

Welcome to [E-XFL.COM](http://E-XFL.COM)

### **Understanding Embedded - FPGAs (Field Programmable Gate Array)**

Embedded - FPGAs, or Field Programmable Gate Arrays, are advanced integrated circuits that offer unparalleled flexibility and performance for digital systems. Unlike traditional fixed-function logic devices, FPGAs can be programmed and reprogrammed to execute a wide array of logical operations, enabling customized functionality tailored to specific applications. This reprogrammability allows developers to iterate designs quickly and implement complex functions without the need for custom hardware.

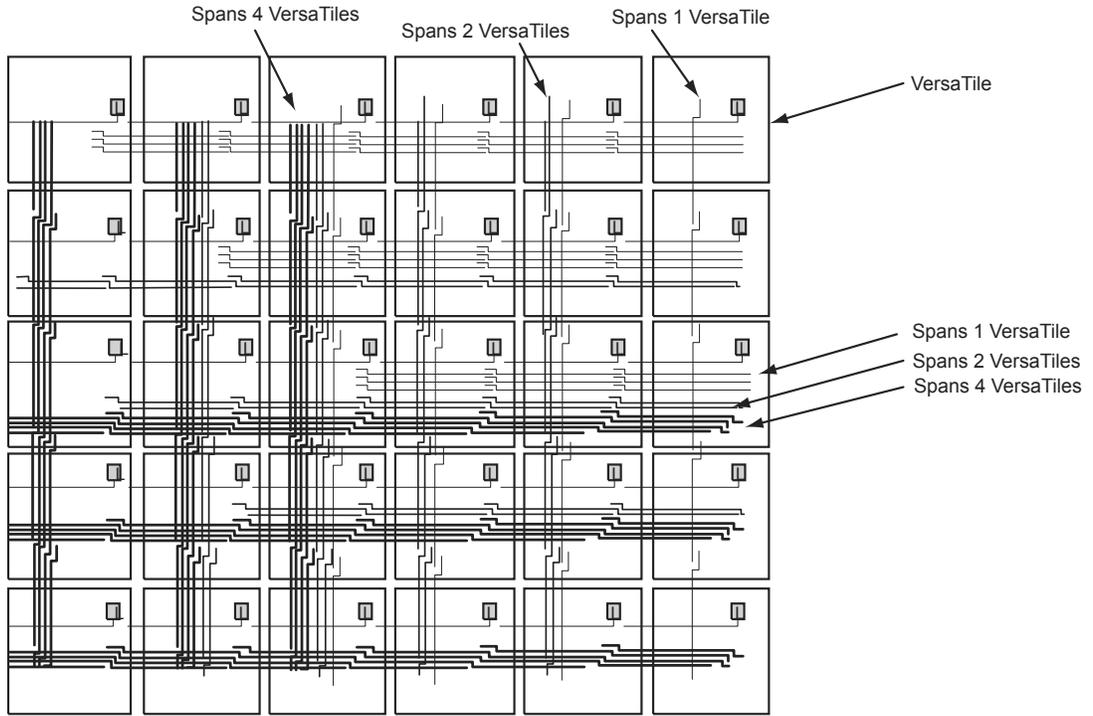
### **Applications of Embedded - FPGAs**

The versatility of Embedded - FPGAs makes them indispensable in numerous fields. In telecommunications.

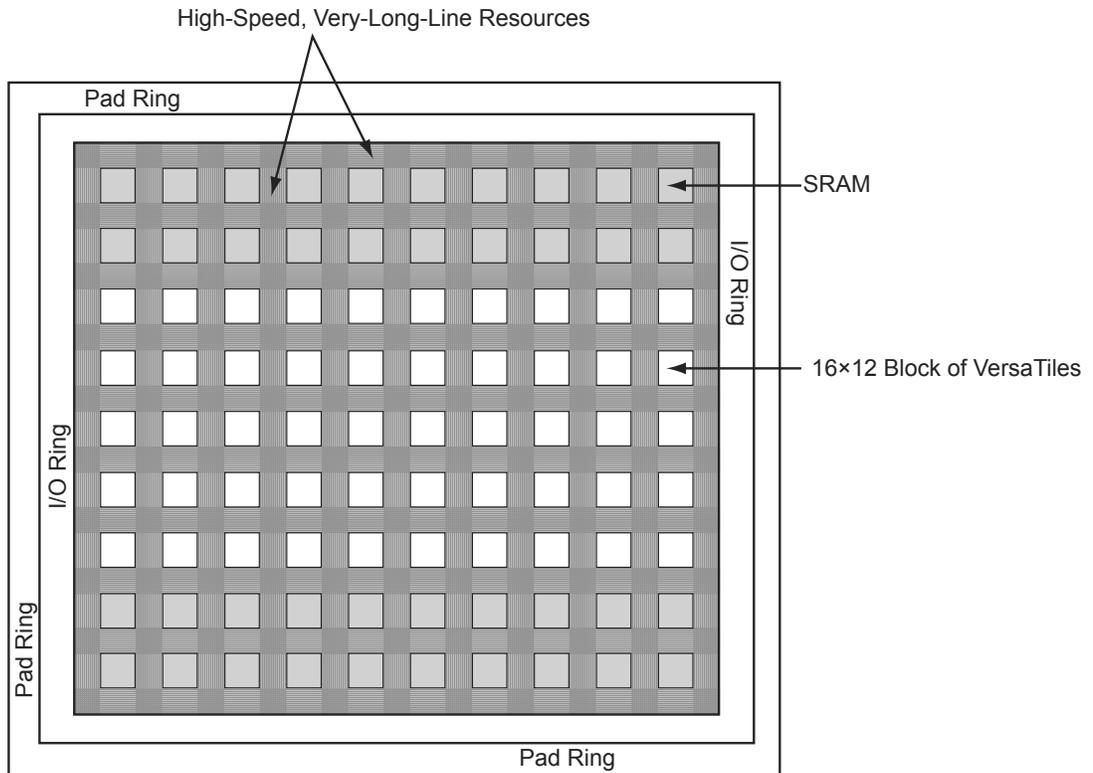
#### **Details**

Product Status	Obsolete
Number of LABs/CLBs	-
Number of Logic Elements/Cells	-
Total RAM Bits	147456
Number of I/O	154
Number of Gates	1000000
Voltage - Supply	1.14V ~ 1.575V
Mounting Type	Surface Mount
Operating Temperature	0°C ~ 85°C (TJ)
Package / Case	208-BFQFP
Supplier Device Package	208-PQFP (28x28)
Purchase URL	<a href="https://www.e-xfl.com/product-detail/microsemi/m1a3p1000l-1pqg208">https://www.e-xfl.com/product-detail/microsemi/m1a3p1000l-1pqg208</a>





