DirectC v4.0

User Guide

Microsemi®

Power Matters.™

Table of Contents

Introduction

This document describes how to enable microprocessor-based embedded ISP (In-System Programming) on Microsemi PolarFire™, RTG4™, IGLOO2™, SmartFusion2™, ProASIC®3 (including ProASIC3 nano), IGLOO™ (including IGLOO nano), SmartFusion™ and Fusion™ devices. In-System Programming refers to an external processor on board programming a Microsemi device through general purpose IOs using a JTAG interface.

The document assumes that the target system contains a microprocessor with a minimum 256 bytes of RAM, a JTAG interface to the target device from the microprocessor, and access to the programming data to be used for programming the FPGA. Access to programming data can be provided by a telecommunications link for most remote systems.

DirectC v4.0 is a set of C code designed to support embedded In-System Programming for AGL, AFS, A3PL, A3PEL, A3P/E, A2F, M2S, M2GL, RTG4, and MPF families. To use DirectC v4.0, you must make some minor modifications to the source code, add the necessary API, and compile the source code and the API together to create a binary executable. The binary executable is downloaded to the system along with the programming data file.

The programming data file is a binary file that can be generated by Libero SoC. The detailed specification of the programming file is included in "Data File Format" on page 24.

DirectC v4.0 contains several compile options to reduce the code size as much as possible. The compile options enable you to disable support for specific device families and features that are not needed in the compile.

DirectC v4.0 supports systems with direct and indirect access to the memory space containing the data file image. With paging support, it is possible to implement the embedded ISP using DirectC on systems with no direct access to the entire memory space containing the data. Paging support is accomplished by making modifications to the data communication functions defined in *dpuser.h*, *dpuser.c*, *dpcom.c*, and *dpcom.h*.

Important Note

This version of DirectC supports RTG4 family of devices. These devices have a feature called Avionics mode. When enabled, it prevents the user from performing programming operation.

To exit out of this mode, the JTAG_TRST pin must be held high and DEVRST_N pin must be toggled.

dp_exit_avionics_mode function is created for that purpose. It is defined in d*puser.c* and must be modified by the user to set JTAG_TRST pin high and toggle DEVRST_N pin.

1 – System Overview

To perform In-System Programming (ISP) for the FPGA, the system must contain the following parameters:

- Control logic (a microprocessor or a softcore microprocessor implemented in another FPGA)
- JTAG interface to the target device
- Access to the data file containing the programming data
- Memory to store and run DirectC code

Note: See your device datasheet for information on power requirements for V_{pump}, V and other power supplies.

Memory requirements depend on the options that are enabled. Table 1-1 is an example of the code size and run time memory required to support the different device families. Refer to "Required Source Code Modifications" on page 12 for more detailed description of available compiler switches.

Text - This is the compiled code size memory requirements.

Data - This is the run time memory requirement, i.e. the free data memory space required to execute the code.

BSS - This is the Block Started by Symbol allocation for variables that do not yet have values, i.e. uninitialized data. It is part of the overall Data size.

***Table 1-1* • Code Memory Requirements- DirectC Code Size on CM3 in Thumb Mode**

Compile Options Enabled	Units are in Bytes		
	Text	Data	BSS
ENABLE_G3_SUPPORT	31168	2393	99
ENABLE_G4_SUPPORT	15200	2398	98
ENABLE_G5_SUPPORT	15960	2422	98
ENABLE_RTG4_SUPPORT	12260	3031	98
All the above	53324	3302	91

Note: All compile options related to conserving code space are relevant to A3P, AGL, Fusion, and SmartFusion device support. If the "ENABLE_G3_SUPPORT" compile option is not defined, these compile options do not make a difference in reducing the memory size required to support M2S/M2GL and RTG4 devices. See "Required Source Code Modifications" on page 12 for details about all compile options.

Systems with Direct Access to Memory

Figure 1-1 shows the overview of a typical system with direct access to the memory space holding the data file. See "Generating Data Files and Integrating DirectC" on page 8 for generating DAT files and Table 1-2 for data storage memory requirements. Systems with Indirect Access to Memory

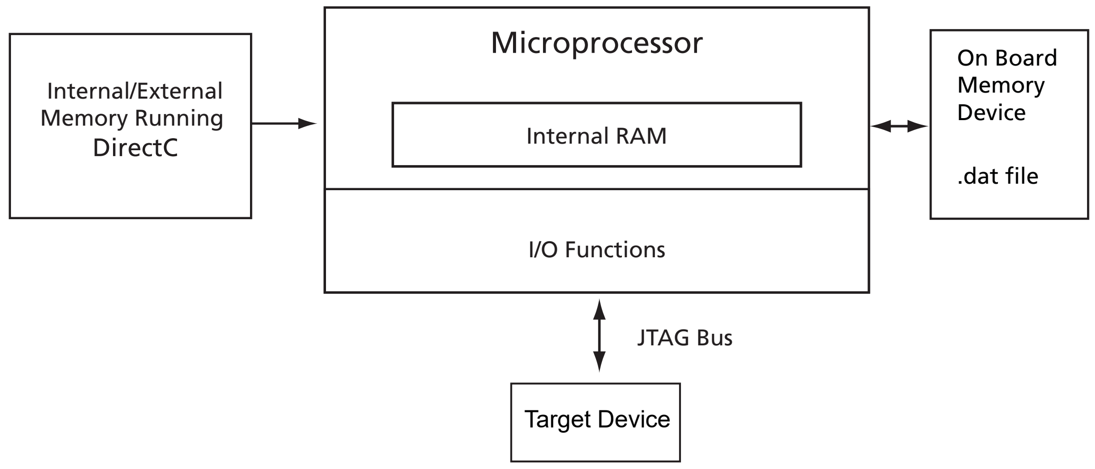


Figure 1-1 • **System with Direct Access to Memory**

Figure 1-2 is an overview of a system with no direct access to the memory space holding the data file. For example, the programming data may be received via a communication interface peripheral that exists between the processor memory and the remote system holds the data file. *dpcom.h* and *dpcom.c* must be modified to interface with the communication peripheral.

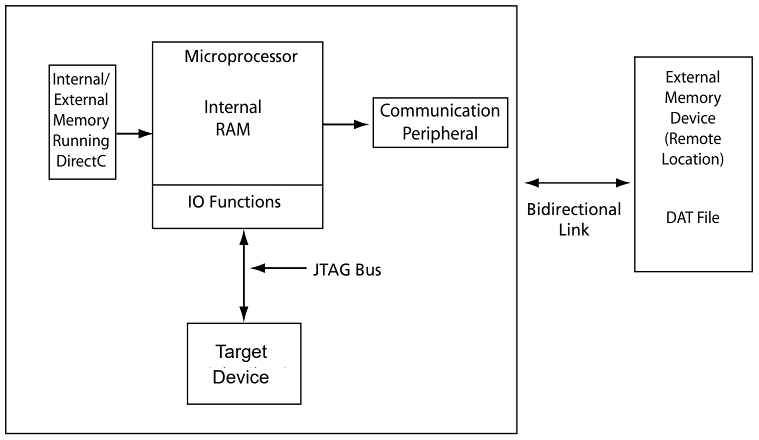


Figure 1-2 • **System With Indirect Access to Memory**

Table 1-2 **• Data Storage Memory Requirements - Data Image Size**

	Data Image Size						
	Core/FPGA Array		**FROM**		**Embedded Flash Memory Block**		
Device	**Plain (kB)**	**Encrypt (kB)**	**Plain (kB)**	**Encrypt (kB)**	**Plain (kB)**	**Encrypt (kB)**	**Security (kB)**
A3PE600	526	647	1	1	N/A	N/A	1
A3PE1500*	1434	1765	1	1	N/A	N/A	1
A3PE3000	2790	3433	1	1	N/A	N/A	1
A3P015	32	N/A	1	N/A	N/A	N/A	1
A3P030	32	N/A	1	N/A	N/A	N/A	1
A3P060	64	79	1	1	N/A	N/A	1
A3P125	127	156	1	1	N/A	N/A	1
A3P250	235	288	1	1	N/A	N/A	1
A3P400	351	432	1	1	N/A	N/A	1
A3P600	523	647	1	1	N/A	N/A	1
A3P1000	915	1126	1	1	N/A	N/A	1
AFS090	96	117	1	1	256	545	1
AFS250	234	288	1	1	256	545	1
AFS600	526	647	1	1	512	1090	1
AFS1500	1434	1765	1	1	2048	2180	1
A2F200M3F	181	222	1	1	256	545	1
A2F500M3G	455	560	1	1	512	1090	1
M2GL010	N/A	557	N/A	N/A	N/A	267	N/S
M2GL025	N/A	1197	N/A	N/A	N/A	267	N/S
M2GL050	N/A	2364	N/A	N/A	N/A	267	N/S
M2S005	N/A	297	N/A	N/A	N/A	137	N/S
M2S010	N/A	557	N/A	N/A	N/A	272	N/S
M2S025	N/A	1197	N/A	N/A	N/A	272	N/S
M2S050	N/A	2364	N/A	N/A	N/A	272	N/S
RT4G150	4992	N/A	N/A	N/A	N/A	N/A	N/A
MPF300	N/A	9472	N/A	N/A	N/A	N/A	N/A

A3PE1500 is not supported with an 8-bit processor.

