

Welcome to [E-XFL.COM](https://www.e-xfl.com)

Understanding Embedded - FPGAs (Field Programmable Gate Array)

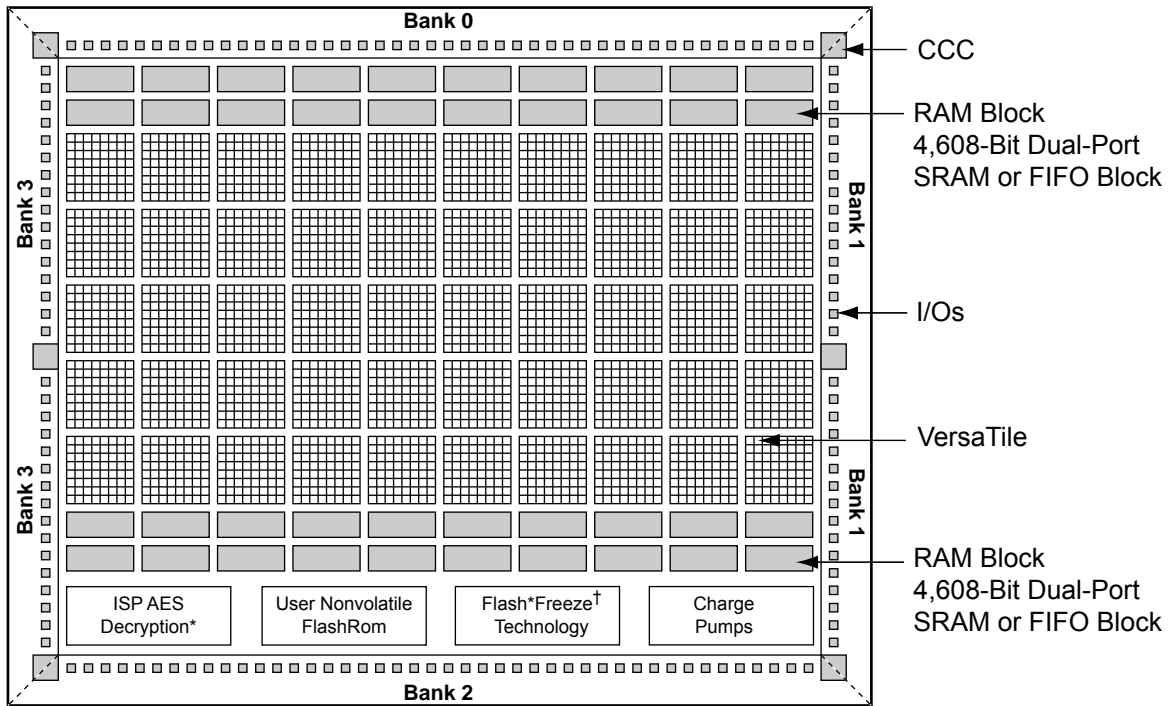
Embedded - FPGAs, or Field Programmable Gate Arrays, are advanced integrated circuits that offer unparalleled flexibility and performance for digital systems. Unlike traditional fixed-function logic devices, FPGAs can be programmed and reprogrammed to execute a wide array of logical operations, enabling 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.

