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	620
Number of Gates	3000000
Voltage - Supply	1.14V ~ 1.575V
Mounting Type	Surface Mount
Operating Temperature	-40°C ~ 100°C (TJ)
Package / Case	896-BGA
Supplier Device Package	896-FBGA (31x31)
Purchase URL	https://www.e-xfl.com/product-detail/microchip-technology/m1a3pe3000l-fgg896i

Introduction

Contents

This user’s guide contains information to help designers understand and use Microsemi's ProASIC®3L devices. Each chapter addresses a specific topic. Most of these chapters apply to other Microsemi device families as well. When a feature or description applies only to a specific device family, this is made clear in the text.

Revision History

The revision history for each chapter is listed at the end of the chapter. Most of these chapters were formerly included in device handbooks. Some were originally application notes or information included in device datasheets.

A "Summary of Changes" table at the end of this user’s guide lists the chapters that were changed in each revision of the document, with links to the "List of Changes" sections for those chapters.

Related Information

Refer to the *ProASIC3L Flash Family FPGAs* datasheet for detailed specifications, timing, and package and pin information.

The website page for ProASIC3L devices is /www.microsemi.com/soc/products/pa3l/default.aspx.

- The INBUF_FF must be driven by a top-level input port of the design.
- The INBUF_FF AND the ULSICC macro must be used to enable type 2 Flash*Freeze mode.
- For type 2 Flash*Freeze mode, the INBUF_FF MUST drive some logic in the design.
- For type 1 Flash*Freeze mode, the INBUF_FF may drive some logic in the design, but it may also be left floating.
- Only one INBUF_FF may be instantiated in a device.
- The FF pin threshold voltages are defined by VCCI and the supported single-ended I/O standard in the corresponding I/O bank.
- The FF pin Schmitt trigger option may be configured in the I/O attribute editor in Microsemi's Designer software. The Schmitt trigger option is only available for IGLOOe, IGLOO nano, IGLOO PLUS, ProASIC3EL, and RT ProASIC3 devices.
- A 2 ns glitch filter resides in the Flash*Freeze Technology block to filter unwanted glitches on the FF pin.

ULSICC

The User Low Static ICC (ULSICC) macro allows the FPGA core to access the Flash*Freeze Technology block so that entering and exiting Flash*Freeze mode can be controlled by the user's design. The ULSICC macro enables a hard block with an available LSICC input port, as shown in Figure 2-3 on page 27 and Figure 2-10 on page 37. Design rules for the ULSICC macro are as follows:

- The ULSICC macro by itself cannot enable Flash*Freeze mode. The INBUF_FF AND the ULSICC macro must both be used to enable type 2 Flash*Freeze mode.
- The ULSICC controls entering the Flash*Freeze mode by asserting the LSICC input (logic '1') of the ULSICC macro. The FF pin must also be asserted (logic '0') to enter Flash*Freeze mode.
- When the LSICC signal is '0', the device cannot enter Flash*Freeze mode; and if already in Flash*Freeze mode, it will exit.
- When the ULSICC macro is not instantiated in the user's design, the LSICC port will be tied High.

Flash*Freeze Management IP

The Flash*Freeze management IP can be configured with the Libero (or SmartGen) core generator in a simple, intuitive interface. With the core configuration tool, users can select the number of clocks to be gated, and select whether or not to implement housekeeping. All port names on the Flash*Freeze management IP block can be renamed by the user.

- The clock gating (filter) blocks include CLKINT buffers for each gated clock output (version 8.3).
- When housekeeping is NOT used, the WAIT_HOUSEKEEPING signal will be automatically fed back into DONE_HOUSEKEEPING inside the core, and the ports will not be available at the IP core interface.
- The INBUF_FF macro is automatically instantiated within the IP core.
- The INBUF_FF port (default name is "Flash_Freeze_N") must be connected to a top-level input port of the design.
- The ULSICC macro is automatically instantiated within the IP core, and the LSICC signal is driven by the FSM.
- Timing analysis can be performed on the clock domain of the source clock (i.e., input to the clock gating filters). For example, if CLKin becomes CLKin_gated, the timing can be performed on the CLKin domain in SmartTime.
- The gated clocks can be added to the clock list if the user wishes to analyze these clocks specifically. The user can locate the gated clocks by looking for instance names such as those below:

```
Top/ff1/ff_1_wrapper_inst/user_ff_1_wrapper/Primary_Filter_Instance/
Latch_For_Clock_Gating:Q
Top/ff1/ff_1_wrapper_inst/user_ff_1_wrapper/genblk1.genblk2.secondary_filter[0].
seconday_filter_instance/Latch_For_Clock_Gating:Q
Top/ff1/ff_1_wrapper_inst/user_ff_1_wrapper/genblk1.genblk2.secondary_filter[1].
seconday_filter_instance/Latch_For_Clock_Gating:Q
```

Spine Architecture

The low power flash device architecture allows the VersaNet global networks to be segmented. Each of these networks contains spines (the vertical branches of the global network tree) and ribs that can reach all the VersaTiles inside its region. The nine spines available in a vertical column reside in global networks with two separate regions of scope: the quadrant global network, which has three spines, and the chip (main) global network, which has six spines. Note that the number of quadrant globals and globals/spines per tree varies depending on the specific device. Refer to Table 3-4 for the clocking resources available for each device. The spines are the vertical branches of the global network tree, shown in Figure 3-3 on page 50. Each spine in a vertical column of a chip (main) global network is further divided into two spine segments of equal lengths: one in the top and one in the bottom half of the die (except in 10 k through 30 k gate devices).

