 */;

 reg [16:0] rom_out;

 always @(posedge clk) begin
   case (addr)
     4'd0: rom_out <= 17'b01100000100011001;
     4'd1: rom_out <= 17'b00001101100111001;
     4'd2: rom_out <= 17'b01011011110110011;
     4'd3: rom_out <= 17'b10100110000000001;
     4'd4: rom_out <= 17'b01101001111111001;
     4'd5: rom_out <= 17'b01010000100010101;
     4'd6: rom_out <= 17'b10101101010001101;
```

```
      4'd7: rom_out <= 17'b01001001010010101;
      4'd8: rom_out <= 17'b01110111000111111;
      4'd9: rom_out <= 17'b10010101111110110;
      4'd10: rom_out <= 17'b01000110000010001;
      4'd11: rom_out <= 17'b11100000110110011;
      4'd12: rom_out <= 17'b10110101101111011;
      4'd13: rom_out <= 17'b01111010011000001;
      4'd14: rom_out <= 17'b00110110100111110;
      4'd15: rom_out <= 17'b11110101000010001;
      default: rom_out <= 17'b00010010110101 01;
    endcase
    out <= (rom_out * A) + B;
  end
endmodule
```

Figure 9-17. syn_preserve Attribute on Registers Associated with MACC_PC_BA_ROM Logic



Example 17: Register triplication of pipeline register with control logic associated with Single Port RAM

Synplify Pro infers RAM1Kx20 and triplicates the pipeline register.

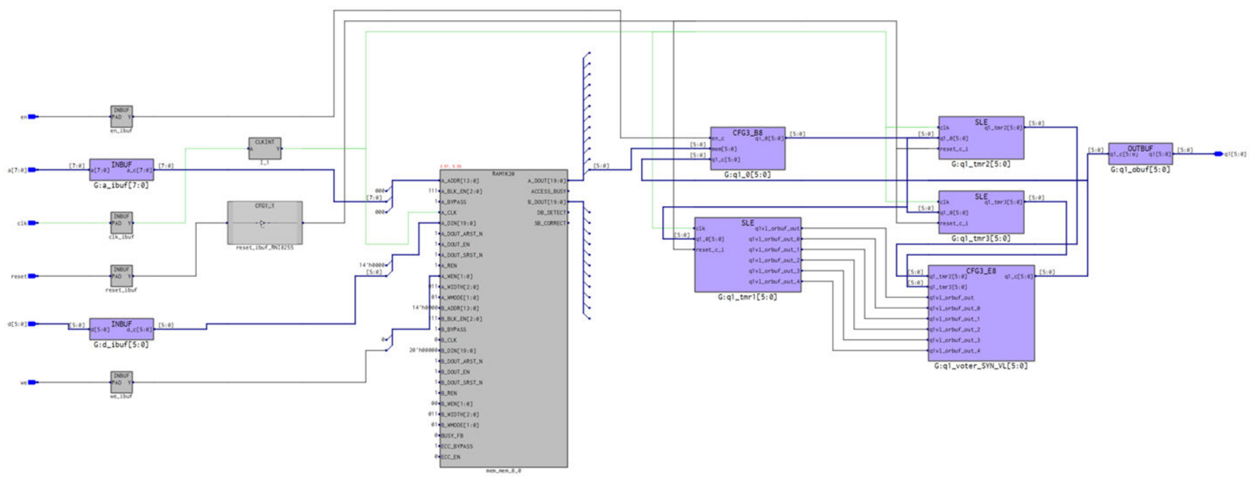
HDL:

```
module ram_singleport_addreg_pipe_areset_en(clk,we,a,d,q1,reset,en);
input [5:0] d;
input [7:0] a;
input clk, we,reset,en;
output [5:0] q1;
wire [5:0] q;
reg [7:0] addr;
reg [5:0] q1;
reg [5:0] mem [255:0];
always @(posedge clk) begin
    addr <= a;
    if(we)
    mem[a] <= d;
end
assign q = mem[addr];
always @ (posedge clk or posedge reset)
begin
    if(reset)
    q1 <= 0;
    else if(en)
     q1 <= q;
end
endmodule
```

FDC:

