Microsemi makes no warranty, representation, or guarantee regarding the information contained herein or the suitability of its products and services for any particular purpose, nor does Microsemi assume any liability whatsoever arising out of the application or use of any product or circuit. The products sold hereunder and any other products sold by Microsemi have been subject to limited testing and should not be used in conjunction with mission-critical equipment or applications. Any performance specifications are believed to be reliable but are not verified, and Buyer must conduct and complete all performance and other testing of the products, alone and together with, or installed in, any end-products. Buyer shall not rely on any data and performance specifications or parameters provided by Microsemi. It is the Buyer’s responsibility to independently determine suitability of any products and to test and verify the same. The information provided by Microsemi hereunder is provided “as is, where is” and with all faults, and the entire risk associated with such information is entirely with the Buyer. Microsemi does not grant, explicitly or implicitly, to any party any patent rights, licenses, or any other IP rights, whether with regard to such information itself or anything described by such information. Information provided in this document is proprietary to Microsemi, and Microsemi reserves the right to make any changes to the information in this document or to any products and services at any time without notice.

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions; setting the world’s standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. Learn more at www.microsemi.com.
Contents

1 Revision History .................................................. 1
  1.1 Revision 7.0 .................................................. 1
  1.2 Revision 6.0 .................................................. 1
  1.3 Revision 5.0 .................................................. 1
  1.4 Revision 4.0 .................................................. 1
  1.5 Revision 3.0 .................................................. 1
  1.6 Revision 2.0 .................................................. 1
  1.7 Revision 1.0 .................................................. 1

2 IGLOO2 FPGA Flash*Freeze Entry and Exit .............................. 2
  2.1 Design Requirements ........................................... 2
  2.2 Prerequisites .................................................. 2
  2.3 Enter and Exit Flash*Freeze ..................................... 3
  2.4 Design Details ................................................ 5
    2.4.1 Design Description ....................................... 6
    2.4.2 Entering Flash*Freeze Mode ............................... 8
    2.4.3 Exiting Flash*Freeze Mode ............................... 9
    2.4.4 Hardware Implementation ............................... 10
  2.5 Conclusion .................................................. 14

3 Appendix: References ............................................. 15

4 Appendix: Importing IP Core to User Vault .............................. 16
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Exiting Flash*Freeze Using Dip Slide Switches</td>
<td>3</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Launching SmartDebug Design Tools</td>
<td>4</td>
</tr>
<tr>
<td>Figure 3</td>
<td>SmartDebug Window - Debug FPGA Array</td>
<td>4</td>
</tr>
<tr>
<td>Figure 4</td>
<td>SRAM Read-Back Content before Flash*Freeze Entry</td>
<td>5</td>
</tr>
<tr>
<td>Figure 5</td>
<td>MSS Clock CCC - Advanced Options</td>
<td>7</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Top-Level Block Diagram of the Design</td>
<td>8</td>
</tr>
<tr>
<td>Figure 7</td>
<td>DIP Switches and the SW4 Connectivity in Top-Level Design</td>
<td>9</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Top-Level Hardware Design</td>
<td>10</td>
</tr>
<tr>
<td>Figure 9</td>
<td>System Builder Configurations for HPMS System Services and eNVM</td>
<td>11</td>
</tr>
<tr>
<td>Figure 10</td>
<td>HPMS System Clocks Configurations</td>
<td>12</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Flash*Freeze Hardware Settings Dialog</td>
<td>12</td>
</tr>
<tr>
<td>Figure 12</td>
<td>CORERESETP Configurator window</td>
<td>13</td>
</tr>
<tr>
<td>Figure 13</td>
<td>SmartDesign Component of CORERESETP</td>
<td>13</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Specifying I/O State and Functionality Options Using I/O Editor</td>
<td>14</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Catalog Tab</td>
<td>16</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Selecting the Add Core to Vault Option</td>
<td>17</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Add Core to Vault Dialog Box</td>
<td>17</td>
</tr>
</tbody>
</table>
## Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Design Requirements</td>
<td>2</td>
</tr>
<tr>
<td>Table 2</td>
<td>Board Jumper Settings</td>
<td>2</td>
</tr>
<tr>
<td>Table 3</td>
<td>LED to Pin Assignment (IGLOO2 Evaluation Kit Board)</td>
<td>7</td>
</tr>
<tr>
<td>Table 4</td>
<td>DIP Switch to Package Pin Assignment</td>
<td>14</td>
</tr>
</tbody>
</table>
1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the most current publication.