Top and bottom spine segments radiating from the center of a device have the same height. However, just as in the ProASIC^{PLUS}® family, signals assigned only to the top and bottom spine cannot access the middle two rows of the die. The spines for quadrant clock networks do not cross the middle of the die and cannot access the middle two rows of the architecture.

Each spine and its associated ribs cover a certain area of the device (the "scope" of the spine; see Figure 3-3 on page 50). Each spine is accessed by the dedicated global network MUX tree architecture, which defines how a particular spine is driven—either by the signal on the global network from a CCC, for example, or by another net defined by the user. Details of the chip (main) global network spine-selection MUX are presented in Figure 3-8 on page 60. The spine drivers for each spine are located in the middle of the die.

Quadrant spines can be driven from user I/Os or an internal signal from the north and south sides of the die. The ability to drive spines in the quadrant global networks can have a significant effect on system performance for high-fanout inputs to a design. Access to the top quadrant spine regions is from the top of the die, and access to the bottom quadrant spine regions is from the bottom of the die. The A3PE3000 device has 28 clock trees and each tree has nine spines; this flexible global network architecture enables users to map up to 252 different internal/external clocks in an A3PE3000 device.

***Table 3-4* • Globals/Spines/Rows for IGLOO and ProASIC3 Devices**

ProASIC3/ ProASIC3L Devices	IGLOO Devices	Chip Globals	Quadrant Globals (4×3)	Clock Trees	Globals/ Spines per Tree	Total Spines per Device	VersaTiles in Each Tree	Total VersaTiles	Rows in Each Spine
A3PN010	AGLN010	4	0	1	0	0	260	260	4
A3PN015	AGLN015	4	0	1	0	0	384	384	6
A3PN020	AGLN020	4	0	1	0	0	520	520	6
A3PN060	AGLN060	6	12	4	9	36	384	1,536	12
A3PN125	AGLN125	6	12	8	9	72	384	3,072	12
A3PN250	AGLN250	6	12	8	9	72	768	6,144	24
A3P015	AGL015	6	0	1	9	9	384	384	12
A3P030	AGL030	6	0	2	9	18	384	768	12
A3P060	AGL060	6	12	4	9	36	384	1,536	12
A3P125	AGL125	6	12	8	9	72	384	3,072	12
A3P250/L	AGL250	6	12	8	9	72	768	6,144	24
A3P400	AGL400	6	12	12	9	108	768	9,216	24
A3P600/L	AGL600	6	12	12	9	108	1,152	13,824	36
A3P1000/L	AGL1000	6	12	16	9	144	1,536	24,576	48
A3PE600/L	AGLE600	6	12	12	9	108	1,120	13,440	35
A3PE1500		6	12	20	9	180	1,888	37,760	59
A3PE3000/L	AGLE3000	6	12	28	9	252	2,656	74,368	83

Spine Access

The physical location of each spine is identified by the letter T (top) or B (bottom) and an accompanying number (T*n* or B*n*). The number *n* indicates the horizontal location of the spine; 1 refers to the first spine on the left side of the die. Since there are six chip spines in each spine tree, there are up to six spines available for each combination of T (or B) and *n* (for example, six T1 spines). Similarly, there are three quadrant spines available for each combination of T (or B) and *n* (for example, four T1 spines), as shown in Figure 3-7.

