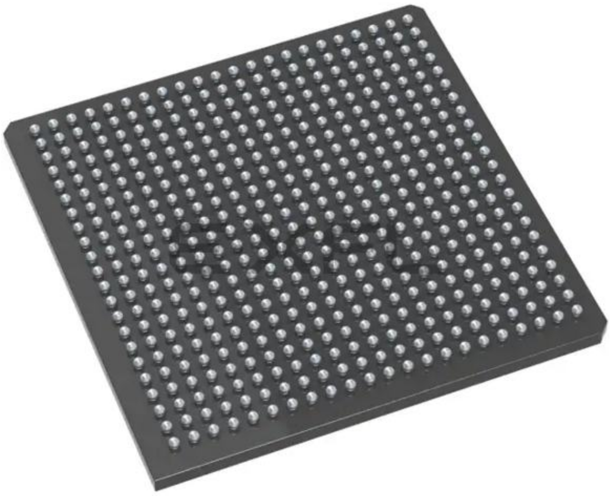


Welcome to E-XFL.COM

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	516096
Number of I/O	341
Number of Gates	3000000
Voltage - Supply	1.14V ~ 1.575V
Mounting Type	Surface Mount
Operating Temperature	-40°C ~ 100°C (TJ)
Package / Case	484-BGA
Supplier Device Package	484-FPBGA (23x23)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/m1a3pe3000l-fg484i

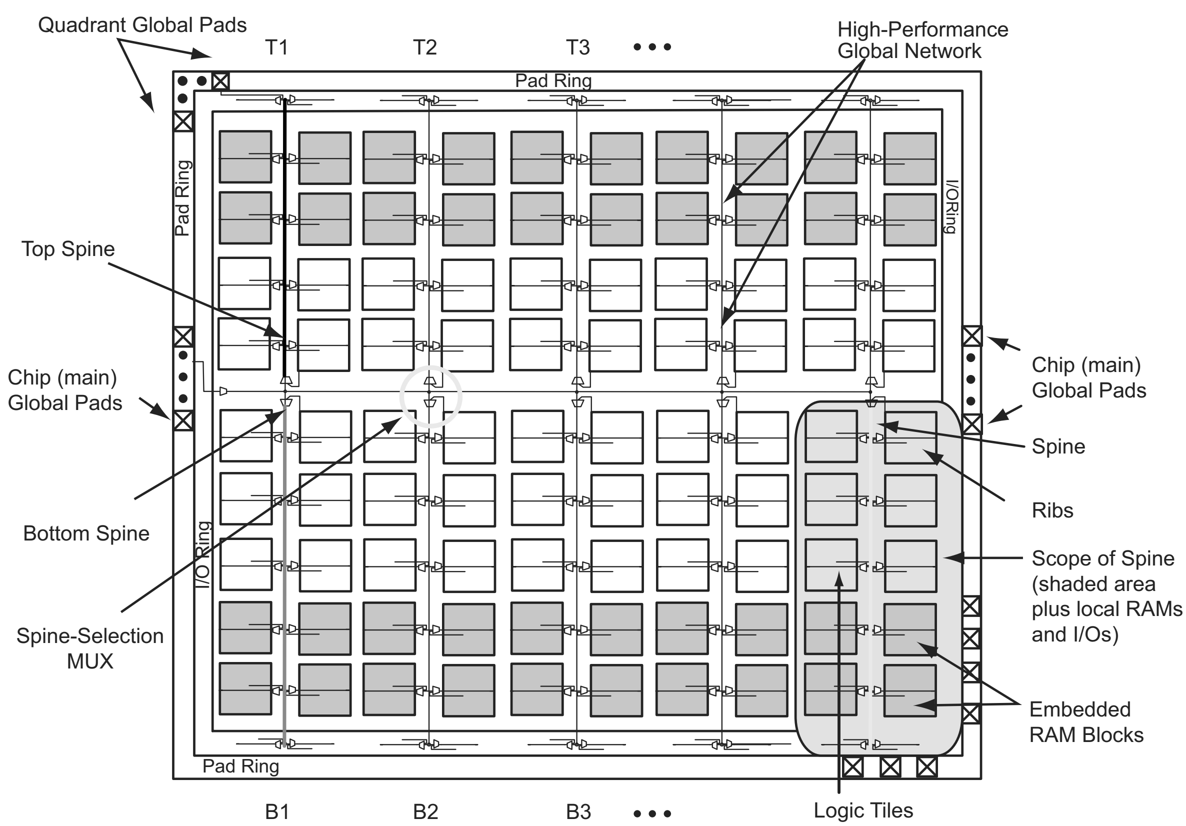
VersaNet Global Network Distribution

One of the architectural benefits of low power flash architecture is the set of powerful, low-delay VersaNet global networks that can access the VersaTiles, SRAM, and I/O tiles of the device. Each device offers a chip global network with six global lines (except for nano 10 k, 15 k, and 20 k gate devices) that are distributed from the center of the FPGA array. In addition, each device (except the 10 k through 30 k gate device) has four quadrant global networks, each consisting of three quadrant global net resources. These quadrant global networks can only drive a signal inside their own quadrant. Each VersaTile has access to nine global line resources—three quadrant and six chip-wide (main) global networks—and a total of 18 globals are available on the device (3 × 4 regional from each quadrant and 6 global).

Figure 3-1 shows an overview of the VersaNet global network and device architecture for devices 60 k and above. Figure 3-2 and Figure 3-3 on page 50 show simplified VersaNet global networks.

The VersaNet global networks are segmented and consist of spines, global ribs, and global multiplexers (MUXes), as shown in Figure 3-1. The global networks are driven from the global rib at the center of the die or quadrant global networks at the north or south side of the die. The global network uses the MUX trees to access the spine, and the spine uses the clock ribs to access the VersaTile. Access is available to the chip or quadrant global networks and the spines through the global MUXes. Access to the spine using the global MUXes is explained in the "Spine Architecture" section on page 57.

These VersaNet global networks offer fast, low-skew routing resources for high-fanout nets, including clock signals. In addition, these highly segmented global networks offer users the flexibility to create low-skew local clock networks using spines for up to 252 internal/external clocks or other high-fanout nets in low power flash devices. Optimal usage of these low-skew networks can result in significant improvement in design performance.