1.1 Revision 7.0
Updated the document for Libero v11.8 software release.

1.2 Revision 6.0
The following is a summary of the changes in revision 6.0 of this document.
• The Libero SoC and FlashPro versions were updated in the Design Requirements, page 2.
• The design files and the document was updated for Libero SoC v11.7 SP3.
• Added a new section, Prerequisites, page 2.
• The block diagram was updated to include the FLASH_FREEZE macro.
• The significance of CoreResetP IP core v8.0.103 was elaborated. For more information, see Hardware Implementation, page 10.

1.3 Revision 5.0
Updated the document for Libero v11.7 software release (SAR 75558).

1.4 Revision 4.0
Updated the document for Libero v11.6 software release (SAR 68372).

1.5 Revision 3.0
Updated the document for Libero v11.5 software release (SAR 62939).

1.6 Revision 2.0
Updated the document for Libero v11.4 software release (SAR 59065).

1.7 Revision 1.0
Revision 1.0 was the first publication of this document.
2 IGLOO2 FPGA Flash*Freeze Entry and Exit

Microsemi IGLOO®2 field programmable gate array (FPGA) devices provide an ultra-low static power solution through Flash*Freeze technology. Entry into the Flash*Freeze mode retains all the SRAM and registers information. Exit from the Flash*Freeze mode achieves rapid recovery to the active mode.

This application note specifies how to enter and exit the Flash*Freeze mode on the IGLOO2 Evaluation Board using the”.stp” programming file. The SRAM content retention capability during Flash*Freeze is also shown in this application note.

The “.stp” file is present at the following location of the design files folder:

m2gl_ac412_flashfreeze_liberov11p8_df\Programming_File

For more information on the Flash*Freeze entry and exit implementation, Flash*Freeze Libero design project, and all the necessary blocks and IP cores instantiated in Libero SoC, see Design Details, page 5.

2.1 Design Requirements

<table>
<thead>
<tr>
<th>Table 1 • Design Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware Requirements</strong></td>
</tr>
<tr>
<td>IGLOO2 Evaluation Kit</td>
</tr>
<tr>
<td>Host PC</td>
</tr>
<tr>
<td><strong>Software Requirements</strong></td>
</tr>
<tr>
<td>Libero SoC</td>
</tr>
<tr>
<td>FlashPro programming software</td>
</tr>
<tr>
<td>Host PC Drivers</td>
</tr>
<tr>
<td>CoreSysServices</td>
</tr>
</tbody>
</table>

2.2 Prerequisites

Before you start:

1. Download and extract the design files from the following link:

   http://soc.microsemi.com/download/rsc/?f=m2gl_ac412_flashfreeze_liberov11p8_df

   The design file has Libero SoC Verilog project, the .mem file for the eNVM data storage client, CPZ file of CoreResetP v8.0.103, and programming files (*.stp) for IGLOO2 Evaluation Kit board. See the Readme.txt file included in the design file for the directory structure and description.

2. Connect the power supply cable to the J6 connector on the board.
3. Connect the FlashPro4 programmer to the PROG HEADER J5 connector on the board.
4. Connect the jumpers to the IGLOO2 FPGA Evaluation Kit board as shown in the following table.

<table>
<thead>
<tr>
<th>Table 2 • Board Jumper Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Jumper</strong></td>
</tr>
<tr>
<td>J3</td>
</tr>
<tr>
<td>J8</td>
</tr>
</tbody>
</table>
2.3 Enter and Exit Flash*Freeze

All the necessary blocks of the device are programmed using the ".stp" file. Flash*Freeze entry and exit service requests can be initiated using the SW2 and SW4 push buttons available on the board. Then, SmartDebug is launched through Libero SoC to read the SRAM content to see that the content were retained during Flash*Freeze.

Follow these steps to program the device:

1. Turn on the board using the SW7 slide switch.
2. Start FlashPro, and click New Project to create a new FlashPro project.
3. Create a new project folder and select the Single device option in the New Project Dialog box.
4. Click the Configure Device and browse the existing ".stp" programming file to load it.
5. Click the Program button to program the device.

The Programmer List Window in the FlashPro, shows the Programmer Name, Programmer Type, Port, Programmer Status, and the Programmer Enabled information.

