JTAG-DirectC User Guide

Introduction

JTAG-DirectC is designed to support embedded In-System Programming for Microchip devices and 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 for compilation.

JTAG-DirectC 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 JTAG-DirectC on systems with no direct access to the entire memory space containing the data. You can enable paging support by modifying the data communication functions defined in `dpuser.h`, `dpuser.c`, `dpcom.c`, and `dpcom.h`.

Supported Device Family

This document describes how to enable microprocessor-based embedded In-System Programming (ISP) on the supported Microchip devices. The following table lists the Microchip devices JTAG-DirectC supports.

Table 1. Device Family Supported by JTAG-DirectC

Device Family	Description
PolarFire®	PolarFire FPGAs deliver the industry's lowest power at mid-range densities with exceptional security and reliability.
PolarFire SoC	PolarFire SoC is the first SoC FPGA with a deterministic, coherent RISC-V CPU cluster, and a deterministic L2 memory subsystem enabling Linux and real-time applications.
SmartFusion®2	SmartFusion2 addresses fundamental requirements for advanced security, high reliability, and low power in critical industrial, military, aviation, communications, and medical applications.
IGLOO®2	IGLOO2 is a low-power mixed-signal programmable solution.
RTG4™	RTG4 is Microchip's family of radiation-tolerant FPGAs. **Note:** This version of JTAG-DirectC supports RTG4 family of devices with Avionics mode. When enabled, the Avionics mode prevents you from programming. To disable this mode, the `JTAG_TRST` pin must be held HIGH and `DEVRST_N` pin must be toggled. Alternatively, you can use the `dp_exit_avionics_mode` function, defined in the `dpuser.c` file, to disable the Avionics mode. The function must be modified to set the `JTAG_TRST` pin HIGH and toggle the `DEVRST_N` pin.
ProASIC®3 (including ProASIC3 nano)	The ProASIC3 FPGAs support portable, consumer, industrial, communications and medical applications with commercial and industrial temperature devices. They also offer specialized screening for automotive and military systems.
IGLOO® (including IGLOO nano)	IGLOO FPGA family devices are designed to meet the demand of low power and small foot print requirements of today's portable and power-conscious electronics.
SmartFusion®	SmartFusion System on Chip (SoC) FPGAs offers the benefits of full customization and IP protection, while still being easy to use.
Fusion®	Fusion mixed-signal FPGAs integrate configurable analog, large Flash memory blocks, comprehensive clock generation and management circuitry, and high-performance, Flash-based programmable logic in a monolithic device.

Table of Contents

1. System Overview

The system must contain the following parameters to perform the In-System Programming (ISP) for the FPGA.

- 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 JTAG-DirectC code.

Note: For information on power requirements for V_{pump} and other supplies, see your product device datasheet.

Memory requirements depend on the options that are enabled. The following table is an example of the code size and run time memory required to support the different device families. For more information on available compiler switches, see 3. Required Source Code Modifications.

Table 1-1. Code Memory Requirements- JTAG DirectC Code Size on CM3 in Thumb Mode

Compile Options Enabled	Units are in Bytes		
	ROM Code[1]	ROM Data[3]	Read/Write Data[2]
ENABLE_G3_SUPPORT	26760	1944	4697
ENABLE_G4_SUPPORT	18882	492	9529
ENABLE_G5_SUPPORT	21174	494	9802
ENABLE_RTG4_SUPPORT	14334	494	9435
All the above	59644	1948	11414

Notes:

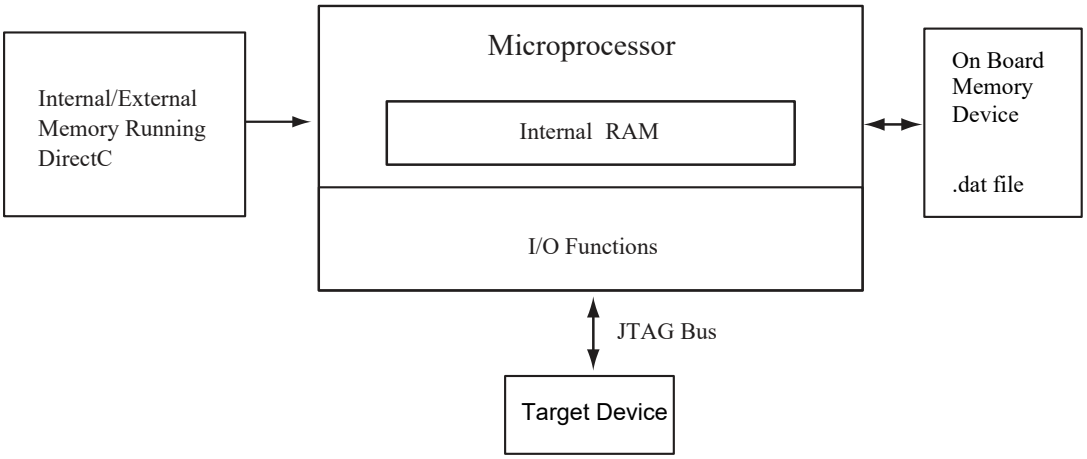
1. ROM Code - This is the compiled code size memory requirements.
2. ROM Data - This is the Block Started by Symbol allocation for variables that do not yet have values, that is uninitialized data. It is part of the overall data size.
3. Read/Write Data - This is the run time memory requirement, that is the free data memory space required to execute the code.

