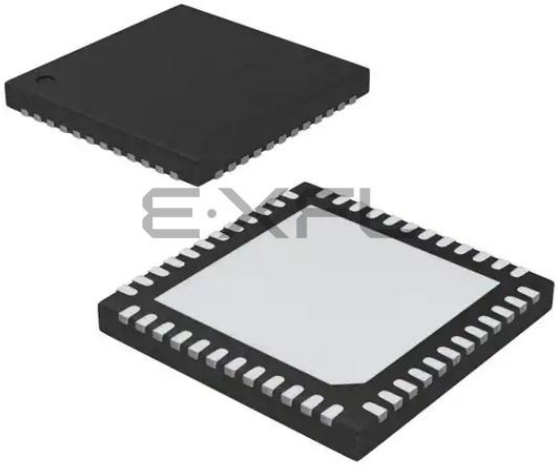

Welcome to E-XFL.COM

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	Active
Number of LABs/CLBs	-
Number of Logic Elements/Cells	-
Total RAM Bits	-
Number of I/O	34
Number of Gates	10000
Voltage - Supply	1.425V ~ 1.575V
Mounting Type	Surface Mount
Operating Temperature	-40°C ~ 100°C (TJ)
Package / Case	48-VFQFN Exposed Pad
Supplier Device Package	48-QFN (6x6)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/a3pn010-1qng48i

FPGA Array Architecture Support

The flash FPGAs listed in Table 1-1 support the architecture features described in this document.

***Table 1-1* • Flash-Based FPGAs**

Series	Family*	Description
IGLOO®	IGLOO	Ultra-low power 1.2 V to 1.5 V FPGAs with Flash*Freeze technology
	IGLOOe	Higher density IGLOO FPGAs with six PLLs and additional I/O standards
	IGLOO nano	The industry's lowest-power, smallest-size solution
	IGLOO PLUS	IGLOO FPGAs with enhanced I/O capabilities
ProASIC®3	ProASIC3	Low power, high-performance 1.5 V FPGAs
	ProASIC3E	Higher density ProASIC3 FPGAs with six PLLs and additional I/O standards
	ProASIC3 nano	Lowest-cost solution with enhanced I/O capabilities
	ProASIC3L	ProASIC3 FPGAs supporting 1.2 V to 1.5 V with Flash*Freeze technology
	RT ProASIC3	Radiation-tolerant RT3PE600L and RT3PE3000L
	Military ProASIC3/EL	Military temperature A3PE600L, A3P1000, and A3PE3000L
	Automotive ProASIC3	ProASIC3 FPGAs qualified for automotive applications
Fusion	Fusion	Mixed signal FPGA integrating ProASIC3 FPGA fabric, programmable analog block, support for ARM® Cortex™-M1 soft processors, and flash memory into a monolithic device

*Note: *The device names link to the appropriate datasheet, including product brief, DC and switching characteristics, and packaging information.*

IGLOO Terminology

In documentation, the terms IGLOO series and IGLOO devices refer to all of the IGLOO devices as listed in Table 1-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

ProASIC3 Terminology

In documentation, the terms ProASIC3 series and ProASIC3 devices refer to all of the ProASIC3 devices as listed in Table 1-1. Where the information applies to only one product line or limited devices, these exclusions will be explicitly stated.

To further understand the differences between the IGLOO and ProASIC3 devices, refer to the *Industry's Lowest Power FPGAs Portfolio*.

standard for CLKBUF is LVTTL in the current Microsemi Libero® System-on-Chip (SoC) and Designer software.

***Table 3-9* • I/O Standards within CLKBUF**

Name	Description
CLKBUF_LVCMOS5	LVCMOS clock buffer with 5.0 V CMOS voltage level
CLKBUF_LVCMOS33	LVCMOS clock buffer with 3.3 V CMOS voltage level
CLKBUF_LVCMOS25	LVCMOS clock buffer with 2.5 V CMOS voltage level[1]
CLKBUF_LVCMOS18	LVCMOS clock buffer with 1.8 V CMOS voltage level
CLKBUF_LVCMOS15	LVCMOS clock buffer with 1.5 V CMOS voltage level
CLKBUF_LVCMOS12	LVCMOS clock buffer with 1.2 V CMOS voltage level
CLKBUF_PCI	PCI clock buffer
CLKBUF_PCIX	PCIX clock buffer
CLKBUF_GTL25	GTL clock buffer with 2.5 V CMOS voltage level[1]
CLKBUF_GTL33	GTL clock buffer with 3.3 V CMOS voltage level[1]
CLKBUF_GTLP25	GTL+ clock buffer with 2.5 V CMOS voltage level[1]
CLKBUF_GTLP33	GTL+ clock buffer with 3.3 V CMOS voltage level[1]
CLKBUF_ HSTL _I	HSTL Class I clock buffer[1]
CLKBUF_ HSTL _II	HSTL Class II clock buffer[1]
CLKBUF_SSTL2_I	SSTL2 Class I clock buffer[1]
CLKBUF_SSTL2_II	SSTL2 Class II clock buffer[1]
CLKBUF_SSTL3_I	SSTL3 Class I clock buffer[1]
CLKBUF_SSTL3_II	SSTL3 Class II clock buffer[1]

Notes:

1. *Supported in only the IGLOOe, ProASIC3E, AFS600, and AFS1500 devices*
2. *By default, the CLKBUF macro uses the 3.3 V LVTTL I/O technology.*

The current synthesis tool libraries only infer the CLKBUF or CLKINT macros in the netlist. All other global macros must be instantiated manually into your HDL code. The following is an example of CLKBUF_LVCMOS25 global macro instantiations that you can copy and paste into your code:

VHDL

```
component clkbuf_lvcmos25
  port (pad : in std_logic; y : out std_logic);
end component

begin
-- concurrent statements
u2 : clkbuf_lvcmos25 port map (pad =>  ext_clk, y => int_clk);
end
```

Verilog

```
module design (______);

input _____;
output ______;

clkbuf_lvcmos25 u2 (.y(int_clk), .pad(ext_clk);

endmodule
```

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 35 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 51 for details.