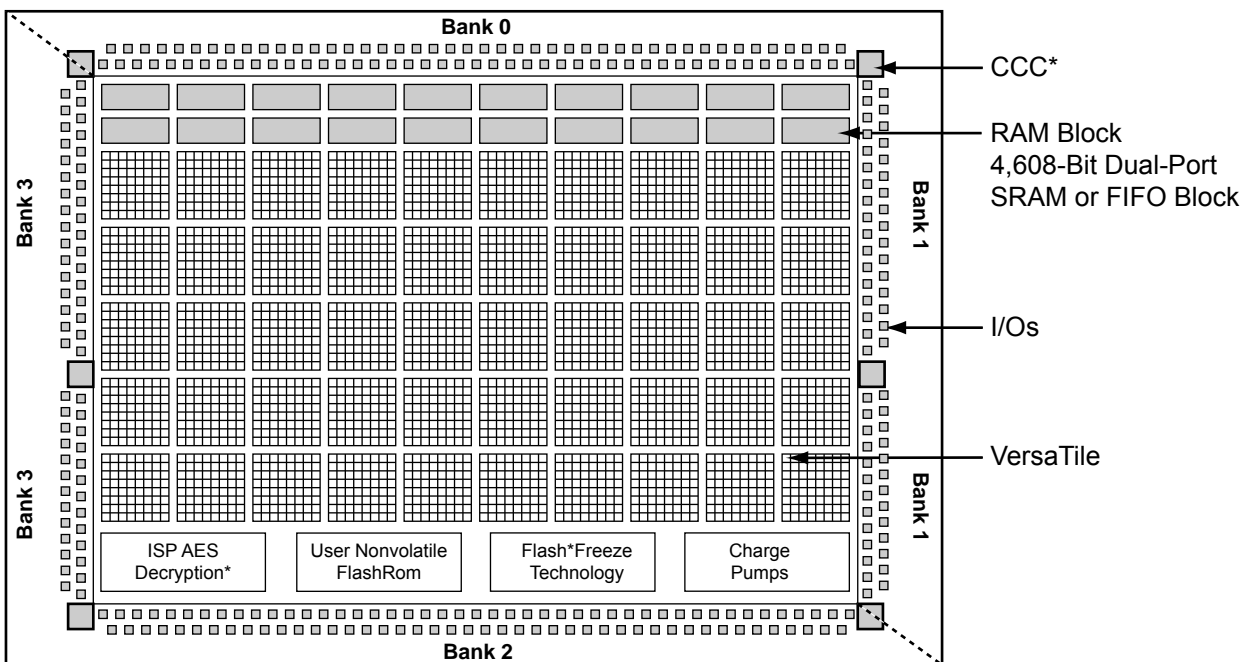
Details

Product Status	Active
Number of LABs/CLBs	-
Number of Logic Elements/Cells	-
Total RAM Bits	-
Number of I/O	49
Number of Gates	20000
Voltage - Supply	1.425V ~ 1.575V
Mounting Type	Surface Mount
Operating Temperature	-20°C ~ 85°C (TJ)
Package / Case	68-VFQFN Exposed Pad
Supplier Device Package	68-QFN (8x8)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/a3pn020-1qng68



Note: Flash*Freeze technology only applies to IGLOO and ProASIC3L families.

Figure 1-5 • IGLOO, IGLOO nano, ProASIC3 nano, and ProASIC3/L Device Architecture Overview with Four I/O Banks (AGL600 device is shown)



Note: * AGLP030 does not contain a PLL or support AES security.

Figure 1-6 • IGLOO PLUS Device Architecture Overview with Four I/O Banks

You can control the maximum number of shared instances allowed for the legalization to take place using the Compile Option dialog box shown in Figure 3-17. Refer to Libero SoC / Designer online help for details on the Compile Option dialog box. A large number of shared instances most likely indicates a floorplanning problem that you should address.

Figure 3-17 • Shared Instances in the Compile Option Dialog Box

Designer Flow for Global Assignment

To achieve the desired result, pay special attention to global management during synthesis and place-and-route. The current Synplify tool does not insert more than six global buffers in the netlist by default. Thus, the default flow will not assign any signal to the quadrant global network. However, you can use attributes in Synplify and increase the default global macro assignment in the netlist. Designer v6.2 supports automatic quadrant global assignment, which was not available in Designer v6.1. Layout will make the choice to assign the correct signals to global. However, you can also utilize PDC and perform manual global assignment to overwrite any automatic assignment. The following step-by-step suggestions guide you in the layout of your design and help you improve timing in Designer:

1. Run Compile and check the Compile report. The Compile report has global information in the "Device Utilization" section that describes the number of chip and quadrant signals in the design. A "Net Report" section describes chip global nets, quadrant global nets, local clock nets, a list of nets listed by fanout, and net candidates for local clock assignment. Review this information. Note that YB or YC are counted as global only when they are used in isolation; if you use YB only and not GLB, this net is not shown in the global/quadrant nets report. Instead, it appears in the Global Utilization report.
2. If some signals have a very high fanout and are candidates for global promotion, promote those signals to global using the compile options or PDC commands. Figure 3-18 on page 54 shows the Globals Management section of the compile options. Select **Promote regular nets whose fanout is greater than** and enter a reasonable value for fanouts.

Using Spines of Occupied Global Networks

When a signal is assigned to a global network, the flash switches are programmed to set the MUX select lines (explained in the "Clock Aggregation Architecture" section on page 45) to drive the spines of that network with the global net. However, if the global net is restricted from reaching into the scope of a spine, the MUX drivers of that spine are available for other high-fanout or critical signals (Figure 3-20).

For example, if you want to limit the CLK1_c signal to the left half of the chip and want to use the right side of the same global network for CLK2_c, you can add the following PDC commands:

```
define_region -name region1 -type inclusive 0 0 34 29
assign_net_macros region1 CLK1_c
assign_local_clock -net CLK2_c -type chip B2
```

Figure 3-20 • Design Example Using Spines of Occupied Global Networks

Conclusion

IGLOO, Fusion, and ProASIC3 devices contain 18 global networks: 6 chip global networks and 12 quadrant global networks. These global networks can be segmented into local low-skew networks called spines. The spines provide low-skew networks for the high-fanout signals of a design. These allow you up to 252 different internal/external clocks in an A3PE3000 device. This document describes the architecture for the global network, plus guidelines and methodologies in assigning signals to globals and spines.