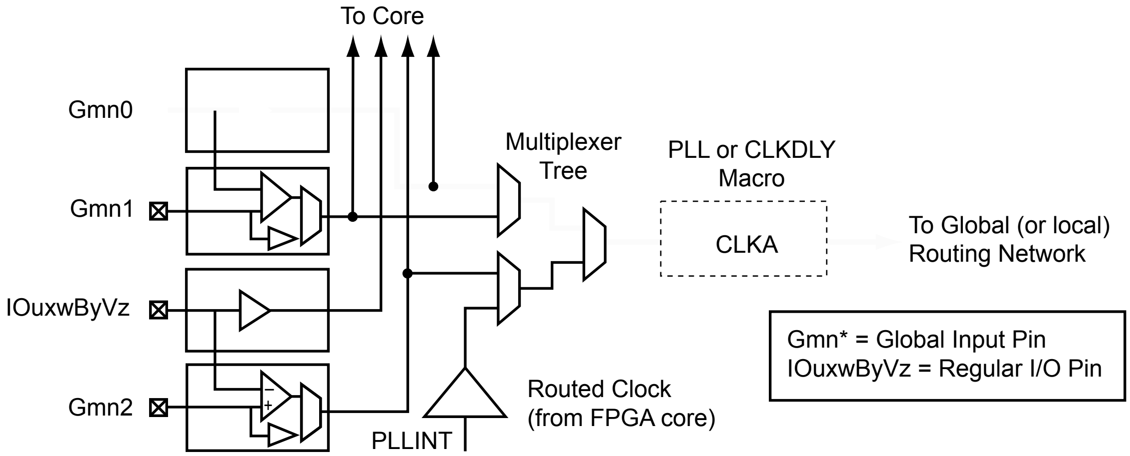


Note: Not applicable to 10 k through 30 k gate devices

Figure 3-1 **• Overview of VersaNet Global Network and Device Architecture**

Figure 3-18 **• Globals Management GUI in Designer**

3. Occasionally, the synthesis tool assigns a global macro to clock nets, even though the fanout is significantly less than other asynchronous signals. Select **Demote global nets whose fanout is less than** and enter a reasonable value for fanouts. This frees up some global networks from the signals that have very low fanouts. This can also be done using PDC.
4. Use a local clock network for the signals that do not need to go to the whole chip but should have low skew. This local clock network assignment can only be done using PDC.
5. Assign the I/O buffer using MVN if you have fixed I/O assignment. As shown in Figure 3-10 on page 61, there are three sets of global pins that have a hardwired connection to each global network. Do not try to put multiple CLKBUF macros in these three sets of global pins. For example, do not assign two CLKBUFs to GAA0x and GAA2x pins.
6. You must click **Commit** at the end of MVN assignment. This runs the pre-layout checker and checks the validity of global assignment.
7. Always run Compile with the **Keep existing physical constraints** option on. This uses the quadrant clock network assignment in the MVN assignment and checks if you have the desired signals on the global networks.
8. Run Layout and check the timing.



Note: Fusion CCCs have additional source selections (RCOSC, XTAL).

Figure 4-9 **• Illustration of Hardwired I/O (global input pins) Usage for IGLOO and ProASIC3 devices 60 k Gates and Larger**

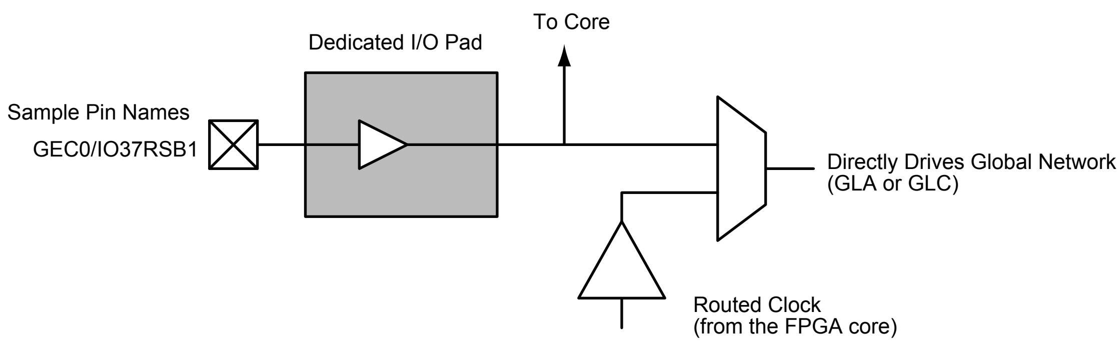


Figure 4-10 **• Illustration of Hardwired I/O (global input pins) Usage for IGLOO and ProASIC3 devices 30 k Gates and Smaller**

External I/O Clock Source

External I/O refers to regular I/O pins. The clock source is instantiated with one of the various INBUF options and accesses the CCCs via internal routing. The user has the option of assigning this input to any of the I/Os labeled with the I/O convention *IOuxwByVz*. Refer to the "User I/O Naming Conventions in I/O Structures" chapter of the appropriate device user's guide, and for Fusion, refer to the *Fusion Family of Mixed Signal FPGAs* datasheet for more information. Figure 4-11 gives a brief explanation of external I/O usage. Choosing this option provides the freedom of selecting any user I/O location but introduces additional delay because the signal connects to the routed clock input through internal routing before connecting to the CCC reference clock input.

For the External I/O option, the routed signal would be instantiated with a PLLINT macro before connecting to the CCC reference clock input. This instantiation is conveniently done automatically by SmartGen when this option is selected. Microsemi recommends using the SmartGen tool to generate the CCC macro. The instantiation of the PLLINT macro results in the use of the routed clock input of the I/O to connect to the PLL clock input. If not using SmartGen, manually instantiate a PLLINT macro before the PLL reference clock to indicate that the regular I/O driving the PLL reference clock should be used (see Figure 4-11 for an example illustration of the connections, shown in red).

In the above two options, the clock source must be instantiated with one of the various INBUF macros. The reference clock pins of the CCC functional block core macros must be driven by regular input macros (INBUFs), not clock input macros.

