# SPI-DirectC v1.2

## User’s Guide

Microsemi®

# Table of Contents

# Introduction

This document describes how to enable processor-based embedded ISP (In-System Programming) on Microsemi IGLOO2™ and SmartFusion2™ devices using the SPI Slave programming method. In-System Programming refers to an external processor on board programming one of the Microsemi IGLOO2™ or SmartFusion2™ devices via SPI peripheral interface.

The document assumes that the target system contains a processor or a soft-core microprocessor with a minimum 1200 bytes of RAM, a SPI interface to the target device from the processor, and access to the programming data to be used for programming the device. Access to programming data can be provided by a telecommunications link for most remote systems.

SPI-DirectC v1.2 is a set of C code designed to support embedded In-System Programming for the M2S and M2GL families of devices. To use SPI-DirectC v1.2, 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 version 11.2 or later. The detailed specification of the programming file is included in "Data File Format" on page 14.

SPI-DirectC v1.2 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 SPI-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, dpcom.c and dpcom.h.

# 1 – System Overview

To perform In-System Programming (ISP) for the SmartFusion2 or IGLOO2 target device, the system must contain the following parameters:

- A microprocessor with at least 1200 bytes of RAM or a softcore processor implemented in another FPGA
- SPI IP to interface to the target device
- Access to the data file containing the programming data
- Memory to store and run SPI-DirectC code

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

Table 1-1 shows the memory requirements.

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 – SPI-DirectC Code Size on M3 16-Bit Mode**

| Text in Bytes | Data in Bytes | BSS in Bytes |
|---|---|---|
| 6434 | 1169 | 60 |

## 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 Table 1-2 for data storage memory requirements.

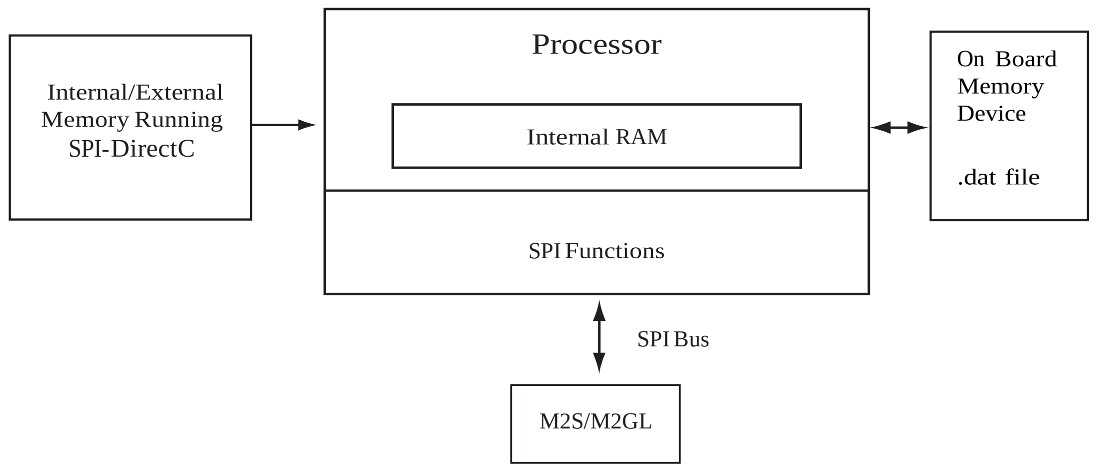

***Figure 1-1* • System with Direct Access to Memory**

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

| Device | Data Image Size | | |
|---|---|---|---|
| | **Core/FPGA Array - Encrypt (kB)** | **Embedded Flash Memory Block - Encrypt (kB)** | **Core/FPGA Array & Security - Encrypt (kB)** |
| M2GL005 | 297 | 133 | 851 |
| M2GL010 | 557 | 267 | 1639 |
| M2GL025 | 1197 | 267 | 2918 |
| M2GL050 | 2364 | 267 | 5253 |
| M2GL090 | 3564 | 532 | 8178 |
| M2GL150 | 5997 | 531 | 13046 |
| M2S005 | 297 | 137 | 860 |
| M2S010 | 557 | 272 | 1648 |
| M2S025 | 1197 | 272 | 2926 |
| M2S050 | 2364 | 272 | 5261 |
| M2S090 | 3564 | 536 | 8186 |
| M2S150 | 5997 | 535 | 13054 |
| The total image size is the sum of all the corresponding enabled blocks for the specific target device. | | | |

## Systems with Indirect 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 holding the data file. *dpcom.h* and *dpcom.c* must be modified to interface with the communication peripheral.