Note: All compile options for 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 SmartFusion2, IGLOO2, RTG4, PolarFire, and PolarFire SoC devices. For details about all compile options, see 3. Required Source Code Modifications.

1.1 Systems with Direct Access to Memory

The following figure shows the overview of a typical system with direct access to the memory space holding the data file. For generating DAT files, see 2. Generating Data Files and Integrating with JTAG-DirectC and the following table for data storage memory requirements.

Figure 1-1. Systems with Direct Access to Memory



The following figure 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

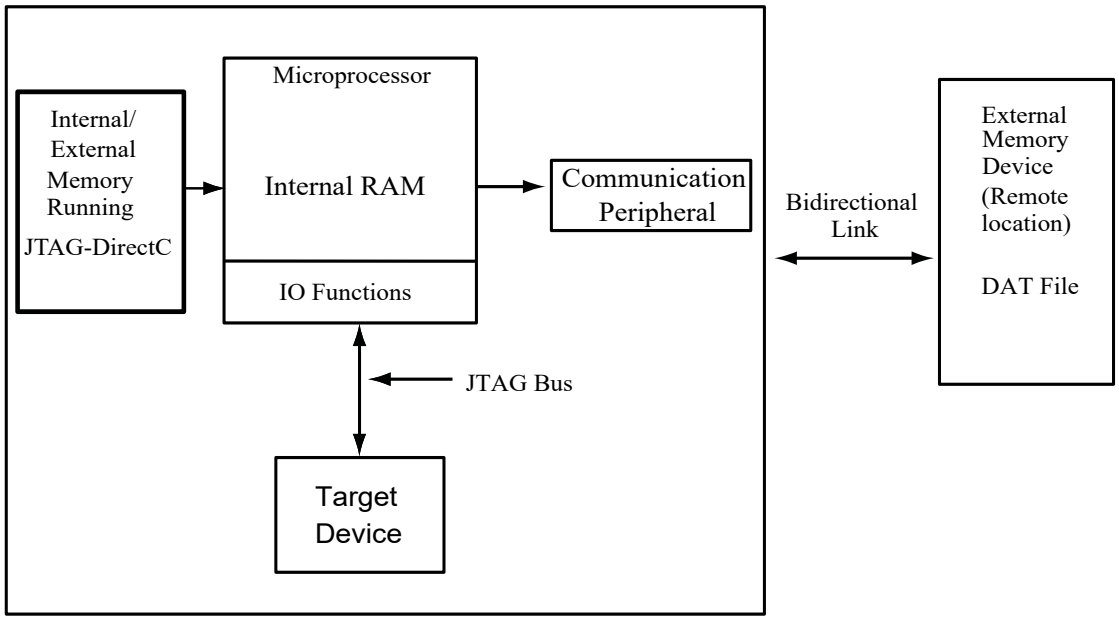


Table 1-2. Data Storage Memory Requirements Data Image Size

Data Image Size							
Device	Core/FPGA Array		FROM		Embedded Flash Memory Block		Security (kB)
	Plain(kB)	Encrypt (kB)	Plain (kB)	Encrypt (kB)	Plain (kB)	Encrypt (kB)	
A3PE600	526	647	1	1	N/A	N/A	1
A3PE1500*	1434	1765	1	1	N/A	N/A	1
A3PE3000 / L	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
MPFS250T	N/A	9542	N/A	N/A	N/A	N/A	N/A
MPF100	N/A	3447	N/A	N/A	N/A	N/A	N/A
MPF200	N/A	5992	N/A	N/A	N/A	N/A	N/A
MPF300	N/A	9256	N/A	N/A	N/A	N/A	N/A
MPF500	N/A	14739	N/A	N/A	N/A	N/A	N/A

.....continued

Data Image Size							
Device	Core/FPGA Array		FROM		Embedded Flash Memory Block		Security (kB)
	Plain(kB)	Encrypt (kB)	Plain (kB)	Encrypt (kB)	Plain (kB)	Encrypt (kB)	

- *A3PE1500 is not supported with an 8-bit processor.
- INA - Information not available currently.
- N/A - Not applicable
- N/S - Not supported
- Data in the table for base FPGA devices applies equally to the M1, M7, P1, and U1 encrypted versions of the devices. For example, 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. See 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 with JTAG-DirectC

This chapter describes how to generate a data file and integrating it with JTAG-DirectC. To generate the DAT file:

1. Launch the Libero SoC Design Suite and open the project.
2. Expand the **Handoff Design for Production** tree on the **Design Flow** tab.
3. Double click **Export Bitstream**. The **Export Bitstream** dialog box opens. The dialog box options depend on the device family, **Custom Security settings**, and **Permanent Locks** for the production settings. For more information on working with the **Export Bitstream**, see the Libero SoC Design Flow User Guide.
4. Program the DAT file into the storage memory.