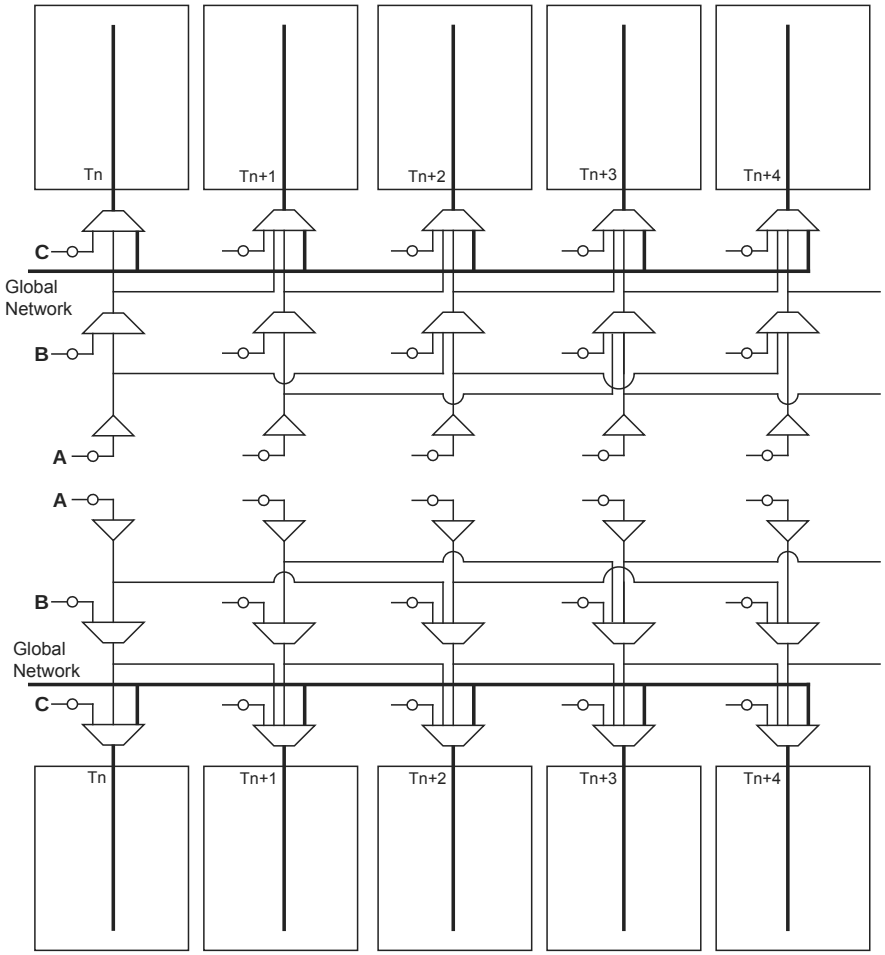


***Figure 3-7* • Chip Global Aggregation**

A spine is also called a local clock network, and is accessed by the dedicated global MUX architecture. These MUXes define how a particular spine is driven. Refer to Figure 3-8 on page 60 for the global MUX architecture. The MUXes for each chip global spine are located in the middle of the die. Access to the top and bottom chip global spine is available from the middle of the die. There is no control dependency between the top and bottom spines. If a top spine, T1, of a chip global network is assigned to a net, B1 is not wasted and can be used by the global clock network. The signal assigned only to the top or bottom spine cannot access the middle two rows of the architecture. However, if a spine is using the top and bottom at the same time (T1 and B1, for instance), the previous restriction is lifted.

The MUXes for each quadrant global spine are located in the north and south sides of the die. Access to the top and bottom quadrant global spines is available from the north and south sides of the die. Since the MUXes for quadrant spines are located in the north and south sides of the die, you should not try to drive T1 and B1 quadrant spines from the same signal.

Step 1

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

Cell usage:

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

Step 2

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

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

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

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 87. 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 102 for a description of the programmable delay types used for the PLL. Also refer to Table 4-14 on page 110 for Programmable Delay Type 1 step delay values, and Table 4-15 on page 110 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 87 shows a block diagram of the PLL, where the programmable delay elements are used for the global networks (Programmable Delay Type 2).

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.

This section outlines the following device information: CCC features, PLL core specifications, functional descriptions, software configuration information, detailed usage information, recommended board-level considerations, and other considerations concerning global networks in low power flash devices.

Clock Conditioning Circuits with Integrated PLLs

Each of the CCCs with integrated PLLs includes the following:

- 1 PLL core, which consists of a phase detector, a low-pass filter, and a four-phase voltage-controlled oscillator
- 3 global multiplexer blocks that steer signals from the global pads and the PLL core onto the global networks
- 6 programmable delays and 1 fixed delay for time advance/delay adjustments
- 5 programmable frequency divider blocks to provide frequency synthesis (automatically configured by the SmartGen macro builder tool)

Clock Conditioning Circuits without Integrated PLLs

There are two types of simplified CCCs without integrated PLLs in low power flash devices.

1. The simplified CCC with programmable delays, which is composed of the following:
   – 3 global multiplexer blocks that steer signals from the global pads and the programmable delay elements onto the global networks
   – 3 programmable delay elements to provide time delay adjustments
2. The simplified CCC (referred to as CCC-GL) without programmable delay elements, which is composed of the following:
   – A global multiplexer block that steer signals from the global pads onto the global networks

Dividers *n* and *m* (the input divider and feedback divider, respectively) provide integer frequency division factors from 1 to 128. The output dividers *u*, *v*, and *w* provide integer division factors from 1 to 32. Frequency scaling of the reference clock CLKA is performed according to the following formulas:

$$f_{GLA} = f_{CLKA} \times m / (n \times u) \text{ – GLA Primary PLL Output Clock}$$

EQ 4-1

$$f_{GLB} = f_{YB} = f_{CLKA} \times m / (n \times v) \text{ – GLB Secondary 1 PLL Output Clock(s)}$$

EQ 4-2

$$f_{GLC} = f_{YC} = f_{CLKA} \times m / (n \times w) \text{ – GLC Secondary 2 PLL Output Clock(s)}$$

EQ 4-3

SmartGen provides a user-friendly method of generating the configured PLL netlist, which includes automatically setting the division factors to achieve the closest possible match to the requested frequencies. Since the five output clocks share the *n* and *m* dividers, the achievable output frequencies are interdependent and related according to the following formula:

$$f_{GLA} = f_{GLB} \times (v / u) = f_{GLC} \times (w / u)$$

EQ 4-4

Clock Delay Adjustment

There are a total of seven configurable delay elements implemented in the PLL architecture.