Related Documents

User's Guides

IGLOO, ProASIC3, SmartFusion, and Fusion Macro Library Guide

http://www.microsemi.com/soc/documents/pa3_libguide_ug.pdf

CLKDLY Macro Usage

When a CLKDLY macro is used in a CCC location, the programmable delay element is used to allow the clock delays to go to the global network. In addition, the user can bypass the PLL in a CCC location integrated with a PLL, but use the programmable delay that is associated with the global network by instantiating the CLKDLY macro. The same is true when using programmable delay elements in a CCC location with no PLLs (the user needs to instantiate the CLKDLY macro). There is no difference between the programmable delay elements used for the PLL and the CLKDLY macro. The CCC will be configured to use the programmable delay elements in accordance with the macro instantiated by the user.

As an example, if the PLL is not used in a particular CCC location, the designer is free to specify up to three CLKDLY macros in the CCC, each of which can have its own input frequency and delay adjustment options. If the PLL core is used, assuming output to only one global clock network, the other two global clock networks are free to be used by either connecting directly from the global inputs or connecting from one or two CLKDLY macros for programmable delay.

The programmable delay elements are shown in the block diagram of the PLL block shown in Figure 4-6 on page 71. Note that any CCC locations with no PLL present contain only the programmable delay blocks going to the global networks (labeled "Programmable Delay Type 2"). Refer to the "Clock Delay Adjustment" section on page 86 for a description of the programmable delay types used for the PLL. Also refer to Table 4-14 on page 94 for Programmable Delay Type 1 step delay values, and Table 4-15 on page 94 for Programmable Delay Type 2 step delay values. CCC locations with a PLL present can be configured to utilize only the programmable delay blocks (Programmable Delay Type 2) going to the global networks A, B, and C.

Global network A can be configured to use only the programmable delay element (bypassing the PLL) if the PLL is not used in the design. Figure 4-6 on page 71 shows a block diagram of the PLL, where the programmable delay elements are used for the global networks (Programmable Delay Type 2).

PLL Macro Signal Descriptions

The PLL macro supports two inputs and up to six outputs. Table 4-3 gives a description of each signal.

Table 4-3 • Input and Output Signals of the PLL Block

Signal	Name	I/O	Description
CLKA	Reference Clock	Input	Reference clock input for PLL core; input clock for primary output clock, GLA
OADIVRST	Reset Signal for the Output Divider A	Input	For Fusion only. OADIVRST can be used when you bypass the PLL core (i.e., OAMUX = 001). The purpose of the OADIVRST signals is to reset the output of the final clock divider to synchronize it with the input to that divider when the PLL is bypassed. The signal is active on a low to high transition. The signal must be low for at least one divider input. If PLL core is used, this signal is "don't care" and the internal circuitry will generate the reset signal for the synchronization purpose.
OADIVHALF	Output A Division by Half	Input	For Fusion only. Active high. Division by half feature. This feature can only be used when users bypass the PLL core (i.e., OAMUX = 001) and the RC Oscillator (RCOSC) drives the CLKA input. This can be used to divide the 100 MHz RC oscillator by a factor of 1.5, 2.5, 3.5, 4.5 ... 14.5). Refer to Table 4-18 on page 95 for more information.
EXTFB	External Feedback	Input	Allows an external signal to be compared to a reference clock in the PLL core's phase detector.
POWERDOWN	Power Down	Input	Active low input that selects power-down mode and disables the PLL. With the POWERDOWN signal asserted, the PLL core sends 0 V signals on all of the outputs.
GLA	Primary Output	Output	Primary output clock to respective global/quadrant clock networks
GLB	Secondary 1 Output	Output	Secondary 1 output clock to respective global/quadrant clock networks
YB	Core 1 Output	Output	Core 1 output clock to local routing network
GLC	Secondary 2 Output	Output	Secondary 2 output clock to respective global/quadrant clock networks
YC	Core 2 Output	Output	Core 2 output clock to local routing network
LOCK	PLL Lock Indicator	Output	Active high signal indicating that steady-state lock has been achieved between CLKA and the PLL feedback signal

Input Clock

The inputs to the input reference clock (CLKA) of the PLL can come from global input pins, regular I/O pins, or internally from the core. For Fusion families, the input reference clock can also be from the embedded RC oscillator or crystal oscillator.

Global Output Clocks

GLA (Primary), GLB (Secondary 1), and GLC (Secondary 2) are the outputs of Global Multiplexer 1, Global Multiplexer 2, and Global Multiplexer 3, respectively. These signals (GLx) can be used to drive the high-speed global and quadrant networks of the low power flash devices.

A global multiplexer block consists of the input routing for selecting the input signal for the GLx clock and the output multiplexer, as well as delay elements associated with that clock.

Core Output Clocks

YB and YC are known as Core Outputs and can be used to drive internal logic without using global network resources. This is especially helpful when global network resources must be conserved and utilized for other timing-critical paths.

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.