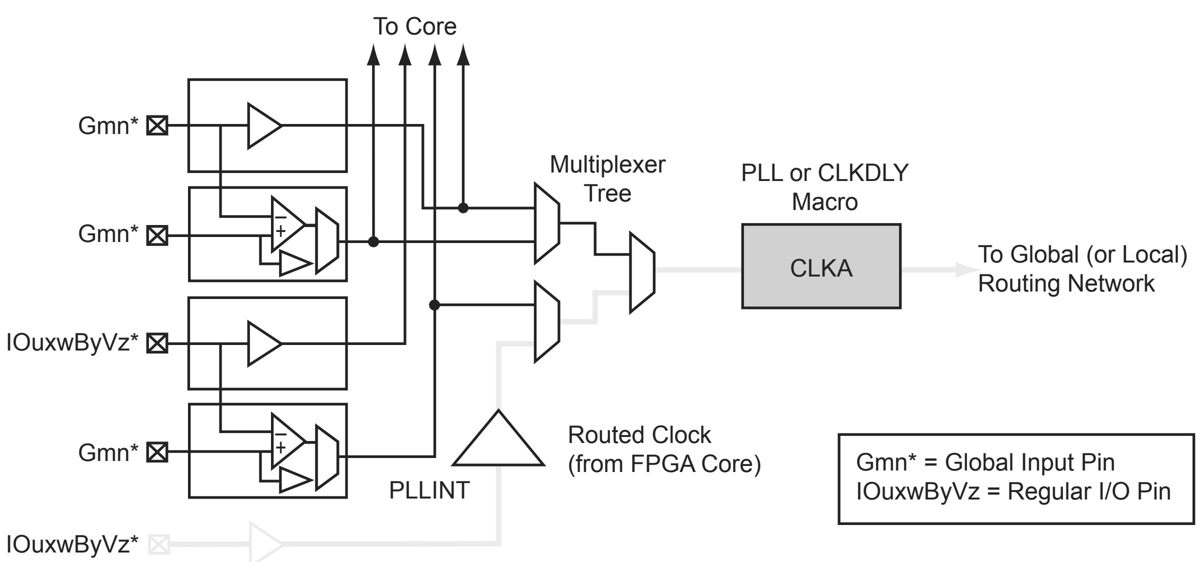


Figure 4-11 **• Illustration of External I/O Usage**

For Fusion devices, the input reference clock can also be from the embedded RC oscillator and crystal oscillator. In this case, the CCC configuration is the same as the hardwired I/O clock source, and users are required to instantiate the RC oscillator or crystal oscillator macro and connect its output to the input reference clock of the CCC block.

Core Logic Clock Source

Core logic refers to internal routed nets. Internal routed signals access the CCC via the FPGA Core Fabric. Similar to the External I/O option, whenever the clock source comes internally from the core itself, the routed signal is instantiated with a PLLINT macro before connecting to the CCC clock input (see Figure 4-12 for an example illustration of the connections, shown in red).

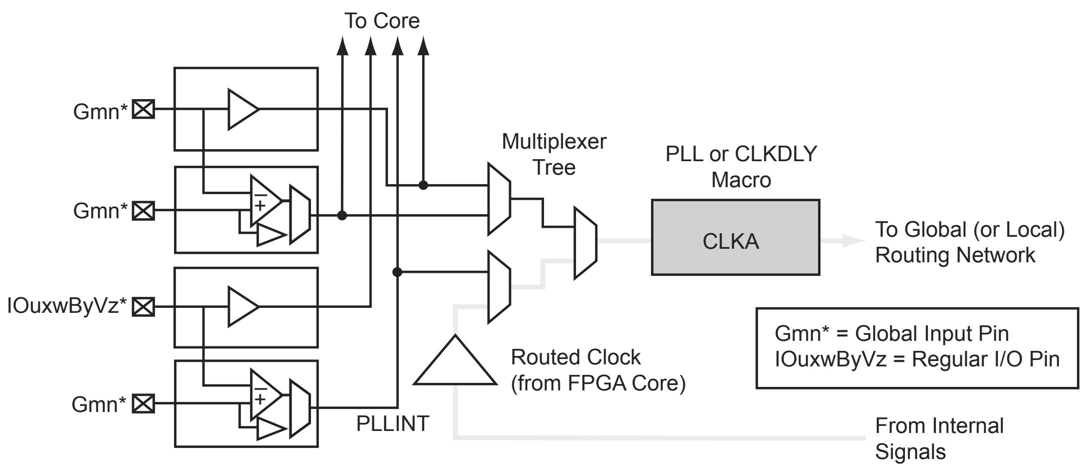


Figure 4-12 **• Illustration of Core Logic Usage**

For Fusion devices, the input reference clock can also be from the embedded RC oscillator and crystal oscillator. In this case, the CCC configuration is the same as the hardwired I/O clock source, and users are required to instantiate the RC oscillator or crystal oscillator macro and connect its output to the input reference clock of the CCC block.

Available I/O Standards

Table 4-4 **• Available I/O Standards within CLKBUF and CLKBUF_LVDS/LVPECL Macros**

CLKBUF_LVCMOS5
CLKBUF_LVCMOS33 [1]
CLKBUF_LVCMOS25 [2]
CLKBUF_LVCMOS18
CLKBUF_LVCMOS15
CLKBUF_PCI
CLKBUF_PCIX [3]
CLKBUF_GTL25 [2,3]
CLKBUF_GTL33 [2,3]
CLKBUF_GTLP25 [2,3]
CLKBUF_GTLP33 [2,3]
CLKBUF_HSTL_I [2,3]
CLKBUF_HSTL_II [2,3]
CLKBUF_SSTL3_I [2,3]
CLKBUF_SSTL3_II [2,3]
CLKBUF_SSTL2_I [2,3]
CLKBUF_SSTL2_II [2,3]
CLKBUF_LVDS [4,5]
CLKBUF_LVPECL[5]

Notes:

1. *By default, the CLKBUF macro uses 3.3 V LVTTL I/O technology. For more details, refer to the IGLOO, ProASIC3, SmartFusion, and Fusion Macro Library Guide.*
2. *I/O standards only supported in ProASIC3E and IGLOOe families.*
3. *I/O standards only supported in the following Fusion devices: AFS600 and AFS1500.*
4. *B-LVDS and M-LVDS standards are supported by CLKBUF_LVDS.*
5. *Not supported for IGLOO nano and ProASIC3 nano devices.*

Global Synthesis Constraints

The Synplify® synthesis tool, by default, allows six clocks in a design for Fusion, IGLOO, and ProASIC3. When more than six clocks are needed in the design, a user synthesis constraint attribute, syn_global_buffers, can be used to control the maximum number of clocks (up to 18) that can be inferred by the synthesis engine.

High-fanout nets will be inferred with clock buffers and/or internal clock buffers. If the design consists of CCC global buffers, they are included in the count of clocks in the design.

The subsections below discuss the clock input source (global buffers with no programmable delays) and the clock conditioning functional block (global buffers with programmable delays and/or PLL function) in detail.

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.