Two of the delays are located in the feedback path, entitled System Delay and Feedback Delay. System Delay provides a fixed delay of 2 ns (typical), and Feedback Delay provides selectable delay values from 0.6 ns to 5.56 ns in 160 ps increments (typical). For PLLs, delays in the feedback path will effectively advance the output signal from the PLL core with respect to the reference clock. Thus, the System and Feedback delays generate negative delay on the output clock. Additionally, each of these delays can be independently bypassed if necessary.

The remaining five delays perform traditional time delay and are located at each of the outputs of the PLL. Besides the fixed global driver delay of 0.755 ns for each of the global networks, the global multiplexer outputs (GLA, GLB, and GLC) each feature an additional selectable delay value, as given in Table 4-7.

***Table 4-7* • Delay Values in Libero SoC Software per Device Family**

Device	Typical	Starting Values	Increments	Ending Value
ProASIC3	200 ps	0 to 735 ps	200 ps	6.735 ns
IGLOO/ProASIC3L 1.5 V	360 ps	0 to 1.610 ns	360 ps	12.410 ns
IGLOO/ProASIC3L 1.2 V	580 ps	0 to 2.880 ns	580 ps	20.280 ns

The additional YB and YC signals have access to a selectable delay from 0.6 ns to 5.56 ns in 160 ps increments (typical). This is the same delay value as the CLKDLY macro. It is similar to CLKDLY, which bypasses the PLL core just to take advantage of the phase adjustment option with the delay value.

The following parameters must be taken into consideration to achieve minimum delay at the outputs (GLA, GLB, GLC, YB, and YC) relative to the reference clock: routing delays from the PLL core to CCC outputs, core outputs and global network output delays, and the feedback path delay. The feedback path delay acts as a time advance of the input clock and will offset any delays introduced beyond the PLL core output. The routing delays are determined from back-annotated simulation and are configuration-dependent.

