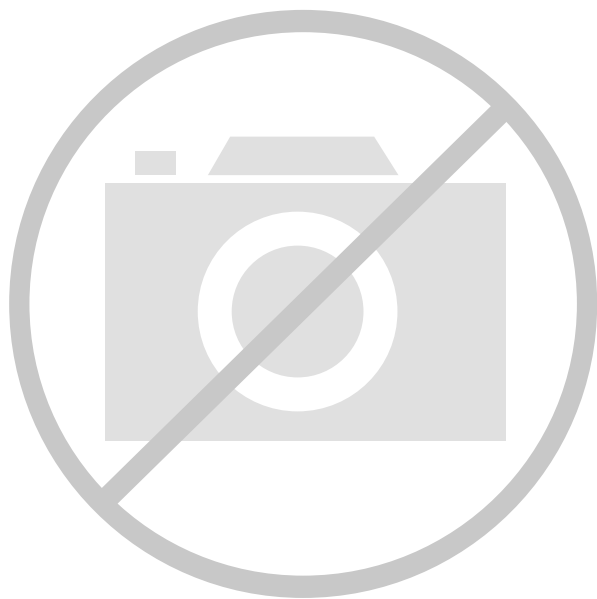




Welcome to [E-XFL.COM](#)

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	221
Number of Gates	3000000
Voltage - Supply	1.14V ~ 1.575V
Mounting Type	Surface Mount
Operating Temperature	-40°C ~ 100°C (TJ)
Package / Case	324-BGA
Supplier Device Package	324-FBGA (19x19)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/a3pe3000l-fg324i

Low Power Modes Overview

Table 2-2 summarizes the low power modes that achieve power consumption reduction when the FPGA or system is idle.

Table 2-2 • Power Modes Summary

Mode		VCCI	VCC	Core	Clocks	ULSICC Macro	To Enter Mode	To Resume Operation	Trigger
Active		On	On	On	On	N/A	Initiate clock	None	–
Static	Idle	On	On	On	Off	N/A	Stop clock	Initiate clock	External
	Flash*Freeze type 1	On	On	On	On*	N/A	Assert FF pin	Deassert FF pin	External
	Flash*Freeze type 2	On	On	On	On*	Used to enter Flash*Freeze mode	Assert FF pin and assert LSICC	Deassert FF pin	External
Sleep		On	Off	Off	Off	N/A	Shut down VCC	Turn on VCC supply	External
Shutdown		Off	Off	Off	Off	N/A	Shut down VCC and VCCI supplies	Turn on VCC and VCCI supplies	External

* External clocks can be left toggling while the device is in Flash*Freeze mode. Clocks generated by the embedded PLL will be turned off automatically.

Static (Idle) Mode

In Static (Idle) mode, none of the clock inputs is switching, and static power is the only power consumed by the device. This mode can be achieved by switching off the incoming clocks to the FPGA, thus benefitting from reduced power consumption. In addition, I/Os draw only minimal leakage current. In this mode, embedded SRAM, I/Os, and registers retain their values so the device can enter and exit this mode just by switching the clocks on or off.

If the device-embedded PLL is used as the clock source, Static (Idle) mode can easily be entered by pulling the PLL POWERDOWN pin LOW (active Low), which will turn off the PLL.

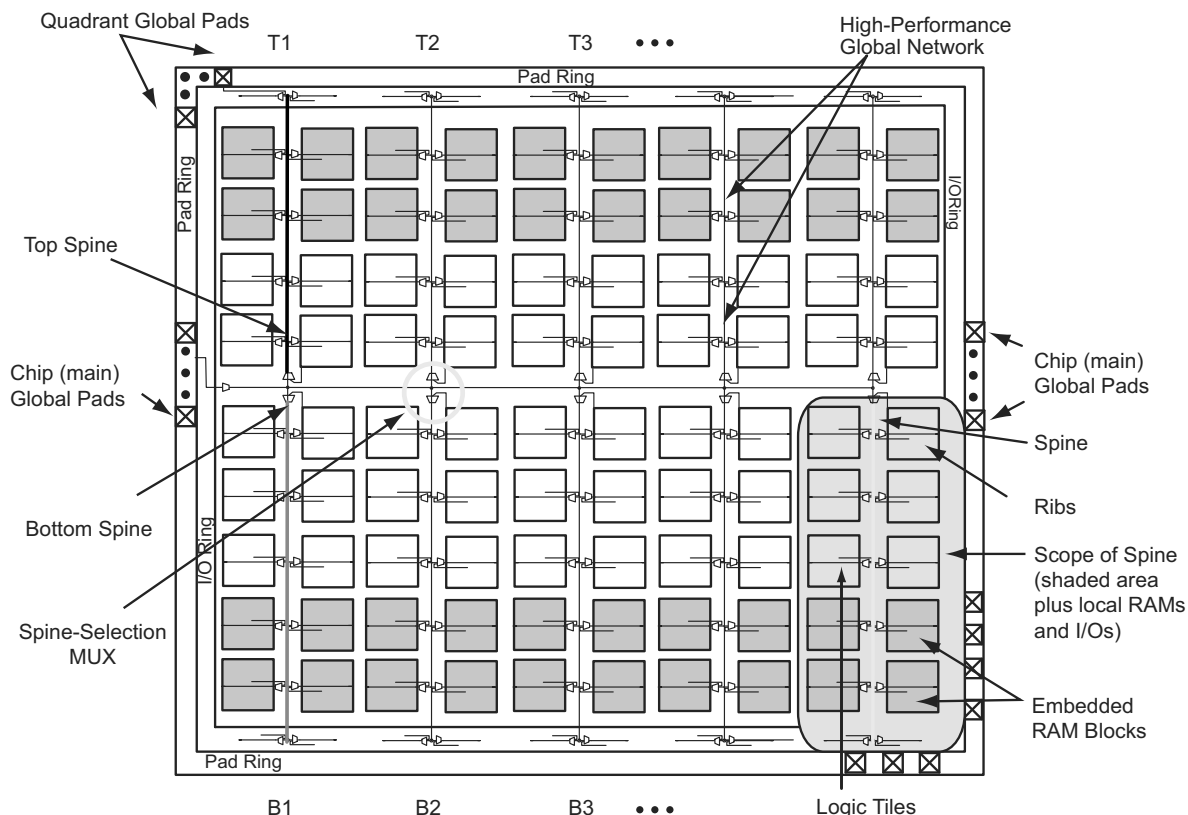
VersaNet Global Network Distribution

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 3-6 shows all nine global inputs for the location A connected to the top left quadrant global network via CCC.