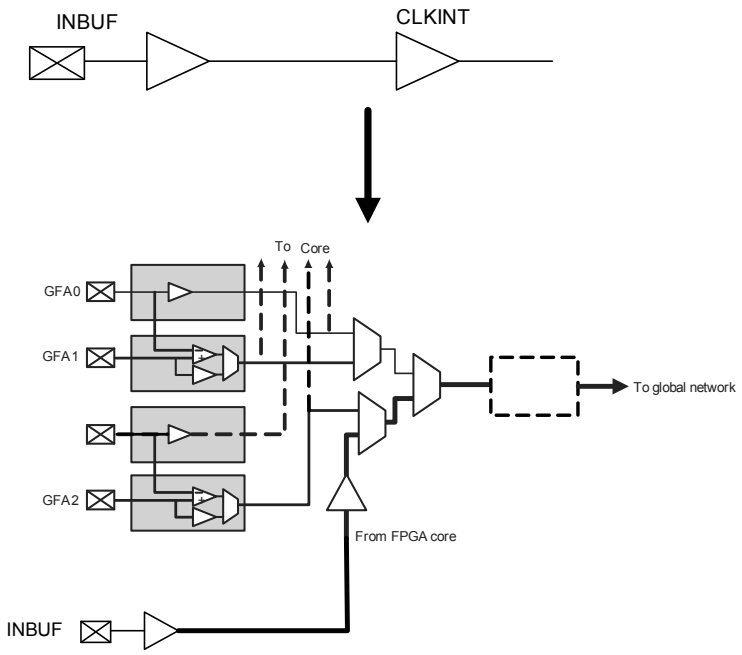

Figure 3-14 • **CLKINT Macro**

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.

Step 1

Run Synthesis with default options. The Synplicity log shows the following device utilization:

Cell usage:

	cell count	area	count*area
DFN1E1C1	1536	2.0	3072.0
BUFF	278	1.0	278.0
INBUF	10	0.0	0.0
VCC	9	0.0	0.0
GND	9	0.0	0.0
OUTBUF	6	0.0	0.0
CLKBUF	3	0.0	0.0
PLL	2	0.0	0.0
TOTAL	1853		3350.0

Step 2

Run Compile with the **Promote regular nets whose fanout is greater than** option selected in Designer; you will see the following in the Compile report:

```
Device utilization report:
==========================
CORE              Used:   1536  Total:  13824   (11.11%)
IO (W/ clocks)    Used:     19  Total:    147   (12.93%)
Differential IO   Used:      0  Total:     65    (0.00%)
GLOBAL            Used:      8  Total:     18   (44.44%)
PLL               Used:      2  Total:      2  (100.00%)
RAM/FIFO          Used:      0  Total:     24    (0.00%)
FlashROM          Used:      0  Total:      1    (0.00%)
……….
The following nets have been assigned to a global resource:
Fanout  Type          Name
-------------------------
1536    INT_NET       Net   : EN_ALL_c
                      Driver: EN_ALL_pad_CLKINT
                      Source: AUTO PROMOTED
1536    SET/RESET_NET Net   : ACLR_c
                      Driver: ACLR_pad_CLKINT
                      Source: AUTO PROMOTED
256     CLK_NET       Net   : QCLK1_c
                      Driver: QCLK1_pad_CLKINT
                      Source: AUTO PROMOTED
256     CLK_NET       Net   : QCLK2_c
                      Driver: QCLK2_pad_CLKINT
                      Source: AUTO PROMOTED
256     CLK_NET       Net   : QCLK3_c
                      Driver: QCLK3_pad_CLKINT
                      Source: AUTO PROMOTED
256     CLK_NET       Net   : $1N14
                      Driver: $1I5/Core
                      Source: ESSENTIAL
256     CLK_NET       Net   : $1N12
                      Driver: $1I6/Core
                      Source: ESSENTIAL
256     CLK_NET       Net   : $1N10
                      Driver: $1I6/Core
                      Source: ESSENTIAL
```

Designer will promote five more signals to global due to high fanout. There are eight signals assigned to global networks.

Each CCC can implement up to three independent global buffers (with or without programmable delay) or a PLL function (programmable frequency division/multiplication, phase shift, and delays) with up to three global outputs. Unused global outputs of a PLL can be used to implement independent global buffers, up to a maximum of three global outputs for a given CCC.

CCC Programming

The CCC block is fully configurable, either via flash configuration bits set in the programming bitstream or through an asynchronous interface. This asynchronous dedicated shift register interface is dynamically accessible from inside the low power flash devices to permit parameter changes, such as PLL divide ratios and delays, during device operation.

To increase the versatility and flexibility of the clock conditioning system, the CCC configuration is determined either by the user during the design process, with configuration data being stored in flash memory as part of the device programming procedure, or by writing data into a dedicated shift register during normal device operation.

This latter mode allows the user to dynamically reconfigure the CCC without the need for core programming. The shift register is accessed through a simple serial interface. Refer to the "UJTAG Applications in Microsemi's Low Power Flash Devices" section on page 297 or the application note *Using Global Resources in Actel Fusion Devices*.

Global Resources

Low power flash and mixed signal devices provide three global routing networks (GLA, GLB, and GLC) for each of the CCC locations. There are potentially many I/O locations; each global I/O location can be chosen from only one of three possibilities. This is controlled by the multiplexer tree circuitry in each global network. Once the I/O location is selected, the user has the option to utilize the CCCs before the signals are connected to the global networks. The CCC in each location (up to six) has the same structure, so generating the CCC macros is always done with an identical software GUI. The CCCs in the corner locations drive the quadrant global networks, and the CCCs in the middle of the east and west chip sides drive the chip global networks. The quadrant global networks span only a quarter of the device, while the chip global networks span the entire device. For more details on global resources offered in low power flash devices, refer to the "Global Resources in Low Power Flash Devices" section on page 31.

A global buffer can be placed in any of the three global locations (CLKA-GLA, CLKB-GLB, or CLKC-GLC) of a given CCC. A PLL macro uses the CLKA CCC input to drive its reference clock. It uses the GLA and, optionally, the GLB and GLC global outputs to drive the global networks. A PLL macro can also drive the YB and YC regular core outputs. The GLB (or GLC) global output cannot be reused if the YB (or YC) output is used. Refer to the "PLL Macro Signal Descriptions" section on page 68 for more information.

Each global buffer, as well as the PLL reference clock, can be driven from one of the following:

- 3 dedicated single-ended I/Os using a hardwired connection
- 2 dedicated differential I/Os using a hardwired connection (not supported for IGLOO nano or ProASIC3 nano devices)
- The FPGA core

PLL Core Specifications

PLL core specifications can be found in the DC and Switching Characteristics chapter of the appropriate family datasheet.

Loop Bandwidth

Common design practice for systems with a low-noise input clock is to have PLLs with small loop bandwidths to reduce the effects of noise sources at the output. Table 4-6 shows the PLL loop bandwidth, providing a measure of the PLL's ability to track the input clock and jitter.

Table 4-6 • **–3 dB Frequency of the PLL**

	Minimum (T_a = +125°C, VCCA = 1.4 V)	**Typical (T_a = +25°C, VCCA = 1.5 V)**	**Maximum (T_a = –55°C, VCCA = 1.6 V)**
–3 dB Frequency	15 kHz	25 kHz	45 kHz

PLL Core Operating Principles

This section briefly describes the basic principles of PLL operation. The PLL core is composed of a phase detector (PD), a low-pass filter (LPF), and a four-phase voltage-controlled oscillator (VCO). Figure 4-19 illustrates a basic single-phase PLL core with a divider and delay in the feedback path.