```
define_global_attribute {syn_radhardlevel} {tmr}
```

Figure 9-18. Register Triplication of Pipeline Register with Control Logic Associated with Single Port RAM



Example 18: Register triplication of output register with control logic associated with Dual Port RAM, use of `syn_keep` and `syn_preserve` attributes

Synplify Pro infers RAM in registers and LUTs. All inferred registers are triplicate, since the attribute is applied globally.

HDL:

```
module ram_dport_addreg_8kx2(data0,data1,waddr0,waddr1,we0,we1,clk, q0, q1);
/*synthesis syn_radhardlevel="tmr" */
parameter d_width = 2;
parameter addr_width = 10;
parameter mem_depth = 1024;
input [d_width-1:0] data0, data1;
input [addr_width-1:0] waddr0, waddr1;
input we0, we1, clk;
reg [d_width-1:0] mem [mem_depth-1:0];
reg [addr_width-1:0] reg_waddr0, reg_waddr1;
output [d_width-1:0] q0 /*synthesis syn_preserve=1 */;
output [d_width-1:0] q1 /*synthesis syn_preserve=1 */;
wire [addr_width-1:0] reg_waddr0_net/* synthesis syn_keep = 1 */;
wire [addr_width-1:0] reg_waddr1_net/* synthesis syn_keep = 1 */;
assign reg_waddr0_net = reg_waddr0;
assign reg_waddr1_net = reg_waddr1;
assign q0 = mem[reg_waddr0_net];
assign q1 = mem[reg_waddr1_net];
always @(posedge clk)
begin
         if (we0)
                 mem[waddr0] <= data0;
         reg_waddr0 <= waddr0;
end
always @(posedge clk)
begin
           if (we1)
                  mem[waddr1] <= data1;
             reg_waddr1 <= waddr1;
end
endmodule
```

9.1. Limitations (Ask a Question)

- The `syn_radhardlevel` TMR attribute triplicates the inferred Sequential Logic Element (SLE) register instance only. SLE Latch instances are not triplicated by this attribute. The SLEs should be inferred by Synplify Pro.
- The `syn_radhardlevel` TMR attribute applied to the SLE, D-type Flip-Flop Output Non-Inverted (DFN), D-type Latch Output Non-Inverted (DLN), and macros manually instantiated in the RTL is not triplicated.
- For `syn_radhardlevel` TMR specified globally, registers associated with memory and multipliers are packed inside RAM and DSP blocks, and are not triplicated.
- Users must add `syn_radhardlevel=none`, if CDC synchronizer FFs are used inside logic.

9.2. Summary (Ask a Question)

This application note outlines the different methods for specifying logic synthesized as a TMR implementation. It also demonstrates that the TMR-implemented register functions in Libero® are spatially separated to further enhance the SEU rate. The examples provided include the correct syntax and placement in the VHDL code, along with useful comments.

10. Revision History (Ask a Question)

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

Table 10-1. Revision History

Revision	Date	Description
D	11/2025	The following is the list of changes made in the revision D of the document: • The document title has been changed from "RT PolarFire TMR and Spatial Separation for Higher Reliability" to "TMR and Spatial Separation for Higher Reliability in Microchip FPGAs". • Added a Note in the Introduction section.
C	03/2025	The following is the list of changes made in the revision C of the document: • Added Design Requirements and Design Prerequisites sections. • Updated the TMR section. • Updated Table 7-1 in the Design Description section. • Added Figure 7-2, Figure 7-3, Figure 7-4, Figure 7-5, and Figure 7-6 in the PATH1 Description section. • Updated Figure 7-10, Figure 7-11, and Figure 7-12 in the Spatial Placement of the R-cells in a TMR Implemented Register Function section. • Updated Figure 9-5, Figure 9-6, Figure 9-7, Figure 9-13, Figure 9-16, and Figure 9-17 in the Design Examples section.
B	02/2023	Updated Spatial Placement.
A	12/2020	Initial Revision

Microchip FPGA Support

Microchip FPGA products group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, and worldwide sales offices. Customers are suggested to visit Microchip online resources prior to contacting support as it is very likely that their queries have been already answered.

Contact Technical Support Center through the website at www.microchip.com/support. Mention the FPGA Device Part number, select appropriate case category, and upload design files while creating a technical support case.

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

- From North America, call **800.262.1060**
- From the rest of the world, call **650.318.4460**
- Fax, from anywhere in the world, **650.318.8044**

Microchip Information

Trademarks

ISBN: 979-8-3371-2302-8

Legal Notice

Microchip Devices Code Protection Feature