When the device is programmed, the Programmer Status column displays the "RUN PASSED" status. And, the H5, H6, J6, and H7 LEDs start blinking.

Follow these steps to enter and exit Flash*Freeze:

1. Press the SW2 push button on the board to enter Flash*Freeze.
   The device enters Flash*Freeze and the H5, H6, J6, and H7 LEDs stop blinking.
2. Press the SW4 push button on the board to exit Flash*Freeze.
   The device comes to active mode and the H5, H6, J6, and H7 LEDs start blinking.

   Note: You can also exit Flash*Freeze using the dip slide switches on the board as shown in the following figure.

Figure 1 • Exiting Flash*Freeze Using Dip Slide Switches

3. While the device is in the active mode, double-click SmartDebug Design from the Design Flow window, as shown in the following figure.
The SmartDebug window appears.

4. Click **Debug FPGA Array**, as shown in the following figure.

**Figure 2 • Launching SmartDebug Design Tools**

The debug file is automatically generated into the Libero SoC project (`<Libero SoC project path>/designer/<top level design name>/<design_name>_debug.txt`). The debug file is automatically loaded into the **SmartDebug** window.

5. Select the **Memory Blocks** tab in the **Debug FPGA Array** window, double-click the Memory Block listed memories and click **Read Block**. The SmartDebug tool reads the SRAM content from the device and shows it in the **Memory Block** data section, as shown in the following figure.
6. Enter Flash*Freeze using the SW2 push button and then exit Flash*Freeze using the SW4 push button.

7. Select the **Memory Blocks** tab in the **Debug FPGA Array** window, double-click the Memory Block listed memories and click **Read Block**. The SmartDebug tool reads the SRAM same content from the device and shows it in the **Memory Block** data section, as shown in the previous figure. This shows that the SRAM content was retained during Flash*Freeze.

### 2.4 Design Details

One of the functions of the system controller in the IGLOO2 device is to handle the system services requests through the communication block (COMM_BLK). The system services are grouped into different services. The IGLOO2 device enters Flash*Freeze mode by using the Flash*Freeze services request that the system controller provides. Some of the Flash*Freeze hardware settings options can be set during the design time, such as the clock source to be used as the standby clock source for the high performance memory subsystem (HPMS) during Flash*Freeze or defining the state of the fabric SRAM during the Flash*Freeze mode.

The HPMS standby clock source and the state of the SRAMs are configured in the Flash*Freeze hardware settings in the Libero® System-on-Chip (SoC) software. The fabric SRAM state during Flash*Freeze can either be sleep or suspend mode. In the suspend mode, the large SRAM (LSRAM) and micro SRAM (µSRAM) contents are retained, when the device exits the Flash*Freeze mode. In the sleep mode, the SRAMs contents are not retained. Exiting Flash*Freeze is achieved by user configurable mechanism through external I/O events (either transitions or pattern matching on I/Os). The state and the role that I/Os play during Flash*Freeze must be specified during the design time using Libero SoC. There are three different settings available. These settings are categorized as the I/O state in the Flash*Freeze mode, I/O availability in the Flash*Freeze mode, and I/O role in exiting the Flash*Freeze mode.

Depending on the type of the I/O, some or all of these options may not be available. For more information, see the **UG0444: SmartFusion2 SoC FPGA and IGLOO2 FPGA Low Power Design User Guide**.

Flash*Freeze entry is implemented using the system services, through the CoreSysServices soft IP, which provides access to the system services. The CoreSysServices soft IP communicates with the COMM_BLK through one of the fabric interface controllers (FICs). Each System Service has a service request phase and a response phase. For more information, see the **CoreSysServices IP Handbook**, which can be accessed through the Libero SoC software.

### 2.4.1 Design Description

The design example consists of the HPMS configured using system builder, a counter, SRAM wrapper logic, IP cores (CoreSysServices, CoreAHBLite, CoreAHBToAPB3, and CoreAPB3), FLASH_FREEZE
macro, fabric AHB master, on-chip 1 MHz RC oscillator, fabric CCC (FCCC), Flash*Freeze request, command generator logic (FF_BLKS), and a synchronizer counter (CLK_Sync_CNTR_Dly) to synchronize the clocks between the fabric and the HPMS system clock after exiting Flash*Freeze. The fabric AHB master along with the SRAM wrapper (AHBMASTER_FIC_RAM) is used to initialize the fabric SRAM by moving data from the embedded non-volatile memory (eNVM) to the fabric SRAM through FIC_0 AHB master and slave interfaces using the AHB master in the fabric. A data storage client is defined in the eNVM with the data to be written to the SRAM. This is used to demonstrate the state of the fabric SRAM content after exiting the Flash*Freeze mode.