INA - Information not available at this time.

N/A - Not applicable

N/S - Not supported

All data in the table for base FPGA devices applies equally to the M1, M7, P1, and U1 encrypted versions of the devices, e.g. data for AFS1500 is equally applicable to M1AFS1500, P1AFS1500, and U1AFS1500. Not all combinations of M1, M7, P1, and U1 are available for all devices. Refer to the product datasheets for available devices.

The total image size is the sum of all the corresponding enabled blocks for the specific target device.

2 – Generating Data Files and Integrating DirectC

This chapter describes the flows for data file generation and DirectC code integration.

To generate your data file:

1. Generate the DAT file using Designer v8.5 or later. DAT file generation is done by running **Export Bitstream** under **Handoff Design for Production** in the Libero Design Flow window. See the latest Libero SoC online help for information about generating a DAT file.
2. Program the DAT file into the storage memory.

Data File Compatibility

DirectC data files can be generated from Designer v8.5 and above. Data files generated from Designer v8.5 are identical to the files generated by the original datgen tool with the exception of the file title. However, data files generated by Designer version v8.6 are enhanced to support nano devices. DirectC v4.0 can detect which version of the file is being used and handle it accordingly.

DirectC v4.0 Code Integration

Figure 2-1 shows the DirectC integration use flow.

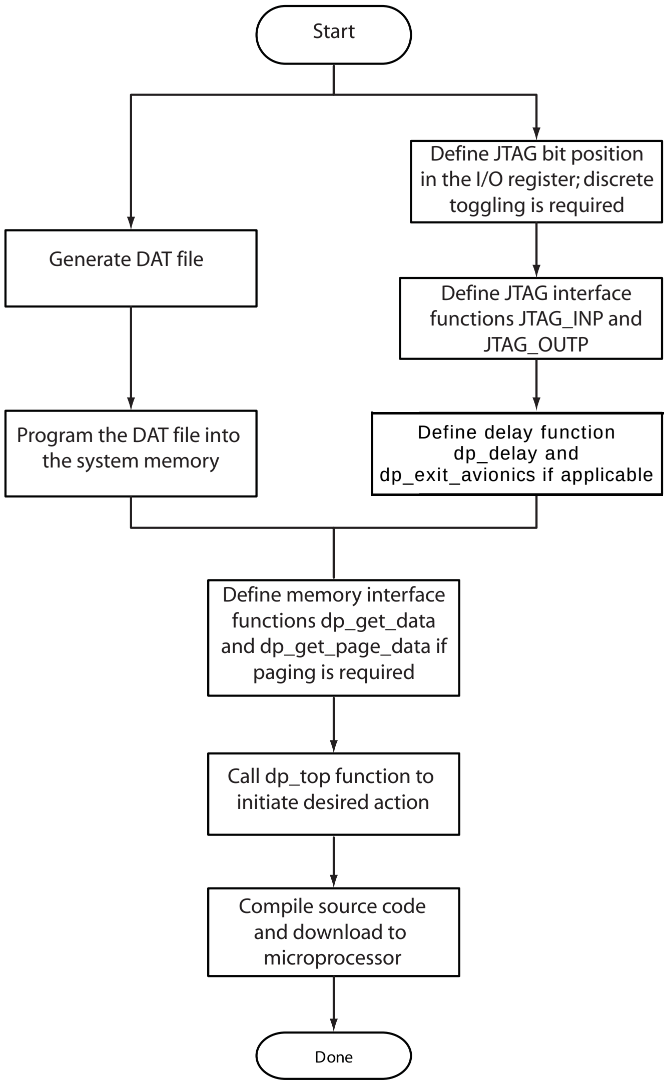
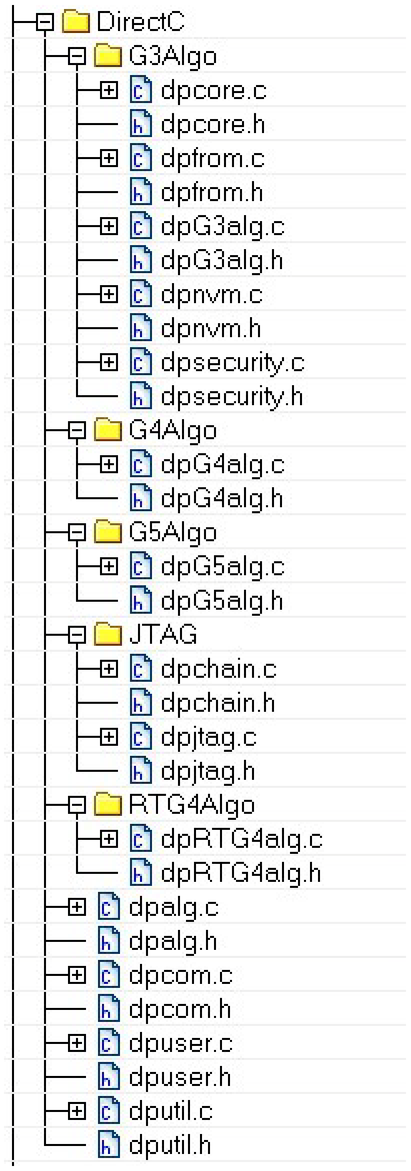

Start
Define JTAG bit position in the I/O register; discrete toggling is required
Generate DAT file
Define JTAG interface functions JTAG_INP and JTAG_OUTP
Program the DAT file into the system memory
Define delay function dp_delay and dp_exit_avionics if applicable
Define memory interface functions dp_get_data and dp_get_page_data if paging is required
Call dp_top function to initiate desired action
Compile source code and download to microprocessor
Done

***Figure 2-1* • Importing DirectC Files**

To use DirectC v4.0 code integration:

1. Import the DirectC v4.0 files shown in Figure 2-2 into your development environment.



```
DirectC
  G3Algo
    dpcore.c
    dpcore.h
    dpfrom.c
    dpfrom.h
    dpG3alg.c
    dpG3alg.h
    dpnvm.c
    dpnvm.h
    dpsecurity.c
    dpsecurity.h
  G4Algo
    dpG4alg.c
    dpG4alg.h
  G5Algo
    dpG5alg.c
    dpG5alg.h
  JTAG
    dpchain.c
    dpchain.h
    dpjtag.c
    dpjtag.h
  RTG4Algo
    dpRTG4alg.c
    dpRTG4alg.h
  dpalg.c
  dpalg.h
  dpcom.c
  dpcom.h
  dpuser.c
  dpuser.h
  dputil.c
  dputil.h
```

Figure 2-2 • **DirectC v4.0 Files to import into your Development Environment**

2. Modify the DirectC code. Refer to Figure 2-1.
   - Define JTAG pin bit locations in the I/O register.
   - For RTG4, assign an additional pin bit to control the devrst pin.
   - Add API to support discrete toggling of the individual JTAG pins.
   - Modify the hardware interface functions (*jtag_inp* and *jtag_outp*) to use the hardware API functions designed to control the JTAG port.
   - Modify the delay function (*dp_delay*).
   - Modify memory access functions to access the data blocks within the image file programmed into the system memory. See "Data File Bit Orientation" on page 31.

   - Call *dp_top* function with the action code desired.
3. Compile the source code. This creates a binary executable that is downloaded to the system for execution.

3 – Required Source Code Modifications

You must modify the *dpuser.h*, *dpuser.c, dpcom.c, dpcom.h, and dpG3alg.h* (if applicable) files when using the DirectC source code. "Source File Description" on page 28 contains a short description of DirectC source code and their function. Functions that must be modified are listed in Table 3-1.