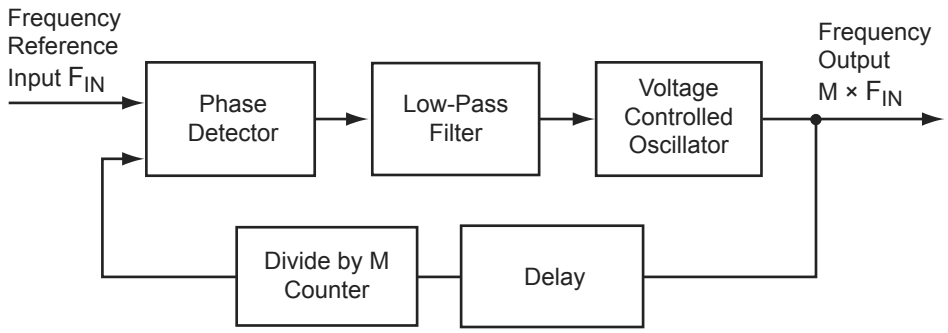

Figure 4-19 • **Simplified PLL Core with Feedback Divider and Delay**

The PLL is an electronic servo loop that phase-aligns the PD feedback signal with the reference input. To achieve this, the PLL dynamically adjusts the VCO output signal according to the average phase difference between the input and feedback signals.

The first element is the PD, which produces a voltage proportional to the phase difference between its inputs. A simple example of a digital phase detector is an Exclusive-OR gate. The second element, the LPF, extracts the average voltage from the phase detector and applies it to the VCO. This applied voltage alters the resonant frequency of the VCO, thus adjusting its output frequency.

Consider Figure 4-19 with the feedback path bypassing the divider and delay elements. If the LPF steadily applies a voltage to the VCO such that the output frequency is identical to the input frequency, this steady-state condition is known as lock. Note that the input and output phases are also identical. The PLL core sets a LOCK output signal HIGH to indicate this condition.

Should the input frequency increase slightly, the PD detects the frequency/phase difference between its reference and feedback input signals. Since the PD output is proportional to the phase difference, the change causes the output from the LPF to increase. This voltage change increases the resonant frequency of the VCO and increases the feedback frequency as a result. The PLL dynamically adjusts in this manner until the PD senses two phase-identical signals and steady-state lock is achieved. The opposite (decreasing PD output signal) occurs when the input frequency decreases.

Now suppose the feedback divider is inserted in the feedback path. As the division factor M (shown in Figure 4-20 on page 85) is increased, the average phase difference increases. The average phase