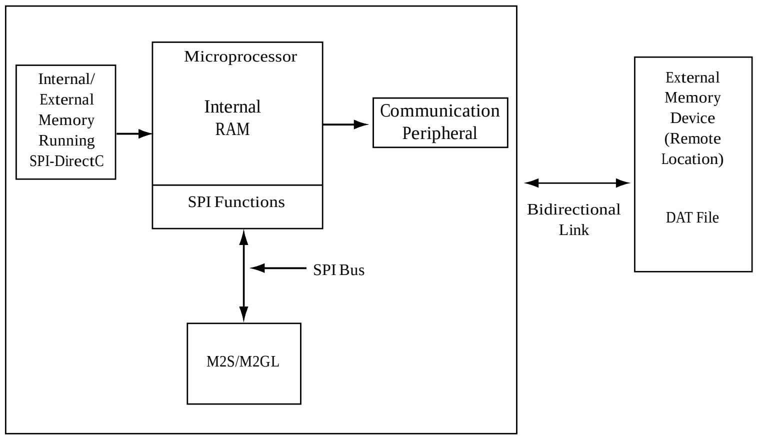

***Figure 1-2* • System With Indirect Access to Memory**

# 2 – Generating Data Files and Integrating SPI-DirectC

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

**To generate your data file:**

1. Generate the DAT file using Libero SoC v11.2 or later. If programming security is required, use Libero SoC v11.4 or later to generate the DAT file. See the latest Libero SoC online help for information on generating a DAT file.
2. Program the DAT file into the storage memory.

## SPI-DirectC v1.2 Code Integration

Figure 2-1 shows the SPI-DirectC integration use flow.

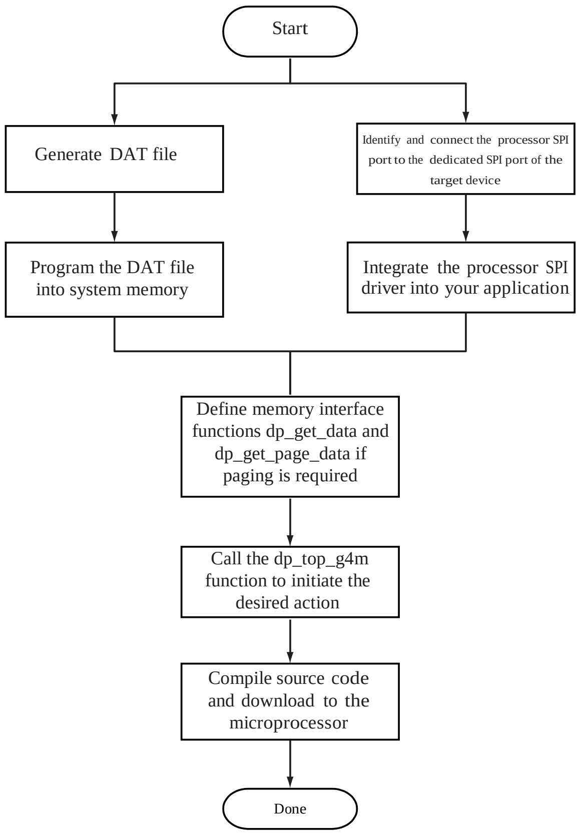
Start
Generate DAT file
Identify and connect the processor SPI port to the dedicated SPI port of the target device
Program the DAT file into system memory
Integrate the processor SPI driver into your application
Define memory interface functions dp_get_data and dp_get_page_data if paging is required
Call the dp_top_g4m function to initiate the desired action
Compile source code and download to the microprocessor
Done