***Table 3-1* • Functions to be Modified by the User**

Function	Source File	Purpose
jtag_inp	dpuser.c	Hardware interface function used to set JTAG pins and read TDO.
jtag_outp	dpuser.c	Hardware interface function used to set JTAG pins.
dp_get_page_data	dpcom.c	Programming file interface function.
dp_delay	dpuser.c	Delay function.
dp_display_text	dpuser.c	Function to display text to an output device. ENABLE_DISPLAY compile option must be defined.
dp_display_value	dpuser.c	Function to display value of a variable to an output device. ENABLE_DISPLAY compile option must be defined.
dp_exit_avionics_ mode	dpuser.c	Function to exit Avionics Mode for RTG4 devices.

Compiler Switches

The compiler switches in Table 3-2 are designed to allow you to easily adjust the compiled code size by enabling or disabling specific support in DirectC. For example, to enable FPGA Array (Core) plain text programming, CORE_SUPPORT and CORE_PLAIN must be defined. Table 3-2 lists the available compiler switches in the project.

***Table 3-2* • Compiler Switches**

Compiler Switch	Source File	Purpose
CORE_SUPPORT	dpG3alg.h	Enables FPGA Array Programming support.
CORE_ENCRYPT	dpG3alg.h	Specify to include FPGA Array Encrypted programming support.
CORE_PLAIN	dpG3alg.h	Specify to include FPGA Array Plain Text programming support.
FROM_SUPPORT	dpG3alg.h	Enables FlashROM Programming support.
FROM_ENCRYPT	dpG3alg.h	Specify to include FlashROM Encrypted programming support.
FROM_PLAIN	dpG3alg.h	Specify to include FlashROM Plain Text programming support.
NVM_SUPPORT	dpG3alg.h	Enables eNVM Programming support.

Table 3-2 • **Compiler Switches (continued)**

Compiler Switch	Source File	Purpose
NVM_ENCRYPT	dpG3alg.h	Specify to include eNVM Encrypted programming support.
NVM_PLAIN	dpG3alg.h	Specify to include eNVM Plain Text programming support.
SECURITY_SUPPORT	dpG3alg.h	Enables Security Programming support.
SILSIG_SUPPORT	dpG3alg.h	Enables SILSIG Programming support
ENABLE_DAS_SUPPORT	dpG3alg.h	Enables support for A3PE1500 rev A devices; support for this feature is not available on some 8-bit microcontrollers because of Run Time Memory requirements.
ENABLE_GPIO_SUPPORT	dpuser.h	This switch must be defined to enable external device programming.
ENABLE_G3_SUPPORT	dpuser.h	Enables support for AGL, AFS, A3PL, A3PEL, A3P/E, and A2F devices.
ENABLE_G4_SUPPORT	dpuser.h	Enables support for M2S and MGL devices.
ENABLE_G5_SUPPORT	dpuser.h	Enables support for MPF devices.
ENABLE_RTG4_SUPPORT	dpuser.h	Enables support for RTG4 devices.
ENABLE_DISPLAY	dpuser.h	Enables display functions.
USE_PAGING	dpuser.h	Used to enable paging implementation for memory access.
CHAIN_SUPPORT	dpuser.h	Used to enable support for chain programming as described in Table 4-2 on page 22.
BSR_SAMPLE	dpuser.h	Enable this option to maintain the last known IO states during programming. **BSR loading and BSR_SAMPLE are not supported for IAP.**
ENABLE_CODE_SPACE_OPTIMIZATION	dpG3alg.h	See "Disabled Features with ENABLE_CODE_SPACE_OPTIMIZATION" on page 30.
DISABLE_CORE_SPECIFIC_ACTIONS	dpG3alg.h	For code size reduction. This option will disable array specific actions such as erase, program and verify array actions.
DISABLE_FROM_SPECIFIC_ACTIONS	dpG3alg.h	For code size reduction. This option will disable FROM specific actions such as erase, program and verify FROM actions.
DISABLE_NVM_SPECIFIC_ACTIONS	dpG3alg.h	For code size reduction. This option will disable NVM specific actions such as program and verify NVM actions.

Table 3-2 • **Compiler Switches (continued)**

Compiler Switch	Source File	Purpose
DISABLE_SEC_SPECIFIC_ACTIONS	dpG3alg.h	For code size reduction. This option will disable security specific actions such as erase and program security actions.
PERFORM_CRC_CHECK	dpuser.h	Enables CRC check of the programming data prior to performing the desired action.

Note: Make sure that the appropriate compiler options are enabled to support all features available in the STAPL/DAT file. Otherwise, DirectC may report an error depending on the requested action. Avoid using source files that have all options enabled. The number of options selected incrementally increases the number of variables that need to be maintained and the amount of memory that is used.

Compiler options defined in dpG3alg.h are specific to the AGL, AFS, A3PL, A3PEL, A3P/E, and A2F families of devices, whereas compiler switches defined in dpuser.h are common to all devices.

Hardware Interface Components

Define JTAG Hardware Bit Assignments (dpuser.h)

Define the JTAG bits corresponding to each JTAG pin. This is usually the bit location of the I/O register controlling the JTAG port of the target device.

```
#define TCK 0x1 /* ... user code goes here ... */
#define TDI 0x2 /* ... user code goes here ... */
#define TMS 0x4 /*... user code goes here ... */
#define TRST 0x0 /* ... user code goes here ... set to zero if does not exist !!!*/
#define TDO 0x80 /*.. user code goes here ... */
```

Hardware Interface Function (dpuser.c)

jtag_inp and *jtag_outp* functions are used to interface with the JTAG port. A register *jtag_port_reg* is an 8 bit register already defined in DirectC. DirectC uses it to track the logical states of all the JTAG pins.

jtag_inp Function

This function returns the logical state of the TDO pin. If it is logic level zero, then this function must return zero. If the logical state is 1, then it must return 0x80.

jag_outp Function

This function takes one argument that is the value of the JTAG port register containing the states of all the JTAG pins. It sets the JTAG pins to the values in this argument.

Delay Function (dpuser.c)

dp_delay function takes one argument which is the amount of time in microseconds. Its purpose is to pause for a minimum of time passed in its argument.

Longer delay time does not impact programming other than programming time.

Display Functions (dpuser.c)

Display functions are only enabled if the ENABLE_DISPLAY compiler switch is enabled. Three functions, *dp_display_array, dp_display_text,* and *dp_display_value*, are available to display text as well as numeric values. You must modify these functions for proper operation.

Memory Interface Functions (dpuser.c)

All access to the memory blocks within the data file is done through *dp_get_data* function within the DirectC code. This is true for all system types.

This function returns an address pointer to the byte containing the first requested bit.

The dp_get_data function takes two arguments as follows:

- var_ID: an integer variable which contains an identifier specifying which block within the data file needs to be accessed.
- bit_index: The bit index addressing the bit to address within the data block specified in Var_ID. Upon completion of this function, the return_bytes variable must hold the total number of valid bytes available for the calling function.

See "Systems with Direct Access to the Memory Containing the Data File" on page 15 and "Systems with Indirect Access to the Data File" on page 15 for details.

Systems with Direct Access to the Memory Containing the Data File

Since the memory space holding the data file is accessible by the microprocessor, it could be treated as an array of unsigned characters. In this case, complete these steps:

1. Disable USE_PAGING compiler switch. See "Compiler Switches" on page 12.
2. Assign the physical address pointer to the first element of the data memory location (*image_buffer* defined in *dpcom.c)*. *image_buffer* is used as the base memory for accessing the information in the programming data in storage memory.

The dp_get_data function calculates the address offset to the requested data and adds it to *image_buffer*. *return_bytes* is the requested data.

An example of *dp_get_data* function implementation follows. This function can be used as is.

```
DPUCHAR* dp_get_data(DPUCHAR var_ID,DPULONG bit_index)
{
    DPUCHAR * data_address = (DPUCHAR*)DPNULL;
    dp_get_data_block_address(var_ID);
    if ((current_block_address == 0U) && (var_ID != Header_ID))
    {
        return_bytes = 0U;
    }
    else
    {
        data_address = dp_get_data_block_element_address(bit_index);
    }
    return data_address;
}
```

Systems with Indirect Access to the Data File

These systems access programming data indirectly via a paging mechanism. Paging is a method of copying a certain range of data from the memory containing the data file and pasting it into a limited size memory buffer that DirectC can access.

