# Maximizing Logic Utilization in eX, SX, and SX-A FPGA Devices Using CC Macros

## Table of Contents



## Introduction

Typically, designers use logic optimization techniques to minimize logic resources, allowing the design to fit into a specific field-programmable gate array (FPGA). This application note introduces an optimization technique where flip-flops are created from combinatorial resources (CC macros).

CC macros were originally developed to allow designers to construct flip-flops to meet radiation-tolerant design requirements. However, designers can also take advantage of this design technique to utilize unused combinatorial cell (C-cell) when the design requires more register cell (R-cell) than are available in the selected device. This can be used to balance the types of logic resources required to allow fitting the design into eX, SX, and SX-A devices.

## SX and Related Architectures

The eX, SX, and SX-A FPGA families use a sea-of-modules architecture. This architecture features two types of logic modules - the C-cell and the R-cell.

The number of dedicated flip-flops in each family is the total number of R-cells available. The unique CC macro features in these families also allows building one flip-flop from every two C-cells. Hence, the maximum number of flip-flops is given by EQ 1:

$$\text{Max Flip-flops} = \text{R-cells} + (\text{C-cells}/2) \quad \textit{EQ 1}$$

Table 1 lists the available flip-flops in each device for the eX, SX, and SX-A families.

***Table 1 •*** **eX, SX, and SX-A FPGA Devices Available Flip-flops**

| **SX-A Family** | **A54SX08A** | **A54SX16A** | **A54SX32A** | **A54SX72A** |
|---|---|---|---|---|
| Dedicated Flip-flops | 256 | 528 | 1,080 | 2,012 |
| Combinatorial Cells | 512 | 924 | 1,800 | 4,024 |
| Maximum Flip-flops | 512 | 990 | 1,980 | 4,024 |
| **SX Family** | **A54SX08** | **A54SX16** | **A54SX16P** | **A54SX32** |
| Dedicated Flip-flops | 256 | 528 | 528 | 1,080 |
| Combinatorial Cells | 512 | 924 | 924 | 1,800 |
| Maximum Flip-flops | 512 | 990 | 990 | 1,980 |
| **eX Family** | **eX64** | **eX128** | **eX256** | |
| Dedicated Flip-flops | 64 | 128 | 256 | |
| Combinatorial Cells | 128 | 256 | 512 | |
| Maximum Flip-flops | 128 | 256 | 512 | |

Consider a design that uses 80 registers and 92 combinatorial cells. Normally, a designer selects the eX128 device instead of the eX64, since the number of registers used in this design is greater than the number of available R-cells in the eX64 device. However, in some cases, the goal is to fit the design into the smallest available device in the family. Since the eX64 has 128 C-cells, this leaves a total of 36 unused C-cells. If the designer uses all 64 R-cells in the eX64 for registers, 16 more flip-flops are still required for the design.

With the 36 C-cells still available, the designer can create up to 18 additional flip-flops, allowing a maximum of 82 available flip-flops for the design—enough to accommodate the required 80 registers. This optimization can be achieved without modifying the functionality of the design.

Another advantage of CC macros is that designs using CC macros in some Microsemi FPGA's are more resistant to single-event upset (SEU) than designs using R-cells. For this reason, CC macro design techniques for ex, SX, and SXA devices are appropriate for space applications and all types of military and high-reliability equipment.

Note: Although CC macros are available for use in Microsemi's RTSX-SU device, it is not necessary to use CC macros for SEU purposes. The R-cells in these families are inherently radiation tolerant because they are implemented with a Triple-Module Redundancy (TMR) voting technique. They can be used to gain additional flip-flops; however, they do not have the same radiation performance or features as the dedicated R-cells.

## CC Macro Flip-flops

CC macros use two C-cells with feedback to form a storage element. The C-cells are gated D-Latches connected serially. The input C-cell is the Master and the output C-cell is the Slave. Figure 1 shows a CC macro flip-flop formed by two C-cells.