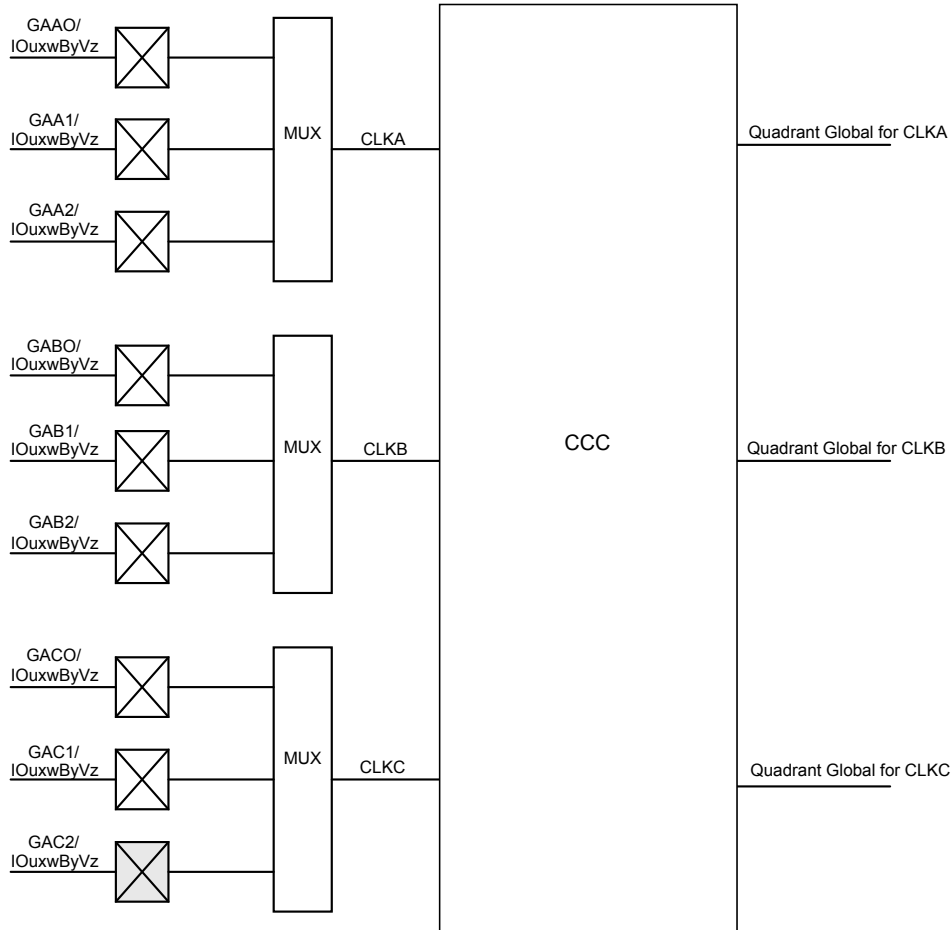


Figure 3-6 • Global Inputs

Since each bank can have a different I/O standard, the user should be careful to choose the correct global I/O for the design. There are 54 global pins available to access 18 global networks. For the single-ended and voltage-referenced I/O standards, you can use any of these three available I/Os to access the global network. For differential I/O standards such as LVDS and LVPECL, the I/O macro needs to be placed on (A0, A1), (B0, B1), (C0, C1), or a similar location. The unassigned global I/Os can be used as regular I/Os. Note that pin names starting with GF and GC are associated with the chip global networks, and GA, GB, GD, and GE are used for quadrant global networks. Table 3-2 on page 54 and Table 3-3 on page 55 show the general chip and quadrant global pin names.

Using Clock Aggregation

Clock aggregation allows for multi-spine clock domains to be assigned using hardwired connections, without adding any extra skew. A MUX tree, shown in Figure 3-8, provides the necessary flexibility to allow long lines, local resources, or I/Os to access domains of one, two, or four global spines. Signal access to the clock aggregation system is achieved through long-line resources in the central rib in the center of the die, and also through local resources in the north and south ribs, allowing I/Os to feed directly into the clock system. As Figure 3-9 indicates, this access system is contiguous.

There is no break in the middle of the chip for the north and south I/O VersaNet access. This is different from the quadrant clocks located in these ribs, which only reach the middle of the rib.