To implement paging:

1. Enable USE_PAGING compiler option. See "Compiler Switches" on page 12.
2. Define *Page_buffer_size*. The recommended minimum buffer size is 16 bytes for efficiency purposes. If eNVM encrypted programming support is required on SmartFusion or Fusion devices, two buffers are needed of Page_buffer_size. Therefore, the run time memory required must be able to hold 2 x Page_buffer_size.
3. Modify the *dp_get_page_data* function. This function copies the requested data from the external memory device into the page buffer. See "Data File Bit Orientation" on page 31 for additional information. Follow these rules for correct operation:
   - Fill the entire page unless the end of the image is reached. See "Data File Format" on page 24
   - Update *return_bytes* to reflect the number of valid bytes in the page.

DirectC programming functions call *dp_get_data* function every time access to a data block within the image data file is needed. The *dp_get_data* function calculates the relative address location of the requested data and checks if it already exists in the current page data. The paging mechanism is triggered if the requested data is not within the page buffer.

Example of dp_get_page_data Function Implementation

dp_get_page_data is the only function that must interface with the communication peripheral of the image data file. Since the requested data blocks may not be contiguous, it must have random access to the data blocks. Its purpose is to fill the page buffer with valid data.

In addition, this function must maintain *start_page_address*, *end_page_address,* and *return_bytes*. These global variables contain the range of data currently in the page as well as the number of valid bytes.

dp_get_page_data takes one argument:

- address_offset - Contains the relative address of the needed element within the data block of the image file.

```
void dp_get_page_data(DPULONG image_requested_address)
{
      DPULONG image_address_index;
      start_page_address=0;

      image_address_index=image_requested_address;
      return_bytes = PAGE_BUFFER_SIZE;
      if (image_requested_address + return_bytes > image_size)
            return_bytes = image_size - image_requested_address;

      while (image_address_index < image_requested_address + return_bytes)
      {
            page_global_buffer[start_page_address]=image_buffer[image_address_index];
            start_page_address++;
            image_address_index++;
      }
      start_page_address = image_requested_address;
      end_page_address = image_requested_address + return_bytes - 1;

      return;
}
```

Main Entry Function

The main entry function is *dp_top* defined in *dpalg.c*. It must be called to initiate the programming operation. Prior to calling the *dp_top* function, a global variable *Action_code* must be assigned a value as defined in *dpalg.h*. Action codes are listed below.

```
#define DP_DEVICE_INFO_ACTION_CODE              1
#define DP_READ_IDCODE_ACTION_CODE              2
#define DP_ERASE_ACTION_CODE                    3
#define DP_PROGRAM_ACTION_CODE                  5
#define DP_VERIFY_ACTION_CODE                   6
#define DP_ENC_DATA_AUTHENTICATION_ACTION_CODE  7
#define DP_ERASE_ARRAY_ACTION_CODE              8
#define DP_PROGRAM_ARRAY_ACTION_CODE            9
#define DP_VERIFY_ARRAY_ACTION_CODE             10
#define DP_ERASE_FROM_ACTION_CODE               11
#define DP_PROGRAM_FROM_ACTION_CODE             12
#define DP_VERIFY_FROM_ACTION_CODE              13
#define DP_ERASE_SECURITY_ACTION_CODE           14
```

```
#define DP_PROGRAM_SECURITY_ACTION_CODE         15
#define DP_PROGRAM_NVM_ACTION_CODE              16
#define DP_VERIFY_NVM_ACTION_CODE               17
#define DP_VERIFY_DEVICE_INFO_CODE              18
#define DP_READ_USERCODE_ACTION_CODE            19
#define DP_PROGRAM_NVM_ACTIVE_ARRAY_CODE        20
#define DP_VERIFY_NVM_ACTIVE_ARRAY_CODE         21
#define DP_IS_CORE_CONFIGURED_ACTION_CODE       22

/* Smart Fusion specific actions */
#define DP_PROGRAM_PRIVATE_CLIENTS_ACTION_CODE  23u
#define DP_VERIFY_PRIVATE_CLIENTS_ACTION_CODE   24u
#define DP_PROGRAM_PRIVATE_CLIENTS_ACTIVE_ARRAY_ACTION_CODE  25u
#define DP_VERIFY_PRIVATE_CLIENTS_ACTIVE_ARRAY_ACTION_CODE   26u
```

The following are the only actions supported on the RTG4, SmartFusion2, IGLOO2, and PolarFire family of devices.

```
#define DP_DEVICE_INFO_ACTION_CODE 1
#define DP_READ_IDCODE_ACTION_CODE 2
#define DP_ERASE_ACTION_CODE 3
#define DP_PROGRAM_ACTION_CODE 5
#define DP_VERIFY_ACTION_CODE 6
#define DP_ENC_DATA_AUTHENTICATION_ACTION_CODE 7
#define DP_VERIFY_DIGEST_ACTION_CODE 28
```

Note: For M2S/M2GL/RTG4 and MPF device families only. Programming of individual blocks such as array or eNVM is not possible with one DAT file that contains both array and eNVM. It will always program all enabled blocks.

To program eNVM or Fabric only, for example, the user must generate DAT files for eNVM or Fabric only. See the Libero online help for more information.

Data Type Definitions

Microsemi uses *DPUCHAR*, *DPUINT*, *DPULONG*, *DPBOOL*, *DPCHAR*, *DPINT*, and *DPLONG* in the DirectC source code. Change the corresponding variable definition if different data type names are used.

```
/************************************************/
/* DPCHAR    -- 8-bit Windows (ANSI) character */
/*              i.e. 8-bit signed integer      */
/* DPINT     -- 16-bit signed integer          */
/* DPLONG    -- 32-bit signed integer          */
/* DPBOOL    -- boolean variable (0 or 1)      */
/* DPUCHAR   -- 8-bit unsigned integer         */
/* DPUSHORT  -- 16-bit unsigned integer        */
/* DPUINT    -- 16-bit unsigned integer        */
/* DPULONG   -- 32-bit unsigned integer        */
\/************************************************/
typedef unsigned char  DPUCHAR;
typedef unsigned short DPUSHORT;
typedef unsigned int   DPUINT;
typedef unsigned long  DPULONG;
typedef unsigned char  DPBOOL;
typedef          char  DPCHAR;
typedef          int   DPINT;
typedef          long  DPLONG;
```

Supported Actions

Table 3-3 lists supported actions and devices. l

Table 3-3 **• Supported Actions**

Action	Supported Devices	Description
DP_DEVICE_INFO_ACTION	All	Displays device security settings and the content of the FROM if not encrypted.
DP_READ_IDCODE_ACTION	All	Reads and displays the content of the IDCODE register.
DP_ERASE_ACTION	All	Erases all supported blocks in the data file.
DP_PROGRAM_ACTION	All	Performs erase, program and verify operations for all the supported blocks in the data file including SmartFusion MSS private clients.
DP_VERIFY_ACTION	All	Performs verify operation for all the supported blocks in the data file including SmartFusion MSS private clients.
DP_ENC_DATA_AUTHENTICATION_ACTION	ProASIC3/E/L, IGLOO/+/E, Fusion, SmartFusion, SmartFusion2, IGLOO2	Valid for encrypted array devices and files only. It performs data authentication for the array to make sure the data was encrypted with the same encryption key as the device.
DP_ERASE_ARRAY_ACTION	ProASIC3/E/L, IGLOO/+/E, Fusion, SmartFusion	Performs erase operation on the array blocks.