**Figure 1-11 • Efficient Long-Line Resources**



**Figure 1-12 • Very-Long-Line Resources**

### External I/O or Local signal as Clock Source

External I/O refers to regular I/O pins are labeled with the I/O convention IOuxwByVz. You can allow the external I/O or internal signal to access the global. To allow the external I/O or internal signal to access the global network, you need to instantiate the CLKINT macro. Refer to Figure 3-4 on page 51 for an example illustration of the connections. Instead of using CLKINT, you can also use PDC to promote signals from external I/O or internal signal to the global network. However, it may cause layout issues because of synthesis logic replication. Refer to the "Global Promotion and Demotion Using PDC" section on page 67 for details.

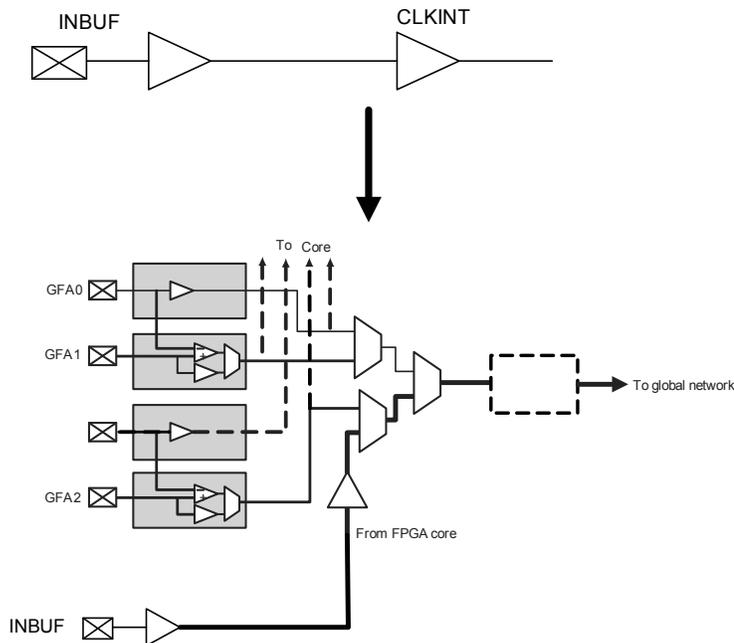


Figure 3-14 • CLKINT Macro

### Using Global Macros in Synplicity

The Synplify® synthesis tool automatically inserts global buffers for nets with high fanout during synthesis. By default, Synplicity® puts six global macros (CLKBUF or CLKINT) in the netlist, including any global instantiation or PLL macro. Synplify always honors your global macro instantiation. If you have a PLL (only primary output is used) in the design, Synplify adds five more global buffers in the netlist. Synplify uses the following global counting rule to add global macros in the netlist:

1. CLKBUF: 1 global buffer
2. CLKINT: 1 global buffer
3. CLKDLY: 1 global buffer
4. PLL: 1 to 3 global buffers
  - GLA, GLB, GLC, YB, and YC are counted as 1 buffer.
  - GLB or YB is used or both are counted as 1 buffer.
  - GLC or YC is used or both are counted as 1 buffer.

---

**Figure 4-31 • Static Timing Analysis Using SmartTime**

**Place-and-Route Stage Considerations**

Several considerations must be noted to properly place the CCC macros for layout.

For CCCs with clock inputs configured with the Hardwired I/O–Driven option:

- PLL macros must have the clock input pad coming from one of the GmA\* locations.
- CLKDLY macros must have the clock input pad coming from one of the Global I/Os.

If a PLL with a Hardwired I/O input is used at a CCC location and a Hardwired I/O–Driven CLKDLY macro is used at the same CCC location, the clock input of the CLKDLY macro must be chosen from one of the GmB\* or GmC\* pin locations. If the PLL is not used or is an External I/O–Driven or Core Logic–Driven PLL, the clock input of the CLKDLY macro can be sourced from the GmA\*, GmB\*, or GmC\* pin locations.

For CCCs with clock inputs configured with the External I/O–Driven option, the clock input pad can be assigned to any regular I/O location (IO\*\*\*\*\* pins). Note that since global I/O pins can also be used as regular I/Os, regardless of CCC function (CLKDLY or PLL), clock inputs can also be placed in any of these I/O locations.

By default, the Designer layout engine will place global nets in the design at one of the six chip globals. When the number of globals in the design is greater than six, the Designer layout engine will automatically assign additional globals to the quadrant global networks of the low power flash devices. If the user wishes to decide which global signals should be assigned to chip globals (six available) and which to the quadrant globals (three per quadrant for a total of 12 available), the assignment can be achieved with PinEditor, ChipPlanner, or by importing a placement constraint file. Layout will fail if the

**Table 7-10 • Hot-Swap Level 3**

<b>Description</b>	Hot-swap while bus idle
<b>Power Applied to Device</b>	Yes
<b>Bus State</b>	Held idle (no ongoing I/O processes during insertion/removal)
<b>Card Ground Connection</b>	Reset must be maintained for 1 ms before, during, and after insertion/removal.
<b>Device Circuitry Connected to Bus Pins</b>	Must remain glitch-free during power-up or power-down
<b>Example Application</b>	Board bus shared with card bus is "frozen," and there is no toggling activity on the bus. It is critical that the logic states set on the bus signal not be disturbed during card insertion/removal.
<b>Compliance of IGLOO and ProASIC3 Devices</b>	30K gate devices, all IGLOOe/ProASIC3E devices: Compliant with two levels of staging (first: GND; second: all other pins) Other IGLOO/ProASIC3 devices: Compliant: Option A – Two levels of staging (first: GND; second: all other pins) together with bus switch on the I/Os Option B – Three levels of staging (first: GND; second: supplies; third: all other pins)