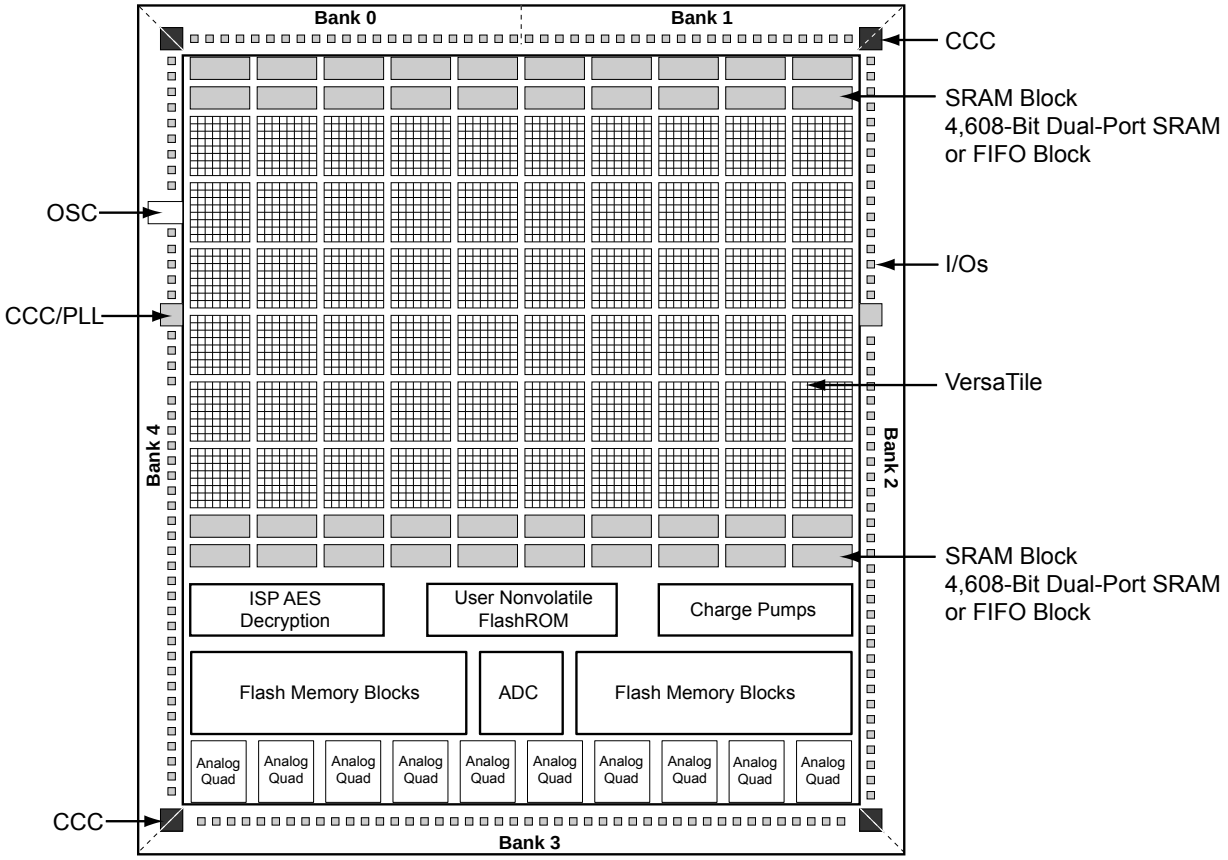

***Figure 5-2* • Fusion Device Architecture Overview (AFS600)**

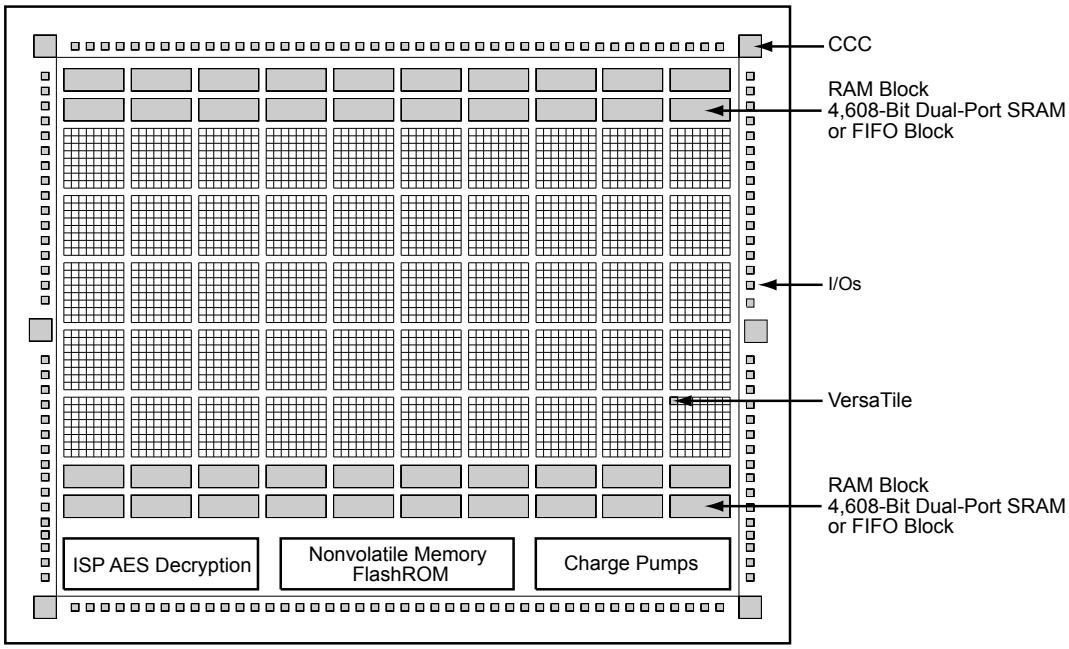

***Figure 5-3* • ProASIC3 and IGLOO Device Architecture**

Simulation of FlashROM Design

The MEM file has 128 rows of 8 bits, each representing the contents of the FlashROM used for simulation. For example, the first row represents page 0, byte 0; the next row is page 0, byte 1; and so the pattern continues. Note that the three MSBs of the address define the page number, and the four LSBs define the byte number. So, if you send address 0000100 to FlashROM, this corresponds to the page 0 and byte 4 location, which is the fifth row in the MEM file. SmartGen defaults to 0s for any unspecified location of the FlashROM. Besides using the MEM file generated by SmartGen, you can create a binary file with 128 rows of 8 bits each and use this as a MEM file. Microsemi recommends that you use different file names if you plan to generate multiple MEM files. During simulation, Libero SoC passes the MEM file used as the generic file in the netlist, along with the design files and testbench. If you want to use different MEM files during simulation, you need to modify the generic file reference in the netlist.

```
……..
UFROM0: UFROM
--generic map(MEMORYFILE => "F:\Appsnotes\FROM\test_designs\testa\smartgen\FROM_a.mem")
--generic map(MEMORYFILE => "F:\Appsnotes\FROM\test_designs\testa\smartgen\FROM_b.mem")
……..
```

The VITAL and Verilog simulation models accept the generics passed by the netlist, read the MEM file, and perform simulation with the data in the file.

Programming File Generation for FlashROM Design

FlashPoint is the programming software used to generate the programming files for flash devices. Depending on the applications, you can use the FlashPoint software to generate a STAPL file with different FlashROM contents. In each case, optional AES decryption is available. To generate a STAPL file that contains the same FPGA core content and different FlashROM contents, the FlashPoint software needs an Array Map file for the core and UFC file(s) for the FlashROM. This final STAPL file represents the combination of the logic of the FPGA core and FlashROM content.

FlashPoint generates the STAPL files you can use to program the desired FlashROM page and/or FPGA core of the FPGA device contents. FlashPoint supports the encryption of the FlashROM content and/or FPGA Array configuration data. In the case of using the FlashROM for device serialization, a sequence of unique FlashROM contents will be generated. When generating a programming file with multiple unique FlashROM contents, you can specify in FlashPoint whether to include all FlashROM content in a single STAPL file or generate a different STAPL file for each FlashROM (Figure 5-11). The programming software (FlashPro) handles the single STAPL file that contains the FlashROM content from multiple devices. It enables you to program the FlashROM content into a series of devices sequentially (Figure 5-11). See the *FlashPro User's Guide* for information on serial programming.

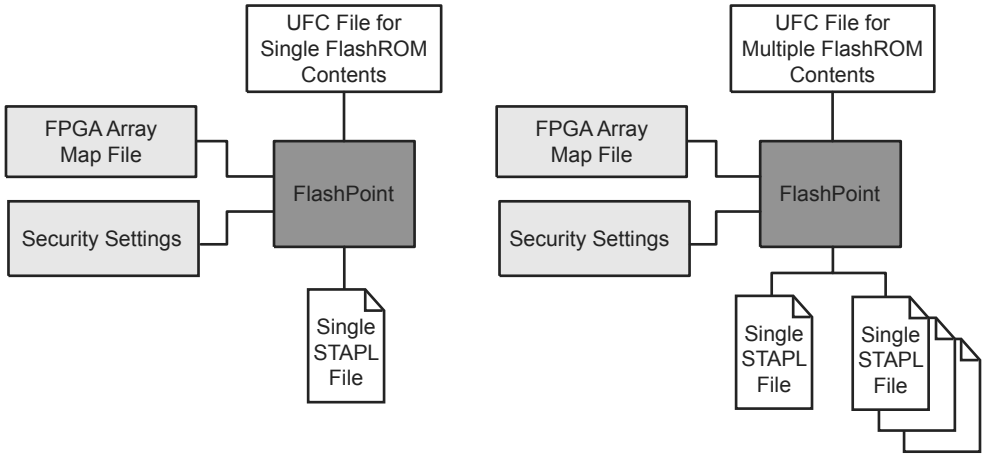

***Figure 5-11* • Single or Multiple Programming File Generation**

without reprogramming the device. Dynamic flag settings are determined by register values and can be altered without reprogramming the device by reloading the register values either from the design or through the UJTAG interface described in the "Initializing the RAM/FIFO" section on page 148.

SmartGen can also configure the FIFO to continue counting after the FIFO is full. In this configuration, the FIFO write counter will wrap after the counter is full and continue to write data. With the FIFO configured to continue to read after the FIFO is empty, the read counter will also wrap and re-read data that was previously read. This mode can be used to continually read back repeating data patterns stored in the FIFO (Figure 6-15).

***Figure 6-15* • SmartGen FIFO Configuration Interface**

FIFOs configured using SmartGen can also make use of the port mapping feature to configure the names of the ports.

Limitations

Users should be aware of the following limitations when configuring SRAM blocks for low power flash devices:

- SmartGen does not track the target device in a family, so it cannot determine if a configured memory block will fit in the target device.
- Dual-port RAMs with different read and write aspect ratios are not supported.
- Cascaded memory blocks can only use a maximum of 64 blocks of RAM.
- The Full flag of the FIFO is sensitive to the maximum depth of the actual physical FIFO block, not the depth requested in the SmartGen interface.

Solution 2

This solution requires one board resistor and one Zener 3.3 V diode, as demonstrated in Figure 7-6.