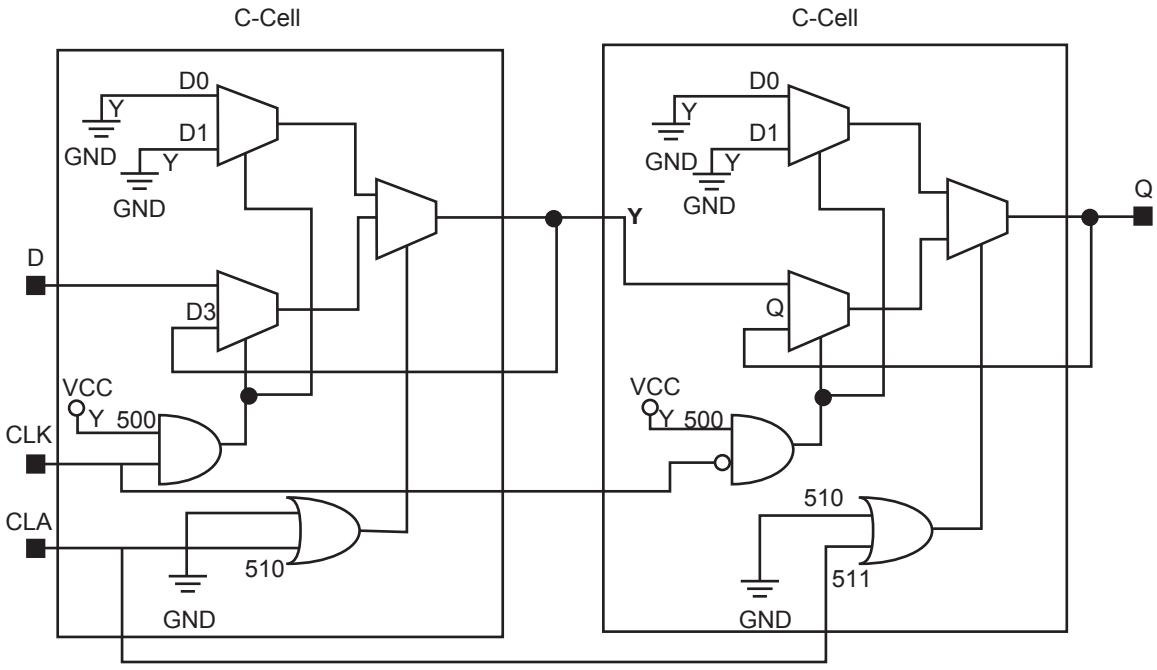


***Figure 1* • o DFC1B_CCMacro**

## Performance of CC Macros Versus R-Cells

This example compares the performance of CC macros compared to R-cells in a 28-bit counter design.

Table 2 shows the cell utilization and performance of a 28-bit counter design when all registers in the counter are formed by dedicated registers (R-cells) in the eX64 device.

***Table 2 •*** **eX64 28-bit Counter Design Using Dedicated Registers (R-cells)**

| Device Utilization | Used (% of Total) | Total |
|---|---|---|
| Sequential | 28 (43.75%) | 64 |
| Combinatorial | 49 (38.28%) | 128 |
| Total Logic | 77 (40.10%) (seq+comb) | 192 |
| I/O with Clocks | 31 | 84 |
| Clock | 1 | 2 |
| HCLK | 1 | 1 |

*Note: Performance: 188 MHz*

Table 3 shows the corresponding cell utilization and performance of a 28-bit counter design when all registers in the counter are formed by CC macro flip-flops in the eX64 device.

***Table 3 •*** **eX64 28-bit Counter Design Using CC Macro Flip-flops**

| Device Utilization | Used (% of Total) | Total |
|---|---|---|
| Sequential | 0 (0.00%) | 64 |
| Combinatorial | 97 (75.78%) | 128 |
| Total Logic | 97 (50.52%) | 192 |
| I/O with Clocks | 31 | 84 |
| Clock | 2 | 2 |
| HCLK | 0 | 1 |

*Note: Performance: 121 MHz*

The performance numbers for both methods show that a design formed by CC macro flip-flops is slower than a design formed by R-cells. Thus, CC macros should NOT be used in the critical paths of a design. In addition, after inserting CC macros, designers should perform static and dynamic timing analyses to verify the performance of the design.