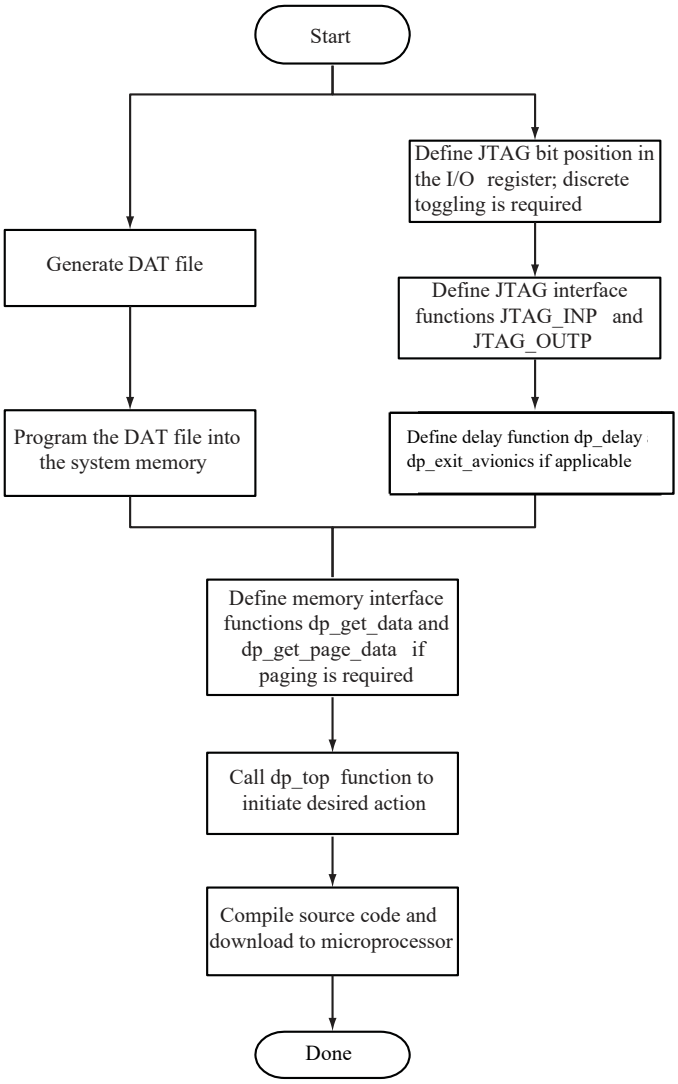
2.1 Data File Compatibility

DirectC data files can be generated from Designer v8.5 and later. Data files generated from Designer v8.5 are identical to the files generated by the original DatGen tool except for the file title. However, data files generated by Designer v8.6 are enhanced to support nano devices. JTAG-DirectC can detect the version of the file used and handles it accordingly.

2.2 JTAG-DirectC Code Integration

The following figure shows the JTAG-DirectC integration use flow.

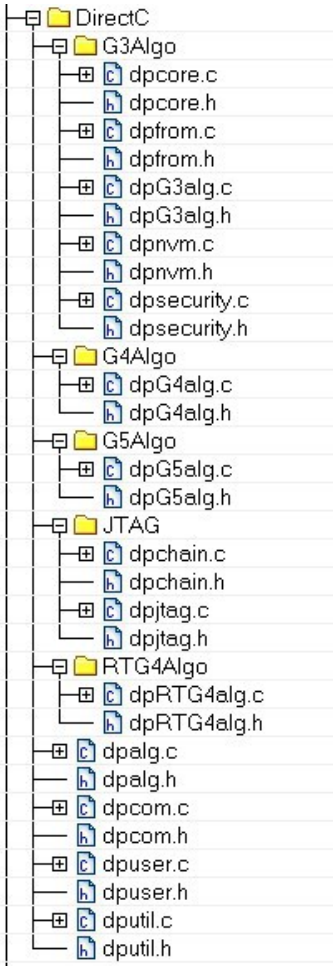
Figure 2-1. Integrating DirectC Files



To use JTAG-DirectC code integration, follow the steps below:

1. Import the JTAG-DirectC files into your development environment, as shown in the following figure.

Figure 2-2. JTAG-DirectC Files to Import into your Development Environment



2. Modify the JTAG-DirectC code.
   a. Define JTAG pin bit locations in the I/O register.
      Note: For RTG4, assign an additional pin bit to control the **`devrst`** pin.
   b. Add API to support discrete toggling of the individual JTAG pins.
   c. Modify the hardware interface functions (`jtag_inp` and `jtag_outp`) to use the hardware API functions designed to control the JTAG port.
   d. Modify the delay function (`dp_delay`).
   e. Modify memory access functions to access the data blocks within the image file programmed into the system memory. See 8. Data File Bit Orientation.
   f. Call `dp_top` function with the action code desired.
3. Compile the source code. This creates a binary executable that is downloaded 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 JTAG-DirectC source code. See 6. Source File Description for a short description of the JTAG-DirectC source code and their function. Functions that must be modified are listed in the following table.

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.

3.1 Compiler Switches

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

.....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. This feature is not supported in 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 and MPFS 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.
BSR_SAMPLE	dpuser.h	This option is only applicable for AGL, AFS, A3PL, A3PEL, A3P/E, and A2F devices. Enable this option to maintain the last known I/O states during programming. BSR loading and BSR_SAMPLE are not supported for IAP. Maintaining the last know IO state for the rest of the device families is data driven. It needs to be enabled while generating the dat file.
ENABLE_CODE_SPACE_OPTIMIZATION	dpG3alg.h	See 7. Disabled Features with ENABLE_CODE_SPACE_OPTIMIZATION
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.

.....continued		
Compiler Switch	**Source File**	**Purpose**
DISABLE_FROM_SPECIFIC_ACTIONS	dpG3alg.h	For code size reduction. This option disables FROM specific actions such as erase, program, and verify FROM actions.
DISABLE_NVM_SPECIFIC_ACTIONS	dpG3alg.h	For code size reduction. This option disables NVM specific actions such as program and verify NVM actions.
DISABLE_SEC_SPECIFIC_ACTIONS	dpG3alg.h	For code size reduction. This option disables 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, JTAG-DirectC may report an error depending on the requested action. 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.

3.2 Hardware Interface Components

This section contains information about the hardware interface components.

3.2.1 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 ... */
```

3.2.2 Hardware Interface Function (dpuser.c)

`jtag_inp` and `jtag_outp` functions are used to interface with the JTAG port. `jtag_port_reg` is an 8-bit register defined in JTAG-DirectC. JTAG-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 the logic level is zero, then this function must return to 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.

3.2.3 Delay Function (dpuser.c)

The `dp_delay` function takes one argument, which is the amount of time in microseconds. The purpose of it is to pause for a minimum of time passed in the argument. Longer delay time does not impact programming operation other than programming time.

3.2.4 Display Functions (dpuser.c)

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

3.2.5 Memory Interface Functions (dpuser.c)

All access to the memory blocks within the data file is done through the dp_get_data function within the JTAG-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 with an identifier specifying the block within the data file that 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.

For more details, see 3.2.6 Systems with Direct Access to the Memory with Data File and 3.2.7 Systems with Indirect Access to the Data File.

3.2.6 Systems with Direct Access to the Memory with Data File

Since the memory space holding the data file that 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. For more details, see 3.1 Compiler Switches.
2. Assign the physical address pointer to the first element of the data memory location (image_buffer is defined in dpcom.c. The image_buffer file 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 toimage_buffer. return_bytes is the requested data.

The following is an example of dp_get_data function implementation. 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;
}
```

3.2.7 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 JTAG-DirectC can access.

To implement paging, follow the steps below:

1. Enable USE_PAGING compiler option. For more details, see 3.1 Compiler Switches.
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 two times the 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. For more information, see 8. Data File Bit Orientation. Follow the below rules for correct operation:
   – Fill the entire page unless the end of the image is reached. See 5. Data File Format for more details.
   – Update `return_bytes` to reflect the number of valid bytes in the page.

JTAG-DirectC programming functions call the `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.

3.2.8 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;
}
```

3.2.9 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 in the following codeblock.