**Table 7-11 • Hot-Swap Level 4**

<b>Description</b>	Hot-swap on an active bus
<b>Power Applied to Device</b>	Yes
<b>Bus State</b>	Bus may have active I/O processes ongoing, but device being inserted or removed must be idle.
<b>Card Ground Connection</b>	Reset must be maintained for 1 ms before, during, and after insertion/removal.
<b>Device Circuitry Connected to Bus Pins</b>	Must remain glitch-free during power-up or power-down
<b>Example Application</b>	There is activity on the system bus, and it is critical that the logic states set on the bus signal not be disturbed during card insertion/removal.
<b>Compliance of IGLOO and ProASIC3 Devices</b>	30K gate devices, all IGLOOe/ProASIC3E devices: Compliant with two levels of staging (first: GND; second: all other pins) Other IGLOO/ProASIC3 devices: Compliant: Option A – Two levels of staging (first: GND; second: all other pins) together with bus switch on the I/Os Option B – Three levels of staging (first: GND; second: supplies; third: all other pins)

- In Active and Static modes:
  - Input buffers with pull-up, driven Low
  - Input buffers with pull-down, driven High
  - Bidirectional buffers with pull-up, driven Low
  - Bidirectional buffers with pull-down, driven High
  - Output buffers with pull-up, driven Low
  - Output buffers with pull-down, driven High
  - Tristate buffers with pull-up, driven Low
  - Tristate buffers with pull-down, driven High
- In Flash\*Freeze mode:
  - Input buffers with pull-up, driven Low
  - Input buffers with pull-down, driven High
  - Bidirectional buffers with pull-up, driven Low
  - Bidirectional buffers with pull-down, driven High

## Electrostatic Discharge Protection

Low power flash devices are tested per JEDEC Standard JESD22-A114-B.

These devices contain clamp diodes at every I/O, global, and power pad. Clamp diodes protect all device pads against damage from ESD as well as from excessive voltage transients.

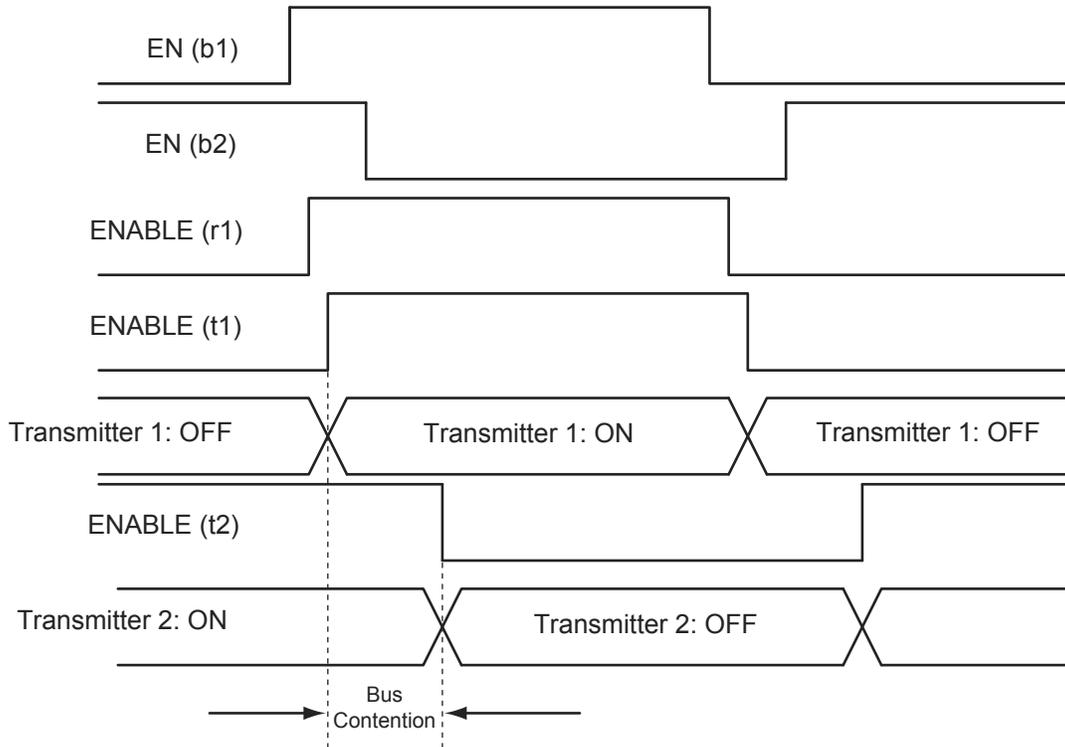
All IGLOO and ProASIC3 devices are tested to the Human Body Model (HBM) and the Charged Device Model (CDM).

Each I/O has two clamp diodes. One diode has its positive (P) side connected to the pad and its negative (N) side connected to VCCI. The second diode has its P side connected to GND and its N side connected to the pad. During operation, these diodes are normally biased in the off state, except when transient voltage is significantly above VCCI or below GND levels.

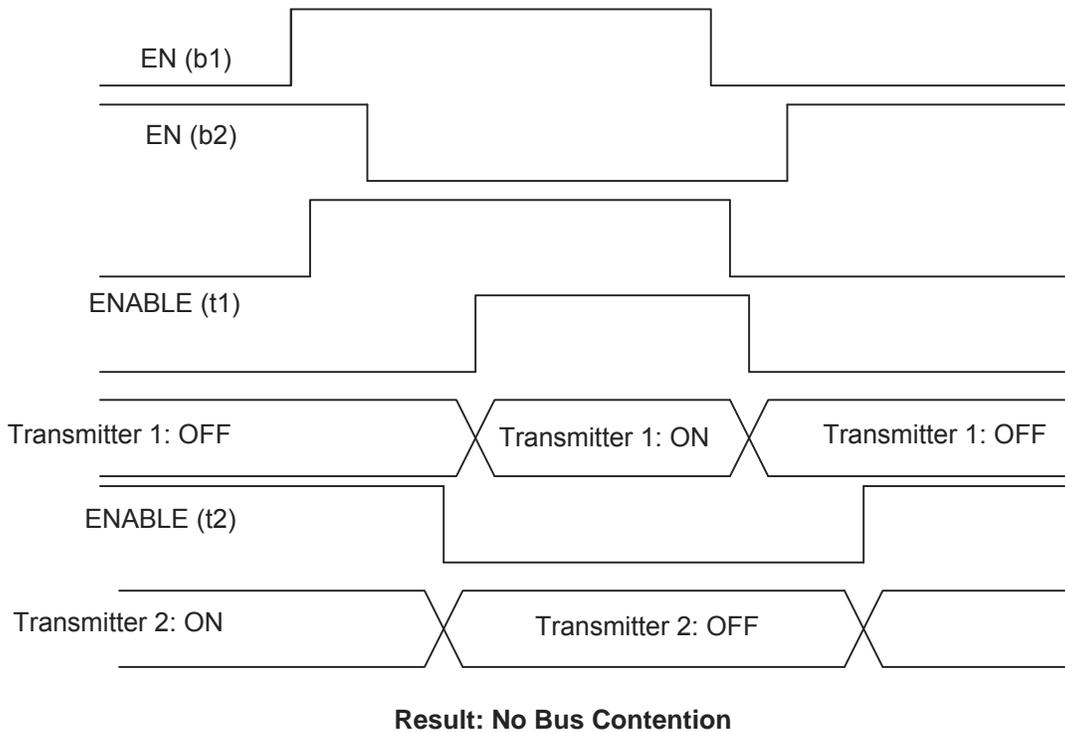
In 30K gate devices, the first diode is always off. In other devices, the clamp diode is always on and cannot be switched off.