Note: Flip-flops constructed from C-cells can only be driven by a routed clock (RCLK network). They cannot be driven by a hardwired clock (HCLK network).

# Design Implementation

CC macros can be implemented in a design using one of the following methods:

- Schematics
- Source file attributes
- Synthesis directives

The use of CC macros does not change the functionality of the design. Using synthesis directives is advantageous because it is not necessary to modify or edit the source code.

Note: Microsemi CC macro flip-flops cannot have both clear and preset inputs simultaneously.

## Schematics

To implement CC macros in a design using schematics, replace the extra register blocks in the schematics with the appropriate CC macro modules. The following CC macros are available:

- DF1_CC
- DFC1B_CC
- DFP1B_CC

These macros correspond with their R-cell equivalents. Figure 2 illustrates the available CC macros in the eX, SX, and SX-A families.

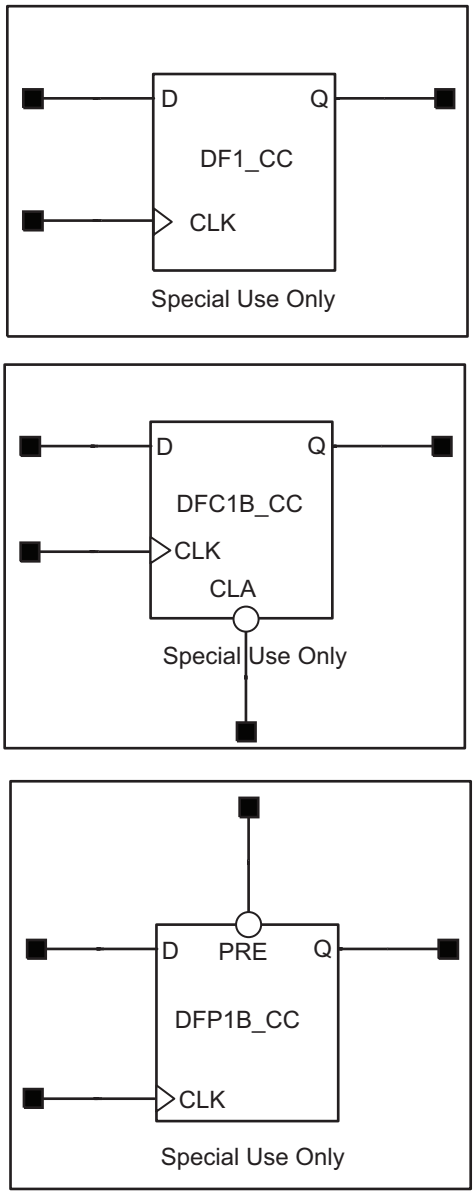


***Figure 2* • DF1_CC, DFC1B_CC, andDFP1B_CC CC Macros**

## Adding Attributes to a Source File

Synplify provides a syn_radhardlevel attribute that specifies the implementation technique for designs. This attribute can be used to create CC macro flip-flops.

It can be applied to a module (Verilog), an architecture (VHDL), or a register.

The syn_radhardlevel attribute can be globally applied to the top-level module or architecture of a design and can be overridden for specific portions. The flip-flop type can even be controlled on a register-by-register basis.

The syn_radhardlevel attribute is only effective if the corresponding Microsemi Verilog (cc_alt.v) or VHDL (cc_alt.vhd) macro files for the design technique are included in the source files list of the Synplify project. These files are located in:

www.microsemi.com/soc/download/rsc/?f=AF_AC201_LF.

The value "cc" for syn_radhardlevel is the instruction for implementing flip-flops as CC macros. This is illustrated below for VHDL and Verilog.

## *VHDL*

In VHDL, the syn_radhardlevel syntax is:

```
attribute syn_radhardlevel of object:
object_type is "cc"
```

where object is an architecture name or a register output signal and object_type is an architecture or a signal.