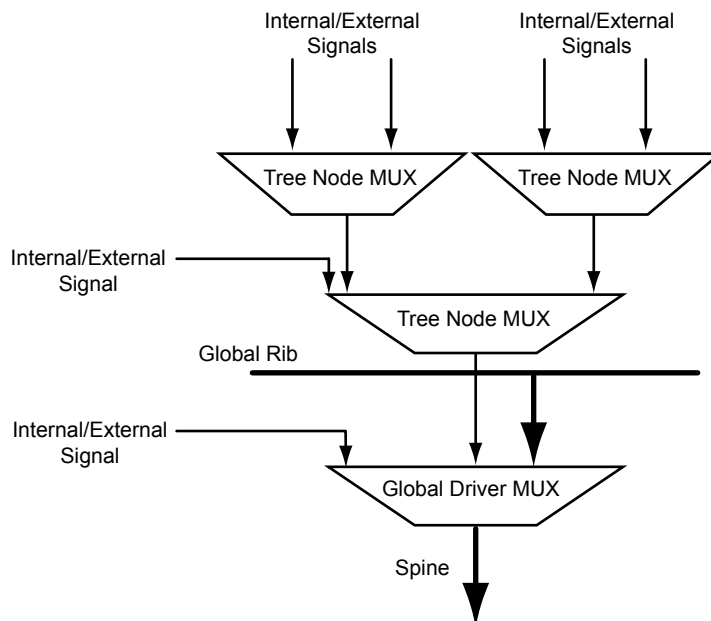


Figure 3-8 • Spine Selection MUX of Global Tree

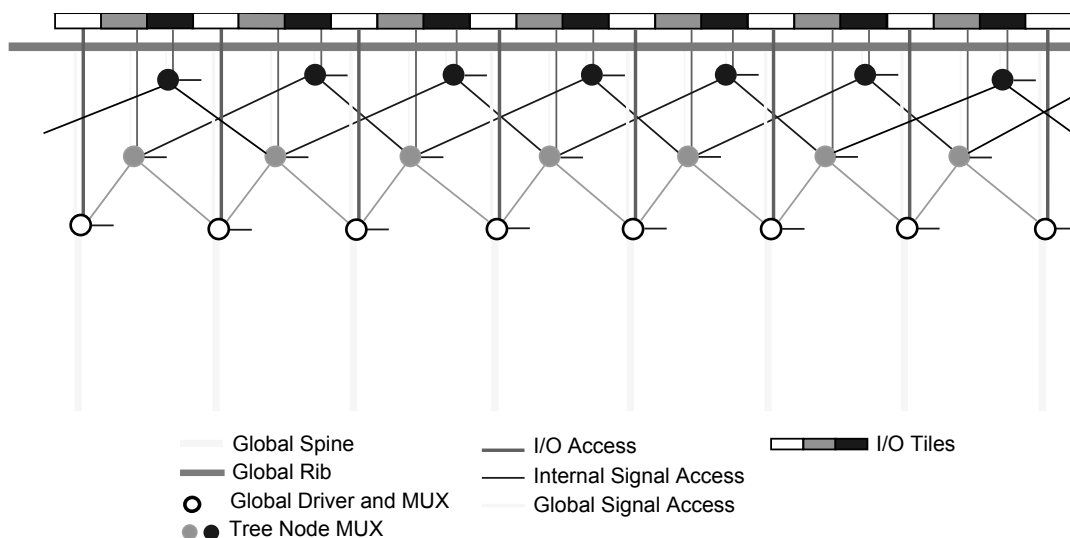


Figure 3-9 • Clock Aggregation Tree Architecture

Clock Aggregation Architecture

This clock aggregation feature allows a balanced clock tree, which improves clock skew. The physical regions for clock aggregation are defined from left to right and shift by one spine. For chip global networks, there are three types of clock aggregation available, as shown in Figure 3-10:

- Long lines that can drive up to four adjacent spines (A)
- Long lines that can drive up to two adjacent spines (B)
- Long lines that can drive one spine (C)

There are three types of clock aggregation available for the quadrant spines, as shown in Figure 3-10:

- I/Os or local resources that can drive up to four adjacent spines
- I/Os or local resources that can drive up to two adjacent spines
- I/Os or local resources that can drive one spine

As an example, A3PE600 and AFS600 devices have twelve spine locations: T1, T2, T3, T4, T5, T6, B1, B2, B3, B4, B5, and B6. Table 3-7 shows the clock aggregation you can have in A3PE600 and AFS600.

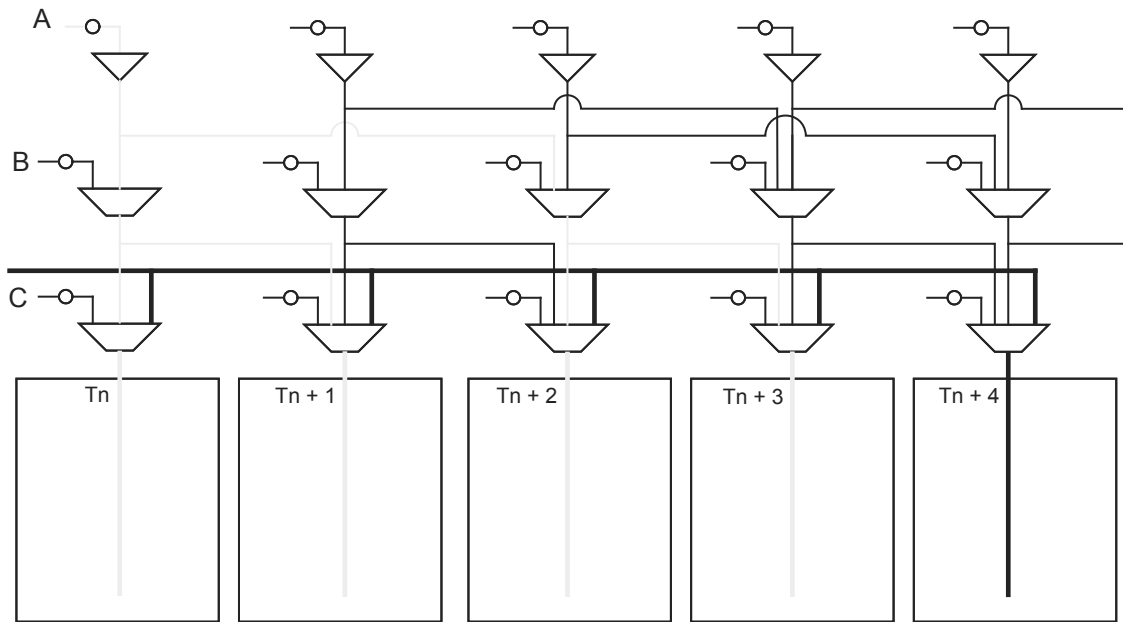


Figure 3-10 • Four Spines Aggregation

Table 3-7 • Spine Aggregation in A3PE600 or AFS600

Clock Aggregation	Spine
1 spine	T1, T2, T3, T4, T5, T6, B1, B2, B3, B4, B5, B6
2 spines	T1:T2, T2:T3, T3:T4, T4:T5, T5:T6, B1:B2, B2:B3, B3:B4, B4:B5, B5:B6
4 spines	B1:B4, B2:B5, B3:B6, T1:T4, T2:T5, T3:T6

The clock aggregation for the quadrant spines can cross over from the left to right quadrant, but not from top to bottom. The quadrant spine assignment T1:T4 is legal, but the quadrant spine assignment T1:B1 is not legal. Note that this clock aggregation is hardwired. You can always assign signals to spine T1 and B2 by instantiating a buffer, but this may add skew in the signal.