By selecting the appropriate I/O configuration, the diode is turned on or off. Refer to Table 7-12 on page 193 for more information about the I/O standards and the clamp diode.

The second diode is always connected to the pad, regardless of the I/O configuration selected.



**Figure 8-18 • Timing Diagram (bypasses skew circuit)**



**Figure 8-19 • Timing Diagram (with skew circuit selected)**

## 9 – I/O Software Control in Low Power Flash Devices

Fusion, IGLOO, and ProASIC3 I/Os provide more design flexibility, allowing the user to control specific features by enabling certain I/O standards. Some features are selectable only for certain I/O standards, whereas others are available for all I/O standards. For example, slew control is not supported by differential I/O standards. Conversely, I/O register combining is supported by all I/O standards. For detailed information about which I/O standards and features are available on each device and each I/O type, refer to the I/O Structures section of the handbook for the device you are using.

Figure 9-1 shows the various points in the software design flow where a user can provide input or control of the I/O selection and parameters. A detailed description is provided throughout this document.

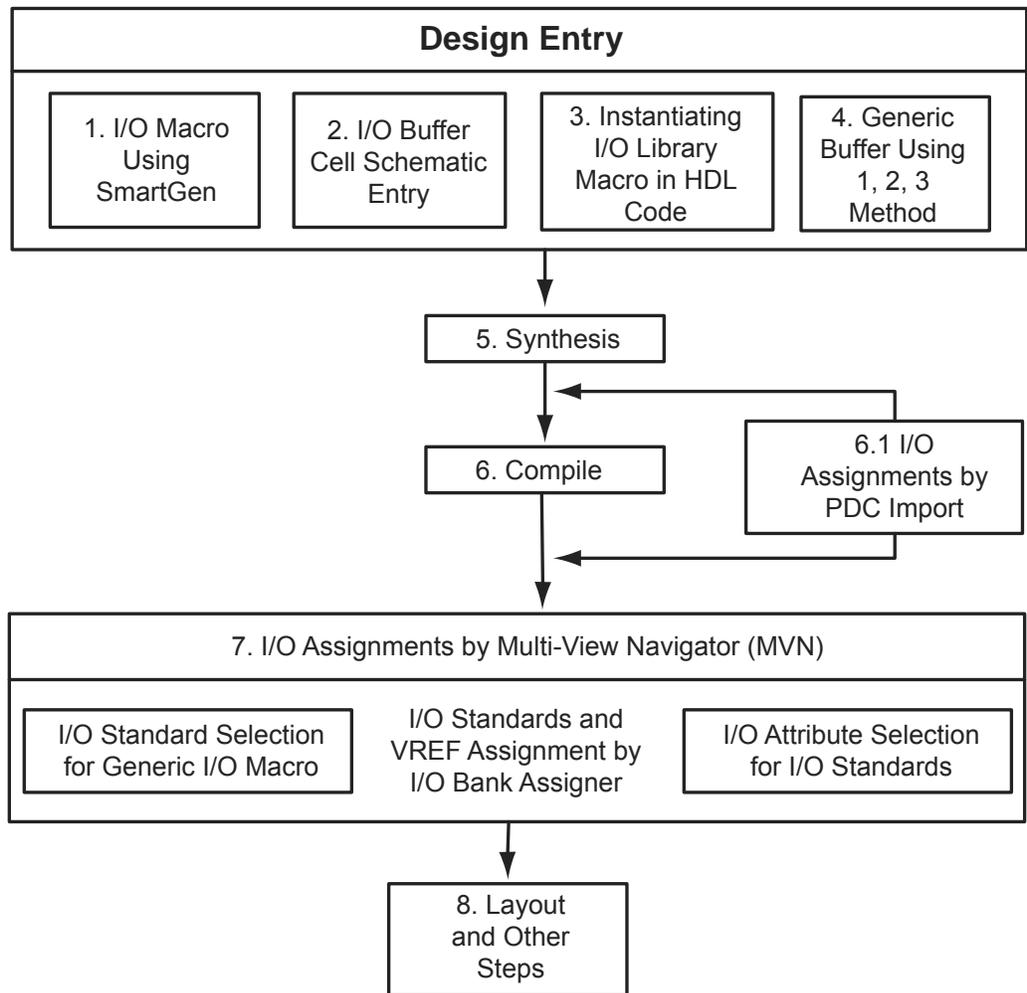


Figure 9-1 • User I/O Assignment Flow Chart

## Instantiating in HDL code

All the supported I/O macros can be instantiated in the top-level HDL code (refer to the *IGLOO*, *ProASIC3*, *SmartFusion*, and *Fusion Macro Library Guide* for a detailed list of all I/O macros). The following is an example:

```
library ieee;
use ieee.std_logic_1164.all;
library proasic3e;

entity TOP is
  port(IN2, IN1 : in std_logic; OUT1 : out std_logic);
end TOP;

architecture DEF_ARCH of TOP is

  component INBUF_LVCMOS5U
    port(PAD : in std_logic := 'U'; Y : out std_logic);
  end component;

  component INBUF_LVCMOS5
    port(PAD : in std_logic := 'U'; Y : out std_logic);
  end component;

  component OUTBUF_SSTL3_II
    port(D : in std_logic := 'U'; PAD : out std_logic);
  end component;

  Other component ....

  signal x, y, z.....other signals : std_logic;

begin

  I1 : INBUF_LVCMOS5U
    port map(PAD => IN1, Y =>x);
  I2 : INBUF_LVCMOS5
    port map(PAD => IN2, Y => y);
  I3 : OUTBUF_SSTL3_II
    port map(D => z, PAD => OUT1);

  other port mapping...

end DEF_ARCH;
```

## Synthesizing the Design

Libero SoC integrates with the Synplify® synthesis tool. Other synthesis tools can also be used with Libero SoC. Refer to the *Libero SoC User's Guide* or Libero online help for details on how to set up the Libero tool profile with synthesis tools from other vendors.