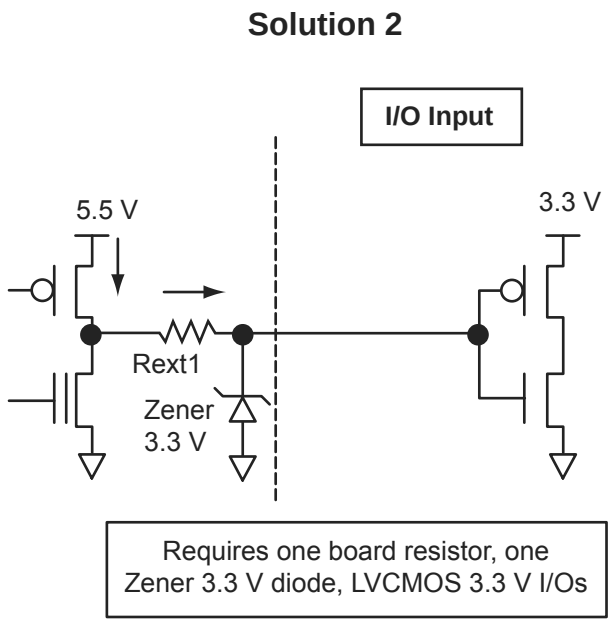

Figure 7-6 • **Solution 2**

Solution 3

This solution requires a bus switch on the board, as demonstrated in Figure 7-7.

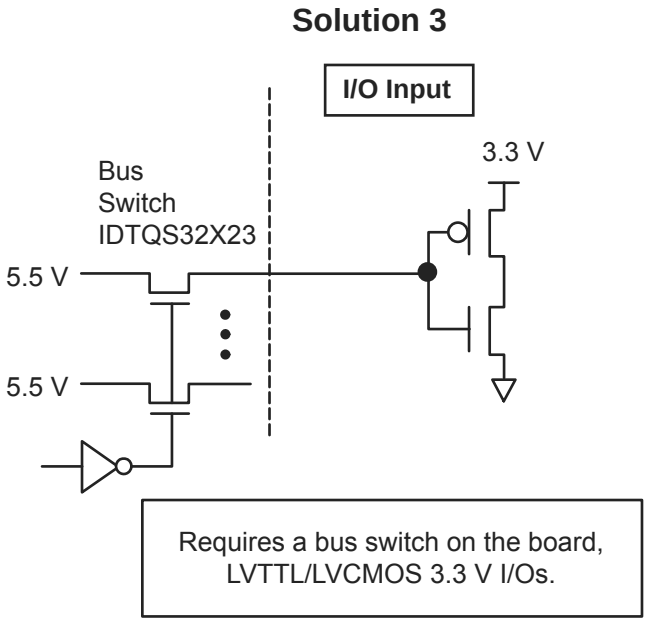

Figure 7-7 • **Solution 3**

I/O Software Support

In Microsemi's Libero software, default settings have been defined for the various I/O standards supported. Changes can be made to the default settings via the use of attributes; however, not all I/O attributes are applicable for all I/O standards.

***Table 7-15* • nano I/O Attributes vs. I/O Standard Applications**

I/O Standard	SLEW (output only)	OUT_DRIVE (output only)	RES_PULL	OUT_LOAD (output only)		Schmitt Trigger	Hold State	Combine Register
				IGLOO nano	ProASIC 3 nano			
LVTTL/ LVCMOS3.3	✓	✓ (8)	✓	✓	✓	✓	✓	✓
LVCMOS2.5	✓	✓ (8)	✓	✓	✓	✓	✓	✓
LVCMOS1.8	✓	✓ (4)	✓	✓	✓	✓	✓	✓
LVCMOS1.5	✓	✓ (2)	✓	✓	✓	✓	✓	✓
LVCMOS1.2	✓	✓ (2)	✓	✓	–	✓	✓	✓
Software Defaults	HIGH	Refer to numbers in parentheses in above cells.	None	All Devices: 5 pF	10 pF or 35 pF*	Off	Off	No

*Note: *10 pF for A3PN010, A3PN015, and A3PN020; 35 pF for A3PN060, A3PN125, and A3PN250.*

***Figure 9-11* • DDR Input/Output Cells as Seen by ChipPlanner for IGLOO/e Devices**

Verilog

```
module Inbuf_ddr(PAD,CLR,CLK,QR,QF);

input PAD, CLR, CLK;
output  QR, QF;

wire Y;

  DDR_REG DDR_REG_0_inst(.D(Y), .CLK(CLK), .CLR(CLR), .QR(QR), .QF(QF));
  INBUF INBUF_0_inst(.PAD(PAD), .Y(Y));

endmodule

module Outbuf_ddr(DataR,DataF,CLR,CLK,PAD);

input DataR, DataF, CLR, CLK;
output  PAD;

wire Q, VCC;

  VCC VCC_1_net(.Y(VCC));
  DDR_OUT DDR_OUT_0_inst(.DR(DataR), .DF(DataF), .CLK(CLK), .CLR(CLR), .Q(Q));
  OUTBUF OUTBUF_0_inst(.D(Q), .PAD(PAD));

endmodule
```

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
	Notes were added where appropriate to point out that IGLOO nano and ProASIC3 nano devices do not support differential inputs (SAR 21449).	N/A
v1.4 (December 2008)	IGLOO nano and ProASIC3 nano devices were added to Table 9-1 • Flash-Based FPGAs.	206
	The "I/O Cell Architecture" section was updated with information applicable to nano devices.	207
	The output buffer (OUTBUF_SSTL3_I) input was changed to D, instead of Q, in Figure 9-1 • DDR Support in Low Power Flash Devices, Figure 9-3 • DDR Output Register (SSTL3 Class I), Figure 9-6 • DDR Output Register (SSTL3 Class I), Figure 9-7 • DDR Tristate Output Register, LOW Enable, 8 mA, Pull-Up (LVTTL), and the output from the DDR_OUT macro was connected to the input of the TRIBUFF macro in Figure 9-7 • DDR Tristate Output Register, LOW Enable, 8 mA, Pull-Up (LVTTL).	205, 209, 212, 213
v1.3 (October 2008)	The "Double Data Rate (DDR) Architecture" section was updated to include mention of the AFS600 and AFS1500 devices.	205
	The "DDR Support in Flash-Based Devices" section was revised to include new families and make the information more concise.	206
v1.2 (June 2008)	The following changes were made to the family descriptions in Table 9-1 • Flash-Based FPGAs: • ProASIC3L was updated to include 1.5 V. • The number of PLLs for ProASIC3E was changed from five to six.	206
v1.1 (March 2008)	The "IGLOO Terminology" section and "ProASIC3 Terminology" section are new.	206