When the external feedback (EXTFB) signal of the PLL in the ProASIC3/E devices is implemented, the phase detector of the PLL core receives the reference clock (CLKA) and EXTFB as inputs. EXTFB must be sourced as an INBUF macro and located at the global/chip clock location associated with the target PLL by Designer software. EXTFB cannot be sourced from the FPGA fabric.

The reference clock, CLKA, can be assigned on GxA0 or GxA1.

External Feedback (EXTFB) signal is assigned on GxB1 by Designer automatically.

x represents global location; can be A, B, C, D, E, or F

Figure 4-5 • CLKA and EXTFB Assigned to Global I/Os

Feedback Configuration

The PLL provides both internal and external feedback delays. Depending on the configuration, various combinations of feedback delays can be achieved.

Internal Feedback Configuration

This configuration essentially sets the feedback multiplexer to route the VCO output of the PLL core as the input to the feedback of the PLL. The feedback signal can be processed with the fixed system and the adjustable feedback delay, as shown in Figure 4-24. The dividers are automatically configured by SmartGen based on the user input.

Indicated below is the System Delay pull-down menu. The System Delay can be bypassed by setting it to 0. When set, it adds a 2 ns delay to the feedback path (which results in delay advancement of the output clock by 2 ns).

Figure 4-24 • Internal Feedback with Selectable System Delay

Figure 4-25 shows the controllable Feedback Delay. If set properly in conjunction with the fixed System Delay, the total output delay can be advanced significantly.

Figure 4-25 • Internal Feedback with Selectable Feedback Delay

```
DYNCCC Core(.CLKA(CLKA), .EXTFB(GND), .POWERDOWN(POWERDOWN), .GLA(GLA), .LOCK(LOCK),
.CLKB(CLKB), .GLB(GLB), .YB(), .CLKC(CLKC), .GLC(GLC), .YC(), .SDIN(SDIN),
.SCLK(SCLK), .SSHIFT(SSHIFT), .SUPDATE(SUPDATE), .MODE(MODE), .SDOUT(SDOUT),
.OADIV0(GND), .OADIV1(GND), .OADIV2(VCC), .OADIV3(GND), .OADIV4(GND), .OAMUX0(GND),
.OAMUX1(GND), .OAMUX2(VCC), .DLYGLA0(GND), .DLYGLA1(GND), .DLYGLA2(GND),
.DLYGLA3(GND), .DLYGLA4(GND), .OBDIV0(GND), .OBDIV1(GND), .OBDIV2(GND),
.OBDIV3(GND), .OBDIV4(GND), .OBMUX0(GND), .OBMUX1(GND), .OBMUX2(GND), .DLYYB0(GND),
.DLYYB1(GND), .DLYYB2(GND), .DLYYB3(GND), .DLYYB4(GND), .DLYGLB0(GND),
.DLYGLB1(GND), .DLYGLB2(GND), .DLYGLB3(GND), .DLYGLB4(GND), .OCDIV0(GND),
.OCDIV1(GND), .OCDIV2(GND), .OCDIV3(GND), .OCDIV4(GND), .OCMUX0(GND), .OCMUX1(GND),
.OCMUX2(GND), .DLYYC0(GND), .DLYYC1(GND), .DLYYC2(GND), .DLYYC3(GND), .DLYYC4(GND),
.DLYGLC0(GND), .DLYGLC1(GND), .DLYGLC2(GND), .DLYGLC3(GND), .DLYGLC4(GND),
.FINDIV0(VCC), .FINDIV1(GND), .FINDIV2(VCC), .FINDIV3(GND), .FINDIV4(GND),
.FINDIV5(GND), .FINDIV6(GND), .FBDIV0(GND), .FBDIV1(GND), .FBDIV2(GND),
.FBDIV3(GND), .FBDIV4(GND), .FBDIV5(VCC), .FBDIV6(GND), .FBDLY0(GND), .FBDLY1(GND),
.FBDLY2(GND), .FBDLY3(GND), .FBDLY4(GND), .FBSEL0(VCC), .FBSEL1(GND),
.XDLYSEL(GND), .VCOSEL0(GND), .VCOSEL1(GND), .VCOSEL2(VCC));
defparam Core.VCOFREQUENCY = 165.000;

endmodule
```

Delayed Clock Configuration

The CLKDLY macro can be generated with the desired delay and input clock source (Hardwired I/O, External I/O, or Core Logic), as in Figure 4-28.

Figure 4-28 • Delayed Clock Configuration Dialog Box

After setting all the required parameters, users can generate one or more PLL configurations with HDL or EDIF descriptions by clicking the **Generate** button. SmartGen gives the option of saving session results and messages in a log file:

```
*****
Macro Parameters
*****
```

Name	: delay_macro
Family	: ProASIC3
Output Format	: Verilog
Type	: Delayed Clock
Delay Index	: 2
CLKA Source	: Hardwired I/O

Total Clock Delay = 0.935 ns.

The resultant CLKDLY macro Verilog netlist is as follows:

```
module delay_macro(GL,CLK);

output GL;
input CLK;
```

The following is an example of a PLL configuration utilizing the clock frequency synthesis and clock delay adjustment features. The steps include generating the PLL core with SmartGen, performing simulation for verification with ModelSim, and performing static timing analysis with SmartTime in Designer.

Parameters of the example PLL configuration:

Input Frequency – 20 MHz

Primary Output Requirement – 20 MHz with clock advancement of 3.02 ns

Secondary 1 Output Requirement – 40 MHz with clock delay of 2.515 ns

Figure 4-29 shows the SmartGen settings. Notice that the overall delays are calculated automatically, allowing the user to adjust the delay elements appropriately to obtain the desired delays.