Table 3-3 • **Supported Actions (continued)**

DP_PROGRAM_ARRAY_ACTION	ProASIC3/E/L, IGLOO/+/E, Fusion, SmartFusion	Performs erase, program and verify operations on the array block and SmartFusion MSS private clients.
DP_VERIFY_ARRAY_ACTION	ProASIC3/E/L, IGLOO/+/E, Fusion, SmartFusion	Performs verify operation on the array block and SmartFusion MSS private clients.
DP_ERASE_FROM_ACTION	ProASIC3/E/L, IGLOO/+/E, Fusion, SmartFusion	Performs erase operation on the FROM block.
DP_PROGRAM_FROM_ACTION	ProASIC3/E/L, IGLOO/+/E, Fusion, SmartFusion	Performs erase, program and verify operations on the FROM block.
DP_VERIFY_FROM_ACTION	ProASIC3/E/L, IGLOO/+/E, Fusion, SmartFusion	Performs verify operation on the FROM block.
DP_ERASE_SECURITY_ACTION	ProASIC3/E/L, IGLOO/+/E, Fusion, SmartFusion	Performs erase operation on the security registers.
DP_PROGRAM_SECURITY_ACTION	ProASIC3/E/L, IGLOO/+/E, Fusion, SmartFusion	Performs erase and program operations on the security registers.
DP_PROGRAM_NVM_ACTION	Fusion, SmartFusion	Performs program and verify operations on all supported NVM blocks in the data file including SmartFusion MSS private clients.
DP_VERIFY_NVM_ACTION	Fusion, SmartFusion	Performs verify operation on all supported NVM blocks in the data file including SmartFusion MSS private clients.
DP_VERIFY_DEVICE_INFO_ACTION	ProASIC3/E/L, IGLOO/+/E, Fusion, SmartFusion	Performs verification of the security settings of the device against the data file security setting.
DP_READ_USERCODE_ACTION	ProASIC3/E/L, IGLOO/+/E, Fusion, SmartFusion	Reads and displays the device usercode while the FPGA Array remains active.
DP_PROGRAM_NVM_ACTIVE_ARRAY	Fusion, SmartFusion	Programs the targeted EFMBs while the FPGA Array remains active including SmartFusion MSS private clients.
DP_VERIFY_NVM_ACTIVE_ARRAY	Fusion, SmartFusion	Verifies the targeted EFMBs while the FPGA Array remains active including SmartFusion MSS private clients.
DP_IS_CORE_CONFIGURED_ACTION_CODE	ProASIC3/E/L, IGLOO/+/E, Fusion, SmartFusion	Performs a quick check on the array to determine if the core is programmed and enabled.
DP_PROGRAM_PRIVATE_CLIENTS_ACTION_CODE	SmartFusion	SmartFusion specific action. This action programs the system boot code as well as initialization clients in SmartFusion used by the MSS.

Table 3-3 • **Supported Actions (continued)**

DP_VERIFY_PRIVATE_CLIENTS_ACTION_CODE	SmartFusion	SmartFusion specific action. This action verifies the system boot code as well as initialization clients in SmartFusion used by the MSS.
DP_PROGRAM_PRIVATE_CLIENTS_ACTIVE_ARRAY_ACTION_CODE	SmartFusion	SmartFusion specific action. This action updates the system boot code as well as initialization clients in smart fusion used by the MSS while the FPGA array remains active.
DP_VERIFY_PRIVATE_CLIENTS_ACTIVE_ARRAY_ACTION_CODE	SmartFusion	SmartFusion specific action. This action updates the system boot code as well as initialization clients in smart fusion used by the MSS while the FPGA array remains active.
DP_VERIFY_DIGEST_ACTION_CODE	SmartFusion2, IGLOO2, RTG4, PolarFire	SmartFusion2 / IGLOO2 / RTG4 / PolarFire specific action. This action checks the digest of a programmed M2S/M2GL/RTG4 device.
DP_CHECK_BITSTREAM_ACTION_CODE	RTG4	Checks the integrity of the bitstream

4 – Chain Programming

Chain programming refers to a chain of devices (from various vendors) connected together serially through a JTAG port. When devices are joined together in a JTAG chain, all of their Instruction Registers (IR) and Data Registers (DR) are put in a long shift register from TDI to TDO. The IR length differs from device to device and the DR length depends on the instruction that shifts into the instruction register.

Pre/Post Data Variable Declaration

The pre/post data variable declaration variables are initialized and used in the *dpchain.c* file. Their default values are 0s. You do not need to change these values if you are programming a standalone device. However, you must correctly set these variables if you are programming Microsemi devices in a daisy chain.

The variables that must be set are defined in dpchain.c and are listed below:

```
DPUINT dp_preir_length = PREIR_LENGTH_VALUE;
DPUINT dp_predr_length = PREDR_LENGTH_VALUE;
DPUINT dp_postir_length = POSTIR_LENGTH_VALUE;
DPUINT dp_postdr_length = POSTDR_LENGTH_VALUE;
```

These variables are used to hold the pre and post IR and DR data:

```
DPUCHAR dp_preir_data[PREIR_DATA_SIZE];
DPUCHAR dp_predr_data[PREDR_DATA_SIZE];
DPUCHAR dp_postir_data[POSTIR_DATA_SIZE];
DPUCHAR dp_postdr_data[POSTDR_DATA_SIZE];

PREIR_DATA_SIZE = (dp_preir_length + 7) / 8;
PREDR_DATA_SIZE = (dp_predr_length + 7) / 8;
POSTIR_DATA_SIZE = (dp_postir_length + 7) / 8;
POSTDR_DATA_SIZE = (dp_postdr_length + 7) / 8;
```

In the example below, the devices sitting in a chain between the need-programming A3P device and the TDO of programming header are called pre-devices. The devices between the need-programming A3P device and the TDI of the programming header are called post-devices. In Figure 4-1, devices one and two are pre-devices, devices four, five, and six are post-devices, and A3P3 is the device that is programmed.

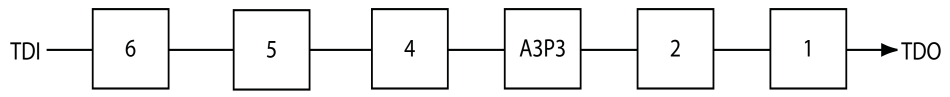


Figure 4-1 • **Devices in the Chain**

If there are N1 pre-devices and N2 post-devices in a chain, L1 is the sum of IR lengths of all the pre-devices. L2 is the sum of IR lengths of all the post devices. Table 4-2 is an example of how to set the values for the dpchain.c file using the variables assuming the values shown in Table 4-1.

Table 4-1 • **Device IR Length**

Device	IR Length
Dev 1	5
Dev 2	8

Table 4-1 • **Device IR Length (continued)**

Device	IR Length
Dev 3	8
Dev 4	3
Dev 5	12
Dev 6	5

L1 = 5 + 8 = 13

L2 = 3 + 12 + 5 = 20

Table 4-2 • **Example Variable Values for dpchain.c File**

Pre/Post Data Values	Comments
#define PREIR_LENGTH_VALUE 13	L1
#define PREDR_LENGTH_VALUE 2	N1
#define POSTIR_LENGTH_VALUE 20	L2
#define POSTDR_LENGTH_VALUE 3	N2
#define PREIR_DATA_SIZE 2	Number of bytes needed to hold L1
#define PREDR_DATA_SIZE 1	Number of bytes needed to hold N1
#define POSTIR_DATA_SIZE 3	Number of bytes needed to hold L2
#define POSTDR_DATA_SIZE 1	Number of bytes needed to hold N2

Initialize the following arrays as follows for this particular example:

```
DPUCHAR dp_preir_data[PREIR_DATA_SIZE]={0xff,0x1f};
DPUCHAR dp_predr_data[PREDR_DATA_SIZE]={0x3};
DPUCHAR dp_postir_data[POSTIR_DATA_SIZE]={0xff,0xff,0xf};
DPUCHAR dp_postdr_data[POSTDR_DATA_SIZE]={0x1f};
```

Note: Chain programming does not support programming multiple devices simultaneously. Instead, it is a method to communicate with one device to perform programming. All other devices must be placed in bypass mode, as implemented in the above example.

Example

The following example shows the definitions of all relevant constants and variables to target a specific device in the chain.