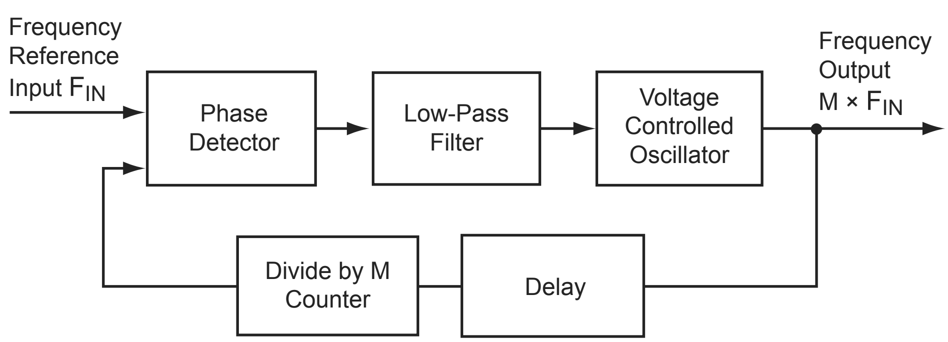


Figure 4-19 **• Simplified PLL Core with Feedback Divider and Delay**

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 101) is increased, the average phase difference increases. The average phase

Each group of control bits is assigned a specific location in the configuration shift register. For a list of the 81 configuration bits (C[80:0]) in the CCC and a description of each, refer to "PLL Configuration Bits Description" on page 106. The configuration register can be serially loaded with the new configuration data and programmed into the CCC using the following ports:

- SDIN: The configuration bits are serially loaded into a shift register through this port. The LSB of the configuration data bits should be loaded first.
- SDOUT: The shift register contents can be shifted out (LSB first) through this port using the shift operation.
- SCLK: This port should be driven by the shift clock.
- SSHIFT: The active-high shift enable signal should drive this port. The configuration data will be shifted into the shift register if this signal is HIGH. Once SSHIFT goes LOW, the data shifting will be halted.
- SUPDATE: The SUPDATE signal is used to configure the CCC with the new configuration bits when shifting is complete.

To access the configuration ports of the shift register (SDIN, SDOUT, SSHIFT, etc.), the user should instantiate the CCC macro in his design with appropriate ports. Microsemi recommends that users choose SmartGen to generate the CCC macros with the required ports for dynamic reconfiguration.

Users must familiarize themselves with the architecture of the CCC core and its input, output, and configuration ports to implement the desired delay and output frequency in the CCC structure. Figure 4-22 shows a model of the CCC with configurable blocks and switches.

Table 4-9 to Table 4-15 on page 110 provide descriptions of the configuration data for the configuration bits.

Table 4-9 **• Input Clock Divider, FINDIV[6:0] (/n)**

FINDIV<6:0> State	Divisor	New Frequency Factor
0	1	1.00000
1	2	0.50000
⋮	⋮	⋮
127	128	0.0078125

Table 4-10 **• Feedback Clock Divider, FBDIV[6:0] (/m)**

FBDIV<6:0> State	Divisor	New Frequency Factor
0	1	1
1	2	2
⋮	⋮	⋮
127	128	128

Table 4-11 **• Output Frequency Dividers**
A **Output Divider, OADIV <4:0> (/u);**
B **Output Divider, OBDIV <4:0> (/v);**
C **Output Divider, OCDIV <4:0> (/w)**

OADIV<4:0>; OBDIV<4:0>; CDIV<4:0> State	Divisor	New Frequency Factor
0	1	1.00000
1	2	0.50000
⋮	⋮	⋮
31	32	0.03125

Table 4-12 **• MUXA, MUXB, MUXC**

OAMUX<2:0>; OBMUX<2:0>; OCMUX<2:0> State	MUX Input Selected
0	None. Six-input MUX and PLL are bypassed. Clock passes only through global MUX and goes directly into HC ribs.
1	Not available
2	PLL feedback delay line output
3	Not used
4	PLL VCO 0° phase shift
5	PLL VCO 270° phase shift
6	PLL VCO 180° phase shift
7	PLL VCO 90° phase shift

FlashROM Design Flow

The Microsemi Libero System-on-Chip (SoC) software has extensive FlashROM support, including FlashROM generation, instantiation, simulation, and programming. Figure 5-9 shows the user flow diagram. In the design flow, there are three main steps:

1. FlashROM generation and instantiation in the design
2. Simulation of FlashROM design
3. Programming file generation for FlashROM design