In the active mode, the HPMS_CCC is configured to provide a 100 MHz clock that is sourced from the FPGA fabric through the CLK_BASE port. The FCCC is configured to provide the 50 MHz CLK_BASE reference. The on-chip 1 MHz oscillator is the reference clock source for the FCCC.

The CoreSysServices IP is configured to use only the Flash*Freeze service option. It sends the Flash*Freeze command to the system controller whenever it receives the Flash*Freeze request enable and command from the FF_BLKS logic. The FF_BLKS logic generates the Flash*Freeze request and command, based on the Flash*Freeze entry input signal (ff_trig). The FF_BLKS logic also monitors the busy signal from the CoreSysServices IP and the FF_TO_START, FF_DONE signals from the FLASH_FREEZE macro.

The FF_TO_START signal is asserted by the system controller to indicate that the Flash*Freeze service is about to start. Only 10 μs are available to do housekeeping before the core is powered off. We recommend using this signal as part of the clock gating process to ensure that any glitches do not cause a sequential element in the design to transition to an unwanted state when entering Flash*Freeze. The FF_DONE signal is asserted by the system controller to indicate that the Flash*Freeze service is about to end. It gets asserted before fabric registers are restored from their corresponding suspend latches and gets de-asserted after fabric restoration is complete. For more information about the FLASH_FREEZE macro, see the UG0450: SmartFusion2 SoC FPGA and IGLOO2 FPGA System Controller User Guide.

When the system enters Flash*Freeze mode, the main clock is switched to a standby clock that is defined by the user, where the user sets the Flash*Freeze hardware settings in the Libero design flow, as shown in Figure 11, page 12.

When the system controller comes out of the Flash*Freeze mode, MSS_CCC still runs off the standby clock. The system controller then waits for the MCCC_MPLL_LOCK assertion and then switches the clock to user system clock. After the lock assertion and before the MSS_CCC clock is switched to user clock, the system controller is ready to communicate with the fabric. To exit Flash*Freeze process, switch from the standby clock to the system clock (user clock) and wait for the MPLL lock and wait for the HPMS and fabric interface to be aligned.

Within the MSS or HPMS CCC, the fabric alignment clock controller (FACC) interfaces with the MPLL, generating the aligned clocks required by the MSS or HPMS sub-blocks, and controls the alignment of the FPGA fabric interface clocks. MCCC_GLMUX_SEL is the register that contains the select line for the four non-glitch multiplexers within FACC, which are related to the aligned clocks. All the four multiplexers are switched by one signal as follows:

- 1: M3_CLK, APB_0_CLK, APB_1_CLK, DDR_SMC_FIC_CLK all driven from CLK_STANDBY
- 0: M3_CLK, APB_0_CLK, APB_1_CLK, DDR_SMC_FIC_CLK all driven from stage B dividers

For more information about the description of the FACC, see the UG0449: SmartFusion2 and IGLOO2 Clocking Resources User Guide.

The sync-up counter logic (CLK_Sync_CNTR_Dly) achieves the following:

- Waits for MCCC_MPLL_LOCK to assert
- Waits for MCCC_GLMUX_SEL to switch to the user clock
- Accounts for the time required for the HPMS clock to switch from the standby clock to the operating clock after PLL achieves the lock and the system controller is ready to communicate with the fabric

When MCCC_MPLL_LOCK achieves lock, MCCC_GLMUX_SEL selects the user clock instead of the standby clock and the required time passes. GL0_EN is asserted to enable the GL0 clock that clocks the fabric logic.
If you are using the system builder, convert the system builder block into the SmartDesign block, to expose the MCCC_MPLL_LOCK and MCCC_GLMUX_SEL signals to the fabric. In the HPMS block, enable the options in the **Advanced Options** tab, as shown in the following figure.

**Figure 5 • MSS Clock CCC - Advanced Options**

![MSS Clock CCC - Advanced Options](image)

The output of a counter is connected to a set of light-emitting diodes (LEDs) to monitor the state of the fabric while entering and exiting the Flash*Freeze mode. The following table shows the LED to pin assignment.