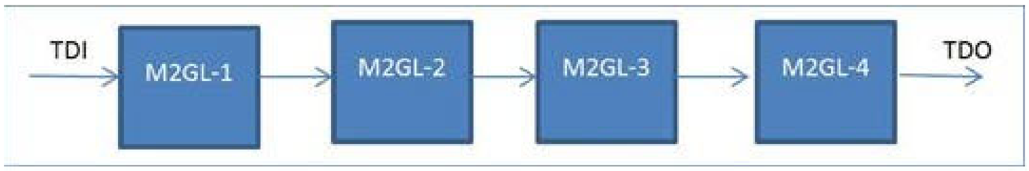


Figure 4-2 • **Constants and Variables Targeting a Specific Device in the Chain**

To program igloo2-1

```
#define PREIR_LENGTH_VALUE 24
#define PREDR_LENGTH_VALUE 3
#define POSTIR_LENGTH_VALUE 0
#define POSTDR_LENGTH_VALUE 0
#define PREIR_DATA_SIZE 3
#define PREDR_DATA_SIZE 1
#define POSTIR_DATA_SIZE 1
#define POSTDR_DATA_SIZE 1

DPUCHAR dp_preir_data[PREIR_DATA_SIZE]={0xff, 0xff , 0xff };
DPUCHAR dp_predr_data[PREDR_DATA_SIZE]={0x7};
DPUCHAR dp_postir_data[POSTIR_DATA_SIZE]={0x0};
DPUCHAR dp_postdr_data[POSTDR_DATA_SIZE]={0x0};
```

To program Igloo2-2

```
#define PREIR_LENGTH_VALUE 16
#define PREDR_LENGTH_VALUE 2
#define POSTIR_LENGTH_VALUE 8
#define POSTDR_LENGTH_VALUE 1
#define PREIR_DATA_SIZE 2
#define PREDR_DATA_SIZE 1
#define POSTIR_DATA_SIZE 1
#define POSTDR_DATA_SIZE 1

DPUCHAR dp_preir_data[PREIR_DATA_SIZE]={0xff, 0xff};
DPUCHAR dp_predr_data[PREDR_DATA_SIZE]={0x3};
DPUCHAR dp_postir_data[POSTIR_DATA_SIZE]={0xff};
DPUCHAR dp_postdr_data[POSTDR_DATA_SIZE]={0x1};
```

To program igloo2-3

```
#define PREIR_LENGTH_VALUE 8
#define PREDR_LENGTH_VALUE 1
#define POSTIR_LENGTH_VALUE 16
#define POSTDR_LENGTH_VALUE 2
#define PREIR_DATA_SIZE 1
#define PREDR_DATA_SIZE 1
#define POSTIR_DATA_SIZE 2
#define POSTDR_DATA_SIZE 1

DPUCHAR dp_preir_data[PREIR_DATA_SIZE]={0xff}
DPUCHAR dp_predr_data[PREDR_DATA_SIZE]={0x1}
DPUCHAR dp_postir_data[POSTIR_DATA_SIZE]={0xff, 0xff}
DPUCHAR dp_postdr_data[POSTDR_DATA_SIZE]={0x3}
```

To program Igloo2-4

```
#define PREIR_LENGTH_VALUE 0
#define PREDR_LENGTH_VALUE 0
#define POSTIR_LENGTH_VALUE 24
#define POSTDR_LENGTH_VALUE 3
#define PREIR_DATA_SIZE 1
#define PREDR_DATA_SIZE 1
#define POSTIR_DATA_SIZE 3
#define POSTDR_DATA_SIZE 1

DPUCHAR dp_preir_data[PREIR_DATA_SIZE]= {0x0}
DPUCHAR dp_predr_data[PREDR_DATA_SIZE]={0x0}
DPUCHAR dp_postir_data[POSTIR_DATA_SIZE]={0xff, 0xff, 0xff}
DPUCHAR dp_postdr_data[POSTDR_DATA_SIZE]={0x7}
```

5 – Data File Format

DAT File Description for AGL, AFS, A3PL, A3PEL, A3P/E, and A2F Devices

The AGL / AFS / A3PL / A3PEL / A3P/3 A2F data file contains the following sections:

- **Header Block** - Contains information identifying the type of the binary file, data size blocks, target device ID and different flags needed in the DirectC code to identify which block is supported and its associated options.
- **Data Lookup Table** - Contains records identifying the starting relative location of all the different data blocks used in the DirectC code and data size of each block. The format is described in Table 5-1.
- **Data Block** - Contains the raw data for all the different variables specified in the lookup table.

Table 5-1 • **DAT Image Description**

Header Section of DAT File	
Information	# of Bytes
Designer version number	24
Header Size	1
Image Size	4
Data Compression Flag	1
M1/P1/M7 Flag	1
Target Device ID	4
Tools Version Number	2
Map Version Number	2
Core Support Flag	1
FORM Support Flag	1
NVM Support Flag	1
NVM Block 0 Support Flag	1
NVM Block 1 Support Flag	1
NVM Block 2 Support Flag	1
NVM Block 3 Support Flag	1
NVM Verify Support Flag	1
PASS Key Support Flag	1
AES Key Support Flag	1

Table 5-1 • **DAT Image Description (continued)**

Header Section of DAT File	
Core Encryption Flag	1
FROM Encryption Flag	1
NVM Block 0 Encryption Flag	1
NVM Block 1 Encryption Flag	1
NVM Block 2 Encryption Flag	1
NVM Block 3 Encryption Flag	1
Device Exception Flag	2
ID Mask	4
SD Tiles	1
Mapped rows	2
BSR Length	2
SE Wait	1
Dual Key Support Flag	1
Number of DirectC data blocks in file	1
Look Up Table	
Information	# of Bytes
Data Identifier # 1	1
Pointer to data 1 memory location in the data block section	4
# of bytes of data 1	4
Data Identifier # 2	1
Pointer to data 2 memory location in the data block section	4
# of bytes of data 2	4
Data Identifier # x	1
Pointer to data x memory location in the data block section	4
# of bytes of data x	4
Data Block	
Information	# of Bytes
Binary Data	Variable
CRC of the entire image	2

DAT File Description for M2GL, M2S, RTG4, and MPF Devices

The M2GL, M2S, RTG4, and MPF data file contains the following sections:

- **Header Block** - Contains information identifying the type of the binary file and data size blocks.
- **Constant Data Block** - Includes device ID, silicon signature and other information needed for programming.
- **Data Lookup Table** - Contains records identifying the starting relative location of all the different data blocks used in the DirectC code and data size of each block. The format is described in Table 5-2.
- **Data Block** - Contains the raw data for all the different variables specified in the lookup table.

***Table 5-2* • DAT Image Description**

Header Section of DAT File	
Information	# of Bytes
Designer version number	24
Header Size	1
Image Size	4
DAT File Version	1
Tools Version Number	2
Map Version Number	2
Feature Flag	2
Device Family	1
Constant Data Block	
Device ID	4
Device ID Mask	4
Silicon Signature	4
Checksum	2
Number of BSR Bits	2
Number of Components	2
Data Size	2
Erase Data Size	2
Verify Data Size	2
ENVM Data Size	2
ENVM Verify Data Size	2
UEK1_EXISTS Flag (Excluding RTG4)	1
UEK2_EXISTS Flag (Excluding RTG4)	1
SEC_ERASE Flag (Excluding RTG4)	1

Table 5-2 • **DAT Image Description (continued)**

Header Section of DAT File	
UEK3_EXISTS Flag (Excluding RTG4 and MPF)	1
Number of Records	1
Look Up Table	
Information	# of Bytes
Data Identifier # 1	1
Pointer to data 1 memory location in the data block section	4
# of bytes of data 1	4
Data Identifier # 2	1
Pointer to data 2 memory location in the data block section	4
# of bytes of data 2	4
Data Identifier # x	1
Pointer to data x memory location in the data block section	4
# of bytes of data x	4
Data Block	
Information	# of Bytes
Binary Data	Variable
CRC of the entire image	2

6 – Source File Description

DPUSER.C and DPUSER.H

These files contain hardware interface functions and require user modification.

DPCOM.C and DPCOM.H

These files contain memory interface functions and require user modification..

DPALG.C and DPALG.H

dpalg.c contains the main entry function *dp_top*.

dpalg.h contains definitions of all the STAPL actions and their corresponding codes.

DPG3ALG.C and DPG3ALG.H

dpG3alg.c contains the main entry function *dp_top_g3* and all other functions common to AGL, AFS, A3PL, A3PEL, A3P/E, and A2F families.

dpG3alg.h contains compile options specific to AGL, AFS, A3PL, A3PEL, A3P/E, and A2F families. User modification may be required.

DPCORE.C and DPCORE.H

Files that contain the specific functions to support array erase, program and verify actions of AGL, AFS, A3PL, A3PEL, A3P/E and A2F families.

DPFROM.C and DPFROM.H

Files that contain the specific functions to support FROM erase, program and verify actions of AGL, AFS, A3PL, A3PEL, A3P/E and A2F families.

DPNVM.C and DPNVM.H

Files that contain the specific functions to support NVM program and verify actions of AFS and A2F families.

DPSECURITY.C and DPSECURITY.H

Files that contain the specific functions to support security erase, program actions of AGL, AFS, A3PL, A3PEL, A3P/E, and A2F families.