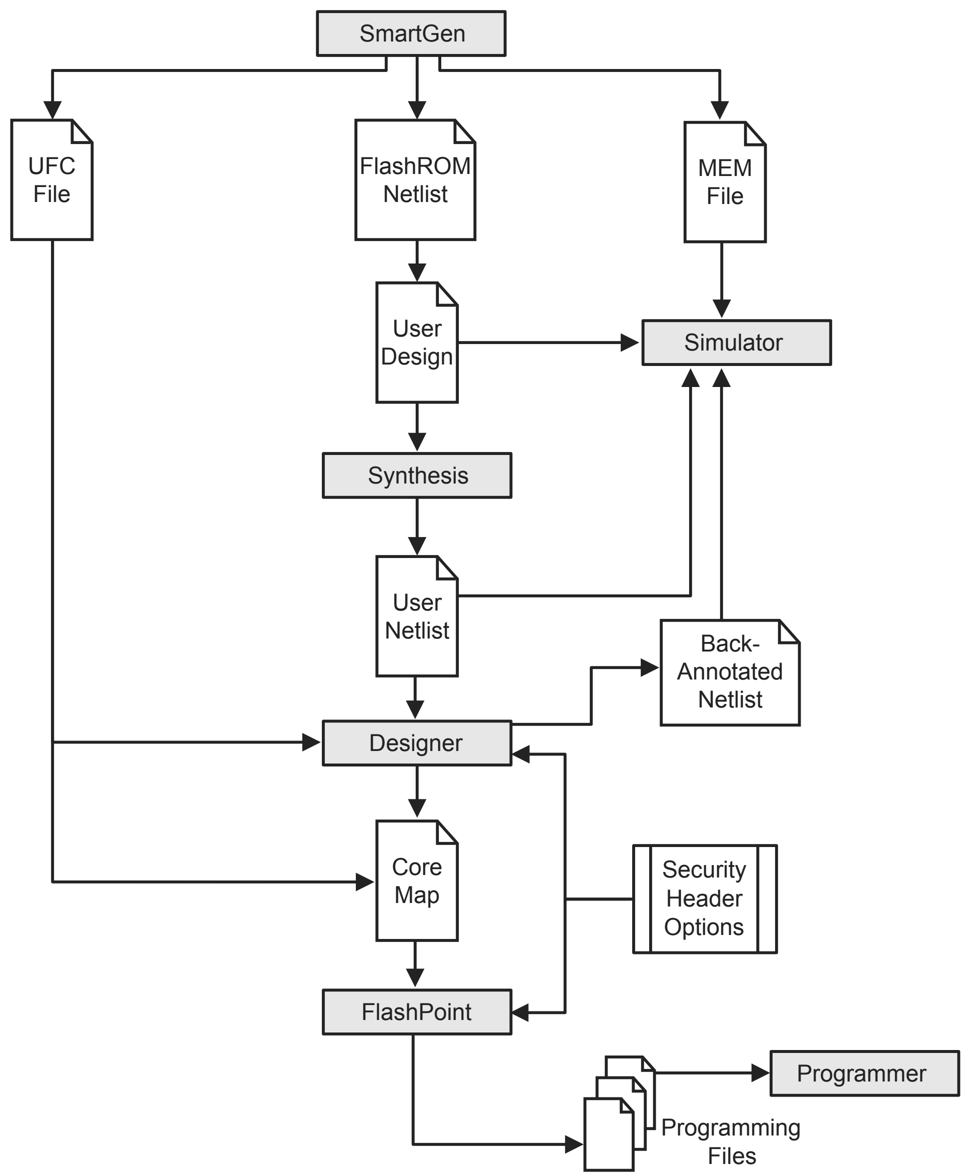


Figure 5-9 • **FlashROM Design Flow**

DEVICE_INFO displays the FlashROM content, serial number, Design Name, and checksum, as shown below:

```
EXPORT IDCODE[32] = 123261CF
EXPORT SILSIG[32] = 00000000
User information :
CHECKSUM: 61A0
Design Name:        TOP
Programming Method: STAPL
Algorithm Version: 1
Programmer: UNKNOWN
=========================================
FlashROM Information :
EXPORT Region_7_0[128] = FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
=========================================
Security Setting :
Encrypted FlashROM Programming Enabled.
Encrypted FPGA Array Programming Enabled.
=========================================
```

The Libero SoC file manager recognizes the UFC and MEM files and displays them in the appropriate view. Libero SoC also recognizes the multiple programming files if you choose the option to generate multiple files for multiple FlashROM contents in Designer. These features enable a user-friendly flow for the FlashROM generation and programming in Libero SoC.

Custom Serialization Using FlashROM

You can use FlashROM for device serialization or inventory control by using the Auto Inc region or Read From File region. FlashPoint will automatically generate the serial number sequence for the Auto Inc region with the **Start Value**, **Max Value**, and **Step Value** provided. If you have a unique serial number generation scheme that you prefer, the Read From File region allows you to import the file with your serial number scheme programmed into the region. See the *FlashPro User's Guide* for custom serialization file format information.

The following steps describe how to perform device serialization or inventory control using FlashROM:

1. Generate FlashROM using SmartGen. From the Properties section in the FlashROM Settings dialog box, select the **Auto Inc** or **Read From File** region. For the Auto Inc region, specify the desired step value. You will not be able to modify this value in the FlashPoint software.
2. Go through the regular design flow and finish place-and-route.
3. Select **Programming File in Designer** and open **Generate Programming File** (Figure 5-12 on page 144).
4. Click **Program FlashROM**, browse to the UFC file, and click **Next**. The FlashROM Settings window appears, as shown in Figure 5-13 on page 144.
5. Select the FlashROM page you want to program and the data value for the configured regions. The STAPL file generated will contain only the data that targets the selected FlashROM page.
6. Modify properties for the serialization.
   – For the Auto Inc region, specify the **Start** and **Max** values.
   – For the Read From File region, select the file name of the custom serialization file.
7. Select the FlashROM programming file type you want to generate from the two options below:
   – Single programming file for all devices: generates one programming file with all FlashROM values.
   – One programming file per device: generates a separate programming file for each FlashROM value.
8. Enter the number of devices you want to program and generate the required programming file.
9. Open the programming software and load the programming file. The programming software, FlashPro3 and Silicon Sculptor II, supports the device serialization feature. If, for some reason, the device fails to program a part during serialization, the software allows you to reuse or skip the serial data. Refer to the *FlashPro User's Guide* for details.

Table 6-2 **• Allowable Aspect Ratio Settings for WIDTHA[1:0]**

WIDTHA[1:0]	WIDTHB[1:0]	D×W
00	00	4k×1
01	01	2k×2
10	10	1k×4
11	11	512×9

Note: The aspect ratio settings are constant and cannot be changed on the fly.

BLKA and BLKB

These signals are active-low and will enable the respective ports when asserted. When a BLKx signal is deasserted, that port’s outputs hold the previous value.

Note: When using the SRAM in single-port mode for Automotive ProASIC3 devices, BLKB should be tied to ground.

WENA and WENB

These signals switch the RAM between read and write modes for the respective ports. A LOW on these signals indicates a write operation, and a HIGH indicates a read.

Note: When using the SRAM in single-port mode for Automotive ProASIC3 devices, WENB should be tied to ground.

CLKA and CLKB

These are the clock signals for the synchronous read and write operations. These can be driven independently or with the same driver.

Note: For Automotive ProASIC3 devices, dual-port mode is supported if the clocks to the two SRAM ports are the same and 180° out of phase (i.e., the port A clock is the inverse of the port B clock). For use of this macro as a single-port SRAM, the inputs and clock of one port should be tied off (grounded) to prevent errors during design compile.

PIPEA and PIPEB

These signals are used to specify pipelined read on the output. A LOW on PIPEA or PIPEB indicates a nonpipelined read, and the data appears on the corresponding output in the same clock cycle. A HIGH indicates a pipelined read, and data appears on the corresponding output in the next clock cycle.

Note: When using the SRAM in single-port mode for Automotive ProASIC3 devices, PIPEB should be tied to ground. For use in dual-port mode, the same clock with an inversion between the two clock pins of the macro should be used in the design to prevent errors during compile.

WMODEA and WMODEB

These signals are used to configure the behavior of the output when the RAM is in write mode. A LOW on these signals makes the output retain data from the previous read. A HIGH indicates pass-through behavior, wherein the data being written will appear immediately on the output. This signal is overridden when the RAM is being read.

Note: When using the SRAM in single-port mode for Automotive ProASIC3 devices, WMODEB should be tied to ground.

RESET

This active-low signal resets the control logic, forces the output hold state registers to zero, disables reads and writes from the SRAM block, and clears the data hold registers when asserted. It does not reset the contents of the memory array.

While the RESET signal is active, read and write operations are disabled. As with any asynchronous reset signal, care must be taken not to assert it too close to the edges of active read and write clocks.

ADDRA and ADDRB

These are used as read or write addresses, and they are 12 bits wide. When a depth of less than 4 k is specified, the unused high-order bits must be grounded (Table 6-3 on page 155).

SRAM Usage

The following descriptions refer to the usage of both RAM4K9 and RAM512X18.

Clocking

The dual-port SRAM blocks are only clocked on the rising edge. SmartGen allows falling-edge-triggered clocks by adding inverters to the netlist, hence achieving dual-port SRAM blocks that are clocked on either edge (rising or falling). For dual-port SRAM, each port can be clocked on either edge and by separate clocks by port. Note that for Automotive ProASIC3, the same clock, with an inversion between the two clock pins of the macro, should be used in design to prevent errors during compile.

Low power flash devices support inversion (bubble-pushing) throughout the FPGA architecture, including the clock input to the SRAM modules. Inversions added to the SRAM clock pin on the design schematic or in the HDL code will be automatically accounted for during design compile without incurring additional delay in the clock path.

The two-port SRAM can be clocked on the rising or falling edge of WCLK and RCLK.