**Table 3 • LED to Pin Assignment (IGLOO2 Evaluation Kit Board)**

<table>
<thead>
<tr>
<th>Counter Output</th>
<th>Package Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>LED_1</td>
<td>F4</td>
</tr>
<tr>
<td>LED_2</td>
<td>F3</td>
</tr>
<tr>
<td>LED_3</td>
<td>G7</td>
</tr>
<tr>
<td>LED_4</td>
<td>H7</td>
</tr>
</tbody>
</table>

The following figure shows the top-level block diagram with the main blocks used in the design.
2.4.2 Entering Flash*Freeze Mode

Entering Flash*Freeze is done through the system services using CoreSysServices IP core. The Flash*Freeze request and command service is generated by initiating the Flash*Freeze entry request through the port ff_trig to the FF_BLKS. Upon the trigger of the ff_trig port, the FF_BLKS sends a service enable request along with a service command byte describing the function to be performed. The Flash*Freeze service requests the system controller to execute the Flash*Freeze entry sequence. When the Flash*Freeze service begins execution, the system controller informs the HPMS by sending a command byte E0H that Flash*Freeze shutdown is imminent. The service is stalled until this command byte is accepted by the COMM_BLK FIFO. If a new service request is received while servicing another request, the new service request is immediately aborted. For more information, see the Flash*Freeze Service section in the UG0450: SmartFusion2 SoC FPGA and IGLOO2 FPGA System Controller User Guide.

As the Flash*Freeze system service command is initiated, the system controller disables the fabric, each eNVM block, or the MSS PLL circuit based on the options specified. All these options are available as system services through CoreSysServices IP core by defining the SERV_OPTION_MODE [2:0] input.
This defines the mode options for Flash*Freeze. For more information, see the CoreSysServices IP Handbook.

2.4.3 Exiting Flash*Freeze Mode

Exiting the Flash*Freeze mode can be initiated by external I/O events. User I/Os (MSIO, MSIOD, or DDRIO) that are single-ended inputs can participate in the Flash*Freeze exit in the following two ways:

- I/O activity: Force Flash*Freeze exit upon an activity (Wake_On_Change)
- I/O signature: Force Flash*Freeze exit upon a signature (Wake_On_1/Wake_On_0) match in which the I/O participates with other I/Os to trigger Flash*Freeze exit. This is a logical AND behavior where all I/Os must meet the Low Power Exit settings.

The external I/O events are specified during the design time using the I/O editor in the Libero SoC software. Only input I/Os participate in the Flash*Freeze exit event.

Note: The Wake_On_Change is logical OR behavior with I/Os that are set as Wake_ON_1/ Wake_ON_0. This means that to wake from Flash*Freeze, it must be {(All Wake-on-0 ANDed) ANDed with (All Wake-on-1 ANDed}) ORed with (All Wake-on-Change ORed).

2.4.3.1 I/O Activity

In the I/O activity mode, an input I/O can be selected to be part of a transition. The value at the pin of the activity I/O is latched before going to the low-power mode. When a change happens on the configured I/O, the device wakes up from the Flash*Freeze mode. The change can either be 1 to 0 or 0 to 1. This option is equivalent to the Wake_On_Change option in the I/O editor. This can be set on more than one I/O. The Wake_On_Change is a logical OR behavior with other I/Os that are set as Wake_On_Change.

2.4.3.2 I/O Signature

Any input I/O can be selected to be a part of a signature match value that is used to wake-up from the Flash*Freeze mode. All the selected I/Os have to match a static predetermined value at the same time. If the configured signature values match the values at I/Os, then the device exits the Flash*Freeze mode. I/Os can be a mixture of different signature settings. An I/O can be configured to participate in the Flash*Freeze exit upon a 0 to 1 or it can be configured to participate in the Flash*Freeze exit upon a 1 to 0 transition. These options are equivalent to Wake_On_1 (transition from 0 to 1) and Wake_On_0 (transition from 1 to 0) settings in the I/O editor in the Libero SoC software.

All other I/Os that are not participating in the Flash*Freeze exit mechanism are tristated or held to the previous state (LAST_VALUE) before entering the Flash*Freeze mode. The selection is set using I/O state in Flash*Freeze mode column options in the I/O editor using the Libero SoC, as shown in Figure 14, page 14.