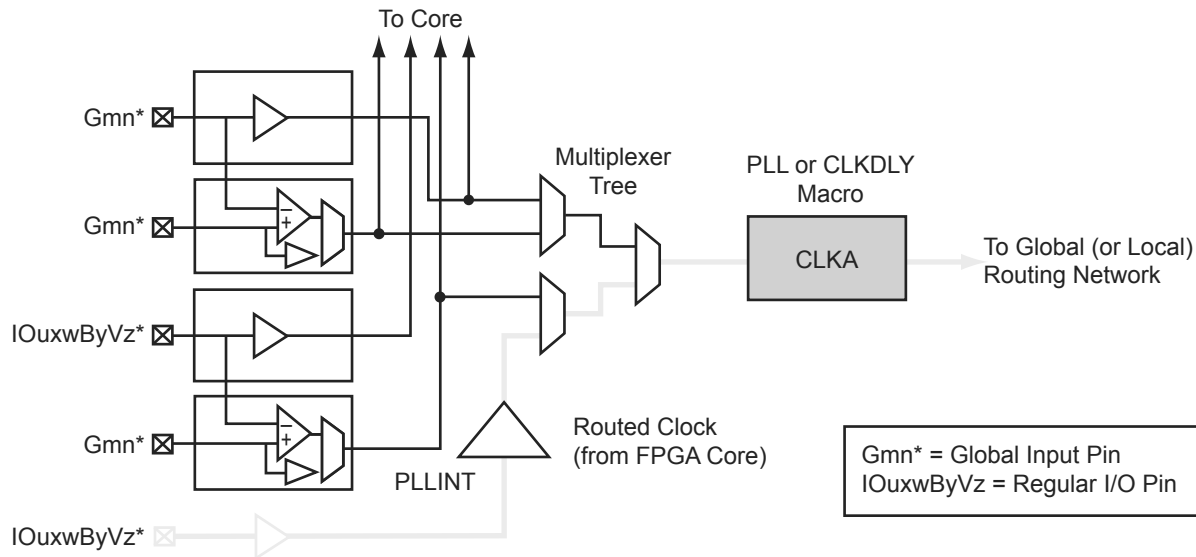


Figure 4-11 • Illustration of External I/O Usage

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.

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 ($T_a = +125^\circ\text{C}$, $V_{CCA} = 1.4\text{ V}$)	Typical ($T_a = +25^\circ\text{C}$, $V_{CCA} = 1.5\text{ V}$)	Maximum ($T_a = -55^\circ\text{C}$, $V_{CCA} = 1.6\text{ V}$)
-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.

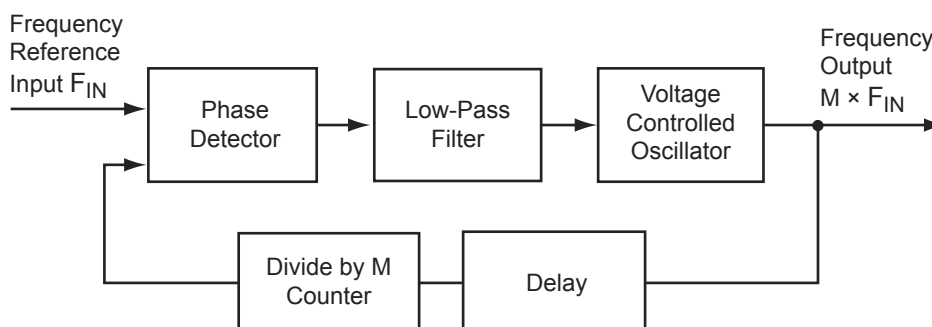


Figure 4-19 • Simplified PLL Core with Feedback Divider and Delay

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

FlashROM Applications

The SmartGen core generator is used to configure FlashROM content. You can configure each page independently. SmartGen enables you to create and modify regions within a page; these regions can be 1 to 16 bytes long (Figure 5-4).

		Byte Number in Page															
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Page Number	7																
	6																
	5																
	4																
	3																
	2																
	1																
	0																

Figure 5-4 • FlashROM Configuration

The FlashROM content can be changed independently of the FPGA core content. It can be easily accessed and programmed via JTAG, depending on the security settings of the device. The SmartGen core generator enables each region to be independently updated (described in the "Programming and Accessing FlashROM" section on page 122). This enables you to change the FlashROM content on a per-part basis while keeping some regions "constant" for all parts. These features allow the FlashROM to be used in diverse system applications. Consider the following possible uses of FlashROM:

- Internet protocol (IP) addressing (wireless or fixed)
- System calibration settings
- Restoring configuration after unpredictable system power-down
- Device serialization and/or inventory control
- Subscription-based business models (e.g., set-top boxes)
- Secure key storage
- Asset management tracking
- Date stamping
- Version management

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 128).
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 128.
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.