Figure 4-29 • SmartGen Settings

After confirming the correct settings, generate a structural netlist of the PLL and verify PLL core settings by checking the log file:

```
Name                : test_pll_delays
Family              : ProASIC3E
Output Format       : VHDL
Type               : Static PLL
Input Freq(MHz)    : 20.000
CLKA Source        : Hardwired I/O
Feedback Delay Value Index : 21
Feedback Mux Select : 2
XDLY Mux Select    : No
Primary Freq(MHz)  : 20.000
Primary PhaseShift : 0
Primary Delay Value Index : 1
Primary Mux Select : 4
Secondary1 Freq(MHz) : 40.000
Use GLB           : YES
Use YB           : NO
...
...
...
Primary Clock frequency 20.000
Primary Clock Phase Shift 0.000
```

Primary Clock Output Delay from CLKA -3.020

Secondary1 Clock frequency 40.000

Secondary1 Clock Phase Shift 0.000

Secondary1 Clock Global Output Delay from CLKA 2.515

Next, perform simulation in ModelSim to verify the correct delays. Figure 4-30 shows the simulation results. The delay values match those reported in the SmartGen PLL Wizard.

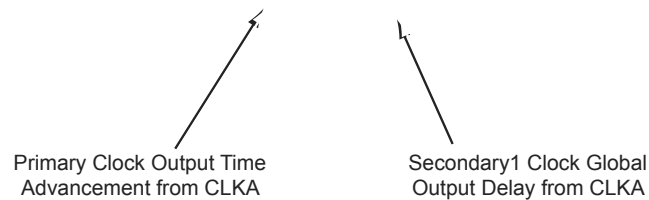


Figure 4-30 • ModelSim Simulation Results

The timing can also be analyzed using SmartTime in Designer. The user should import the synthesized netlist to Designer, perform Compile and Layout, and then invoke SmartTime. Go to **Tools > Options** and change the maximum delay operating conditions to **Typical Case**. Then expand the Clock-to-Out paths of GLA and GLB and the individual components of the path delays are shown. The path of GLA is shown in Figure 4-31 on page 123 displaying the same delay value.

Figure 4-34 • Cascade PLL Configuration

Using internal feedback, we know from EQ 4-1 on page 102 that the maximum achievable output frequency from the primary output is

$$f_{GLA} = f_{CLKA} \times m / (n \times u) = 2 \text{ MHz} \times 128 / (1 \times 1) = 256 \text{ MHz}$$

EQ 4-5

Figure 4-35 shows the settings of the initial PLL. When configuring the initial PLL, specify the input to be either Hardwired I/O–Driven or External I/O–Driven. This generates a netlist with the initial PLL routed from an I/O. Do not specify the input to be Core Logic–Driven, as this prohibits the connection from the I/O pin to the input of the PLL.

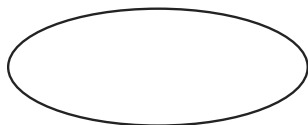


Figure 4-35 • First-Stage PLL Showing Input of 2 MHz and Output of 256 MHz

A second PLL can be connected serially to achieve the required frequency. EQ 4-1 on page 102 to EQ 4-3 on page 102 are extended as follows:

$$f_{GLA2} = f_{GLA} \times m_2 / (n_2 \times u_2) = f_{CLKA1} \times m_1 \times m_2 / (n_1 \times u_1 \times n_2 \times u_2) - \text{Primary PLL Output Clock}$$

EQ 4-6

$$f_{GLB2} = f_{YB2} = f_{CLKA1} \times m_1 \times m_2 / (n_1 \times n_2 \times v_1 \times v_2) - \text{Secondary 1 PLL Output Clock(s)}$$

EQ 4-7

$$f_{GLC2} = f_{YC2} = f_{CLKA1} \times m_1 \times m_2 / (n_1 \times n_2 \times w_1 \times w_2) - \text{Secondary 2 PLL Output Clock(s)}$$

EQ 4-8

In the example, the final output frequency (f_{output}) from the primary output of the second PLL will be as follows (EQ 4-9):

$$f_{\text{output}} = f_{GLA2} = f_{GLA} \times m_2 / (n_2 \times u_2) = 256 \text{ MHz} \times 70 / (64 \times 1) = 280 \text{ MHz}$$

EQ 4-9

Figure 4-36 on page 127 shows the settings of the second PLL. When configuring the second PLL (or any subsequent-stage PLLs), specify the input to be Core Logic–Driven. This generates a netlist with the second PLL routed internally from the core. Do not specify the input to be Hardwired I/O–Driven or External I/O–Driven, as these options prohibit the connection from the output of the first PLL to the input of the second PLL.