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

**To use SPI-DirectC v1.2 code integration:**

1. Import the SPI-DirectC v1.2 files shown in Figure 2-2 into your development environment.

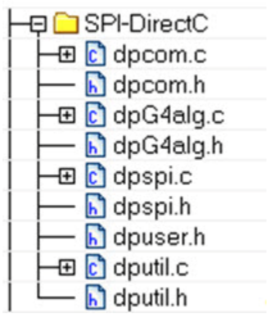
```
SPI-DirectC
├─ dpcom.c
├─ dpcom.h
├─ dpG4alg.c
├─ dpG4alg.h
├─ dpspi.c
├─ dpspi.h
├─ dpuser.h
├─ dputil.c
└─ dputil.h
```

***Figure 2-2* • SPI-DirectC v1.2 Files to import into your Development Environment**

2. Modify the SPI-DirectC code.
   – Add the SPI driver (available with the processor used to run SPI-DirectC).
   – Modify the hardware interface functions (*do_SPI_SCAN_in* and *do_SPI_SCAN_out*) to use the hardware API functions designed to control the SPI port.
   – 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 17.
   – Call *Dp_top_g4m* with the action code desired. See "DPG4ALG.C and DPG4ALG.H" on page 13 for supported actions and their corresponding codes.
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, dpspi.c, dpcom.c* and *dputil.c* files when using the SPI-DirectC source code. "Source File Description" on page 16 contains a short description of SPI-DirectC source code and their functions. Functions that must be modified are listed in Table 3-1.

***Table 3-1*** **• Modified Functions**

| Function | Source File | Purpose |
|---|---|---|
| Do_SPI_SCAN_in | Dpspi.c | Hardware interface function used to scan data in using the SPI driver |
| Do_SPI_SCAN_out | Dpspi.c | Hardware interface function used to scan data out using the SPI driver |
| dp_get_page_data | Dpcom.c | Programming file interface function |
| dp_display_text | Dpuser.c | Function to display text to an output device |
| dp_display_value | Dpuser.c | Function to display value of a variable to an output device |

## Compiler Switches

The compiler switch is shown in Table 3-2.

***Table 3-2*** **• Compiler Switches**

| Compiler Switch | Source File | Purpose |
|---|---|---|
| USE_PAGING | dpuser.h | Enables paging implementation for memory access |

## Hardware Interface Components

### Hardware Interface Function (dpspi.c)

*do_SPI_SCAN_in* and *do_SPI_SCAN_out* functions are used to interface with the SPI port to clock data into and out of the target device. These functions should use the SPI driver API available for the targeted device processor.

#### *Do_SPI_SCAN_in Function*

This function takes three arguments:

- Command: 8-bit variable holding the command value
- Data_bits: The number of bits to clock into the device.
- input_buffer: pointer to the buffer which holds valid data to be clocked into the device.

#### *Dp_SPI_SCAN_OUT Function*

This function takes four arguments:

- Command bits: The number of bits to clock in for the command portion of the frame. This value should be 8 as all SPI commands are 8 bit long.
- Command: 8-bit variable holding the command value
- Data_bits: The number of bits to read from the device.
- Ouput_buffer: pointer to the buffer to hold the data read from the target device.

## Display Functions

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

All access to the memory blocks within the data file is done through the *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.

*Dp_get_data* function takes two arguments:

- 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 the function, it is expected that return_bytes will indicate the total number of valid bytes available for the client of the function.

See "Systems with Direct Access to the Memory Containing the Data File" and "Systems with Indirect Access to the Data File" on page 10 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 can be treated as an array of unsigned characters. In this case:

1. Disable the USE_PAGING compiler switch. See "Compiler Switches" on page 8.
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 the *dp_get_data* function implementation is:

```
DPUCHAR* dp_get_data(DPUCHAR var_ID,DPULONG bit_index)
{
      DPULONG image_requested_address;
      if (var_ID == Header_ID)
            current_block_address = 0;
      else
dp_get_data_block_address(var_ID);

    if ((current_block_address ==0) && (var_ID != Header_ID))
    {
        return_bytes = 0;
        return NULL;
    }

/* Calculating the relative address of the data block needed within the image */
      image_requested_address = current_block_address + bit_index / 8;

     return_bytes=image_size - image_requested_address;
     return image_buffer+image_requested_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 the USE_PAGING compiler option. See "Compiler Switches" on page 8.
2. Define *Page_buffer_size*. The minimum buffer size is 16 bytes.
3. Modify the *dp_get_data* function. For correct operation:
   - The function must return a pointer to the byte which contains the first bit to be processed.
   - The function must update the *return_bytes* variable which specifies the number of valid bytes in the page buffer.
4. 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 17 for additional information. For correct operation:
   - Fill the entire page unless the end of the image is reached. See "Data File Format" on page 14.
   - Update *return_bytes* to reflect the number of valid bytes in the page.

SPI-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.

## Example of dp_get_data Function Implementation

```
DPUCHAR* dp_get_data(DPUCHAR var_ID,DPULONG bit_index)
{
      DPULONG image_requested_address;

      DPULONG page_address_offset;

      if (var_ID == Header_ID)
            current_block_address = 0;
      else
            dp_get_data_block_address(var_ID);

    if ((current_block_address ==0) && (var_ID != Header_ID))
    {
        return_bytes = 0;
        return NULL;
    }

      /* Calculating the relative address of the data block needed within the image */
      image_requested_address = current_block_address + bit_index / 8;

      /* If the data is within the page, adjust the pointer to point to the particular
element requested */
      if ((image_requested_address >= start_page_address) && (image_requested_address
<= end_page_address))
      {
            page_address_offset = image_requested_address - start_page_address;
            return_bytes = end_page_address - image_requested_address + 1;
      }
      /* Otherwise, call dp_get_page_data which would fill the page with a new data
block */
      else
      {
            dp_get_page_data(image_requested_address);
            page_address_offset = 0;
      }
      return &page_global_buffer[page_address_offset];
}
```

### *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* and *end_page_address* that contain the range of data currently in the page.

*dp_get_page_data* takes two arguments:

- address_offset - Contains the relative address of the needed element within the data block of the image file.
- *preturn_bytes* - Points to the *return_bytes* variable that should be updated with the number of valid bytes available.

```
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_g4m* defined in *dpG4alg.c*. It must be called to initiate the programming operation. Prior to calling the function, a global variable Action_code must be assigned a value as defined in dpuser.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
```

Note: Programming of individual blocks, such as array only, eNVM only, or security only is not possible with one data file because of how the data is constructed. If you wish to use such a feature you must generate multiple data files.

## Data Type Definitions

Microsemi uses *DPUCHAR*, *DPUINT*, *DPULONG*, *DPBOOL*, *DPCHAR*, *DPINT*, and *DPLONG* in the SPI-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

**Action:** DP_DEVICE_INFO_ACTION
**Purpose:** Displays device security settings and design and checksum information

**Action:** DP_READ_IDCODE_ACTION

**Purpose:** Reads and displays the content of the IDCODE register

**Action:** DP_ERASE_ACTION

**Purpose:** Erases the FPGA and security information, if supported in the data file

**Action:** DP_PROGRAM_ACTION

**Purpose:** Performs erase, program and verify operations of FPGA, eNVM and security if supported in the data file

**Action:** DP_VERIFY_ACTION

**Purpose:** Performs verify operation for all the supported blocks in the data file

**Action:** DP_ENC_DATA_AUTHENTICATION_ACTION

**Purpose:** Performs data authentication to make sure the data was encrypted with the same encryption key as the device

# 4 – Data File Format

## DAT File Description for M2GL and M2S Devices

The M2GL and M2S 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 SPI-DirectC code and data size of each block. The format is described in Table 4-1.
- **Data Block** - Contains the raw data for all the different variables specified in the lookup table.

***Table 4-1* • 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 |

*Table 4-1* • **DAT Image Description (continued)**

| **Header Section of DAT File** | |
|---|---|
| ENVM Verify Data Size | 2 |
| 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 |

# 5 – Source File Description

## DPUSER.H

File contains definitions of all Action codes as well as possible error codes that could be reported within SPI-DirectC code.

## DPCOM.C and DPCOM.H

These files contain memory interface functions.

## DPG4ALG.C and DPG4ALG.H

These files contain the main entry function dp_top_g4 and all other functions common to M2S and MGL families.

## DPSPI.C and DPSPI.H

These files contain the SPI interface function declaration and definition to the target device.

## DPUTIL.C and DPUTIL.H

These files contain utility functions needed in the SPI-DirectC code.

# 6 – Data File Bit Orientation

This chapter specifies the data orientation of the binary data file generated by the Libero software. The SPI-DirectC implementation must be in sync with the specified data orientation. Table 6-1 illustrates how the data is stored in the binary data file. See "Data File Format" on page 14 for additional information on the data file.

*Table 6-1* • **Binary Data File Example**

| Byte O | Byte 1 | Byte 2 | Byte 3 | .. | .. | Byte N |
|---|---|---|---|---|---|---|
| Bit7..Bit0 | Bit15..Bit8 | Bit23..Bit16 | Bit31..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. An 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
1000001110011O 0000101010011010101100000110111110101100011110001010011O tdi
Bit 69                                                          Bit 0
```

