

# Using Synopsys to Design Actel's Radiation-Hardened FPGAs

The use of synthesis to design Actel radiation-hardened parts requires a methodology to avoid radiation hazard cells. This application note describes a simple process of excluding these cells or replacing them with fault-tolerance cells in synthesis. In addition to the cell exclusion process or redundancy logic by manual means, there is a demand for automation in synthesis tools customized to your needs. To meet this demand, we are introducing ways to automate the task by using Synopsys FPGA compiler synthesis tools and its scripting capabilities. Other synthesis vendors have shown a strong interest in working with Actel to automate these capabilities in their tools. Please contact the synthesis vendor of choice for more information.

## Requirements

We assume that you are familiar with Actel's Synopsys FPGA compiler tool and setups and that you have an understanding of general synthesis methodology using Synopsys tools. We further assume that you are a moderate user of the Synopsys FPGA compiler script language DC-Shell. For more information about the use of Synopsys FPGA compiler tools

with Actel devices and general synthesis methodology, refer to Actel's *Synopsys Synthesis Methodology Guide*.

## Goals

This application note is designed to address customers of two major categories of Actel radiation-hardened parts, using Synopsys automation scripts:

- Customers using Actel combination register cells (C-C) instead of normal sequential cells
- Customers implementing redundancy register logic (triple modular redundancy, TMR) to replace normal sequential cells

Using Actel combination register cells (C-C) instead of normal sequential cells

Actel's radiation-hardened devices (ACT 2 RadHard) are equipped with registers built of pure combinational logic with a high tolerance for radiation, compared with normal register cells (flip-flops and latches). These cells are available in Actel's Synopsys libraries (ACT 2). You may prefer to use these cells instead of soft sequential cells. (See Table 1.)

**Table 1 • ACT 2 Combinational Models, Flip-Flops and Latches**

DFP1	D-Type Flip-Flop, Active High Preset
DFP1A	D-Type Flip-Flop, Active High Preset & Active Low Clock
DFP1B	D-Type Flip-Flop, Active Low Preset
DFP1D	D-Type Flip-Flop, Active Low Preset & Active Low Clock
DFPC	D-Type Flip-Flop, Active High Preset & Active Low Clear & Active High Clock
DLC1	Data Latch, Active High Clear
DLC1A	Data Latch, Active High Clear & Active Low Clock
DLE2C	Data Latch, Active Low Enable & Active Low Clock & Active High Clear
DLE3B	Data Latch, Active Low Enable & Active High Preset & Active Low Clock
DLE3C	Data Latch, Active Low Enable & Active Low Preset & Active Low Clock
DLP1	Data Latch, Active High Preset & Active High Clock
DLP1A	Data Latch, Active High Preset & Active Low Clock
DLP1B	Data Latch, Active Low Preset & Active High Clock
DLP1C	Data Latch, Active Low Preset & Active Low Clock

The following Synopsys FPGA compiler DC-Shell scripts and comments explain how to avoid using soft sequential cells and to force the synthesis to map to Actel radiation-hardened cells. Note that the sample exclusion script (my\_exclude.scr)

is run after Actel's Synopsys setup files and before synthesis mapping and compile (my\_design.scr).

Figure 1 is a sample Synopsys compile script to map my\_design, including the radiation-hard exclusion script.

---

### my\_design.scr

```
include actsetup.scr /*Actel/Synopsys setup scripts for ACT 2 parts */
read -f vhdl my_design.vhd /* Read in "my_design" hdl file */

current_design my_design /* Set the design level */
create_clock -period 20 clk /* Create master clock */
set_dont_touch_network clk /* Avoid adding any buffers to clock network */

set_port_is_pad /* Set for IO pads */
insert_pads /* Inserting IOs */

include my_exclude.scr /* Include the exclusion script to avoid seq.reg.module */

check_design /* Check design for DRC problems */
compile -map_effort medium /* Map to ACT 2 family */

write -f edif -h -o my_design.edn /* Save the EDIF netlist */
write my_design.db /* Save "my_design" in Synopsys DB format */

quit
```

---

### Figure 1 • Sample Synopsys Compile Script to Map my\_design to non\_seq Register

Figure 2 is a sample script to exclude soft sequential cells in Actel's Synopsys libraries.