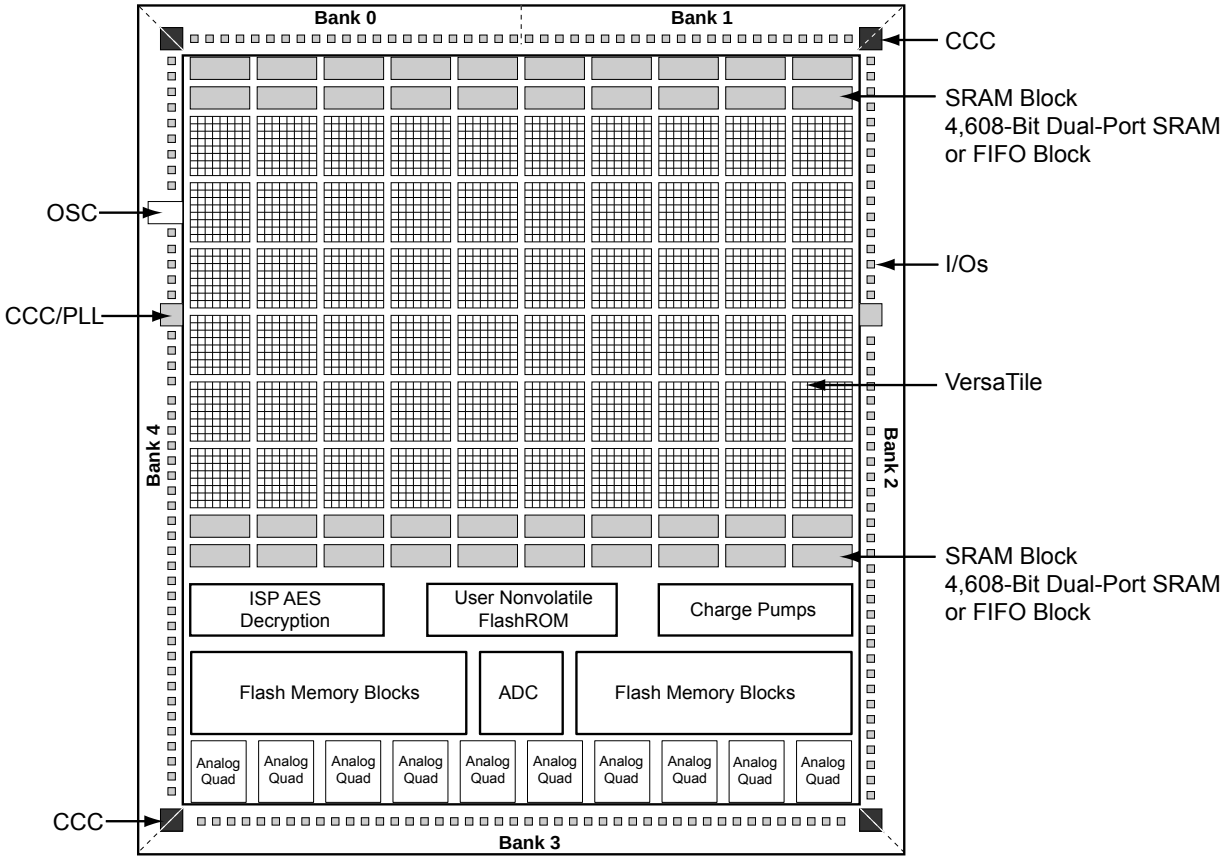


***Figure 5-2* • Fusion Device Architecture Overview (AFS600)**

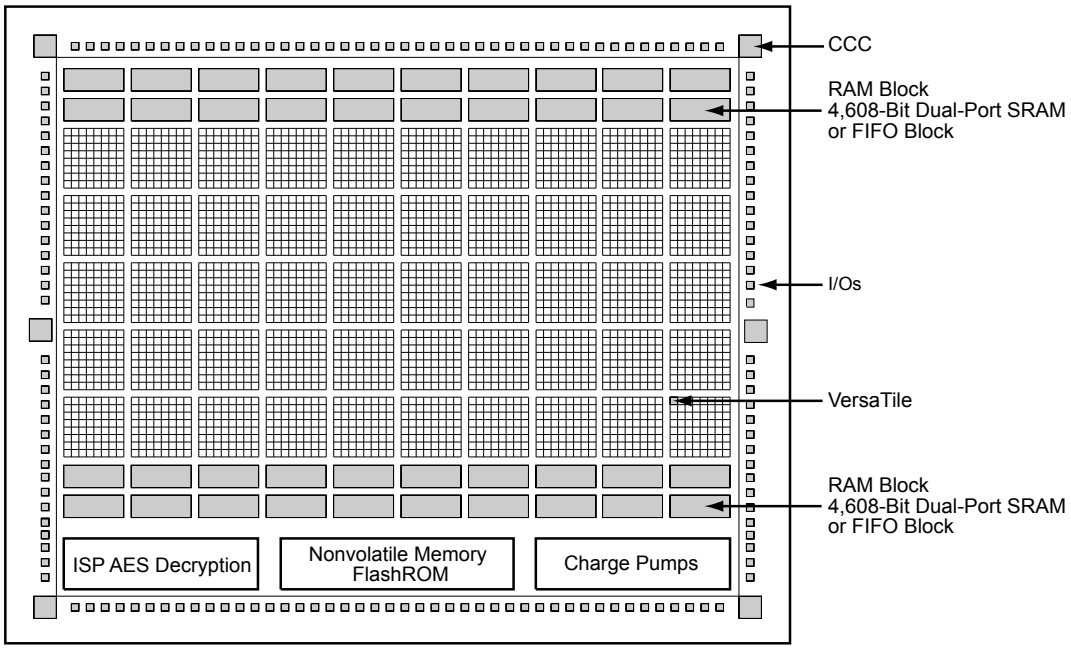


***Figure 5-3* • ProASIC3 and IGLOO Device Architecture**

256×18 FIFO is full, even though a 128×18 FIFO was requested. For this example, the Almost-Full flag can be used instead of the Full flag to signal when the 128th data word is reached.

To accommodate different aspect ratios, the almost-full and almost-empty values are expressed in terms of data bits instead of data words. SmartGen translates the user's input, expressed in data words, into data bits internally. SmartGen allows the user to select the thresholds for the Almost-Empty and Almost-Full flags in terms of either the read data words or the write data words, and makes the appropriate conversions for each flag.

After the empty or full states are reached, the FIFO can be configured so the FIFO counters either stop or continue counting. For timing numbers, refer to the appropriate family datasheet.

Signal Descriptions for FIFO4K18

The following signals are used to configure the FIFO4K18 memory element:

WW and RW

These signals enable the FIFO to be configured in one of the five allowable aspect ratios (Table 6-6).

***Table 6-6* • Aspect Ratio Settings for WW[2:0]**

WW[2:0]	RW[2:0]	D×W
000	000	4k×1
001	001	2k×2
010	010	1k×4
011	011	512×9
100	100	256×18
101, 110, 111	101, 110, 111	Reserved

WBLK and RBLK

These signals are active-low and will enable the respective ports when LOW. When the RBLK signal is HIGH, that port's outputs hold the previous value.

WEN and REN

Read and write enables. WEN is active-low and REN is active-high by default. These signals can be configured as active-high or -low.

WCLK and RCLK

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

Note: For the Automotive ProASIC3 FIFO4K18, for the same clock, 180° out of phase (inverted) between clock pins should be used.