### Attribute Settings for a Register

This example uses a CC macro to implement the single register (data_b):

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
library synplify;
entity test (
----
----
end test;
architecture rtl of top is
signal data_a: std_logic;
signal data_b: std_logic_vector (7 downto 0);
-- using cc macro's
attribute syn_radhardlevel : string;
attribute syn_radhardlevel of data_b : signal
is "cc";
-- Other code
```

### Attribute Settings for an Architecture

This example uses CC macros to implement all registers declared in an architecture:

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
library synplify;
entity test (
----
----
end test;
architecture rtl of top is attribute
syn_radhardlevel : string;
attribute syn_radhardlevel of rtl : architecture
is "cc";
-- Other code
```

## *Verilog*

In Verilog, the syn_radhardlevel syntax is:

```
Object/* synthesis syn_radhardlevel="cc"*/;
```

where object is a module or a register output signal.

### Attribute Settings for a Register

This example uses a CC macro to implement the register dataout [3:0]:

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

### Attribute Settings for a Module

This example uses CC macros to implement all registers in a module:

```
module top (clk, dataout, a, b) /* synthesis
syn_radhardlevel ="cc" */;
input clk;
input a;
input b;
output dataout [3:0];
reg [3:0] dataout;
register [7:0] data_a;
// Other code
```

## Adding Attributes during Synthesis

Using synthesis attributes, CC macros can be inferred into a design while synthesizing. In addition to requiring no source code modification, designers can easily experiment with different CC-macro/dedicated-register combinations until an optimal solution is found.

The syn_radhardlevel attribute can be set either through a constraint file (.sdc) or through the SCOPE constraint editor in Synplify.

### *Constraint File*

This example illustrates the use of the syn_radhardlevel attribute in a design constraint file:

define_attribute {dataout[3:0]}

syn_radhardlevel {"cc"}

### *SCOPE Editor*

The Microsemi Libero® Integrated Design Environment (IDE) Platinum package supports the Synplicity GUI-based constraints editor, SCOPE (this is not supported in Libero IDE Silver and Gold). To set constraints using SCOPE (Figure 3):

1. Compile the design by selecting **Compile Only** from the **Synplicity Run** menu.
2. Start SCOPE in the open project window by clicking the **SCOPE** icon on the toolbar.
3. Open the **Constraint File**.
4. Click the **Attributes** tab at the bottom of the SCOPE window. The spreadsheet displays columns of attributes. From this tab, set the object and object type values for each attribute.
5. From the **Object** column, scroll down and select **v.work.<top_level_entityname>**.
6. Set the **Attribute** column to syn_radhardlevel and change the value from v.work.<top_level_entityname> to "none". This prevents all flip-flops in the entire design from being implemented with CC macros.
7. In the subsequent row(s), select **register** from the **Object Type** column.
8. Select the specific register to implement as a CC macro from the **Object** column.
9. Select **syn_radhardlevel** in the Attribute column and set the value to "cc". This defines the CC macro for the object(s) specified in SCOPE.
10. Repeat steps 7-9 for all registers to be implemented with CC macros.

| | Enabled | Object Type | Object | Attribute | Value | Val Type | Description |
|---|---|---|---|---|---|---|---|
| 1 | ☑ | | v:work.top_level | syn_radhardlevel | none | | |
| 2 | ☑ | register | make_scc.tx_hold_reg[7:0] | syn_radhardlevel | cc | string | Radiation-hardened implementation style |

***Figure 3* • Example of Setting Constraints in Synplify using the SCOPE Editor**

To synthesize a design using CC macros, include the family's Synplicity library file (eX.vhd, for example) and the cc_alt.vhd file along with all HDL design files as shown in Figure 4. In Project View, place cc_alt.vhd and the package file (ex.vhd) at the top as illustrated in Figure 4.