```
/* Action Names -- match actions function */
/* These codes are passed to the main entry function "dp_top" to indicate
* which action to perform */
#define DP_DEVICE_INFO_ACTION_CODE              1u
#define DP_READ_IDCODE_ACTION_CODE              2u
#define DP_ERASE_ACTION_CODE                    3u
#define DP_PROGRAM_ACTION_CODE                  5u
#define DP_VERIFY_ACTION_CODE                   6u
/* Array only actions */
#define DP_ENC_DATA_AUTHENTICATION_ACTION_CODE  7u
#define DP_ERASE_ARRAY_ACTION_CODE              8u
#define DP_PROGRAM_ARRAY_ACTION_CODE            9u
#define DP_VERIFY_ARRAY_ACTION_CODE             10u
/* FROM only actions */
#define DP_ERASE_FROM_ACTION_CODE               11u
#define DP_PROGRAM_FROM_ACTION_CODE             12u
#define DP_VERIFY_FROM_ACTION_CODE              13u
/* Security only actions */
#define DP_ERASE_SECURITY_ACTION_CODE           14u
#define DP_PROGRAM_SECURITY_ACTION_CODE         15u
/* NVM only actions */
#define DP_PROGRAM_NVM_ACTION_CODE              16u
```

```
#define DP_VERIFY_NVM_ACTION_CODE               17u
#define DP_VERIFY_DEVICE_INFO_ACTION_CODE       18u
#define DP_READ_USERCODE_ACTION_CODE            19u
/* For P1 device, The following two actions are only supported with data files
* generated form V86 or later.  ENABLE_V85_DAT_SUPPORT must be disabled */
#define DP_PROGRAM_NVM_ACTIVE_ARRAY_ACTION_CODE 20u
#define DP_VERIFY_NVM_ACTIVE_ARRAY_ACTION_CODE  21u
#define DP_IS_CORE_CONFIGURED_ACTION_CODE       22u
#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
#define DP_CHECK_BITSTREAM_ACTION_CODE            27u
#define DP_VERIFY_DIGEST_ACTION_CODE            28u
#define DP_VALIDATE_USER_ENC_KEYS_ACTION_CODE    29u
#define DP_READ_DEVICE_CERTIFICATE_ACTION_CODE    30u
#define DP_ZEROIZE_LIKE_NEW_ACTION_CODE         31u
#define DP_ZEROIZE_UNRECOVERABLE_ACTION_CODE    32u
```

Note: This list is 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 programs all enabled blocks. To program eNVM or Fabric only, you must generate DAT files for eNVM or Fabric. For more information see the Libero SoC Design Flow User Guide.

3.2.10 Data Type Definitions

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

```
/*********************************************/
/* DPCHAR -- 8-bit Windows (ANSI) character */
/* that is, 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;
```

3.2.11 Supported Actions

The following table lists supported actions and devices.

Table 3-3. Supported Actions

Action	Supported Devices	Description
DP_DEVICE_INFO_ACTION	All	Displays device design information and status including security settings.
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.

.....continued

Action	Supported Devices	Description
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	All excluding RTG4	Valid for encrypted array devices and files only. It performs data authentication for the array to make sure that 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.
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 user code 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.

.....continued

Action	Supported Devices	Description
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_COD E	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 MSS.
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 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 SmartFusion used by the MSS while the FPGA array remains active.
DP_VERIFY_PRIVATE_CLIENTS_ACTIVE_A RRAY_ACTION_CODE	SmartFusion	SmartFusion specific action. This action updates the system boot code as well as initialization clients in SmartFusion used by the MSS while the FPGA array remains active.
DP_VERIFY_DIGEST_ACTION_CODE	SmartFusion2, IGLOO2, RTG4, PolarFire, PolarFire SoC	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.
DP_READ_DEVICE_CERTIFICATE_ACTION_CODE	SmartFusion2, IGLOO2, PolarFire, PolarFire SoC	Reads and displays device certificate.
DP_ZEROIZE_LIKE_NEW_ACTION_CODE	PolarFire, PolarFire SoC	Performs zeroization. Device is recoverable.
DP_ZEROIZE_UNRECOVERABLE_ACTION_CODE	PolarFire, PolarFire SoC	Performs zeroization. Device is not recoverable.

4. Chain Programming

Chain programming refers to a chain of devices (from various vendors) connected serially through a JTAG port. When devices are joined together in a JTAG chain, all 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 Microchip devices in a daisy chain. The following is a list of variables that must be set and defined in `dpchain.c`:

```
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:

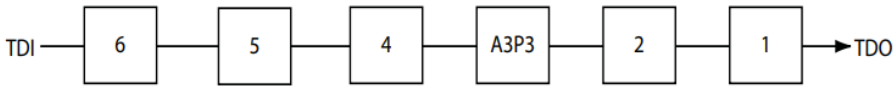
```
DPUCHAR dp_preir_data[PREIR_DATA_SIZE];
DPUCHAR dp_predr_data[PREDR_DATA_SIZE];
DPUCHAR dp_postir_data[POSTIR_DATA_SIZE];
DPUCHAR dp_postdr_data[POSTDR_DATA_SIZE];

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 following example, the devices 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. The following figure shows,

- Devices one and two that are pre-devices.
- Devices four, five, and six that are post-devices.
- A3P3 that 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. The following table is an example of how to set the values for the `dpchain.c` file using the variables.

Table 4-1. Device IR Length

Device	IR Length
Dev 1	5
Dev 2	8
Dev 3	8
Dev 4	3
Dev 5	12
Dev 6	5

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

Notes:

1. L1 = 5 + 8 = 13
2. L2 = 3 + 12 + 5 = 20

Initialize the following arrays as in this example:

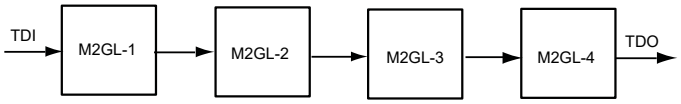
```
DPUCHAR dp_preir_data[PREIR_DATA_SIZE]={0xff,0x1f};
DPUCHAR dp_predr_data[PREDR_DATA_SIZE]={0x3};
DPUCHAR dp_postir_data[POSTIR_DATA_SIZE]={0xff,0xff,0xf};
DPUCHAR dp_postdr_data[POSTDR_DATA_SIZE]={0x1f};
```

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.

4.1 Example

The following example shows the definitions of all relevant constants and variables to target a specific device (IGLOO2 is used in the example) 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}
```

Table 4-3. IR Bit Length

Device Family	IR Bit Length
Fusion	8
SmartFusion	8
SmartFusion2	8
IGLOO2	8
RTG4	8
PolarFire / PolarFire SoC	8
ProASIC3/E/L, IGLOO/+/E	8

5. Data File Format

JTAG-DirectC is a set of C code designed to support embedded In-System Programming for Microchip devices. To use JTAG-DirectC, 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 along with the programming data file. The programming data file is a binary file that can be generated by Libero® SoC.

5.1 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

The header block contains information identifying the type of the binary file, data size blocks, target Device ID, and different flags needed in the JTAG-DirectC code to identify the block that is supported and its associated options.

Table 5-1. Header Section Description

Information	Number 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
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

.....continued	
Information	**Number Of Bytes**
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

Data Look-up Table
The Look-up table block contains records identifying the starting relative location of all the different data blocks used in the JTAG-DirectC code and data size of each block. The format is described in the following table.

Table 5-2. Look-up Table Description

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
The data block contains the raw data for all the different variables specified in the Look-up Table (LUT).

Table 5-3. Data Block Description

Information	# Of Bytes
CRC of the entire image	2
Binary Data	Variable

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

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

Header Block
The header block contains information identifying the type of the binary file and data size blocks.

Table 5-4. Header Section Description

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

The constant data block includes Device ID, silicon signature, and other information needed for programming.

Table 5-5. DAT Image Description

Information	# Of Bytes
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
Number of Records	1
UEK3_EXISTS Flag (Excluding RTG4 and MPF)	1

Data Lookup Table

The data look-up table contains records identifying the starting relative location of all the different data blocks used in the JTAG-DirectC code and data size of each block. The format is described in the following table.

Table 5-6. DAT Image Description

Information	# Of Bytes
Data Identifier # 1	1

.....continued	
Information	**# Of Bytes**
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

The data block contains the raw data for all the different variables specified in the lookup table.

Table 5-7. DAT Image Description

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 JTAG-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 and MPFS 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:	JTAG-DirectC cannot identify if the device is standard Fusion or ARM enabled device. JTAG-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 and prevents 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 does not function.

8. Data File Bit Orientation

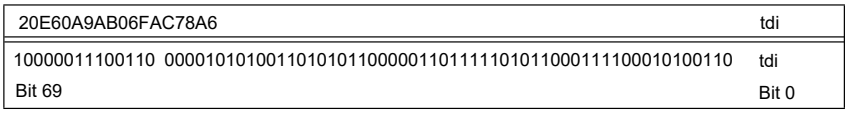
This topic specifies the data orientation of the binary data file generated by Libero software. JTAG-DirectC implementation must be in sync with the specified data orientation. The following table illustrates how the data is stored in the binary data file. See 5. Data File Format for additional information 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 following example 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 is shown in the following figure.

Figure 8-1. Binary Representation of data



This data is stored in the data block section. The following table shows how the data is stored in the data block.

Table 8-2. Data Block Section Example

Byte 0	Byte 1	Byte 2	Byte3	Byte4	..	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 v7.40. It is designed to work on SmartFusion2 Security Evaluation Kit with the SmartFusion2 M2S090-FGG484 device.