DPG4ALG.C and DPG4ALG.H

dpG4alg.c contains the main entry function *dp_top_g4* and all other functions common to M2S and MGL families.

DPJTAG.C and DPJTAG.H

The JTAG related functions are declared in *dpjtag.h* and implemented in *dpjtag.c*.

DPCHAIN.C and DPCHAIN.H

Files that contain the specific functions to support chain programming.

dpchain.c contains pre- and post-IR/DR data definition to support chain programming. User modification to set up a chain may be required.

DPUTIL.C and DPUTIL.H

These files contain utility functions needed in the DirectC code.

DPRTG4ALG.C and DPRTG4ALG.H

dpRTG4alg.c contains the main entry function *dp_top_rtg4* and all other functions specific to RTG4 devices.

DPG5ALG.C and DPG5ALG.H

dpG5alg.c contains the main entry function *dp_top_g5* and all other functions specific to MPF devices.

7 – Disabled Features with ENABLE_CODE_SPACE_OPTIMIZATION

DMK Verification for ARM Enabled Devices

This feature identifies whether the target device is M1, M7, or P1 device.

Affected devices: ARM enabled devices

Impact if removed: DirectC will be unable to identify if the device is standard Fusion or ARM enabled device. DirectC still supports programming; however, it relies on the data file processing the target device as an ARM enabled device.

030/015 Device Check

This feature identifies if the target device is a 015 or 030 device; needed to prevent the wrong design from being programmed into the device.

Affected devices: A3P and AGL 015 / 030 device

Impact if removed: If the design does not match the target device, programming may pass, but the device may not function

8 – Data File Bit Orientation

This section specifies the data orientation of the binary data file generated by Libero software. DirectC implementation must be in sync with the specified data orientation. Table 8-1 illustrates how the data is stored in the binary data file. See "Data File Format" on page 24 for additional information about the data file.

Table 8-1 • **Binary Data File Example**

Byte 0	Byte 1	Byte 2	Byte 3	..	..	Byte N
Bit7..Bit0	Bit15..Bit8	Bit23..Bit16	Bit35..Bit24	..	..	Bit(8N+7)..Bit(8N)
Valid Data	Valid Data	Valid Data	Valid Data	..	..	o <-Valid Data

If the number of bits in a data block is not a multiple of eight, the rest of the most significant bits (msb) in the last byte are filled with zeros. The example below shows a given 70-bit data to be shifted into the target shift register from the least significant bit (lsb) to the most significant bit (msb). A binary representation of the same data follows.

20E60A9AB06FAC78A6	tdi
1000001110011 0 000010101001101010110000011011111010110001111000 10100110 tdi	
Bit 69	Bit 0

This data is stored in the data block section. Table 8-2 shows how the data is stored in the data block.

Table 8-2 • **Data Block Section Example**

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	..	Byte 8
Bit7...Bit0	Bit15..Bit8	Bit23..Bit16	Bit35..Bit24	Bit43..Bit36	..	Bit71..Bit64
10100110	01111000	10101100	01101111	10110000		00100000
A6	78	AC	6F	B0		20

9 – Sample Project

The sample project, IAR_JTAG_DirectC.zip, available with this release of DirectC, is based on IAR Embedded Workbench version 6.40. It is designed to work on SmartFusion Security Evaluation Kit with the SmartFusion2 M2S090-FGG484 device.

Project Requirements

You will need the following hardware and software to run the sample project:

Hardware:

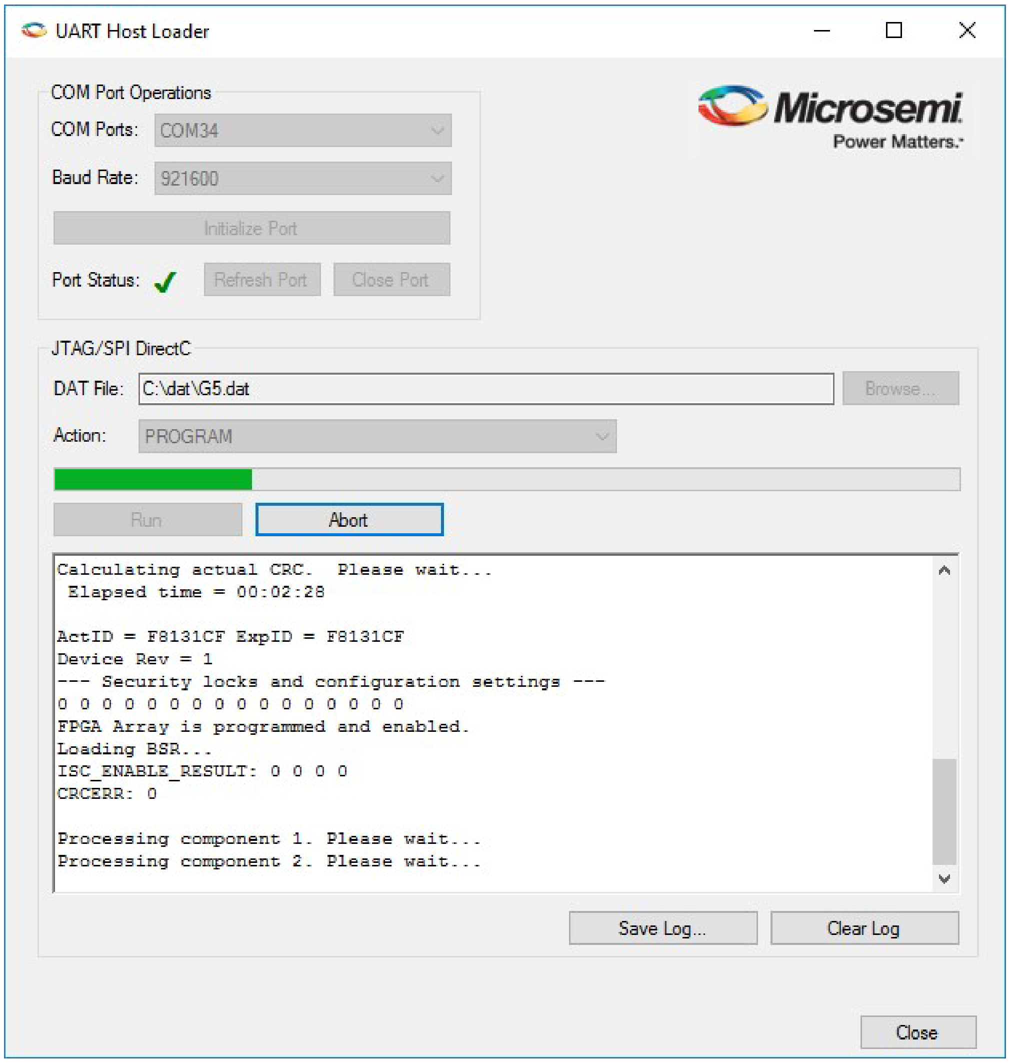
- SmartFusion2 Security Evaluation Kit with SmartFusion2 M2S090-FGG484 device.
- jLink from IAR.
- Target board with Microsemi device to be programmed.

Software:

- IAR Embedded Workbench version 6.4.
- UART Host Loader available with this release package.

Procedure

1. Program the evaluation kit with JTAG_DC_top.stp under the *M2S Eval Kit Files* directory. The M2S090 design connects specific MSS IO pins and SPI1 port to specific J1 header pins for JTAG and SPI access.
2. Connect the JTAG pins as described in HeaderPinAssignment.xlsx available under the *M2S Eval Kit Files* directory.
3. Connect the Mini USB (J18) to your PC. The mini USB is connected to the FTDI FT4232h device used as a USB to UART bridge.
4. Make sure the appropriate drivers are installed on your PC to communicate with this chip.
5. Run UART Host Loader available with this release package.
6. There should be four com ports available in the serial port setup window. Select the 4th one from the list and configure the Baud Rate as shown below. If more than 4 ports are available, disconnect the J18 header and refresh the com ports in the UARTHostLoader application to identify exiting ports. Reconnect the J18 header and refresh the USB ports. Select the 4th port from the newly generated port list.
7. Click **Initialize Port** to establish connection with the selected COM port.
8. Select the programming file and desired action.
9. Click **Run**. The UART Host Loader application waits for data from the SmartFusion2 evaluation kit.
10. The STAPL file programmed into the evaluation kit has a DirectC sample project that supports SmartFusion2, IGLOO2, RTG4, and PolarFire devices. Resetting the board runs the embedded application and performs the action selected. To run another action or select a different programming file, select it from the UART Host Loader and click **Run** again.