During synthesis, the following rules apply:

- Generic macros:
  - Users can instantiate generic INBUF, OUTBUF, TRIBUF, and BIBUF macros.
  - Synthesis will automatically infer generic I/O macros.
  - The default I/O technology for these macros is LVTTTL.
  - Users will need to use the I/O Attribute Editor in Designer to change the default I/O standard if needed (see Figure 9-6 on page 259).
- Technology-specific I/O macros:
  - Technology-specific I/O macros, such as INBUF\_LVCMO25 and OUTBUF\_GTL25, can be instantiated in the design. Synthesis will infer these I/O macros in the netlist.

- The I/O standard of technology-specific I/O macros cannot be changed in the I/O Attribute Editor (see Figure 9-6).
- The user MUST instantiate differential I/O macros (LVDS/LVPECL) in the design. This is the only way to use these standards in the design (IGLOO nano and ProASIC3 nano devices do not support differential inputs).
- To implement the DDR I/O function, the user must instantiate a DDR\_REG or DDR\_OUT macro. This is the only way to use a DDR macro in the design.

**Figure 9-6 • Assigning a Different I/O Standard to the Generic I/O Macro**

## Performing Place-and-Route on the Design

The netlist created by the synthesis tool should now be imported into Designer and compiled. During Compile, the user can specify the I/O placement and attributes by importing the PDC file. The user can also specify the I/O placement and attributes using ChipPlanner and the I/O Attribute Editor under MVN.

### Defining I/O Assignments in the PDC File

A PDC file is a Tcl script file specifying physical constraints. This file can be imported to and exported from Designer.

Table 9-3 shows I/O assignment constraints supported in the PDC file.

**Table 9-3 • PDC I/O Constraints**

Command	Action	Example	Comment
<b>I/O Banks Setting Constraints</b>			
set_iobank	Sets the I/O supply voltage, $V_{CCI}$ , and the input reference voltage, $V_{REF}$ , for the specified I/O bank.	<pre>set_iobank bankname [-vcci vcci_voltage] [-vref vref_voltage]  set_iobank Bank7 -vcci 1.50 -vref 0.75</pre>	Must use in case of mixed I/O voltage ( $V_{CCI}$ ) design
set_vref	Assigns a $V_{REF}$ pin to a bank.	<pre>set_vref -bank [bankname] [pinnum]  set_vref -bank Bank0 685 704 723 742 761</pre>	Must use if voltage-referenced I/Os are used
set_vref_defaults	Sets the default $V_{REF}$ pins for the specified bank. This command is ignored if the bank does not need a $V_{REF}$ pin.	<pre>set_vref_defaults bankname  set_vref_defaults bank2</pre>	

*Note: Refer to the Libero SoC User's Guide for detailed rules on PDC naming and syntax conventions.*

## Instantiating DDR Registers

Using SmartGen is the simplest way to generate the appropriate RTL files for use in the design. Figure 10-4 shows an example of using SmartGen to generate a DDR SSTL2 Class I input register. SmartGen provides the capability to generate all of the DDR I/O cells as described. The user, through the graphical user interface, can select from among the many supported I/O standards. The output formats supported are Verilog, VHDL, and EDIF.

Figure 10-5 on page 277 through Figure 10-8 on page 280 show the I/O cell configured for DDR using SSTL2 Class I technology. For each I/O standard, the I/O pad is buffered by a special primitive that indicates the I/O standard type.

---

---

**Figure 10-4 • Example of Using SmartGen to Generate a DDR SSTL2 Class I Input Register**

# General Flash Programming Information

## Programming Basics

When choosing a programming solution, there are a number of options available. This section provides a brief overview of those options. The next sections provide more detail on those options as they apply to Microsemi FPGAs.

### ***Reprogrammable or One-Time-Programmable (OTP)***

Depending on the technology chosen, devices may be reprogrammable or one-time-programmable. As the name implies, a reprogrammable device can be programmed many times. Generally, the contents of such a device will be completely overwritten when it is reprogrammed. All Microsemi flash devices are reprogrammable.

An OTP device is programmable one time only. Once programmed, no more changes can be made to the contents. Microsemi flash devices provide the option of disabling the reprogrammability for security purposes. This combines the convenience of reprogrammability during design verification with the security of an OTP technology for highly sensitive designs.

### ***Device Programmer or In-System Programming***

There are two fundamental ways to program an FPGA: using a device programmer or, if the technology permits, using in-system programming. A device programmer is a piece of equipment in a lab or on the production floor that is used for programming FPGA devices. The devices are placed into a socket mounted in a programming adapter module, and the appropriate electrical interface is applied. The programmed device can then be placed on the board. A typical programmer, used during development, programs a single device at a time and is referred to as a single-site engineering programmer.

With ISP, the device is already mounted onto the system printed circuit board when programming occurs. Typically, ISP programming is performed via a JTAG interface on the FPGA. The JTAG pins can be controlled either by an on-board resource, such as a microprocessor, or by an off-board programmer through a header connection. Once mounted, it can be programmed repeatedly and erased. If the application requires it, the system can be designed to reprogram itself using a microprocessor, without the use of any external programmer.

If multiple devices need to be programmed with the same program, various multi-site programming hardware is available in order to program many devices in parallel. Microsemi In House Programming is also available for this purpose.