Project Requirements

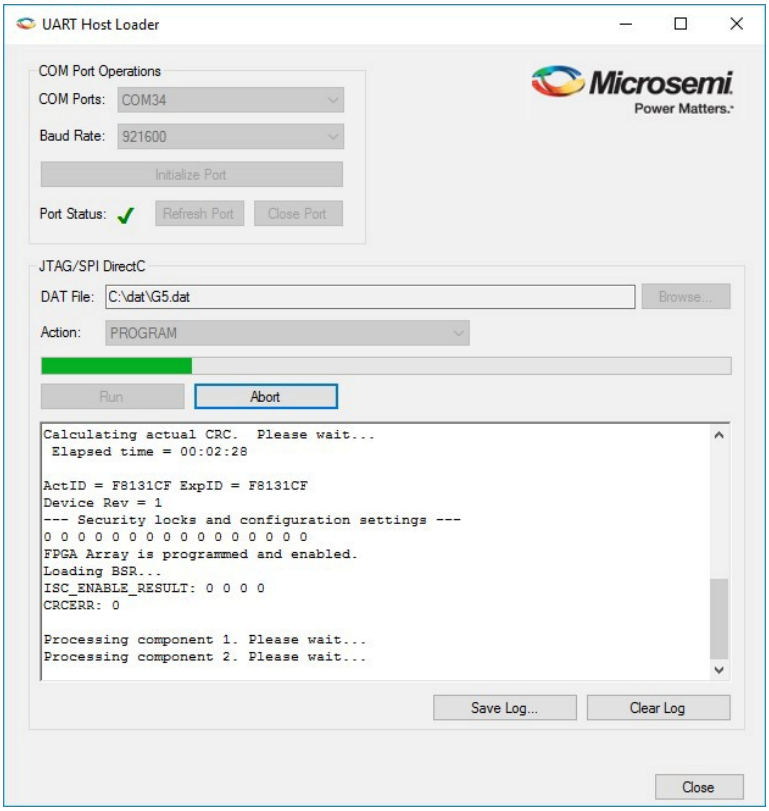
The following hardware and software are required to work with the sample project:

- **Hardware:**
  - SmartFusion2 Security Evaluation Kit with SmartFusion2 M2S090-FGG484 device.
  - jLink from IAR.
  - Target board with Microchip device to be programmed.
- **Software:**
  - IAR Embedded Workbench version 7.4.
  - UART Host Loader available with this release package.

Procedure

1. Program the evaluation kit with `JTAG_DC_top.job` 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 EvalKit Files directory.
3. Connect the Mini USB (J18) to your PC. The mini-USB is connected to the FTDI FT4232h device that is used as a USB to UART bridge.
   Note: Make sure the appropriate drivers are installed on your PC to communicate with this chip.
4. Run **UART Host Loader** available with this release package.
5. Select the fourth com port amongst the four com ports available in the serial port setup window. Configure the Baud Rate as shown in the following figure.

Figure 9-1. UART Host Loader



6. If more than four ports are available, disconnect the J18 header and refresh the com ports in the **UART Host Loader** application to identify exiting ports.
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. Reset the board and run the embedded application to perform the action selected. To run another action or select a different programming file, select it from the **UART Host Loader** and click **Run**.
    The programming file programmed into the evaluation kit has a JTAG-DirectC sample project that supports SmartFusion2, IGLOO2, RTG4, PolarFire, and PolarFire SoC devices.
11. Run the IAR workbench to make changes to the embedded project 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 and Troubleshooting Tips

The information in this chapter might help you solve or identify a problem while using JTAG-DirectC code. See the following table 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. For example, the 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 must 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 is permanently secured and cannot be unlocked. Solution: • Load the correct DAT file.
10	Failed to program FlashROM	Solution: • Check Vpump level. • Try with new device. • Measure JTAG pins and noise or reflection.

.....continued

Exit Code	Error Message	Action/Solution
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	Solution: • Check Vpump level. • Try with new device. • Measure JTAG pins and noise or reflection.
18	Failed to authenticate the encrypted data.	Solution: • Make sure that the AES key used to encrypt the data matches the AES key programmed in the device.
20	Failed to verify FlashROM at row ###.	Solution: • Check Vpump level. • Try with new device. • Measure JTAG pins and noise or reflection. • Make sure that the device is programmed with the correct design.
22	Failed to program pass key	Solution: • Check Vpump level. • Try with new device. • Measure JTAG pins and noise or reflection.
23	Failed to program AES key	Solution: • Check Vpump level. • Try with new device. • Measure JTAG pins and noise or reflection.

.....continued

Exit Code	Error Message	Action/Solution
24	Failed to program UROW	Solution: • Check Vpump level. • Try with new device. • Measure JTAG pins and noise or reflection. • Make sure that you mounted 0.01 uF and 0.33 uF caps on Vpump (close to the pin).
25	Failed to enter programming mode	Solution: • 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.	Solution: • 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.	Solution: • Provide a data file with a valid pass key.
31	Failed to program DMK	Solution: • 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.	Solution: • Provide a data file with an encrypted FlashROM.
35	Pass key match failure.	Solution: • 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.	Solution: • Make sure that the device is properly secured with AES encryption protection is 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.	Solution: • Make sure that the device is properly secured with the AES encryption protection is ON for FPGA Array. • Provide the correct data file for programming.