SW5 (four different dual in-line package (DIP) switches) on the IGLOO2 Evaluation Kit board is used to demonstrate the pattern matching wake-up mechanism. Four different inputs are created in the top-level design where each input is assigned to a DIP switch.

SW4 on the Evaluation Kit board is used to demonstrate the transition (Wake_On_Change) wake-up event mechanism, as shown in the following figure.

Figure 7 • DIP Switches and the SW4 Connectivity in Top-Level Design
2.4.4 Hardware Implementation

The hardware implementation involves configuring the HPMS and the necessary Flash*Freeze settings. The HPMS configuration is done using the system builder. The design example consists of the HPMS, a counter, SRAM wrapper logic, IP cores (CoreSysServices, CoreAHBLite, CoreAHBToAPB3, and CoreAPB3), FLASH_FREEZE macro, fabric AHB master, on-chip 25/50 MHz RC oscillator, FCCC, and FF_BLKS as shown in Figure 8, page 10. The FLASH_FREEZE macro in the FF_BLKS provides the FF_TO_START signal to indicate the start of flash freeze to user logic and the FF_DONE signal connected to the HPMS system builder. The IP cores along with the SRAM wrapper are used to initialize the fabric SRAM (AHBMASTER_FIC_RAM) by moving data from the eNVM to the fabric SRAM through FIC_0 AHB master and slave interfaces. A data storage client is defined in the eNVM with the data to be written to the SRAM. This is used to demonstrate the state of the fabric SRAM content after exiting Flash*Freeze.

Figure 8 • Top-Level Hardware Design

The HPMS is configured using the Device Features page in the system builder, to use HPMS system services and HPMS on-chip Flash Memory (eNVM), as shown in Figure 9, page 11. The HPMS is also configured to provide the clock and reset signals to all the blocks including the CoreSysServices IP and FF_BLKS.
The eNVM data storage client is defined using the configure flash memory option under the Memories page in the system builder configurator. The ".mem" file defines the data storage client at `<project location>\IGLOO2_FlashFreeze\constraint\folder`.

The HPMS_CCC clock source is sourced from the FPGA Fabric Input through the CLK_BASE port where an FCCC is used. The FCCC is configured to provide a 50 MHz CLK_BASE clock using GL0 output. The reference clock for the FCCC is the on-chip 50 MHz RC oscillator. The following figure shows the system clocks configurations for the HPMS_CLK and FIC_0_CLK clock settings. The system builder automatically instantiates FCCC and RCOSC and configures them accordingly.
**Figure 10 • HPMS System Clocks Configurations**

![HPMS System Clocks Configurations](image)

**Note:** Connect the inverted FF_DONE signal to all the fabric CCC reset inputs (PLL_ARST_N) for resetting the CCC during Flash*Freeze.

The standby clock source for the HPMS and the state of the SRAMs (µSRAM and LSRAM) during the Flash*Freeze mode are configured using the **Flash*Freeze Hardware Settings** dialog-box in the Libero SoC software, as shown in Figure 11, page 12. The following are the HPMS clock source options that are available to be used during the Flash*Freeze mode:

- On-chip 1 MHz RC oscillator
- On-chip 50 MHz RC oscillator

Suspend and sleep modes are the µSRAM/LSRAM state options that are available to be used during the Flash*Freeze mode.

**Figure 11 • Flash*Freeze Hardware Settings Dialog**

![Flash Freeze Hardware Settings](image)
We recommend using CoreResetP IP core v8.0.103 included in the design files to ensure that FF_DONE signal is used to gate any signal that is used as asynchronous resets or presets in fabric and signals that are intended for use as inputs to ASIC blocks on the device (MDDR, FDDR and SERDES). This is to avoid any spurious resets as we exit Flash*Freeze.

You can implement Flash*Freeze in your existing design by importing the CoreResetP IP core. For more information on importing this IP core, see Appendix: Importing IP Core to User Vault, page 16.

The following figure shows how to enable Flash*Freeze support using CORERESETP configurator window.

*Figure 12 • CORERESETP Configurator window*

The following figure shows the SmartDesign component of CORERESETP with Flash*Freeze support enabled.