## Programming Features for Microsemi Devices

### ***Flash Devices***

The flash devices supplied by Microsemi are reprogrammable by either a generic device programmer or ISP. Microsemi supports ISP using JTAG, which is supported by the FlashPro4 and FlashPro3, FlashPro Lite, Silicon Sculptor 3, and Silicon Sculptor II programmers.

Levels of ISP support vary depending on the device chosen:

- All SmartFusion, Fusion, IGLOO, and ProASIC3 devices support ISP.
- IGLOO, IGLOOe, IGLOO nano V5, and IGLOO PLUS devices can be programmed in-system when the device is using a 1.5 V supply voltage to the FPGA core.
- IGLOO nano V2 devices can be programmed at 1.2 V core voltage (when using FlashPro4 only) or 1.5 V. IGLOO nano V5 devices are programmed with a VCC core voltage of 1.5 V.

## Related Documents

Below is a list of related documents, their location on the Microsemi SoC Products Group website, and a brief summary of each document.

### Application Notes

*Programming Antifuse Devices*

[http://www.microsemi.com/soc/documents/AntifuseProgram\\_AN.pdf](http://www.microsemi.com/soc/documents/AntifuseProgram_AN.pdf)

*Implementation of Security in Actel's ProASIC and ProASIC<sup>PLUS</sup> Flash-Based FPGAs*

[http://www.microsemi.com/soc/documents/Flash\\_Security\\_AN.pdf](http://www.microsemi.com/soc/documents/Flash_Security_AN.pdf)

### User's Guides

#### **FlashPro Programmers**

FlashPro4,<sup>1</sup> FlashPro3, FlashPro Lite, and FlashPro<sup>2</sup>

[http://www.microsemi.com/soc/products/hardware/program\\_debug/flashpro/default.aspx](http://www.microsemi.com/soc/products/hardware/program_debug/flashpro/default.aspx)

*FlashPro User's Guide*

[http://www.microsemi.com/soc/documents/FlashPro\\_UG.pdf](http://www.microsemi.com/soc/documents/FlashPro_UG.pdf)

The FlashPro User's Guide includes hardware and software setup, self-test instructions, use instructions, and a troubleshooting / error message guide.

#### **Silicon Sculptor 3 and Silicon Sculptor II**

[http://www.microsemi.com/soc/products/hardware/program\\_debug/ss/default.aspx](http://www.microsemi.com/soc/products/hardware/program_debug/ss/default.aspx)

### Other Documents

<http://www.microsemi.com/soc/products/solutions/security/default.aspx#flashlock>

The security resource center describes security in Microsemi Flash FPGAs.

*Quality and Reliability Guide*

<http://www.microsemi.com/soc/documents/RelGuide.pdf>

*Programming and Functional Failure Guidelines*

[http://www.microsemi.com/soc/documents/FA\\_Policies\\_Guidelines\\_5-06-00002.pdf](http://www.microsemi.com/soc/documents/FA_Policies_Guidelines_5-06-00002.pdf)

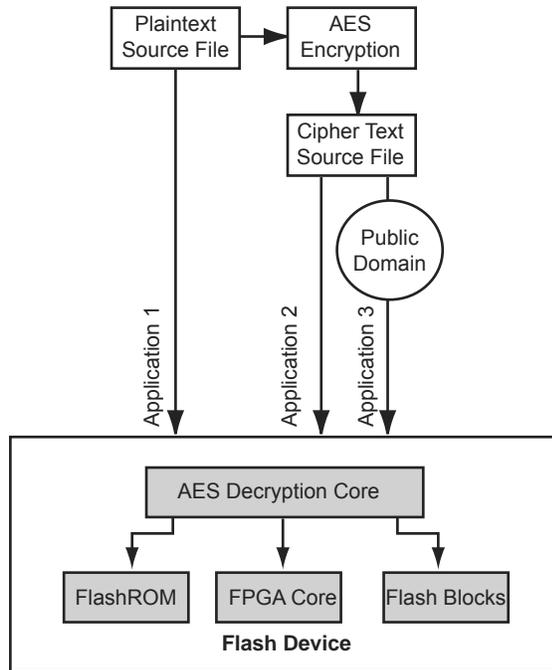
---

1. FlashPro4 replaced FlashPro3 in Q1 2010.  
2. FlashPro is no longer available.



## Security in Action

This section illustrates some applications of the security advantages of Microsemi's devices (Figure 12-6).



*Note: Flash blocks are only used in Fusion devices*

**Figure 12-6 • Security Options**

## FlashROM Security Use Models

Each of the subsequent sections describes in detail the available selections in Microsemi Designer as an aid to understanding security applications and generating appropriate programming files for those applications. Before proceeding, it is helpful to review Figure 12-7 on page 309, which gives a general overview of the programming file generation flow within the Designer software as well as what occurs during the device programming stage. Specific settings are discussed in the following sections.

In Figure 12-7 on page 309, the flow consists of two sub-flows. Sub-flow 1 describes programming security settings to the device only, and sub-flow 2 describes programming the design contents only.

In Application 1, described in the "Application 1: Trusted Environment" section on page 309, the user does not need to generate separate files but can generate one programming file containing both security settings and design contents. Then programming of the security settings and design contents is done in one step. Both sub-flow 1 and sub-flow 2 are used.

In Application 2, described in the "Application 2: Nontrusted Environment—Unsecured Location" section on page 309, the trusted site should follow sub-flows 1 and 2 separately to generate two separate programming files. The programming file from sub-flow 1 will be used at the trusted site to program the device(s) first. The programming file from sub-flow 2 will be sent off-site for production programming.

In Application 3, described in the "Application 3: Nontrusted Environment—Field Updates/Upgrades" section on page 310, typically only sub-flow 2 will be used, because only updates to the design content portion are needed and no security settings need to be changed.

In the event that update of the security settings is necessary, see the "Reprogramming Devices" section on page 321 for details. For more information on programming low power flash devices, refer to the "In-System Programming (ISP) of Microsemi's Low Power Flash Devices Using FlashPro4/3/3X" section on page 327.