Date	Changes	Page
v1.1 (continued)	Table 6-1 • Flash-Based FPGAs and associated text were updated to include the IGLOO PLUS family. The "IGLOO Terminology" section and "ProASIC3 Terminology" section are new.	134
	The text introducing Table 6-8 • Memory Availability per IGLOO and ProASIC3 Device was updated to replace "A3P030 and AGL030" with "15 k and 30 k gate devices." Table 6-8 • Memory Availability per IGLOO and ProASIC3 Device was updated to remove AGL400 and AGL1500 and include IGLOO PLUS and ProASIC3L devices.	146

I/O Bank Architecture and CCC Naming Conventions

The nano products feature varying bank architectures which have been optimized to balance silicon area with I/O and clocking flexibility. The A standard naming scheme is used to illustrate the I/O Bank architecture and the CCCs associated with each architecture.

Table 7-16 • A Standard Naming Scheme

Name	Description
Bank x	Refers to the specific bank number within which an I/O resides
CCC	Clock Condition Circuit with simple clock delay operations as well as clock spine access
CCC-GL	Clock Condition Circuit with Global Locations for chip reach clocking. These CCCs support programmable delays but do not have an integrated PLL.
CCC-PLL	Clock Condition Circuit with integrated PLL and programmable delays
Chip Reach	Access to chip global lines
Quadrant Reach	Access to quadrant global lines

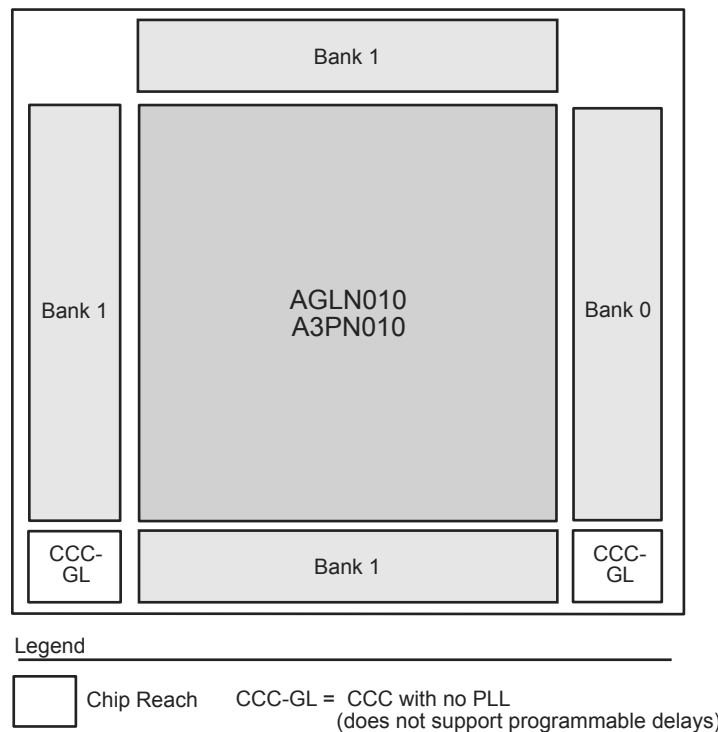


Figure 7-9 • I/O Bank Architecture of AGLN010 and A3PN010 Devices

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 8-6 on page 193).
- 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.

VREF for GTL+ 3.3 V



Figure 8-13 • Selecting VREF Voltage for the I/O Bank

Assigning VREF Pins for a Bank

The user can use default pins for VREF. In this case, select the **Use default pins for VREFs** check box (Figure 8-13). This option guarantees full VREF coverage of the bank. The equivalent PDC command is as follows:

```
set_vref_default [bank name]
```

To be able to choose VREF pins, adequate VREF pins must be created to allow legal placement of the compatible voltage-referenced I/Os.

To assign VREF pins manually, the PDC command is as follows:

```
set_vref -bank [bank name] [package pin numbers]
```

For ChipPlanner/PinEditor to show the range of a VREF pin, perform the following steps:

1. Assign VCCI to a bank using **MVN > Edit > I/O Bank Settings**.
2. Open **ChipPlanner**. Zoom in on an I/O package pin in that bank.
3. Highlight the pin and then right-click. Choose **Use Pin for VREF**.

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

Verilog

```
module Inbuf_dds(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_dds(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
```