---

### my\_exclude.scr

```
/* Following will EXCLUDE all normal sequential flip-flops in Actel ACT 2 family */
set_dont_use act2/DF*
/* Following will EXCLUDE all normal sequential latches in Actel ACT 2 family */
set_dont_use act2/DL*
/* Following will EXCLUDE all JK Seq. flip-flops in Actel ACT2 family */
set_dont_use act2/J*
/* Following will EXCLUDE all T type Seq. flip-flops in Actel ACT 2 family */
set_dont_use act2/T*
/* Following will INCLUDE C-C flip-flops to use in Actel ACT 2 family */
remove_attribute act2/DFP1 dont_use
remove_attribute act2/DFP1A dont_use
remove_attribute act2/DFP1B dont_use
remove_attribute act2/DFP1D dont_use
remove_attribute act2/DFPC dont_use
/* Following will INCLUDE C-C Latches to use in Actel ACT 2 family */
remove_attribute act2/DLC1 dont_use
remove_attribute act2/DLC1A dont_use
remove_attribute act2/DLE2C dont_use
remove_attribute act2/DLE3B dont_use
remove_attribute act2/DLE3C dont_use
remove_attribute act2/DLP1 dont_use
remove_attribute act2/DLP1A dont_use
remove_attribute act2/DLP1B dont_use
remove_attribute act2/DLP1C dont_use
```

---

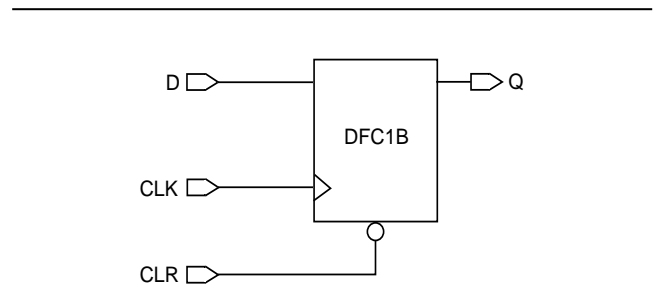
### Figure 2 • Script to Exclude Soft Sequential Cells

## Implementing Redundancy Register Logic (TMR) to Replace Normal Sequential Cells

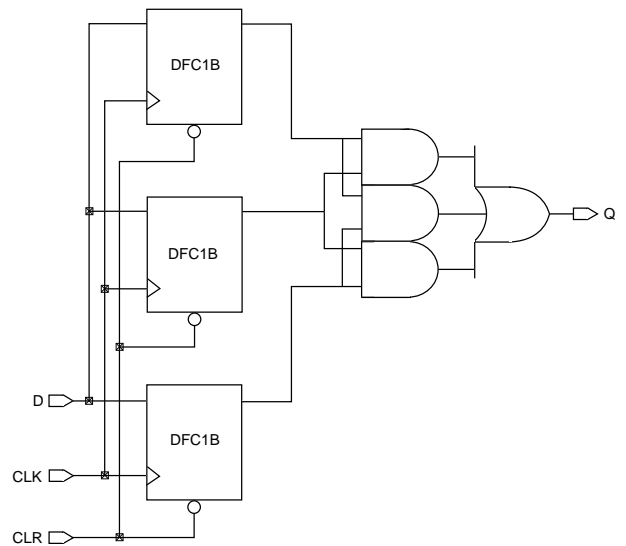
When designing for fault tolerance, you implement a variety of techniques that introduce register sequential elements with voting logic to identify and correct the faulty logic.

This application note explains a Synopsys DC-Shell script to replace a simple ASYNC reset flip-flop (DFC1B) with a TMR voting logic flip-flop (DFC1B\_TMR), where the faulty output is identified and corrected by "best of three" voting logic (Figure 3). Note that the TMR circuitry logic can be coded in High-level languages such as VHDL or Verilog (DFC1B\_TMR is in both VHDL and Verilog) and should be imported to the Synopsys FPGA compiler (`my_design.scr`) before running the replacement script (`my_replace.scr`). This is a generic TMR cell, and you are encouraged to explore and customize special TMR cells in VHDL or Verilog and to use the same Synopsys replacement script. Also, note that the replacement script replaces new macro cells for an old implementation pin-pin base. Further, the replacement script will match every input/output pin name (e.g., D, CLK, CLR, Q) and will substitute the new macro and connect them.

Figure 5 is a sample Synopsys compile scripts to map to `my_design`, including the replacement script.