This data is stored in the data block section. Table 6-2 shows how the data is stored in the data block.

*Table 6-2* • **Data Block Section Example**

| Byte O | Byte 1 | Byte 2 | Byte 3 | Byte 4 | .. | Byte 8 |
|---|---|---|---|---|---|---|
| Bit7...Bit0 | Bit15..Bit8 | Bit23..Bit16 | Bit31..Bit24 | Bit43..Bit32 | .. | Bit71..Bit64 |
| 10100110 | 01111000 | 10101100 | 01101111 | 10110000 | | 00100000 |
| A6 | 78 | AC | 6F | B0 | | 20 |

# 7 – Error Messages and Troubleshooting Tips

The information in this chapter may help you solve or identify a problem when using SPI-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 at 1-800-262-1060.

See Table 7-1 for a description of exit codes and their solutions.

***Table 7-1*** **• Exit Codes**

| Exit Code | Error Message | Action/Solution |
|---|---|---|
| 0 | This code does not indicate an error. | This message indicates success |
| 2 | Data processing failed. | - Check the Vpump level.<br>- Try with a new device.<br>- Measure SPI pins and noise or reflection.<br>- Load the correct DAT file. |
| 6 | The IDCODE of the target device does not match the expected value in the DAT file image. | Possible Causes:<br>- The data file loaded was compiled for a different device. Example: M2S010 DAT file loaded to program M2S050 device.<br>- Noise or reflections on one or more of the SPI pins causing incorrect read-back of the SDO Bits.<br>Solution:<br>- Choose the correct DAT file for the target device.<br>- Cut down the extra length of ground connection. |
| 7 | Device polling error. | - Check the Vpump level<br>- Try with a new device<br>- Measure SPI pins and noise or reflection.<br>- Load the correct DAT file. |
| 8 | FPGA failed during the Erase operation. | Possible Causes:<br>- The device is secured, and the corresponding data file is not loaded. The device has been permanently secured and cannot be unlocked.<br>Solution:<br>- Load the correct DAT file. |
| 10 | Failed to program device. | - Check Vpump level.<br>- Try with new device.<br>- Measure SPI pins and noise or reflection. |

***Table 7-1* • Exit Codes (continued)**

| Exit Code | Error Message | Action/Solution |
|---|---|---|
| 11 | FPGA failed verify. | Possible Cause:<br>- The device is secured, and the corresponding DAT file is not loaded.<br>- The device is programmed with an incorrect design.<br>Solution:<br>- Load the correct DAT file.<br>- Check Vpump level.<br>- Measure SPI pins and noise or reflection. |
| 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. |
| 25 | Device initialization failure. | - Check Vpump level.<br>- Try with new device.<br>- Measure SPI pins and noise or reflection. |
| 100 | CRC data error. Data file is corrupted or programming on system board is not successful. | - Regenerate data file.<br>- Reprogram data file into system memory. |
| 150 | Request action is not found. | Check spelling. |
| 151 | Action is not supported because required data block is missing from the data file. | Regenerate DAT file with the needed block/feature support. |

# 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 http://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.

## 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@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.

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; 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 3,600 employees globally. Learn more at **www.microsemi.com**.

**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