COM Ports: COM34

Baud Rate: 921600

Port Status: ✓

DAT File: C:\dat\G5.dat

Action: PROGRAM

```
Calculating actual CRC.  Please wait...
 Elapsed time = 00:02:28

ActID = F8131CF ExpID = F8131CF
Device Rev = 1
--- Security locks and configuration settings ---
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
FPGA Array is programmed and enabled.
Loading BSR...
ISC_ENABLE_RESULT: 0 0 0 0
CRCERR: 0

Processing component 1. Please wait...
Processing component 2. Please wait...
```

11. To make changes to the embedded project, run IAR workbench and modify the compile options as desired. You can download the embedded application using jLink as follows:
    a. Connect jLink to RVI/IAR header.
    b. Set the JTAG select jumper low.
    c. Click on download and run from IAR.

10 – Error Messages & Troubleshooting Tips

The information in this chapter may help you solve or identify a problem when using DirectC code. If you have a problem that you cannot solve, visit the Microsemi website at http://www.microsemi.com/products/fpga-soc/design-support/fpga-soc-support or contact Microsemi Customer Technical Support at tech@microsemi.com or call our hotline 1-800-262-1060.

See Table 10-1 for a description of exit codes and their solutions.

***Table 10-1* • Exit Codes**

Exit Code	Error Message	Action/Solution
0	This code does not indicate an error	This message indicates success
6	JEDEC standard message. The IDCODE of the target device does not match the expected value in the DAT file image.	Possible Causes: - The data file loaded was compiled for a different device. Example AFS250 DAT file loaded to program AFS600 device. - Device TRST pin is grounded - Noise or reflections on one or more of the JTAG pins causing incorrect read-back of the IR Bits. Solutions: - Choose the correct DAT file for the target device. - Measure JTAG pins and noise or reflection. TRST should be floating or tied high. - Cut down the extra length of ground connection.
8	This message occurs when the FPGA failed during the Erase operation.	Possible Causes: - The device is secured, and the corresponding data file is not loaded. The device has been permanently secured and cannot be unlocked. Solution: - Load the correct DAT file.
10	Failed to program FlashROM	- Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection.
11	The message occurs when the FPGA failed verify.	Possible Cause: - The device is secured, and the corresponding DAT file is not loaded. - The device is programmed with an incorrect design. Solution: - Load the correct DAT file. - Check Vpump level. - Measure JTAG pins and noise or reflection.
14	Failed to program Silicon Signature	- Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection.

Table 10-1 • **Exit Codes (continued)**

Exit Code	Error Message	Action/Solution
18	Failed to authenticate the encrypted data.	- Make sure the AES key used to encrypt the data matches the AES key programmed in the device.
20	Failed to verify FlashROM at row ###.	- Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection. -Make sure the device is programmed with the correct design.
22	Failed to program pass key	- Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection.
23	Failed to program AES key	- Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection.
24	Failed to program UROW	- Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection. - Make sure you mounted 0.01uF and 0.33ufF caps on Vpump (close to the pin).
25	Failed to enter programming mode	- Try programming with a new device. - Measure JTAG pins and noise or reflection.
27	FlashROM Write/Erase is protected by the pass key. A valid pass key needs to be provided.	- Provide a data file with a pass key.
30	FPGA Array verification is protected by a pass key. A valid pass key needs to be provided.	- Provide a data file with a valid pass key.
31	Failed to program DMK	- Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection.
33	FPGA Array encryption is enforced. Plain text programming is prohibited.	Provide a data file with an encrypted FPGA Array.
34	FlashROM encryption is enforced. Plain text programming is prohibited.	- Provide a data file with an encrypted FlashROM.
35	Pass key match failure.	- Provide a data file with correct pass key.
36	FlashROM Encryption is not enforced. AES key may not be present in the target device. Unable to proceed with Encrypted FlashROM programming.	- Make sure the device is properly secured with AES encryption protection turned on. - Provide correct DAT file for programming.
37	FPGA Array Encryption is not enforced. Cannot guarantee valid AES key present in target device. Unable to proceed with Encrypted FPGA Array programming.	- Make sure the device is properly secured with the AES encryption protection turned on for FPGA Array. - Provide the correct data file for programming.

Table 10-1 • **Exit Codes (continued)**

Exit Code	Error Message	Action/Solution
38	Failed to program pass key.	- Check that the device is not already secured with a different pass key. - Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection.
39	Failed the Embedded Flash Block verification.	- Check that the device is not read secured already with a different pass key. - Measure JTAG pins and noise or reflection.
41	Failed to program Embedded Flash Block.	- Check Vpump level. - Try with new device. - Measure JTAG pins and noise or reflection.
42	User lock bits do not match the lock bits in the data file.	Provide a data file with the correct lock bits data.
43	User urow information does not match the urow information in the data file.	Provide a data file with the correct urow information data.
47	NVM encryption is enforced. Plain text programming is prohibited.	Provide a data file with an encrypted NVM.
49	NVM encryption is not enforced. Cannot guarantee valid AES key present in target device. Unable to proceed with encrypted NVM programming.	- Make sure the device is properly secured with the AES encryption protection turned on for NVM. - Provide the correct data file for programming.
100	CRC data error. Data file is corrupted or programming on system board is not successful.	- Regenerate data file. - Reprogram data file into system memory.
150	Requested action is not found.	Check spelling.
151	Action is not supported because required data block is missing from the data file.	Regenerate STAPL/DAT file with the needed block/ feature support.
152	Compiled code does not support the requested action.	Compile DirectC code with the appropriate compile options enabled.
153	Data file contain data for the protected portion of NVM0 block	Regenerate the data file from the latest Designer software
154	Device security settings do not match with the data file	Regenerate the data file with the correct device security settings

A – Product Support

Microsemi SoC Products Group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, electronic mail, and worldwide sales offices. This appendix contains information about contacting Microsemi SoC Products Group and using these support services.

Customer Service

Contact Customer Service for non-technical product support, such as product pricing, product upgrades, update information, order status, and authorization.

From North America, call **800.262.1060**
From the rest of the world, call **650.318.4460**
Fax, from anywhere in the world, **650.318.8044**

Customer Technical Support Center

Microsemi SoC Products Group staffs its Customer Technical Support Center with highly skilled engineers who can help answer your hardware, software, and design questions about Microsemi SoC Products. The Customer Technical Support Center spends a great deal of time creating application notes, answers to common design cycle questions, documentation of known issues, and various FAQs. So, before you contact us, please visit our online resources. It is very likely we have already answered your questions.

Technical Support

For Microsemi SoC Products Support, visit http://www.microsemi.com/products/fpga-soc/design-support/fpga-soc-support.

Website

You can browse a variety of technical and non-technical information on the Microsemi SoC Products Group home page, at www.microsemi.com/soc.

Contacting the Customer Technical Support Center

Highly skilled engineers staff the Technical Support Center. The Technical Support Center can be contacted by email or through the Microsemi SoC Products Group website.

Email

You can communicate your technical questions to our email address and receive answers back by email, fax, or phone. Also, if you have design problems, you can email your design files to receive assistance. We constantly monitor the email account throughout the day. When sending your request to us, please be sure to include your full name, company name, and your contact information for efficient processing of your request.

The technical support email address is soc_tech@microsemi.com.

My Cases

Microsemi SoC Products Group customers may submit and track technical cases online by going to My Cases.

Outside the U.S.

Customers needing assistance outside the US time zones can either contact technical support via email (soc_tech@microsemi.com) or contact a local sales office.

Visit About Us for sales office listings and corporate contacts.

Sales office listings can be found at www.microsemi.com/soc/company/contact/default.aspx.

ITAR Technical Support

For technical support on RH and RT FPGAs that are regulated by International Traffic in Arms Regulations (ITAR), contact us via soc_tech_itar@microsemi.com. Alternatively, within My Cases, select **Yes** in the ITAR drop-down list. For a complete list of ITAR-regulated Microsemi FPGAs, visit the ITAR web page.

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense & security, aerospace 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, Calif. and has approximately 4,800 employees globally. Learn more at **www.microsemi.com**.

Microsemi
Power Matters.™

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA

Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Sales: +1 (949) 380-6136
Fax: +1 (949) 215-4996

E-mail: sales.support@microsemi.com

5-13-00109-3/10.17