**Figure 3 • Actel Sequential Cell (DFC1B)**



**Figure 4 • Actel TMR Sequential Cell (DFC1B\_TMR)**

### `my_design.scr`

```
include actsetup.scr /* Actel/Synopsys setup scripts for ACT 2 parts */
read -f vhd1 my_design.vhd /* Read in "my_design" hdl file */
read -f vhd1 dfc1b_tmr.vhd /* Read in the TMR register for replacement */

current_design my_design /* Set to design level to be mapped */
create_clock -period 20 clk /* Create master clock */
set_dont_touch_network clk /* Avoid adding any buffers to clock network */

set_port_is_pad /* Set for IO pads */
insert_pads /* Inserting IOs */

check_design /* Check design for DRC problems */
compile -map_effort medium /* Map to ACT 2 family */

write -f edif -h -o before_my_design.edn /* Save the EDIF netlist BEFORE TMR replacement */
include my_replace.scr /* Replacement of normal Seq. DFC1B with DCF1B-TMR */
write -f edif -h -o after_my_design.edn /* Save the EDIF netlist AFTER TMR replacement */
write my_design.db /* Save "my_design" in Synopsys DB format */
quit
```

**Figure 5 • Sample Synopsys Compile Scripts including Replacement Scripts**

Figure 6 is a sample script to replace the simple DCF1B flip-flop with the TMR voting flip-flop DFC1B\_TMR.

---

#### my\_replace.scr

```
/** Fill in the list that specify the macro to be replaced with the replacement macro in the
following format. The variable name used for this purpose is swap_macro_list.
    {{macroToBEReplaced0 replacementMacro0}
        {macroToBEReplaced1 replacementMacro1}
    {macroToBEReplacedN replacementMacroN}
    }
***/
swap_macro_list = {{DFC1B DFC1B_TMR} \
}
/** Get a list of all cells in the current design ***/
find cell ""
all_cells_in_design = dc_shell_status

/** For each element of the list cells ***/
foreach(sub_list, swap_macro_list) {
/** count tracks the nth element of the sublist ***/
count = 0
foreach(macro_name, sub_list) {
if (count == 0) {
/** First element in the sublist is the macro to be replaced ***/
source_macro_name = macro_name
filter(all_cells_in_design, "@ref_name == source_macro_name") >>/dev/null
/** Find all the instances of the macro to be replaced ***/
all_instances_of_source_macro = dc_shell_status
}
if (count == 1) {
/** Second element in the sublist is the replacement macro ***/
replacement_macro_name = macro_name
if (all_instances_of_source_macro != {} ) {
/** This command replaces the macro of the instance ***/
echo CHANGING source_macro_name TO replacement_macro_name FOR
all_instances_of_source_macro
change_link all_instances_of_source_macro replacement_macro_name
}
}
count = count + 1
}
}
```

---

**Figure 6 • Sample Script to Replace DCF1B flip-flop with the TMR voting flip-flop**

Figure 7 is a sample VHDL file of simple DFC1B TMR circuitry.

---

#### DFC1B\_TMR.vhd

```
library ieee;
use ieee.std_logic_1164.all;
library ACT 2;
use ACT 2.components.all;

entity DFC1B_TMR is
port (D      : in std_logic;
      Q      : out std_logic;
      CLR    : in std_logic;
      CLK    : in std_logic);

end DFC1B_TMR;

architecture tmr_arch of DFC1B_TMR is
signal wire0, wire1, wire2 : std_logic;
begin

U0: DFC1B
  port map (D, CLK, CLR, wire0);
U1: DFC1B
  port map (D, CLK, CLR, wire1);
U2: DFC1B
  port map (D, CLK, CLR, wire2);

U3: MAJ3
  port map (wire0, wire1, wire2, Q);

end tmr_arch;
```

---

**Figure 7 • Sample VHDL file of Simple DFC1B TMB Circuitry**

Figure 8 is a sample Verilog file of simple DFC1B TMR circuitry.

---

#### DFC1B\_TMR.v

```
module DFC1B_TMR(D,CLK,CLR,Q);

  input  D,CLK,CLR;
  output Q;

  wire wire0, wire1, wire2;

  DFC1B U0(D,CLK,CLR,wire0);
  DFC1B U1(D,CLK,CLR,wire1);
  DFC1B U2(D,CLK,CLR,wire2);

  MAJ3  U4(wire0,wire1,wire2,Q);

endmodule
```

---

**Figure 8 • Sample Verilog file of Simple DFC1B TMB Circuitry**





Actel and the Actel logo are registered trademarks of Actel Corporation.  
All other trademarks are the property of their owners.



**Take it to a higher level.**

<http://www.actel.com>

**Actel Europe Ltd.**

Daneshill House, Lutyens Close  
Basingstoke, Hampshire RG24 8AG  
United Kingdom

**Tel:** +44(0).1256.305600

**Fax:** +44(0).1256.355420

**Actel Corporation**

955 East Arques Avenue  
Sunnyvale, California 94086  
USA

**Tel:** 408.739.1010

**Fax:** 408.739.1540

**Actel Japan**

EXOS Ebisu Bldg. 4F  
1-24-14 Ebisu Shibuya-ka  
Tokyo 150 Japan

**Tel:** +81.(0)3445.7671

**Fax:** +81.(0)3445.7668