.....continued

Exit Code	Error Message	Action/Solution
38	Failed to program pass key.	Solution: • 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.	Solution: • 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.	Solution: • 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.	Solution: • 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.	Solution: • Regenerate data file. • Reprogram the data file into the 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 files 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.

11. Revision History

Revision	Date	Description
A	09/2021	• Migrated to the Microchip standard template format. • Renamed document title from **DirectC User Guide** to **JTAG-DirectC User Guide**. • Updated for the new version numbering schema (v202x.x) for DirectC solutions.

Microchip FPGA Support

Microchip FPGA products group backs its products with various support services, including Customer Service, Customer Technical Support Center, a website, and worldwide sales offices. Customers are suggested to visit Microchip online resources prior to contacting support as it is very likely that their queries have been already answered.

Contact Technical Support Center through the website at www.microchip.com/support. Mention the FPGA Device Part number, select appropriate case category, and upload design files while creating a technical support case.

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**

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner and under normal conditions.

- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable." Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-8866-8

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com **Atlanta** Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455 **Austin, TX** Tel: 512-257-3370 **Boston** Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088 **Chicago** Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075 **Dallas** Addison, TX Tel: 972-818-7423 Fax: 972-818-2924 **Detroit** Novi, MI Tel: 248-848-4000 **Houston, TX** Tel: 281-894-5983 **Indianapolis** Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380 **Los Angeles** Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800 **Raleigh, NC** Tel: 919-844-7510 **New York, NY** Tel: 631-435-6000 **San Jose, CA** Tel: 408-735-9110 Tel: 408-436-4270 **Canada - Toronto** Tel: 905-695-1980 Fax: 905-695-2078	**Australia - Sydney** Tel: 61-2-9868-6733 **China - Beijing** Tel: 86-10-8569-7000 **China - Chengdu** Tel: 86-28-8665-5511 **China - Chongqing** Tel: 86-23-8980-9588 **China - Dongguan** Tel: 86-769-8702-9880 **China - Guangzhou** Tel: 86-20-8755-8029 **China - Hangzhou** Tel: 86-571-8792-8115 **China - Hong Kong SAR** Tel: 852-2943-5100 **China - Nanjing** Tel: 86-25-8473-2460 **China - Qingdao** Tel: 86-532-8502-7355 **China - Shanghai** Tel: 86-21-3326-8000 **China - Shenyang** Tel: 86-24-2334-2829 **China - Shenzhen** Tel: 86-755-8864-2200 **China - Suzhou** Tel: 86-186-6233-1526 **China - Wuhan** Tel: 86-27-5980-5300 **China - Xian** Tel: 86-29-8833-7252 **China - Xiamen** Tel: 86-592-2388138 **China - Zhuhai** Tel: 86-756-3210040	**India - Bangalore** Tel: 91-80-3090-4444 **India - New Delhi** Tel: 91-11-4160-8631 **India - Pune** Tel: 91-20-4121-0141 **Japan - Osaka** Tel: 81-6-6152-7160 **Japan - Tokyo** Tel: 81-3-6880- 3770 **Korea - Daegu** Tel: 82-53-744-4301 **Korea - Seoul** Tel: 82-2-554-7200 **Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906 **Malaysia - Penang** Tel: 60-4-227-8870 **Philippines - Manila** Tel: 63-2-634-9065 **Singapore** Tel: 65-6334-8870 **Taiwan - Hsin Chu** Tel: 886-3-577-8366 **Taiwan - Kaohsiung** Tel: 886-7-213-7830 **Taiwan - Taipei** Tel: 886-2-2508-8600 **Thailand - Bangkok** Tel: 66-2-694-1351 **Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393 **Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829 **Finland - Espoo** Tel: 358-9-4520-820 **France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79 **Germany - Garching** Tel: 49-8931-9700 **Germany - Haan** Tel: 49-2129-3766400 **Germany - Heilbronn** Tel: 49-7131-72400 **Germany - Karlsruhe** Tel: 49-721-625370 **Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44 **Germany - Rosenheim** Tel: 49-8031-354-560 **Israel - Ra'anana** Tel: 972-9-744-7705 **Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781 **Italy - Padova** Tel: 39-049-7625286 **Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340 **Norway - Trondheim** Tel: 47-72884388 **Poland - Warsaw** Tel: 48-22-3325737 **Romania - Bucharest** Tel: 40-21-407-87-50 **Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91 **Sweden - Gothenberg** Tel: 46-31-704-60-40 **Sweden - Stockholm** Tel: 46-8-5090-4654 **UK - Wokingham** Tel: 44-118-921-5800 Fax: 44-118-921-5820