If negative-edge RAM and FIFO clocking is selected for memory macros, clock edge inversion management (bubble-pushing) is automatically used within the development tools, without performance penalty.

Modes of Operation

There are two read modes and one write mode:

- Read Nonpipelined (synchronous—1 clock edge): In the standard read mode, new data is driven onto the RD bus in the same clock cycle following RA and REN valid. The read address is registered on the read port clock active edge, and data appears at RD after the RAM access time. Setting PIPE to OFF enables this mode.
- Read Pipelined (synchronous—2 clock edges): The pipelined mode incurs an additional clock delay from address to data but enables operation at a much higher frequency. The read address is registered on the read port active clock edge, and the read data is registered and appears at RD after the second read clock edge. Setting PIPE to ON enables this mode.
- Write (synchronous—1 clock edge): On the write clock active edge, the write data is written into the SRAM at the write address when WEN is HIGH. The setup times of the write address, write enables, and write data are minimal with respect to the write clock.

RAM Initialization

Each SRAM block can be individually initialized on power-up by means of the JTAG port using the UJTAG mechanism. The shift register for a target block can be selected and loaded with the proper bit configuration to enable serial loading. The 4,608 bits of data can be loaded in a single operation.

FIFO Features

The FIFO4KX18 macro is created by merging the RAM block with dedicated FIFO logic (Figure 6-6 on page 158). Since the FIFO logic can only be used in conjunction with the memory block, there is no separate FIFO controller macro. As with the RAM blocks, the FIFO4KX18 nomenclature does not refer to a possible aspect ratio, but rather to the deepest possible data depth and the widest possible data width. FIFO4KX18 can be configured into the following aspect ratios: 4,096×1, 2,048×2, 1,024×4, 512×9, and 256×18. In addition to being fully synchronous, the FIFO4KX18 also has the following features:

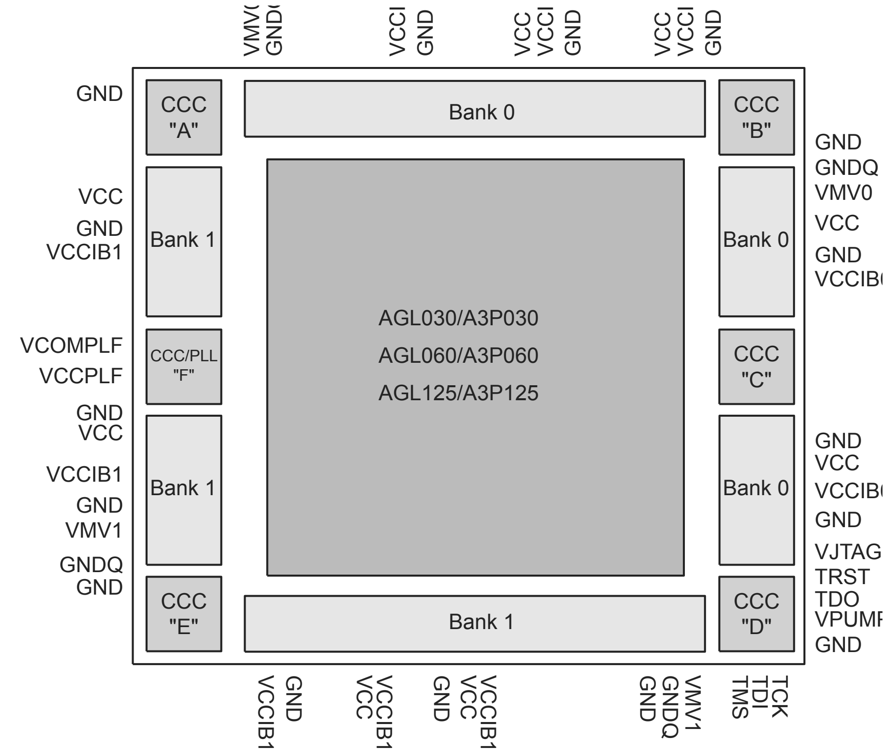
- Four FIFO flags: Empty, Full, Almost-Empty, and Almost-Full
- Empty flag is synchronized to the read clock
- Full flag is synchronized to the write clock
- Both Almost-Empty and Almost-Full flags have programmable thresholds
- Active-low asynchronous reset
- Active-low block enable
- Active-low write enable
- Active-high read enable
- Ability to configure the FIFO to either stop counting after the empty or full states are reached or to allow the FIFO counters to continue

Features Supported on Every I/O

Table 7-5 lists all features supported by transmitter/receiver for single-ended and differential I/Os. Table 7-6 on page 180 lists the performance of each I/O technology.

***Table 7-5* • I/O Features**

Feature	Description
All I/O	• High performance (Table 7-6 on page 180) • Electrostatic discharge (ESD) protection • I/O register combining option
Single-Ended Transmitter Features	• Hot-swap: – 30K gate devices: hot-swap in every mode – All other IGLOO and ProASIC3 devices: no hot-swap • Output slew rate: 2 slew rates (except 30K gate devices) • Weak pull-up and pull-down resistors • Output drive: 3 drive strengths • Programmable output loading • Skew between output buffer enable/disable time: 2 ns delay on rising edge and 0 ns delay on falling edge (see the "Selectable Skew between Output Buffer Enable and Disable Times" section on page 199 for more information) • LVTTL/LVCMOS 3.3 V outputs compatible with 5 V TTL inputs
Single-Ended Receiver Features	• 5 V–input–tolerant receiver (Table 7-12 on page 193) • Separate ground plane for GNDQ pin and power plane for VMV pin are used for input buffer to reduce output-induced noise.
Differential Receiver Features—250K through 1M Gate Devices	• Separate ground plane for GNDQ pin and power plane for VMV pin are used for input buffer to reduce output-induced noise.
CMOS-Style LVDS, B-LVDS, M-LVDS, or LVPECL Transmitter	• Two I/Os and external resistors are used to provide a CMOS-style LVDS, DDR LVDS, B-LVDS, and M-LVDS/LVPECL transmitter solution. • High slew rate • Weak pull-up and pull-down resistors • Programmable output loading



Note: The 30 k gate devices do not support a PLL (V_{COMPLF} and V_{CCPLF} pins).