RPIPE

This signal is used to specify pipelined read on the output. A LOW on RPIPE indicates a nonpipelined read, and the data appears on the output in the same clock cycle. A HIGH indicates a pipelined read, and data appears on the output in the next clock cycle.

RESET

This active-low signal resets the control logic and forces the output hold state registers to zero when asserted. It does not reset the contents of the memory array (Table 6-7 on page 160).

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.

WD

This is the input data bus and is 18 bits wide. Not all 18 bits are valid in all configurations. When a data width less than 18 is specified, unused higher-order signals must be grounded (Table 6-7 on page 160).

Initializing the RAM/FIFO

The SRAM blocks can be initialized with data to use as a lookup table (LUT). Data initialization can be accomplished either by loading the data through the design logic or through the UJTAG interface. The UJTAG macro is used to allow access from the JTAG port to the internal logic in the device. By sending the appropriate initialization string to the JTAG Test Access Port (TAP) Controller, the designer can put the JTAG circuitry into a mode that allows the user to shift data into the array logic through the JTAG port using the UJTAG macro. For a more detailed explanation of the UJTAG macro, refer to the "FlashROM in Microsemi's Low Power Flash Devices" section on page 133.

A user interface is required to receive the user command, initialization data, and clock from the UJTAG macro. The interface must synchronize and load the data into the correct RAM block of the design. The main outputs of the user interface block are the following:

- Memory block chip select: Selects a memory block for initialization. The chip selects signals for each memory block that can be generated from different user-defined pockets or simple logic, such as a ring counter (see below).
- Memory block write address: Identifies the address of the memory cell that needs to be initialized.
- Memory block write data: The interface block receives the data serially from the UTDI port of the UJTAG macro and loads it in parallel into the write data ports of the memory blocks.
- Memory block write clock: Drives the WCLK of the memory block and synchronizes the write data, write address, and chip select signals.

Figure 6-8 shows the user interface between UJTAG and the memory blocks.

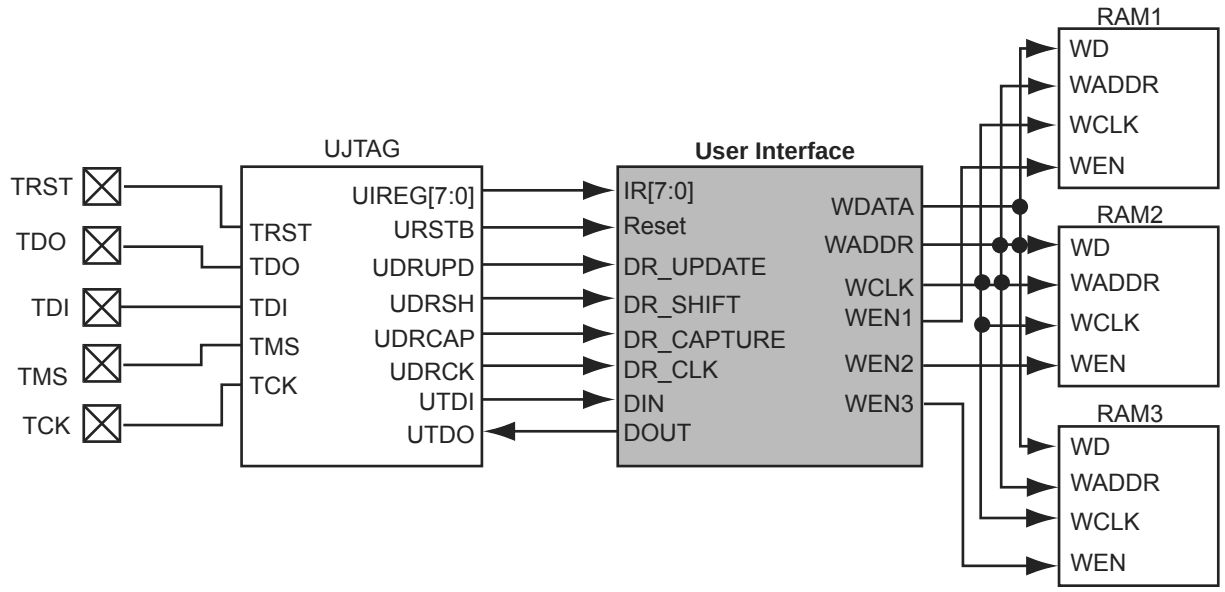


Figure 6-8 • **Interfacing TAP Ports and SRAM Blocks**

An important component of the interface between the UJTAG macro and the RAM blocks is a serial-in/parallel-out shift register. The width of the shift register should equal the data width of the RAM blocks. The RAM data arrives serially from the UTDI output of the UJTAG macro. The data must be shifted into a shift register clocked by the JTAG clock (provided at the UDRCK output of the UJTAG macro).

Then, after the shift register is fully loaded, the data must be transferred to the write data port of the RAM block. To synchronize the loading of the write data with the write address and write clock, the output of the shift register can be pipelined before driving the RAM block.