## Related Documents

### User's Guides

*FlashPro User's Guide*

[http://www.microsemi.com/soc/documents/flashpro\\_ug.pdf](http://www.microsemi.com/soc/documents/flashpro_ug.pdf)

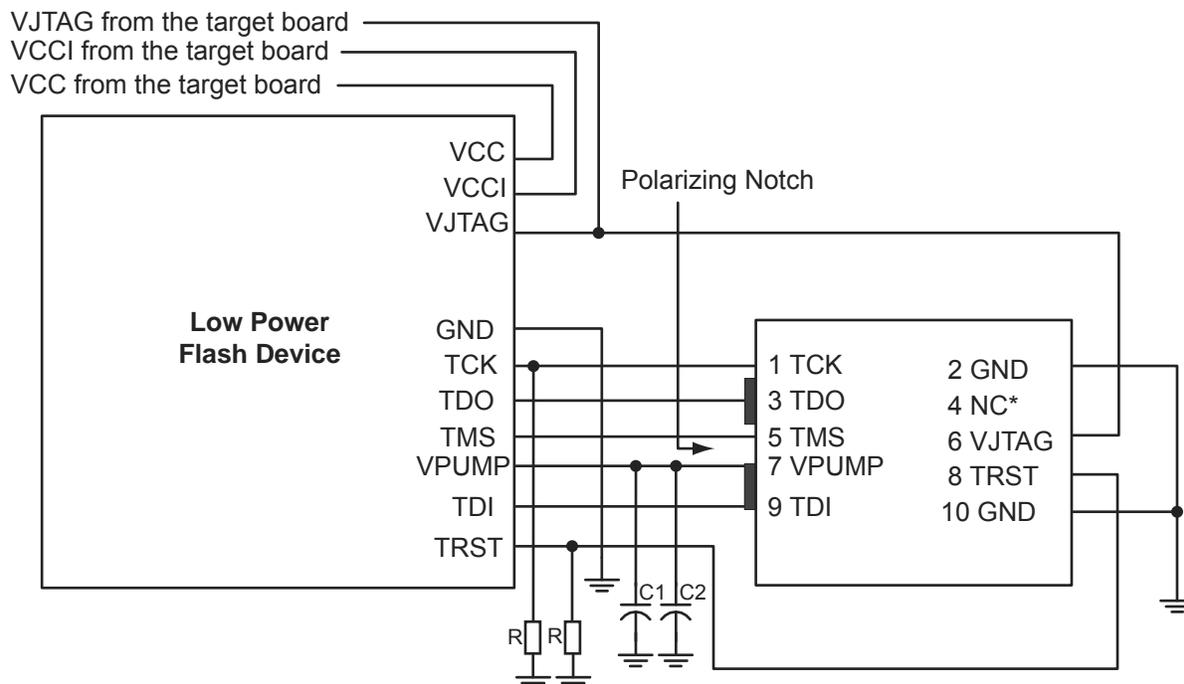
## List of Changes

The following table lists critical changes that were made in each revision of the chapter.

Date	Changes	Page
July 2010	This chapter is no longer published separately with its own part number and version but is now part of several FPGA fabric user's guides.	N/A
v1.5 (August 2009)	The "CoreMP7 Device Security" section was removed from "Security in ARM-Enabled Low Power Flash Devices", since M7-enabled devices are no longer supported.	304
v1.4 (December 2008)	IGLOO nano and ProASIC3 nano devices were added to Table 12-1 • Flash-Based FPGAs.	302
v1.3 (October 2008)	The "Security Support in Flash-Based Devices" section was revised to include new families and make the information more concise.	302
v1.2 (June 2008)	The following changes were made to the family descriptions in Table 12-1 • Flash-Based FPGAs: <ul style="list-style-type: none"> <li>• ProASIC3L was updated to include 1.5 V.</li> <li>• The number of PLLs for ProASIC3E was changed from five to six.</li> </ul>	302
v1.1 (March 2008)	The chapter was updated to include the IGLOO PLUS family and information regarding 15 k gate devices.	N/A
	The "IGLOO Terminology" section and "ProASIC3 Terminology" section are new.	302

## Board-Level Considerations

A bypass capacitor is required from VPUMP to GND for all low power flash devices during programming. This bypass capacitor protects the devices from voltage spikes that may occur on the VPUMP supplies during the erase and programming cycles. Refer to the "Pin Descriptions and Packaging" chapter of the appropriate device datasheet for specific recommendations. For proper programming, 0.01  $\mu\text{F}$  and 0.33  $\mu\text{F}$  capacitors (both rated at 16 V) are to be connected in parallel across VPUMP and GND, and positioned as close to the FPGA pins as possible. The bypass capacitor must be placed within 2.5 cm of the device pins.



*Note: \*NC (FlashPro3/3X); Prog\_Mode (FlashPro4). Prog\_Mode on FlashPro4 is an output signal that goes High during device programming and returns to Low when programming is complete. This signal can be used to drive a system to provide a 1.5 V programming signal to IGLOO nano, ProASIC3L, and RT ProASIC3 devices that can run with 1.2 V core voltage but require 1.5 V for programming. IGLOO nano V2 devices can be programmed at 1.2 V core voltage (when using FlashPro4 only), but IGLOO nano V5 devices are programmed with a VCC core voltage of 1.5 V.*

**Figure 13-6 • Board Layout and Programming Header Top View**

## Troubleshooting Signal Integrity

### Symptoms of a Signal Integrity Problem

A signal integrity problem can manifest itself in many ways. The problem may show up as extra or dropped bits during serial communication, changing the meaning of the communication. There is a normal variation of threshold voltage and frequency response between parts even from the same lot. Because of this, the effects of signal integrity may not always affect different devices on the same board in the same way. Sometimes, replacing a device appears to make signal integrity problems go away, but this is just masking the problem. Different parts on identical boards will exhibit the same problem sooner or later. It is important to fix signal integrity problems early. Unless the signal integrity problems are severe enough to completely block all communication between the device and the programmer, they may show up as subtle problems. Some of the FlashPro4/3/3X exit codes that are caused by signal integrity problems are listed below. Signal integrity problems are not the only possible cause of these