***Figure 7-19* • Naming Conventions of IGLOO and ProASIC3 Devices with Two I/O Banks – Top View**

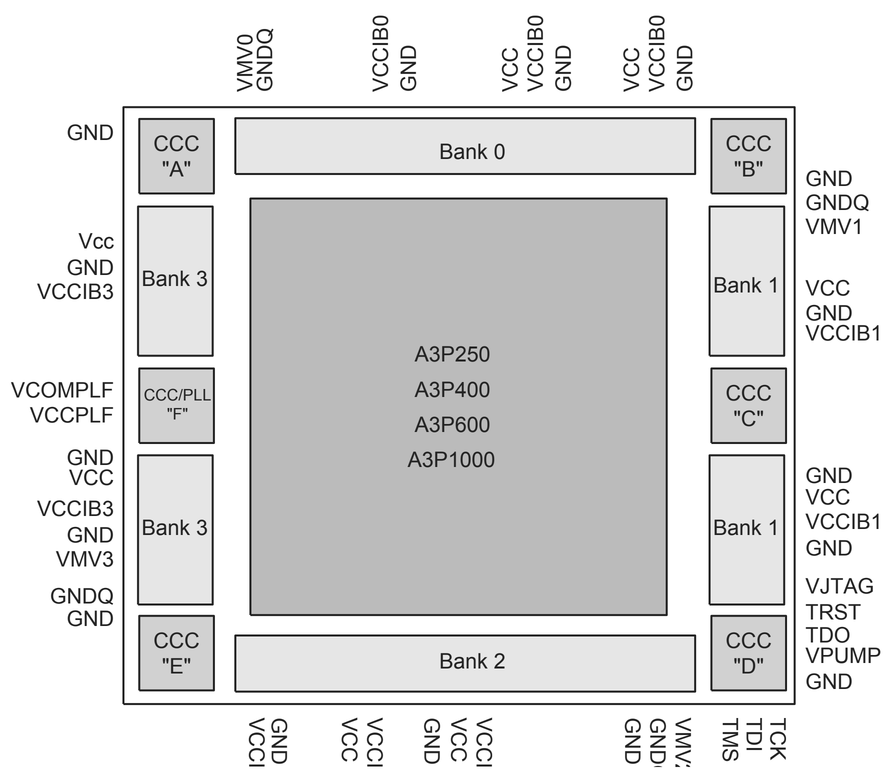


***Figure 7-20* • Naming Conventions of IGLOO and ProASIC3 Devices with Four I/O Banks – Top View**

Implementing I/Os in Microsemi Software

Microsemi Libero SoC software is integrated with design entry tools such as the SmartGen macro builder, the ViewDraw schematic entry tool, and an HDL editor. It is also integrated with the synthesis and Designer tools. In this section, all necessary steps to implement the I/Os are discussed.

Design Entry

There are three ways to implement I/Os in a design:

1. Use the SmartGen macro builder to configure I/Os by generating specific I/O library macros and then instantiating them in top-level code. This is especially useful when creating I/O bus structures.
2. Use an I/O buffer cell in a schematic design.
3. Manually instantiate specific I/O macros in the top-level code.

If technology-specific macros, such as INBUF_LVCMOS33 and OUTBUF_PCI, are used in the HDL code or schematic, the user will not be able to change the I/O standard later on in Designer. If generic I/O macros are used, such as INBUF, OUTBUF, TRIBUF, CLKBUF, and BIBUF, the user can change the I/O standard using the Designer I/O Attribute Editor tool.

Using SmartGen for I/O Configuration

The SmartGen tool in Libero SoC provides a GUI-based method of configuring the I/O attributes. The user can select certain I/O attributes while configuring the I/O macro in SmartGen. The steps to configure an I/O macro with specific I/O attributes are as follows:

1. Open Libero SoC.
2. On the left-hand side of the Catalog View, select **I/O**, as shown in Figure 9-2.

Figure 9-2 **• SmartGen Catalog**

Volume Programming Services

Device Type Supported: Flash and Antifuse

Once the design is stable for applications with large production volumes, preprogrammed devices can be purchased. Table 11-2 describes the volume programming services.

Table 11-2 **• Volume Programming Services**

Programmer	Vendor	Availability
In-House Programming	Microsemi	Contact Microsemi Sales
Distributor Programming Centers	Memec Unique	Contact Distribution
Independent Programming Centers	Various	Contact Vendor

Advantages: As programming is outsourced, this solution is easier to implement than creating a substantial in-house programming capability. As programming houses specialize in large-volume programming, this is often the most cost-effective solution.

Limitations: There are some logistical issues with the use of a programming service provider, such as the transfer of programming files and the approval of First Articles. By definition, the programming file must be released to a third-party programming house. Nondisclosure agreements (NDAs) can be signed to help ensure data protection; however, for extremely security-conscious designs, this may not be an option.

- Microsemi In-House Programming

  When purchasing Microsemi devices in volume, IHP can be requested as part of the purchase. If this option is chosen, there is a small cost adder for each device programmed. Each device is marked with a special mark to distinguish it from blank parts. Programming files for the design will be sent to Microsemi. Sample parts with the design programmed, First Articles, will be returned for customer approval. Once approval of First Articles has been received, Microsemi will proceed with programming the remainder of the order. To request Microsemi IHP, contact your local Microsemi representative.

- Distributor Programming Centers

  If purchases are made through a distributor, many distributors will provide programming for their customers. Consult with your preferred distributor about this option.

Application 3: Nontrusted Environment—Field Updates/Upgrades

Programming or reprogramming of devices may occur at remote locations. Reconfiguration of devices in consumer products/equipment through public networks is one example. Typically, the remote system is already programmed with particular design contents. When design update (FPGA array contents update) and/or data upgrade (FlashROM and/or FB contents upgrade) is necessary, an updated programming file with AES encryption can be generated, sent across public networks, and transmitted to the remote system. Reprogramming can then be done using this AES-encrypted programming file, providing easy and secure field upgrades. Low power flash devices support this secure ISP using AES. The detailed flow for this application is shown in Figure 12-8. Refer to the "Microprocessor Programming of Microsemi's Low Power Flash Devices" chapter of an appropriate FPGA fabric user's guide for more information.

To prepare devices for this scenario, the user can initially generate a programming file with the available security setting options. This programming file is programmed into the devices before shipment. During the programming file generation step, the user has the option of making the security settings permanent or not. In situations where no changes to the security settings are necessary, the user can select this feature in the software to generate the programming file with permanent security settings. Microsemi recommends that the programming file use encryption with an AES key, especially when ISP is done via public domain.

For example, if the designer wants to use an AES key for the FPGA array and the FlashROM, **Permanent** needs to be chosen for this setting. At first, the user chooses the options to use an AES key for the FPGA array and the FlashROM, and then chooses **Permanently lock the security settings**. A unique AES key is chosen. Once this programming file is generated and programmed to the devices, the AES key is permanently stored in the on-chip memory, where it is secured safely. The devices are sent to distant locations for the intended application. When an update is needed, a new programming file must be generated. The programming file must use the same AES key for encryption; otherwise, the authentication will fail and the file will not be programmed in the device.