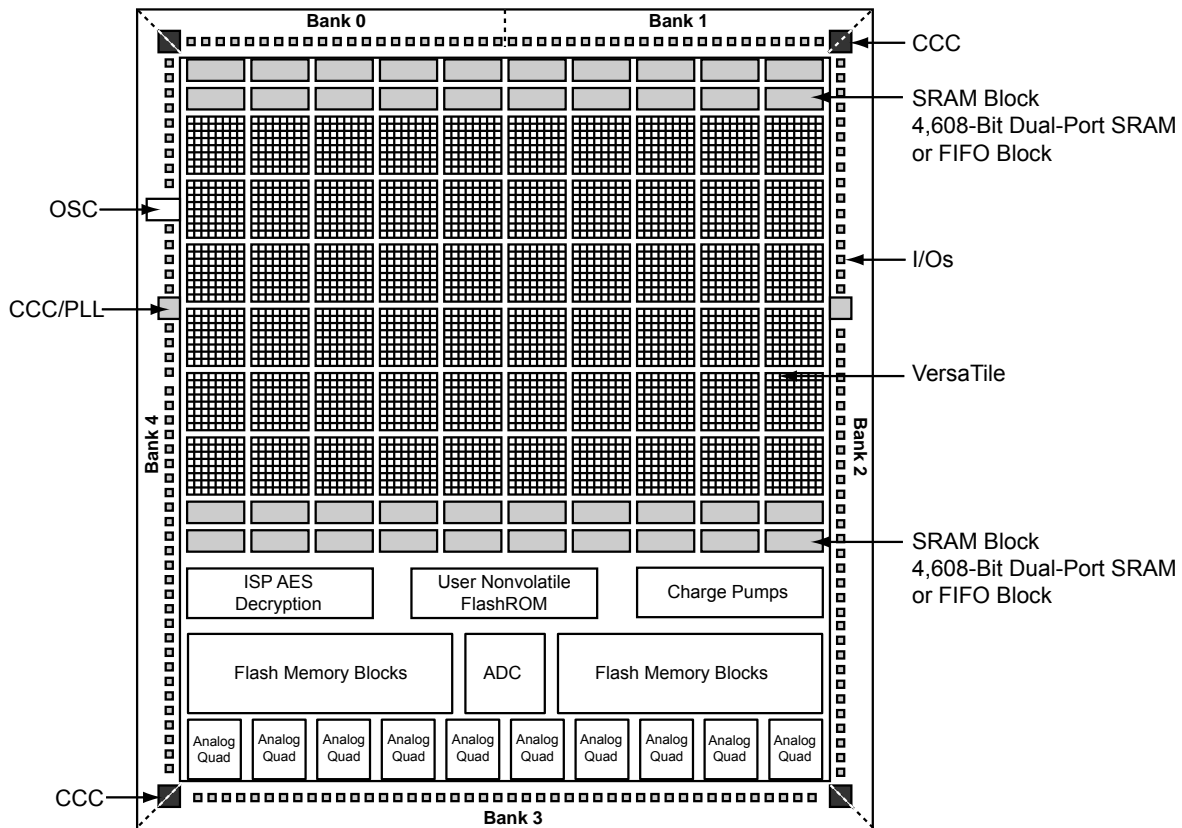


Figure 11-3 • Block Representation of the AES Decryption Core in a Fusion AFS600 FPGA

Security Features

IGLOO and ProASIC3 devices have two entities inside: FlashROM and the FPGA core fabric. Fusion devices contain three entities: FlashROM, FBs, and the FPGA core fabric. The parts can be programmed or updated independently with a STAPL programming file. The programming files can be AES-encrypted or plaintext. This allows maximum flexibility in providing security to the entire device. Refer to the "Programming Flash Devices" section on page 221 for information on the FlashROM structure.

Unlike SRAM-based FPGA devices, which require a separate boot PROM to store programming data, low power flash devices are nonvolatile, and the secured configuration data is stored in on-chip flash cells that are part of the FPGA fabric. Once programmed, this data is an inherent part of the FPGA array and does not need to be loaded at system power-up. SRAM-based FPGAs load the configuration bitstream upon power-up; therefore, the configuration is exposed and can be read easily.

The built-in FPGA core, FBs, and FlashROM support programming files encrypted with the 128-bit AES (FIPS-192) block ciphers. The AES key is stored in dedicated, on-chip flash memory and can be programmed before the device is shipped to other parties (allowing secure remote field updates).

Security in ARM-Enabled Low Power Flash Devices

There are slight differences between the regular flash devices and the ARM®-enabled flash devices, which have the M1 and M7 prefix.

The AES key is used by Microsemi and preprogrammed into the device to protect the ARM IP. As a result, the design is encrypted along with the ARM IP, according to the details below.

Note: The settings in this figure are used to show the generation of an AES-encrypted programming file for the FPGA array, FlashROM, and FB contents. One or all locations may be selected for encryption.

Figure 11-17 • Settings to Program a Device Secured with FlashLock and using AES Encryption

Choose the **High** security level to reprogram devices using both the FlashLock Pass Key and AES key protection (Figure 11-18 on page 255). Enter the AES key and click **Next**.

A device that has already been secured with FlashLock and has an AES key loaded must recognize the AES key to program the device and generate a valid bitstream in authentication. The FlashLock Key is only required to unlock the device and change the security settings.

This is what makes it possible to program in an untrusted environment. The AES key is protected inside the device by the FlashLock Key, so you can only program if you have the correct AES key. In fact, the AES key is not in the programming file either. It is the key used to encrypt the data in the file. The same key previously programmed with the FlashLock Key matches to decrypt the file.

An AES-encrypted file programmed to a device without FlashLock would not be secure, since without FlashLock to protect the AES key, someone could simply reprogram the AES key first, then program with any AES key desired or no AES key at all. This option is therefore not available in the software.