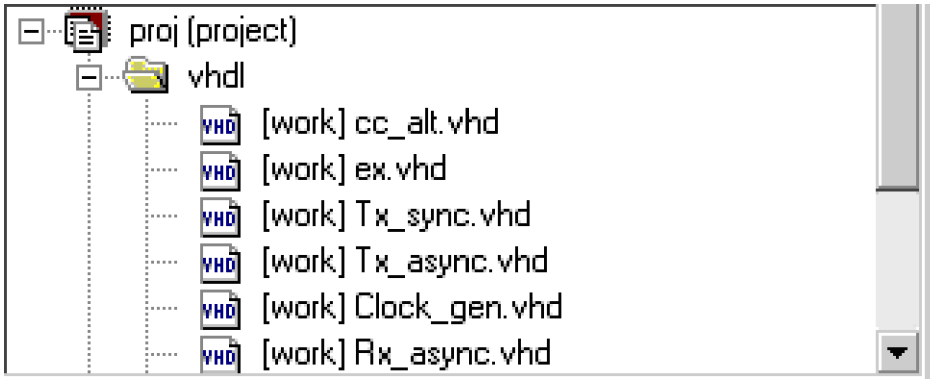


***Figure 4* • Importing the Source Files in Synplify**

# Layout Considerations

CC macros should use a global clock to ensure low skew between master and slave elements and hence correct flip-flop functionality. For the eX, SX, and SX-A families, all CC macros should be connected to the low-skew CLKA/B global resources. The use of the low-skew QCLK network is also acceptable when using the A54SX72A device.

Note: Ensuring close placement of the master and slave elements is helpful, but does not guarantee functionality. In an effort to ensure close placement, Microsemi's Designer software produces warnings similar to the following for unacceptable placement in an SX-A device:

```
A master-slave flip-flop FC driven by
Z_1I18/U1 could not be established.
If you require better timing, please
unplace the macros and try layout again!
```

The 'FC' in the above warning refers to the FastConnect routing resource in the SX-A architecture. FastConnect enables horizontal routing between any two logic modules within a given SuperCluster and vertical routing with the SuperCluster immediately below it. Refer to the device datasheet at www.microsemi.com/soc/documents/SXA_DS.pdf for more information.

# Conclusion

CC macros are useful when a design runs out of registers and C-cells are still available in the device. In this situation it is not necessary to use a bigger device in the family—utilizing CC macro flip-flops enables the design to fit into the same device.

Since the performance of CC macros differs from R-cells, they should not be used in the critical paths of the design. They are used in the timing critical paths of the design may result in a 30% decline in performance. Also, CC macros do not support simultaneous CLR/PRE and cannot be routed to the HCLK network. Finally, you should connect CC macros to low-skew RCLK and QCLK networks to ensure the correct functionality.

# List of Changes

| Revision * | Changes | Page |
|---|---|---|
| Revision 2 (March 2012) | Removed Axcelerator (SAR 22126). | |
| | Table 1 was revised (SAR 22126). | 2 |
| | The "SX and Related Architectures" section was modified (SAR 22126). | 1 |
| | The "Layout Considerations" section was modified (SAR 22126). | 8 |
| Revision 1 (February 2003) | EQ 1 was updated. | 1 |
| | Table 1 was updated to include Axcelerator. | 2 |
| | The "SCOPE Editor" section is new | 7 |
| | The "Layout Considerations" section is new. | 8 |
| *Note:* **The revision number is located in the part number after the hyphen. The part number is displayed at the bottom of the last page of the document. The digits following the slash indicate the month and year of publication.* | | |

**Microsemi Corporate Headquarters**
One Enterprise, Aliso Viejo CA 92656 USA
Within the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

Microsemi Corporation (NASDAQ: MSCC) offers a comprehensive portfolio of semiconductor solutions for: aerospace, defense and security; enterprise and communications; and industrial and alternative energy markets. Products include high-performance, high-reliability analog and RF devices, mixed signal and RF integrated circuits, customizable SoCs, FPGAs, and complete subsystems. Microsemi is headquartered in Aliso Viejo, Calif. Learn more at **www.microsemi.com**.



51900011-2/3.12