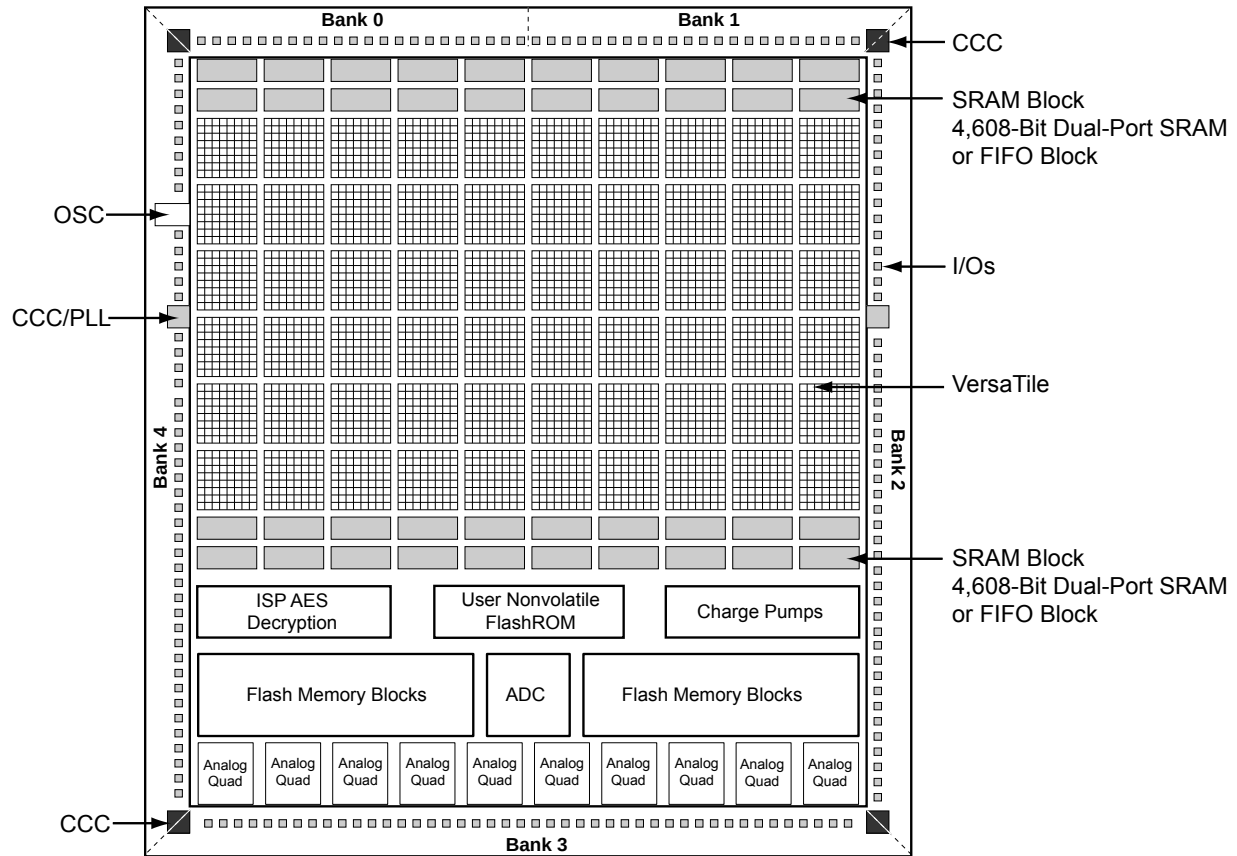


Figure 5-2 • Fusion Device Architecture Overview (AFS600)

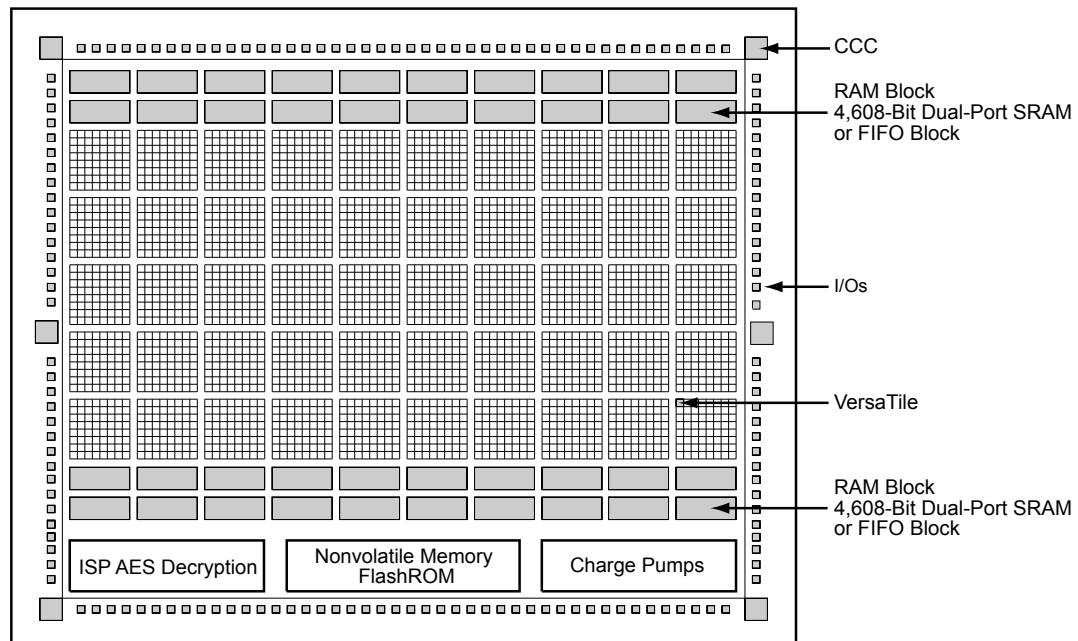


Figure 5-3 • ProASIC3 and IGLOO Device Architecture

SRAM and FIFO Architecture

To meet the needs of high-performance designs, the memory blocks operate strictly in synchronous mode for both read and write operations. The read and write clocks are completely independent, and each can operate at any desired frequency up to 250 MHz.

- 4k×1, 2k×2, 1k×4, 512×9 (dual-port RAM—2 read / 2 write or 1 read / 1 write)
- 512×9, 256×18 (2-port RAM—1 read / 1 write)
- Sync write, sync pipelined / nonpipelined read

Automotive ProASIC3 devices support single-port SRAM capabilities or dual-port SRAM only under specific conditions. 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). The Libero SoC software macro libraries support a dual-port macro only. 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. 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.

The memory block includes dedicated FIFO control logic to generate internal addresses and external flag logic (FULL, EMPTY, AFULL, AEMPTY).

Simultaneous dual-port read/write and write/write operations at the same address are allowed when certain timing requirements are met.

During RAM operation, addresses are sourced by the user logic, and the FIFO controller is ignored. In FIFO mode, the internal addresses are generated by the FIFO controller and routed to the RAM array by internal MUXes.

The low power flash device architecture enables the read and write sizes of RAMs to be organized independently, allowing for bus conversion. For example, the write size can be set to 256×18 and the read size to 512×9.

Both the write width and read width for the RAM blocks can be specified independently with the WW (write width) and RW (read width) pins. The different D×W configurations are 256×18, 512×9, 1k×4, 2k×2, and 4k×1. When widths of one, two, or four are selected, the ninth bit is unused. For example, when writing nine-bit values and reading four-bit values, only the first four bits and the second four bits of each nine-bit value are addressable for read operations. The ninth bit is not accessible.

Conversely, when writing four-bit values and reading nine-bit values, the ninth bit of a read operation will be undefined. The RAM blocks employ little-endian byte order for read and write operations.

Memory Blocks and Macros

Memory blocks can be configured with many different aspect ratios, but are generically supported in the macro libraries as one of two memory elements: RAM4K9 or RAM512X18. The RAM4K9 is configured as a true dual-port memory block, and the RAM512X18 is configured as a two-port memory block. Dual-port memory allows the RAM to both read from and write to either port independently. Two-port memory allows the RAM to read from one port and write to the other using a common clock or independent read and write clocks. If needed, the RAM4K9 blocks can be configured as two-port memory blocks. The memory block can be configured as a FIFO by combining the basic memory block with dedicated FIFO controller logic. The FIFO macro is named FIFO4KX18 (Figure 6-3 on page 152).