The write address can be generated in different ways. It can be imported through the TAP using a different instruction opcode and another shift register, or generated internally using a simple counter. Using a counter to generate the address bits and sweep through the address range of the RAM blocks is

without reprogramming the device. Dynamic flag settings are determined by register values and can be altered without reprogramming the device by reloading the register values either from the design or through the UJTAG interface described in the "Initializing the RAM/FIFO" section on page 164.

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

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

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

Limitations

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

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

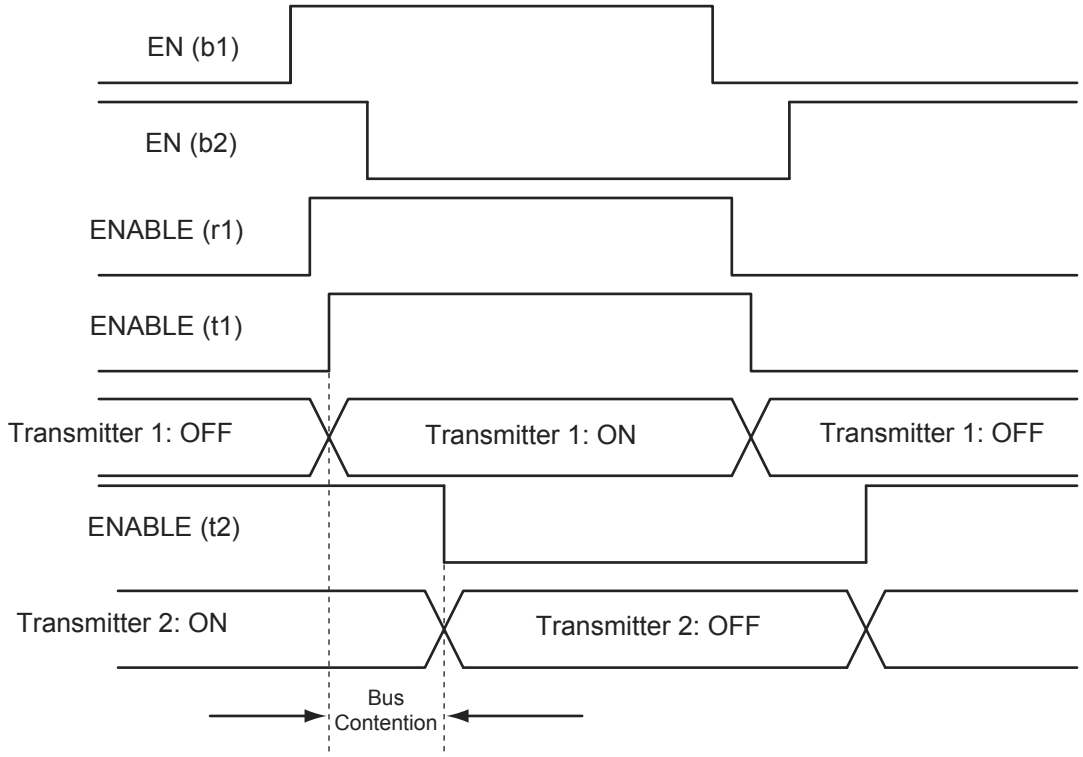
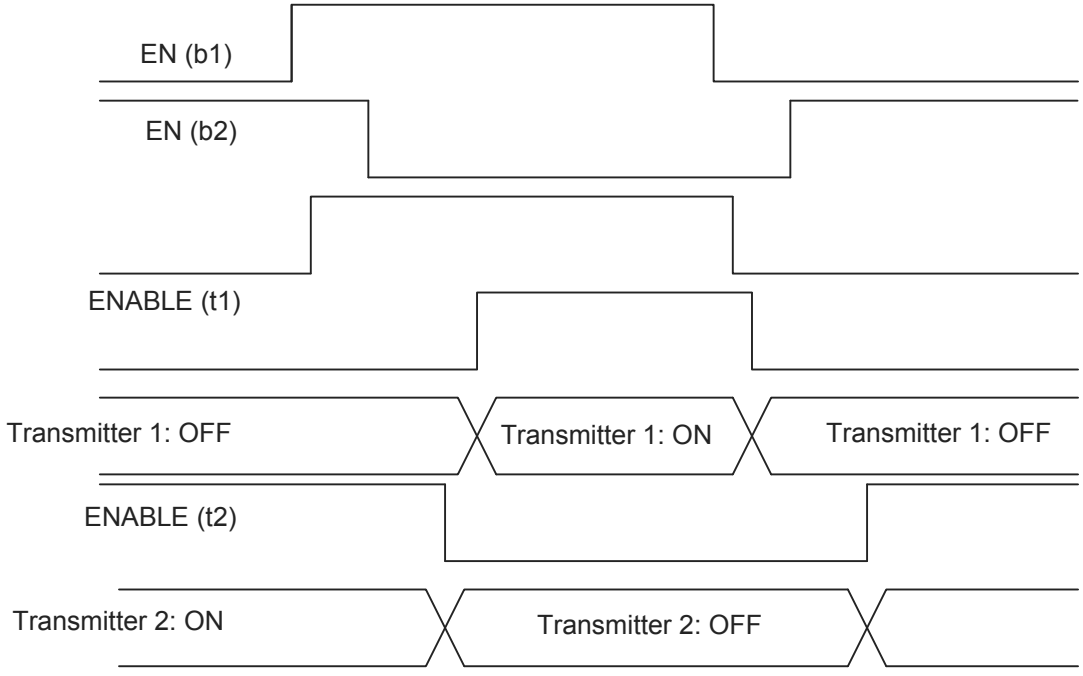