Date	Changes	Page
v1.3 (December 2008)	The "Programming Support in Flash Devices" section was updated to include IGLOO nano and ProASIC3 nano devices.	222
	The "Flash Devices" section was updated to include information for IGLOO nano devices. The following sentence was added: IGLOO PLUS devices can also be operated at any voltage between 1.2 V and 1.5 V; the Designer software allows 50 mV increments in the voltage.	223
	Table 10-4 · Programming Ordering Codes was updated to replace FP3-26PIN-ADAPTER with FP3-10PIN-ADAPTER-KIT.	228
	Table 14-6 · Programmer Device Support was updated to add IGLOO nano and ProASIC3 nano devices. AGL400 was added to the IGLOO portion of the table.	317
v1.2 (October 2008)	The "Programming Support in Flash Devices" section was revised to include new families and make the information more concise.	222
	Figure 10-1 · FlashPro Programming Setup and the "Programming Support in Flash Devices" section are new.	221, 222
	Table 14-6 · Programmer Device Support was updated to include A3PE600L with the other ProASIC3L devices, and the RT ProASIC3 family was added.	317
v1.1 (March 2008)	The "Flash Devices" section was updated to include the IGLOO PLUS family. The text, "Voltage switching is required in-system to switch from a 1.2 V core to 1.5 V core for programming," was revised to state, "Although the device can operate at 1.2 V core voltage, the device can only be reprogrammed when the core voltage is 1.5 V. Voltage switching is required in-system to switch from a 1.2 V supply (V_{CC}, V_{CCI}, and V_{JTAG}) to 1.5 V for programming."	223
	The ProASIC3L family was added to Table 14-6 · Programmer Device Support as a separate set of rows rather than combined with ProASIC3 and ProASIC3E devices. The IGLOO PLUS family was included, and AGL015 and A3P015 were added.	317

Figure 12-2 shows different applications for ISP programming.

1. In a trusted programming environment, you can program the device using the unencrypted (plaintext) programming file.
2. You can program the AES Key in a trusted programming environment and finish the final programming in an untrusted environment using the AES-encrypted (cipher text) programming file.
3. For the remote ISP updating/reprogramming, the AES Key stored in the device enables the encrypted programming bitstream to be transmitted through the untrusted network connection.

Microsemi low power flash devices also provide the unique Microsemi FlashLock feature, which protects the Pass Key and AES Key. Unless the original FlashLock Pass Key is used to unlock the device, security settings cannot be modified. Microsemi does not support read-back of FPGA core-programmed data; however, the FlashROM contents can selectively be read back (or disabled) via the JTAG port based on the security settings established by the Microsemi Designer software. Refer to the "Security in Low Power Flash Devices" section on page 235 for more information.

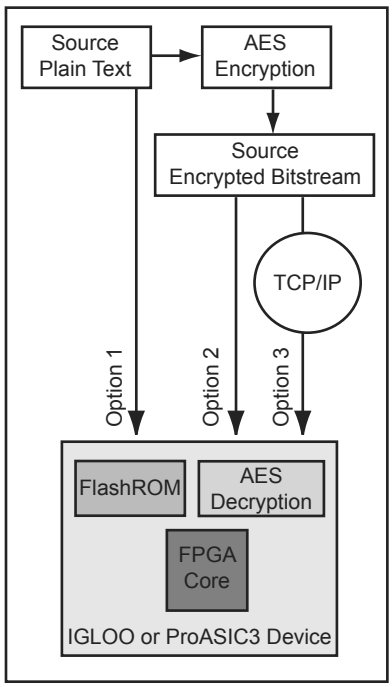

***Figure 12-2* • Different ISP Use Models**

STAPL vs. DirectC

Programming the low power flash devices is performed using DirectC or the STAPL player. Both tools use the STAPL file as an input. DirectC is a compiled language, whereas STAPL is an interpreted language. Microprocessors will be able to load the FPGA using DirectC much more quickly than STAPL. This speed advantage becomes more apparent when lower clock speeds of 8- or 16-bit microprocessors are used. DirectC also requires less memory than STAPL, since the programming algorithm is directly implemented. STAPL does have one advantage over DirectC—the ability to upgrade. When a new programming algorithm is required, the STAPL user simply needs to regenerate a STAPL file using the latest version of the Designer software and download it to the system. The DirectC user must download the latest version of DirectC from Microsemi, compile everything, and download the result into the system (Figure 14-4).