Clocks for the RAM blocks can be driven by the VersaNet (global resources) or by regular nets. When using local clock segments, the clock segment region that encompasses the RAM blocks can drive the RAMs. In the dual-port configuration (RAM4K9), each memory block port can be driven by either rising-edge or falling-edge clocks. Each port can be driven by clocks with different edges. Though only a rising-edge clock can drive the physical block itself, the Microsemi Designer software will automatically bubble-push the inversion to properly implement the falling-edge trigger for the RAM block.

Temporary overshoots are allowed according to the overshoot and undershoot table in the datasheet.

Solution 1

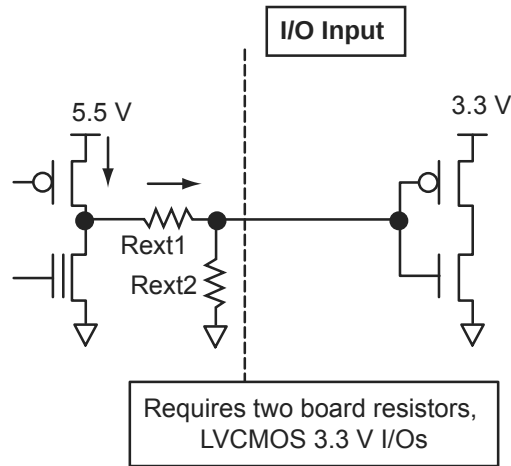


Figure 7-9 • Solution 1

Solution 2

The board-level design must ensure that the reflected waveform at the pad does not exceed the voltage overshoot/undershoot limits provided in the datasheet. This is a requirement to ensure long-term reliability.

This scheme will also work for a 3.3 V PCI/PCI-X configuration, but the internal diode should not be used for clamping, and the voltage must be limited by the external resistors and Zener, as shown in Figure 7-10. Relying on the diode clamping would create an excessive pad DC voltage of $3.3\text{ V} + 0.7\text{ V} = 4\text{ V}$.

Solution 2

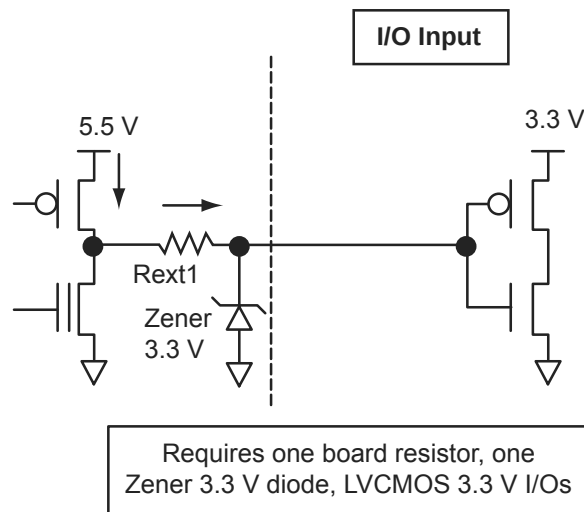


Figure 7-10 • Solution 2

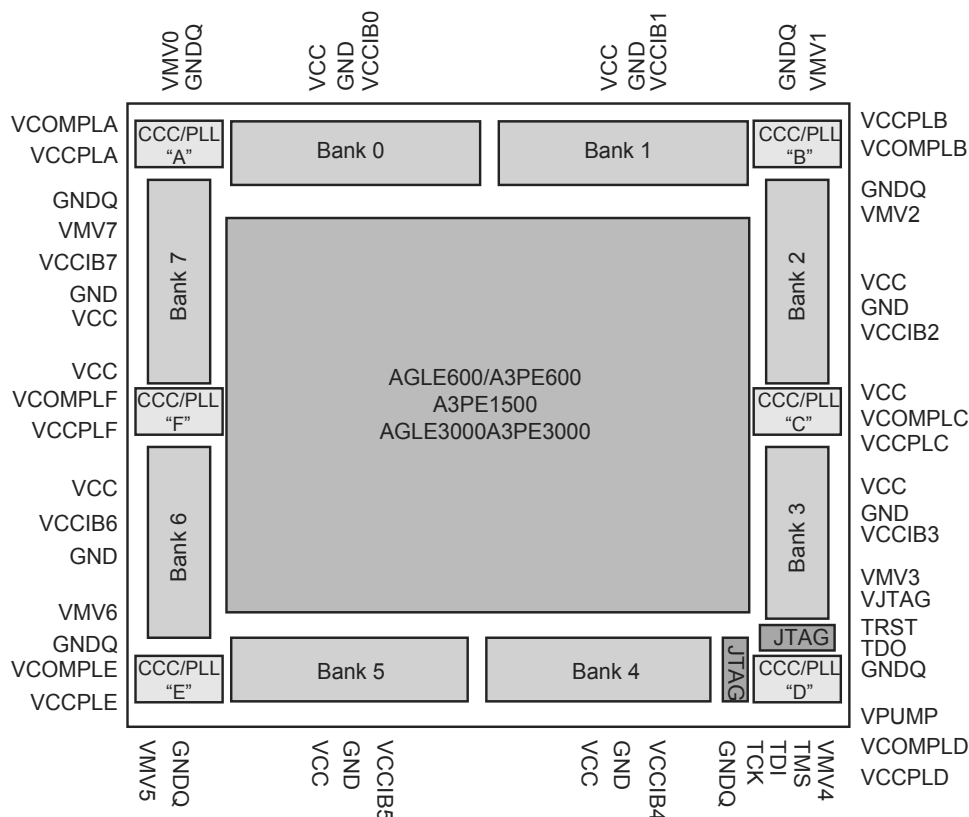


Figure 8-20 • User I/O Naming Conventions of IGLOOe and ProASIC3E Devices – Top View

Board-Level Considerations

Low power flash devices have robust I/O features that can help in reducing board-level components. The devices offer single-chip solutions, which makes the board layout simpler and more immune to signal integrity issues. Although, in many cases, these devices resolve board-level issues, special attention should always be given to overall signal integrity. This section covers important board-level considerations to facilitate optimum device performance.