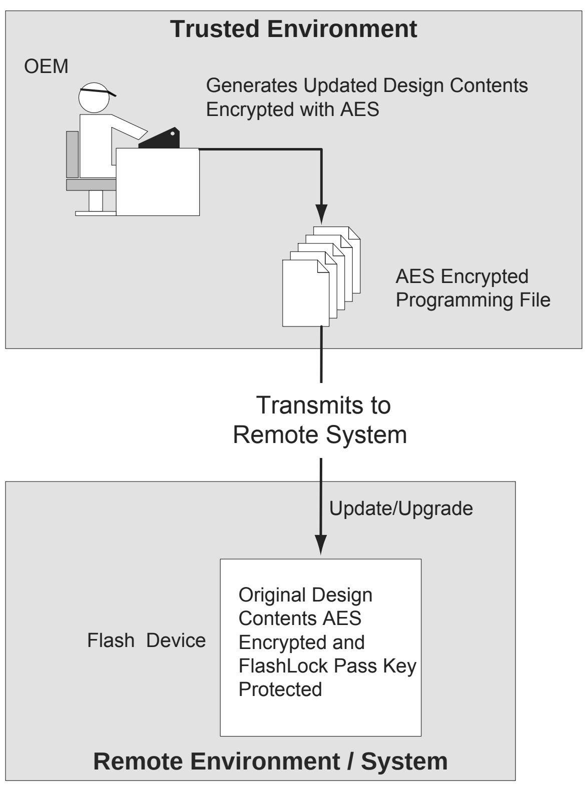


Figure 12-8 • **Application 3: Nontrusted Environment—Field Updates/Upgrades**

Generating Programming Files

Generation of the Programming File in a Trusted Environment—Application 1

As discussed in the "Application 1: Trusted Environment" section on page 309, in a trusted environment, the user can choose to program the device with plaintext bitstream content. It is possible to use plaintext for programming even when the FlashLock Pass Key option has been selected. In this application, it is not necessary to employ AES encryption protection. For AES encryption settings, refer to the next sections.

The generated programming file will include the security setting (if selected) and the plaintext programming file content for the FPGA array, FlashROM, and/or FBs. These options are indicated in Table 12-2 and Table 12-3.

Table 12-2 **• IGLOO and ProASIC3 Plaintext Security Options, No AES**

Security Protection	FlashROM Only	FPGA Core Only	Both FlashROM and FPGA
No AES / no FlashLock	✓	✓	✓
FlashLock only	✓	✓	✓
AES and FlashLock	–	–	–

Table 12-3 **• Fusion Plaintext Security Options**

Security Protection	FlashROM Only	FPGA Core Only	FB Core Only	All
No AES / no FlashLock	✓	✓	✓	✓
FlashLock	✓	✓	✓	✓
AES and FlashLock	–	–	–	–

Note: For all instructions, the programming of Flash Blocks refers to Fusion only.

For this scenario, generate the programming file as follows:

1. Select the **Silicon features to be programmed** (Security Settings, FPGA Array, FlashROM, Flash Memory Blocks), as shown in Figure 12-10 on page 314 and Figure 12-11 on page 314. Click **Next**.

   If **Security Settings** is selected (i.e., the FlashLock security Pass Key feature), an additional dialog will be displayed to prompt you to select the security level setting. If no security setting is selected, you will be directed to Step 3.

Silicon Testing and Debugging

In many applications, the design needs to be tested, debugged, and verified on real silicon or in the final embedded application. To debug and test the functionality of designs, users may need to monitor some internal logic (or nets) during device operation. The approach of adding design test pins to monitor the critical internal signals has many disadvantages, such as limiting the number of user I/Os. Furthermore, adding external I/Os for test purposes may require additional or dedicated board area for testing and debugging.

The UJTAG tiles of low power flash devices offer a flexible and cost-effective solution for silicon test and debug applications. In this solution, the signals under test are shifted out to the TDO pin of the TAP Controller. The main advantage is that all the test signals are monitored from the TDO pin; no pins or additional board-level resources are required. Figure 17-6 illustrates this technique. Multiple test nets are brought into an internal MUX architecture. The selection of the MUX is done using the contents of the TAP Controller instruction register, where individual instructions (values from 16 to 127) correspond to different signals under test. The selected test signal can be synchronized with the rising or falling edge of TCK (optional) and sent out to UTDO to drive the TDO output of JTAG.

For flash devices, TDO (the output) is configured as low slew and the highest drive strength available in the technology and/or device. Here are some examples:

1. If the device is A3P1000 and VCCI is 3.3 V, TDO will be configured as LVTTL 3.3 V output, 24 mA, low slew.
2. If the device is AGLN020 and VCCI is 1.8 V, TDO will be configured as LVCMOS 1.8 V output, 4 mA, low slew.
3. If the device is AGLE300 and VCCI is 2.5 V, TDO will be configured as LVCMOS 2.5 V output, 24 mA, low slew.

The test and debug procedure is not limited to the example in Figure 17-5 on page 369. Users can customize the debug and test interface to make it appropriate for their applications. For example, multiple test signals can be registered and then sent out through UTDO, each at a different edge of TCK. In other words, *n* signals are sampled with an F_{TCK} / *n* sampling rate. The bandwidth of the information sent out to TDO is always proportional to the frequency of TCK.

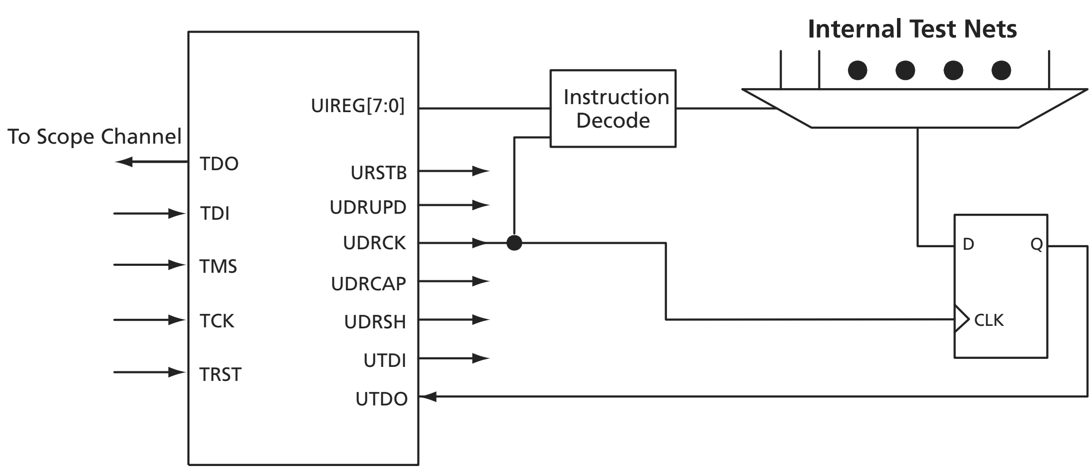


Figure 17-6 **• UJTAG Usage Example in Test and Debug Applications**