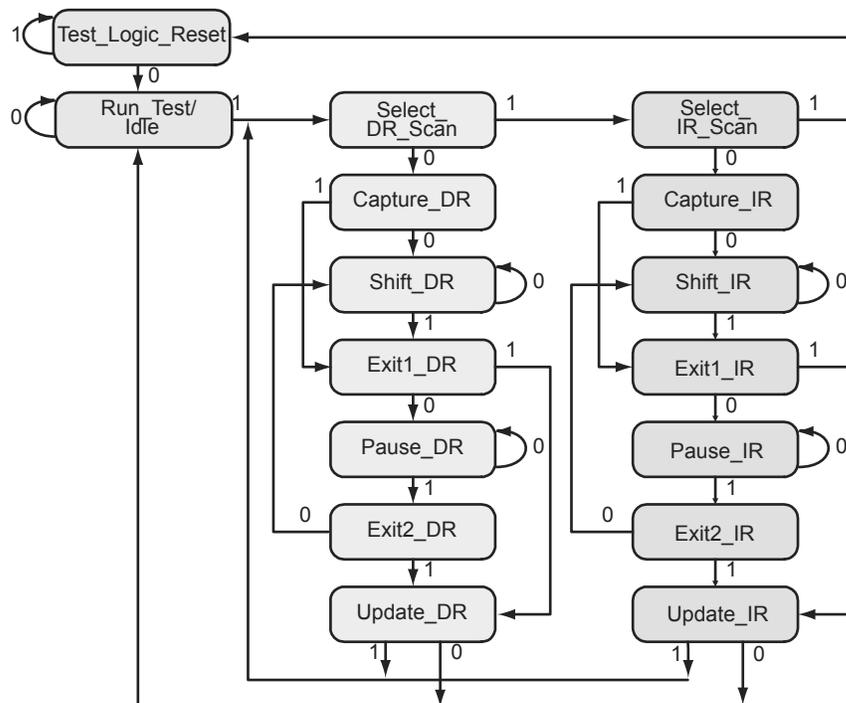


Figure 17-4 • TAP Controller State Diagram

## UJTAG Port Usage

UIREG[7:0] hold the contents of the JTAG instruction register. The UIREG vector value is updated when the TAP Controller state machine enters the Update\_IR state. Instructions 16 to 127 are user-defined and can be employed to encode multiple applications and commands within an application. Loading new instructions into the UIREG vector requires users to send appropriate logic to TMS to put the TAP Controller in a full IR cycle starting from the Select IR\_Scan state and ending with the Update\_IR state.

UTDI, UTDO, and UDRCK are directly connected to the JTAG TDI, TDO, and TCK ports, respectively. The TDI input can be used to provide either data (TAP Controller in the Shift\_DR state) or the new contents of the instruction register (TAP Controller in the Shift\_IR state).

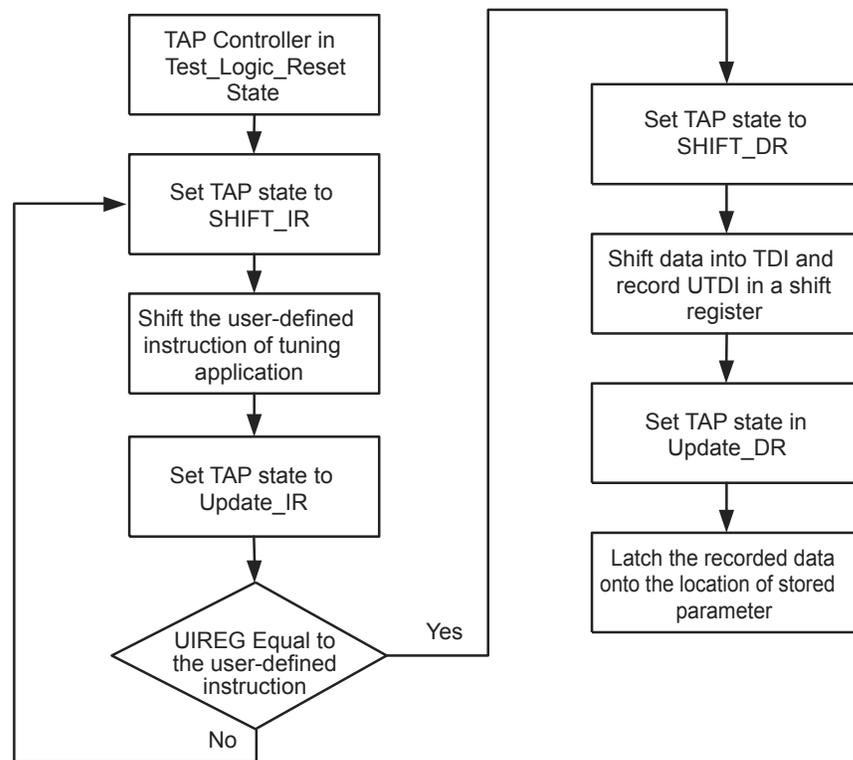
UDRSH, UDRUPD, and UDRCAP are HIGH when the TAP Controller state machine is in the Shift\_DR, Update\_DR, and Capture\_DR states, respectively. Therefore, they act as flags to indicate the stages of the data shift process. These flags are useful for applications in which blocks of data are shifted into the design from JTAG pins. For example, an active UDRSH can indicate that UTDI contains the data bitstream, and UDRUPD is a candidate for the end-of-data-stream flag.

As mentioned earlier, users should not connect the TDI, TDO, TCK, TMS, and TRST ports of the UJTAG macro to any port or net of the design netlist. The Designer software will automatically handle the port connection.

## Fine Tuning

In some applications, design constants or parameters need to be modified after programming the original design. The tuning process can be done using the UJTAG tile without reprogramming the device with new values. If the parameters or constants of a design are stored in distributed registers or embedded SRAM blocks, the new values can be shifted onto the JTAG TAP Controller pins, replacing the old values. The UJTAG tile is used as the “bridge” for data transfer between the JTAG pins and the FPGA VersaTiles or SRAM logic. Figure 17-5 shows a flow chart example for fine-tuning application steps using the UJTAG tile.

In Figure 17-5, the TMS signal sets the TAP Controller state machine to the appropriate states. The flow mainly consists of two steps: a) shifting the defined instruction and b) shifting the new data. If the target parameter is constantly used in the design, the new data can be shifted into a temporary shift register from UTDI. The UDRSH output of UJTAG can be used as a shift-enable signal, and UDRCK is the shift clock to the shift register. Once the shift process is completed and the TAP Controller state is moved to the Update\_DR state, the UDRUPD output of the UJTAG can latch the new parameter value from the temporary register into a permanent location. This avoids any interruption or malfunctioning during the serial shift of the new value.



**Figure 17-5 • Flow Chart Example of Fine-Tuning an Application Using UJTAG**