Termination

Proper termination of all signals is essential for good signal quality. Nonterminated signals, especially clock signals, can cause malfunctioning of the device.

For general termination guidelines, refer to the *Board-Level Considerations* application note for Microsemi FPGAs. Also refer to the "Pin Descriptions" chapter of the appropriate datasheet for termination requirements for specific pins.

Low power flash I/Os are equipped with on-chip pull-up/-down resistors. The user can enable these resistors by instantiating them either in the top level of the design (refer to the *IGLOO*, *Fusion*, and *ProASIC3 Macro Library Guide* for the available I/O macros with pull-up/-down) or in the I/O Attribute Editor in Designer if generic input or output buffers are instantiated in the top level. Unused I/O pins are configured as inputs with pull-up resistors.

As mentioned earlier, low power flash devices have multiple programmable drive strengths, and the user can eliminate unwanted overshoot and undershoot by adjusting the drive strengths.

those banks, the user does not need to assign the same VCCI voltage to another bank. The user needs to assign the other three VCCI voltages to three more banks.

Assigning Technologies and VREF to I/O Banks

Low power flash devices offer a wide variety of I/O standards, including voltage-referenced standards. Before proceeding to Layout, each bank must have the required VCCI voltage assigned for the corresponding I/O technologies used for that bank. The voltage-referenced standards require the use of a reference voltage (VREF). This assignment can be done manually or automatically. The following sections describe this in detail.

Manually Assigning Technologies to I/O Banks

The user can import the PDC at this point and resolve this requirement. The PDC command is `set_iobank [bank name] -vcci [vcci value]`

Another method is to use the I/O Bank Settings dialog box (**MVN > Edit > I/O Bank Settings**) to set up the V_{CCI} voltage for the bank (Figure 9-12).

Figure 9-12 • Setting VCCI for a Bank

Figure 12-10 • All Silicon Features Selected for IGLOO and ProASIC3 Devices

Figure 12-11 • All Silicon Features Selected for Fusion

15 – 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.

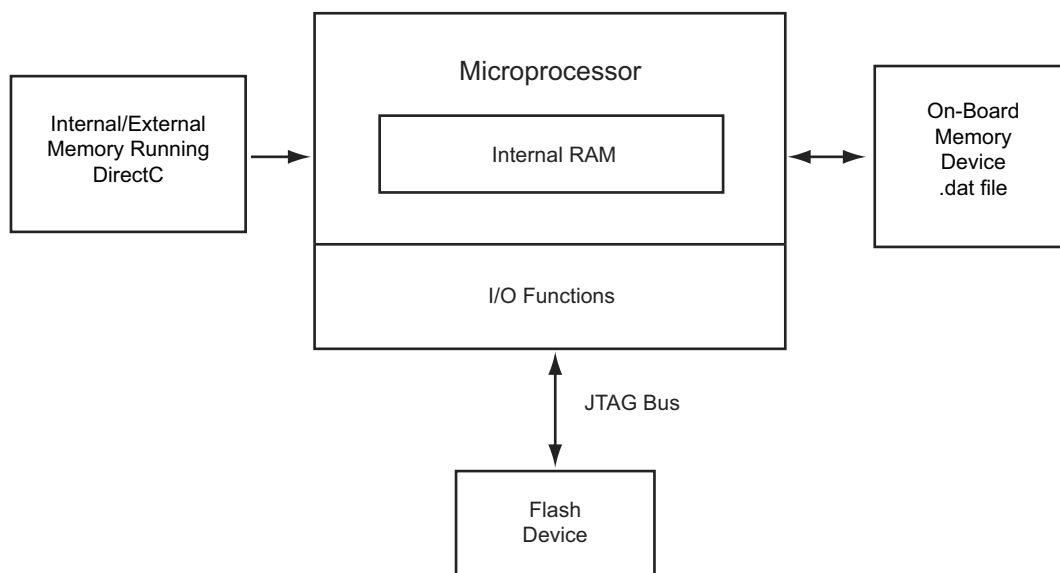


Figure 15-1 • ISP Using Microprocessor

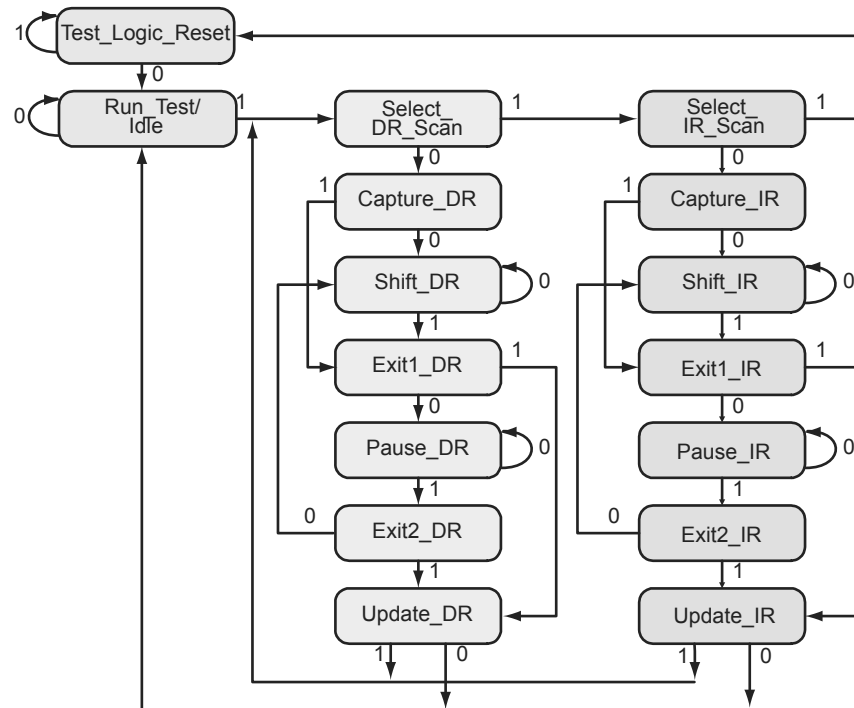


Figure 17-4 • TAP Controller State Diagram

UJTAG Port Usage

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

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

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

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