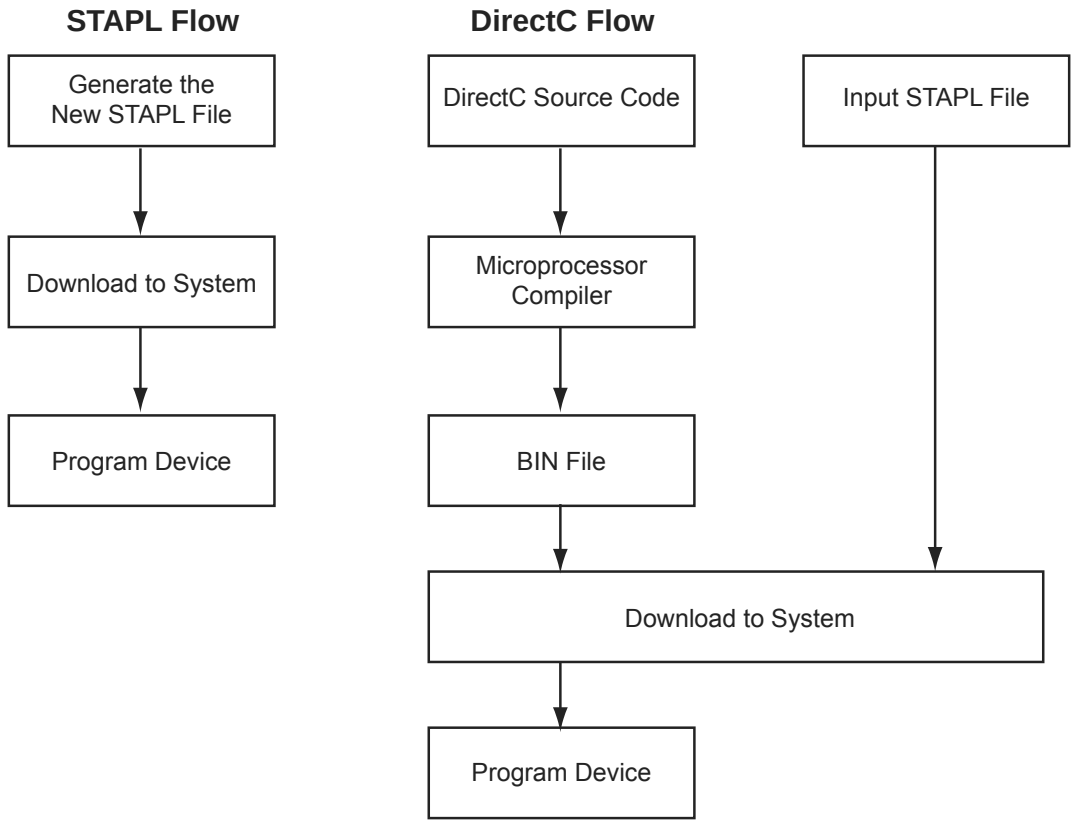

***Figure 14-4* • STAPL vs. DirectC**

16 – UJTAG Applications in Microsemi's Low Power Flash Devices

Introduction

In Fusion, IGLOO, and ProASIC3 devices, there is bidirectional access from the JTAG port to the core VersaTiles during normal operation of the device (Figure 16-1). User JTAG (UJTAG) is the ability for the design to use the JTAG ports for access to the device for updates, etc. While regular JTAG is used, the UJTAG tiles, located at the southeast area of the die, are directly connected to the JTAG Test Access Port (TAP) Controller in normal operating mode. As a result, all the functional blocks of the device, such as Clock Conditioning Circuits (CCCs) with PLLs, SRAM blocks, embedded FlashROM, flash memory blocks, and I/O tiles, can be reached via the JTAG ports. The UJTAG functionality is available by instantiating the UJTAG macro directly in the source code of a design. Access to the FPGA core VersaTiles from the JTAG ports enables users to implement different applications using the TAP Controller (JTAG port). This document introduces the UJTAG tile functionality and discusses a few application examples. However, the possible applications are not limited to what is presented in this document. UJTAG can serve different purposes in many designs as an elementary or auxiliary part of the design. For detailed usage information, refer to the "Boundary Scan in Low Power Flash Devices" section on page 291.

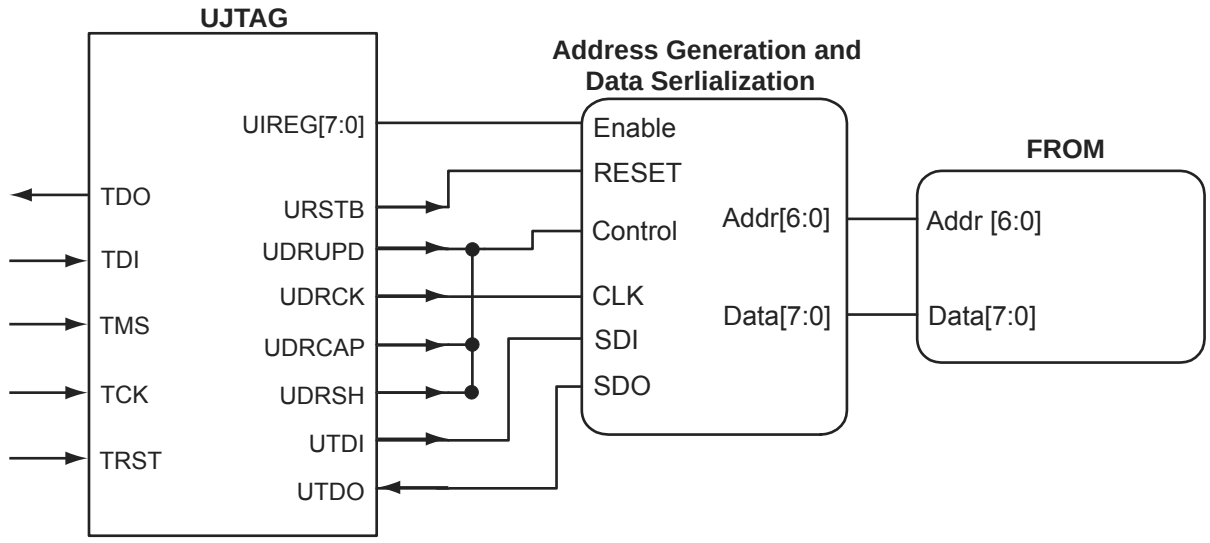

***Figure 16-1* • Block Diagram of Using UJTAG to Read FlashROM Contents**

UJTAG Macro

The UJTAG tiles can be instantiated in a design using the UJTAG macro from the Fusion, IGLOO, or ProASIC3 macro library. Note that "UJTAG" is a reserved name and cannot be used for any other user-defined blocks. A block symbol of the UJTAG tile macro is presented in Figure 16-2. In this figure, the ports on the left side of the block are connected to the JTAG TAP Controller, and the right-side ports are accessible by the FPGA core VersaTiles. The TDI, TMS, TDO, TCK, and TRST ports of UJTAG are only provided for design simulation purposes and should be treated as external signals in the design netlist. However, these ports must NOT be connected to any I/O buffer in the netlist. Figure 16-3 on page 300 illustrates the correct connection of the UJTAG macro to the user design netlist. Microsemi Designer software will automatically connect these ports to the TAP during place-and-route. Table 16-2 gives the port descriptions for the rest of the UJTAG ports:

Table 16-2 • **UJTAG Port Descriptions**

Port	Description
UIREG [7:0]	This 8-bit bus carries the contents of the JTAG Instruction Register of each device. Instruction Register values 16 to 127 are not reserved and can be employed as user-defined instructions.
URSTB	URSTB is an active-low signal and will be asserted when the TAP Controller is in Test-Logic-Reset mode. URSTB is asserted at power-up, and a power-on reset signal resets the TAP Controller. URSTB will stay asserted until an external TAP access changes the TAP Controller state.
UTDI	This port is directly connected to the TAP's TDI signal.
UTDO	This port is the user TDO output. Inputs to the UTDO port are sent to the TAP TDO output MUX when the IR address is in user range.
UDRSH	Active-high signal enabled in the ShiftDR TAP state
UDRCAP	Active-high signal enabled in the CaptureDR TAP state
UDRCK	This port is directly connected to the TAP's TCK signal.
UDRUPD	Active-high signal enabled in the UpdateDR TAP state

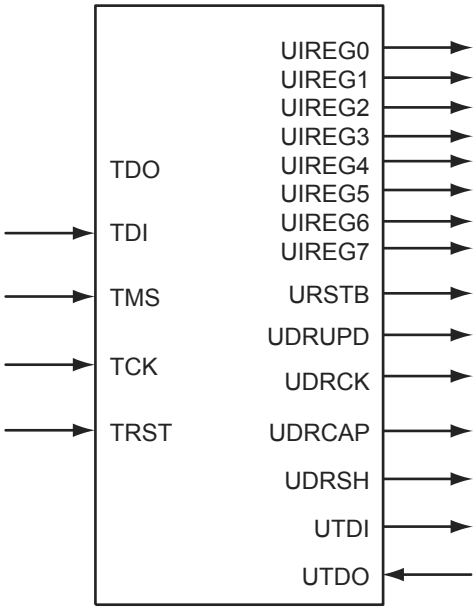

Figure 16-2 • **UJTAG Tile Block Symbol**

SRAM Initialization

Users can also initialize embedded SRAMs of the low power flash devices. The initialization of the embedded SRAM blocks of the design can be done using UJTAG tiles, where the initialization data is imported using the TAP Controller. Similar functionality is available in ProASIC^{PLUS} devices using JTAG. The guidelines for implementation and design examples are given in the *RAM Initialization and ROM Emulation in ProASIC^{PLUS} Devices* application note.