***Figure 7-17* • Timing Diagram (bypasses skew circuit)**



Result: No Bus Contention

***Figure 7-18* • Timing Diagram (with skew circuit selected)**

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

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

Performing Place-and-Route on the Design

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

Defining I/O Assignments in the PDC File

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

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

***Table 9-3* • PDC I/O Constraints**

Command	Action	Example	Comment
I/O Banks Setting Constraints			

set_iobank	Sets the I/O supply voltage, V_{CCI}, and the input reference voltage, V_{REF}, for the specified I/O bank.	`set_iobank bankname [-vcci vcci_voltage] [-vref vref_voltage]`
		
		`set_iobank Bank7 -vcci 1.50 -vref 0.75`
			Must use in case of mixed I/O voltage (V_{CCI}) design

set_vref	Assigns a V_{REF} pin to a bank.	`set_vref -bank [bankname] [pinnum]`
		
		`set_vref -bank Bank0 685 704 723 742 761`
			Must use if voltage-referenced I/Os are used

set_vref_defaults	Sets the default V_{REF} pins for the specified bank. This command is ignored if the bank does not need a V_{REF} pin.	`set_vref_defaults bankname`
		
		`set_vref_defaults bank2`	

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

If the assignment is not successful, an error message appears in the Output window.

To undo the I/O bank assignments, choose **Undo** from the **Edit** menu. Undo removes the I/O technologies assigned by the IOBA. It does not remove the I/O technologies previously assigned.

To redo the changes undone by the Undo command, choose **Redo** from the **Edit** menu.

To clear I/O bank assignments made before using the Undo command, manually unassign or reassign I/O technologies to banks. To do so, choose **I/O Bank Settings** from the **Edit** menu to display the I/O Bank Settings dialog box.

Conclusion

Fusion, IGLOO, and ProASIC3 support for multiple I/O standards minimizes board-level components and makes possible a wide variety of applications. The Microsemi Designer software, integrated with Libero SoC, presents a clear visual display of I/O assignments, allowing users to verify I/O and board-level design requirements before programming the device. The device I/O features and functionalities ensure board designers can produce low-cost and low power FPGA applications fulfilling the complexities of contemporary design needs.

Related Documents

User's Guides

Libero SoC User's Guide

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

IGLOO, ProASIC3, SmartFusion, and Fusion Macro Library Guide

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

SmartGen Core Reference Guide

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

DDR Output Register

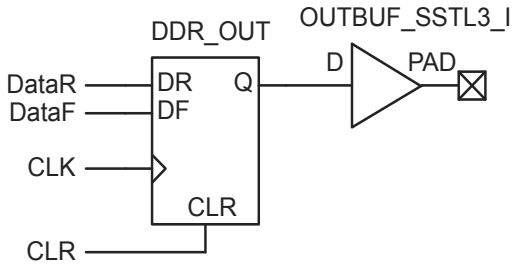


***Figure 10-6* • DDR Output Register (SSTL3 Class I)**

Verilog

```
module DDR_OutBuf_SSTL3_I(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_SSTL3_I OUTBUF_SSTL3_I_0_inst(.D(Q),.PAD(PAD));

endmodule
```

VHDL

```
library ieee;
use ieee.std_logic_1164.all;
library proasic3; use proasic3.all;

entity DDR_OutBuf_SSTL3_I is
  port(DataR, DataF, CLR, CLK : in std_logic;  PAD : out std_logic) ;
end DDR_OutBuf_SSTL3_I;

architecture DEF_ARCH of  DDR_OutBuf_SSTL3_I is

  component DDR_OUT
    port(DR, DF, CLK, CLR : in std_logic := 'U'; Q : out std_logic) ;
  end component;

  component OUTBUF_SSTL3_I
    port(D : in std_logic := 'U'; PAD : out std_logic) ;
  end component;

  component VCC
    port( Y : out std_logic);
  end component;

signal Q, VCC_1_net : std_logic ;

begin

  VCC_2_net : VCC port map(Y => VCC_1_net);
  DDR_OUT_0_inst : DDR_OUT
  port map(DR => DataR, DF => DataF, CLK => CLK, CLR => CLR, Q => Q);
  OUTBUF_SSTL3_I_0_inst : OUTBUF_SSTL3_I
  port map(D => Q, PAD => PAD);

end DEF_ARCH;
```

FlashROM Security Use Models

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

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

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

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

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

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