Date	Changes	Page
July 2010 (continued)	The "Chain Integrity Test Error Analyze Chain Failure" section was renamed to the "Scan Chain Failure" section, and the Analyze Chain command was changed to Scan Chain. It was noted that occasionally a faulty programmer can cause scan chain failures.	272
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.	265
v1.4 (December 2008)	The "ISP Architecture" section was revised to include information about core voltage for IGLOO V2 and ProASIC3L devices, as well as 50 mV increments allowable in Designer software.	261
	IGLOO nano and ProASIC3 nano devices were added to Table 12-1 • Flash-Based FPGAs Supporting ISP.	262
	A second capacitor was added to Figure 12-6 • Board Layout and Programming Header Top View.	271
v1.3 (October 2008)	The "ISP Support in Flash-Based Devices" section was revised to include new families and make the information more concise.	262
v1.2 (June 2008)	The following changes were made to the family descriptions in Table 12-1 • Flash-Based FPGAs Supporting ISP: <ul style="list-style-type: none"> ProASIC3L was updated to include 1.5 V. The number of PLLs for ProASIC3E was changed from five to six. 	262
v1.1 (March 2008)	The "ISP Architecture" section was updated to included the IGLOO PLUS family in the discussion of family-specific support. The text, "When 1.2 V is used, the device can be reprogrammed in-system at 1.5 V only," was revised to state, "Although the device can operate at 1.2 V core voltage, the device can only be reprogrammed when all supplies (VCC, VCCI, and VJTAG) are at 1.5 V."	261
	The "ISP Support in Flash-Based Devices" section and Table 12-1 • Flash-Based FPGAs Supporting ISP were updated to include the IGLOO PLUS family. The "IGLOO Terminology" section and "ProASIC3 Terminology" section are new.	262
	The "Security" section was updated to mention that 15 k gate devices do not have a built-in 128-bit decryption core.	264
	Table 12-2 • Power Supplies was revised to remove the Normal Operation column and add a table note stating, "All supply voltages should be at 1.5 V or higher, regardless of the setting during normal operation."	263
	The "ISP Programming Header Information" section was revised to change FP3-26PIN-ADAPTER to FP3-10PIN-ADAPTER-KIT. Table 12-3 • Programming Header Ordering Codes was updated with the same change, as well as adding the part number FFSD-05-D-06.00-01-N, a 10-pin cable with 50-mil-pitch sockets.	269
	The "Board-Level Considerations" section was updated to describe connecting two capacitors in parallel across VPUMP and GND for proper programming.	271
v1.0 (January 2008)	Information was added to the "Programming Voltage (VPUMP) and VJTAG" section about the JTAG interface pin.	263
51900055-2/7.06	ACTgen was changed to SmartGen.	N/A
	In Figure 12-6 • Board Layout and Programming Header Top View, the order of the text was changed to: VJTAG from the target board VCCI from the target board VCC from the target board	271

14 – Microprocessor Programming of Microsemi's Low Power Flash Devices

Introduction

The Fusion, IGLOO, and ProASIC3 families of flash FPGAs support in-system programming (ISP) with the use of a microprocessor. Flash-based FPGAs store their configuration information in the actual cells within the FPGA fabric. SRAM-based devices need an external configuration memory, and hybrid nonvolatile devices store the configuration in a flash memory inside the same package as the SRAM FPGA. Since the programming of a true flash FPGA is simpler, requiring only one stage, it makes sense that programming with a microprocessor in-system should be simpler than with other SRAM FPGAs. This reduces bill-of-materials costs and printed circuit board (PCB) area, and increases system reliability.

Nonvolatile flash technology also gives the low power flash devices the advantage of a secure, low power, live-at-power-up, and single-chip solution. Low power flash devices are reprogrammable and offer time-to-market benefits at an ASIC-level unit cost. These features enable engineers to create high-density systems using existing ASIC or FPGA design flows and tools.

This document is an introduction to microprocessor programming only. To explain the difference between the options available, user's guides for DirectC and STAPL provide more detail on implementing each style.