*Figure 13 • SmartDesign Component of CORERESETP*

The I/O Flash*Freeze exit mechanism is specified using the Low Power Exit setting in the I/O editor, as shown in *Figure 14*, page 14. Please note the following points:

- The I/O available in Flash*Freeze option applies only to I/Os allocated to the HPMS peripherals.
- When I/Os are set to be available during the Flash*Freeze mode, the I/O state in Flash*Freeze option does not apply.
• Only inputs or bidirectional I/Os participate in signature/activity Flash*Freeze exit. This means that
the low-power exit options are available to be set on inputs and/or bidirectional I/Os only.

**Figure 14 • Specifying I/O State and Functionality Options Using I/O Editor**

<table>
<thead>
<tr>
<th>Part Name</th>
<th>Pin Number</th>
<th>I/O state in Flash*Freeze mode</th>
<th>Resistor Pull</th>
<th>I/O available in Flash*Freeze mode</th>
<th>Low Power Exit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wake_On_Change_SW4</td>
<td>J18</td>
<td>TRISTATE</td>
<td>None</td>
<td>No</td>
<td>Off</td>
</tr>
<tr>
<td>DIP1 on 1 to 1</td>
<td>L19</td>
<td>TRISTATE</td>
<td>Down</td>
<td>No</td>
<td>Wake_On_1</td>
</tr>
<tr>
<td>DIP2 On 1 to 2</td>
<td>L18</td>
<td>TRISTATE</td>
<td>Down</td>
<td>No</td>
<td>Wake_On_1</td>
</tr>
<tr>
<td>DIP3 On 0 to 1</td>
<td>K21</td>
<td>TRISTATE</td>
<td>Up</td>
<td>No</td>
<td>Wake_On_0</td>
</tr>
<tr>
<td>DIP4 On 0 to 1</td>
<td>K20</td>
<td>TRISTATE</td>
<td>Up</td>
<td>No</td>
<td>Wake_On_0</td>
</tr>
<tr>
<td>ff_trig SW2 K16</td>
<td>K16</td>
<td>TRISTATE</td>
<td>None</td>
<td>No</td>
<td>Off</td>
</tr>
</tbody>
</table>

The Flash*Freeze exit behavior of input I/Os (DIP1-4) and SW5 are configured using the I/O editor, as shown in the previous figure.

The DIP switch-to-package pin assignment for the IGLOO2 Evaluation Kit is shown in the following figure.

**Table 4 • DIP Switch to Package Pin Assignment**

<table>
<thead>
<tr>
<th>Input DIP Switch</th>
<th>Package Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIP1</td>
<td>L19</td>
</tr>
<tr>
<td>DIP2</td>
<td>L18</td>
</tr>
<tr>
<td>DIP3</td>
<td>K21</td>
</tr>
<tr>
<td>DIP4</td>
<td>K20</td>
</tr>
<tr>
<td>SW4</td>
<td>J18</td>
</tr>
<tr>
<td>SW2 (ff_trig)</td>
<td>K16</td>
</tr>
</tbody>
</table>

**2.5 Conclusion**

This application note specified how to enter and exit Flash*Freeze on the IGLOO2 device using the "*.stp" programming file. The SRAM content retention capability during Flash*Freeze was also shown in this application note.
The following references complement and help in understanding the relevant Microsemi IGLOO2 FPGA device features and flows that are demonstrated in this document.

- For information on the Flash*Freeze services provided by the System Controller, see UG0450: *SmartFusion2 SoC FPGA and IGLOO2 FPGA System Controller User Guide*.
- For information on the Flash*Freeze technology supported by SmartFusion2 and IGLOO2 devices, see UG0444: *SmartFusion2 SoC FPGA and IGLOO2 FPGA Low Power Design User Guide*.
- For more information on the system services, see *CoreSysServices IP Handbook*. 
4 Appendix: Importing IP Core to User Vault

The following steps describe how to import the CoreResetP IP core v8.0.103 to User Vault in Libero SoC.

1. Select the **Catalog** tab in Libero SOC as shown in the following figure.

   *Figure 15 • Catalog Tab*

   ![Catalog Tab](image)

2. Click the **Settings** drop-down and select the **Add Core to Vault** option as shown in the *Figure 16*, page 17.
Figure 16 • Selecting the Add Core to Vault Option

The **Add Core to Vault** dialog box opens. Change file type to core files (.ccz,.cpz) from the drop down list and navigate the IP core location as shown in the following figure.

Figure 17 • Add Core to Vault Dialog Box

You have successfully imported the CoreResetP IP core to the User Vault.