SRAMs are volatile by nature; data is lost in the absence of power. Therefore, the initialization process should be done at each power-up if necessary.

FlashROM Read-Back Using JTAG

The low power flash architecture contains a dedicated nonvolatile FlashROM block, which is formatted into eight 128-bit pages. For more information on FlashROM, refer to the "FlashROM in Microsemi's Low Power Flash Devices" section on page 117. The contents of FlashROM are available to the VersaTiles during normal operation through a read operation. As a result, the UJTAG macro can be used to provide the FlashROM contents to the JTAG port during normal operation. Figure 16-7 illustrates a simple block diagram of using UJTAG to read the contents of FlashROM during normal operation.

The FlashROM read address can be provided from outside the FPGA through the TDI input or can be generated internally using the core logic. In either case, data serialization logic is required (Figure 16-7) and should be designed using the VersaTile core logic. FlashROM contents are read asynchronously in parallel from the flash memory and shifted out in a synchronous serial format to TDO. Shifting the serial data out of the serialization block should be performed while the TAP is in UDRSH mode. The coordination between TCK and the data shift procedure can be done using the TAP state machine by monitoring UDRSH, UDRCAP, and UDRUPD.

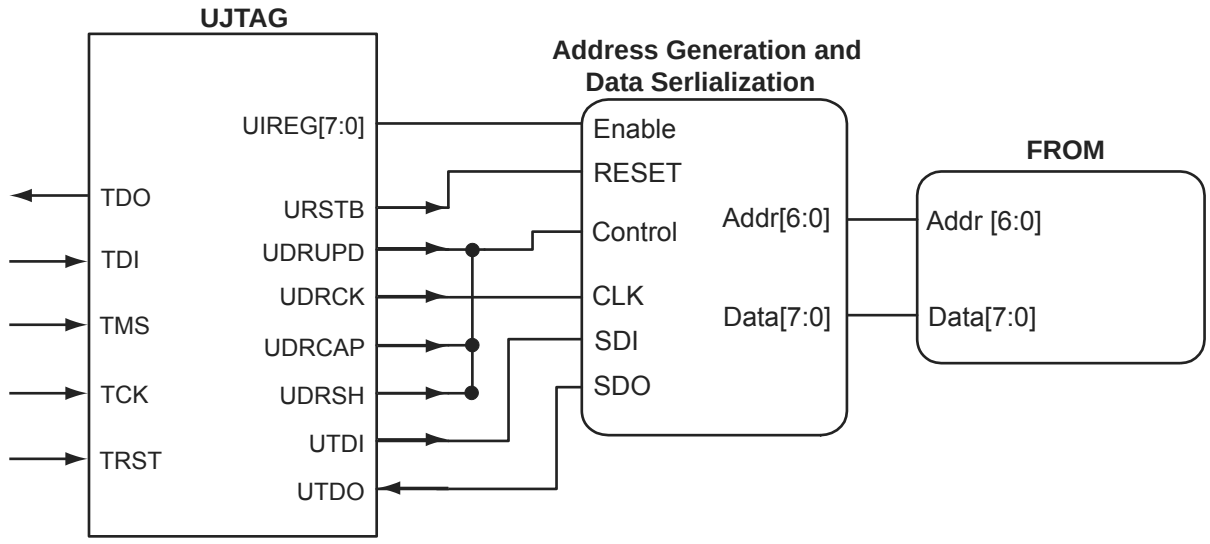

***Figure 16-7* • Block Diagram of Using UJTAG to Read FlashROM Contents**

Power-Up/-Down Sequence and Transient Current

Microsemi's low power flash devices use the following main voltage pins during normal operation:[2]

- VCCPLX
- VJTAG
- VCC: Voltage supply to the FPGA core
  - VCC is 1.5 V ± 0.075 V for IGLOO, IGLOO nano, IGLOO PLUS, and ProASIC3 devices operating at 1.5 V.
  - VCC is 1.2 V ± 0.06 V for IGLOO, IGLOO nano, IGLOO PLUS, and ProASIC3L devices operating at 1.2 V.
  - V5 devices will require a 1.5 V VCC supply, whereas V2 devices can utilize either a 1.2 V or 1.5 V VCC.
- VCCIBx: Supply voltage to the bank's I/O output buffers and I/O logic. Bx is the I/O bank number.
- VMVx: Quiet supply voltage to the input buffers of each I/O bank. x is the bank number. (Note: IGLOO nano, IGLOO PLUS, and ProASIC3 nano devices do not have VMVx supply pins.)

The I/O bank VMV pin must be tied to the VCCI pin within the same bank. Therefore, the supplies that need to be powered up/down during normal operation are VCC and VCCI. These power supplies can be powered up/down in any sequence during normal operation of IGLOO, IGLOO nano, IGLOO PLUS, ProASIC3L, ProASIC3, and ProASIC3 nano FPGAs. During power-up, I/Os in each bank will remain tristated until the last supply (either VCCIBx or VCC) reaches its functional activation voltage. Similarly, during power-down, I/Os of each bank are tristated once the first supply reaches its brownout deactivation voltage.

Although Microsemi's low power flash devices have no power-up or power-down sequencing requirements, Microsemi identifies the following power conditions that will result in higher than normal transient current. Use this information to help maximize power savings:

Microsemi recommends tying VCCPLX to VCC and using proper filtering circuits to decouple VCC noise from the PLL.

a. If VCCPLX is powered up before VCC, a static current of up to 5 mA (typical) per PLL may be measured on VCCPLX.

   The current vanishes as soon as VCC reaches the VCCPLX voltage level.

   The same current is observed at power-down (VCC before VCCPLX).

b. If VCCPLX is powered up simultaneously or after VCC:

   i. Microsemi's low power flash devices exhibit very low transient current on VCC. For ProASIC3 devices, the maximum transient current on V_{CC} does not exceed the maximum standby current specified in the device datasheet.

The source of transient current, also known as inrush current, varies depending on the FPGA technology. Due to their volatile technology, the internal registers in SRAM FPGAs must be initialized before configuration can start. This initialization is the source of significant inrush current in SRAM FPGAs during power-up. Due to the nonvolatile nature of flash technology, low power flash devices do not require any initialization at power-up, and there is very little or no crossbar current through PMOS and NMOS devices. Therefore, the transient current at power-up is significantly less than for SRAM FPGAs. Figure 17-1 on page 310 illustrates the types of power consumption by SRAM FPGAs compared to Microsemi's antifuse and flash FPGAs.

2. *For more information on Microsemi FPGA voltage supplies, refer to the appropriate datasheet located at http://www.microsemi.com/soc/techdocs/ds.*