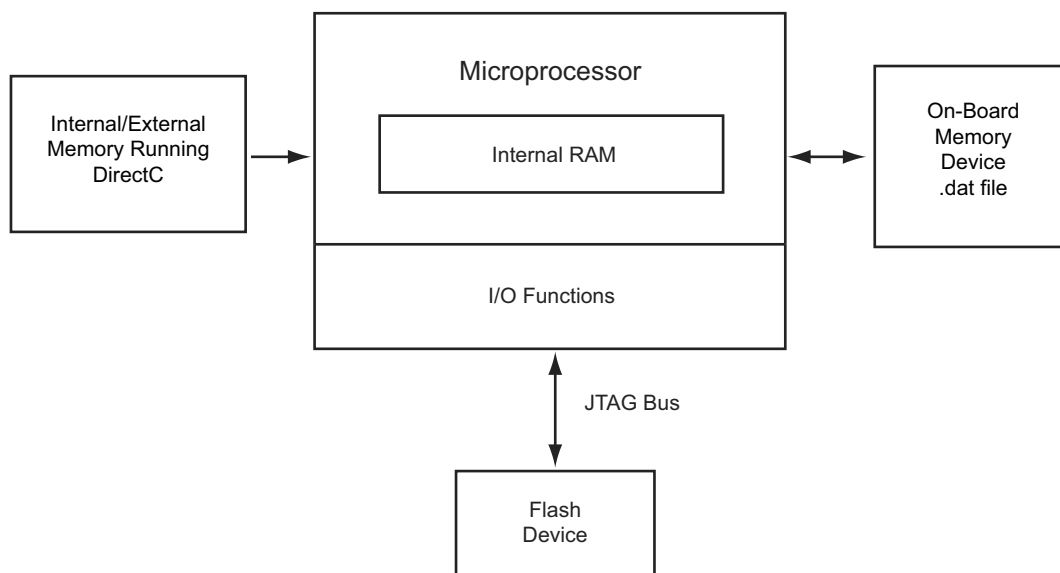


Figure 14-1 • ISP Using Microprocessor

17 – Power-Up/-Down Behavior of Low Power Flash Devices

Introduction

Microsemi's low power flash devices are flash-based FPGAs manufactured on a 0.13 μm process node. These devices offer a single-chip, reprogrammable solution and support Level 0 live at power-up (LAPU) due to their nonvolatile architecture.

Microsemi's low power flash FPGA families are optimized for logic area, I/O features, and performance. IGLOO[®] devices are optimized for power, making them the industry's lowest power programmable solution. IGLOO PLUS FPGAs offer enhanced I/O features beyond those of the IGLOO ultra-low power solution for I/O-intensive low power applications. IGLOO nano devices are the industry's lowest-power cost-effective solution. ProASIC3[®]L FPGAs balance low power with high performance. The ProASIC3 family is Microsemi's high-performance flash FPGA solution. ProASIC3 nano devices offer the lowest-cost solution with enhanced I/O capabilities.

Microsemi's low power flash devices exhibit very low transient current on each power supply during power-up. The peak value of the transient current depends on the device size, temperature, voltage levels, and power-up sequence.

The following devices can have inputs driven in while the device is not powered:

- IGLOO (AGL015 and AGL030)
- IGLOO nano (all devices)
- IGLOO PLUS (AGLP030, AGLP060, AGLP125)
- IGLOOe (AGLE600, AGLE3000)
- ProASIC3L (A3PE3000L)
- ProASIC3 (A3P015, A3P030)
- ProASIC3 nano (all devices)
- ProASIC3E (A3PE600, A3PE1500, A3PE3000)
- Military ProASIC3EL (A3PE600L, A3PE3000L, but not A3P1000)
- RT ProASIC3 (RT3PE600L, RT3PE3000L)

The driven I/Os do not pull up power planes, and the current draw is limited to very small leakage current, making them suitable for applications that require cold-sparing. These devices are hot-swappable, meaning they can be inserted in a live power system.¹

1. For more details on the levels of hot-swap compatibility in Microsemi's low power flash devices, refer to the "Hot-Swap Support" section in the I/O Structures chapter of the FPGA fabric user's guide for the device you are using.

Figure 17-3 • I/O State when VCCI Is Powered before VCC

Power-Up to Functional Time

At power-up, device I/Os exit the tristate mode and become functional once the last voltage supply in the power-up sequence (VCCI or VCC) reaches its functional activation level. The power-up-to-functional time is the time it takes for the last supply to power up from zero to its functional level. Note that the functional level of the power supply during power-up may vary slightly within the specification at different ramp-rates. Refer to Table 17-2 for the functional level of the voltage supplies at power-up.

Typical I/O behavior during power-up-to-functional time is illustrated in Figure 17-2 on page 311 and Figure 17-3.

Table 17-2 • Power-Up Functional Activation Levels for VCC and VCCI

Device	VCC Functional Activation Level (V)	VCCI Functional Activation Level (V)
ProASIC3, ProASIC3 nano, IGLOO, IGLOO nano, IGLOO PLUS, and ProASIC3L devices running at VCC = 1.5 V*	0.85 V \pm 0.25 V	0.9 V \pm 0.3 V
IGLOO, IGLOO nano, IGLOO PLUS, and ProASIC3L devices running at VCC = 1.2 V*	0.85 V \pm 0.2 V	0.9 V \pm 0.15 V

Note: *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.

Microsemi's low power flash devices meet Level 0 LAPU; that is, they can be functional prior to V_{CC} reaching the regulated voltage required. This important advantage distinguishes low power flash devices from their SRAM-based counterparts. SRAM-based FPGAs, due to their volatile technology, require hundreds of milliseconds after power-up to configure the design bitstream before they become functional. Refer to Figure 17-4 on page 313 and Figure 17-5 on page 314 for more information.