AC491
Application Note
PolarFire EDAC and Scrubbing of Fabric RAMs

Microsemi
a Microchip company

Microsemi 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
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi, a wholly owned subsidiary of Microchip Technology Inc. (Nasdaq: MCHP), offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Learn more at www.microsemi.com.

51900491. 2.0 3/21

Contents

Figures

Tables

1 Revision History

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

1.1 Revision 2.0

Added Appendix 2: Running the TCL Script, page 20.

1.2 Revision 1.0

The first publication of this document.

2 EDAC and Scrubbing of Fabric RAMs

The CoreEDAC IP generates EDAC circuitry for both internal (on-chip) and external RAM blocks. The user data is fed to the EDAC encoder, which calculates the parity bits and appends these to the user data, forming a codeword. The codeword is stored into the RAM. During user read, the read codeword is decoded first, which detects and corrects errors (if any), discards parity bits, and outputs the corrected user data word. Scrubbing periodically checks every memory location using the ECC decoder. If a location contains a corrupted word, the decoder detects and corrects the word. The scrubbing circuitry then writes the corrected word back to the same location. To provide normal access to the RAM and prevent decreasing performance, scrubbing is only done during idle periods. The scrubbing circuitry sets a proper write address and write enable signals, writing the corrected codeword back to the RAM. Writeback occurs only upon detecting an error.

The application note design can be programmed using the following option:

- Using the job file: To program the device using the job file provided along with the design files, see Appendix 1: Programming the Device Using FlashPro Express.

2.1 Design Requirements

The following table lists the hardware and software requirements for this application note design.

Table 1 • **Design Requirements**

Requirement	Version
Operating system	64-bit Windows 7, 8, or 10
Hardware	
PolarFire Evaluation Kit (MPF300-EVAL-KIT) -12 V/5 A AC power adapter and cord -USB 2.0 A to mini-B cable for UART and programming	Rev D or later
Software	
Libero SoC	**Note:** Refer to the `readme.txt` file provided in the design files for the software versions used with this reference design.
FlashPro Express	

Note: Libero SmartDesign and configuration screen shots shown in this guide are for illustration purpose only. Open the Libero design to see the latest updates.

2.2 Prerequisites

Before you start:

1. Download the design files from:
   http://soc.microsemi.com/download/rsc/?f=mpf_ac491_df
2. Download and install Libero SoC from:
   https://www.microsemi.com/product-directory/design-resources/1750-libero-soc

2.3 Application Note Design

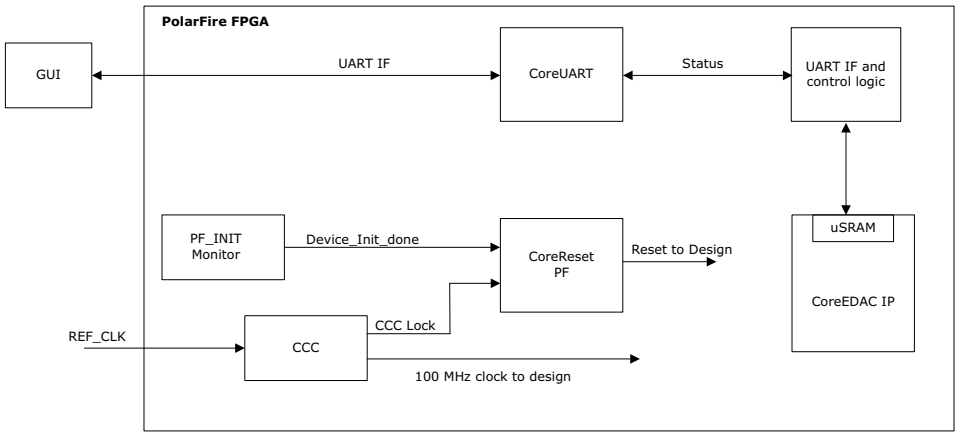
EDAC detects a 1-bit error or 2-bit error when data is read from the memory. If EDAC detects the 1-bit error, the EDAC controller corrects the same error bit. If EDAC is enabled for all the 1-bit and 2-bit errors, corresponding error counters will get incremented. To demonstrate this, an error is introduced manually and detection and correction is observed. The application note design shows how to use CoreEDAC IP with internal µSRAMs for the following:

1. Detect and correct single-bit errors
2. Detect double-bit errors
3. Perform auto memory scrubbing and on-demand memory scrubbing

The similar approach can be used for on-chip LSRAMs. This application note design does not demonstrate the use of EDAC with external memories.

The block diagram of the design is shown in Figure 1.

Figure 1 • **Block Diagram**



1. Configure CoreEDAC IP to use internal µSRAM blocks and set the memory depth to 1kb. Enable error inject test port to induce an errors into the memory. Enable scrubbing logic to correct the errors.
2. The user RTL logic is used to write and read 1kb data to and from the memory. This logic will also monitor the status flags set by the CoreEDAC controller for monitoring the scrubbing request and detecting single-bit and double-bit errors.
3. UART is used to interface with GUI to provide commands to the design and display the error counters and status flags.
4. The application note design performs the following operations using the GUI.

- **Memory Write**: Initialize 1kb µSRAM memory with incremental data.
- **Memory Read**: Read 1kb data from µSRAM memory to verify the memory is initialized. Initially, the memory is not corrupted; hence the error counters must be zero.
- **Inject Error**: Inject Single-bit or double-bit error in memory locations.
- **Memory Read**: Read the data from corrupted µSRAM memory.
  - **Single-bit error**: The relevant counter will be incremented and the corrected data will be available.
  - **Double-bit error**: The relevant counter will be incremented and the erroneous data will be available.
- **Memory Scrub**: Performs memory scrubbing and correct single bit errors in µSRAM memory.
- **Single Memory Write**: User can access the entire 1kb memory through this operation. User can provide the memory address and the data to be written during this operation.
- **Single Memory Read**: User can access the entire 1kb memory through this operation. User can provide the memory address and the data to be read during this operation.
- **Loop Test**: User can use this option to perform the operations from step to step in one click.

2.3.1 Design Description

This application note design involves implementation of following tasks:

- Writing data to µSRAM
- Reading data from µSRAM
- Corrupting one or two bits
- Reading the data

Note: In the case of a 1-bit error, the EDAC controller corrects the error. In the case of a 2-bit error, the EDAC controller does not correct the error.

The following tests are implemented in this application note design.

- Loop Test
- Manual Test

Note: These tests are applicable to both 1-bit and 2-bit errors.

2.3.1.1 Loop Test

Loop Test is executed when the PolarFire FPGA receives a loop test command from the GUI. Initially, all the error counters and EDAC related registers are placed in the RESET state.

The following steps are executed for each iteration:

1. 1kb incremental data is written into the memory.
2. 1kb data is read from the memory.
3. 50 single bit errors are injected into the memory.
4. The memory is read and the error count is displayed.
5. Scrubbing is performed on the memory and the scrubbing correction count is displayed.
6. 1-bit or 2-bit error detection and correction is sent to the GUI.

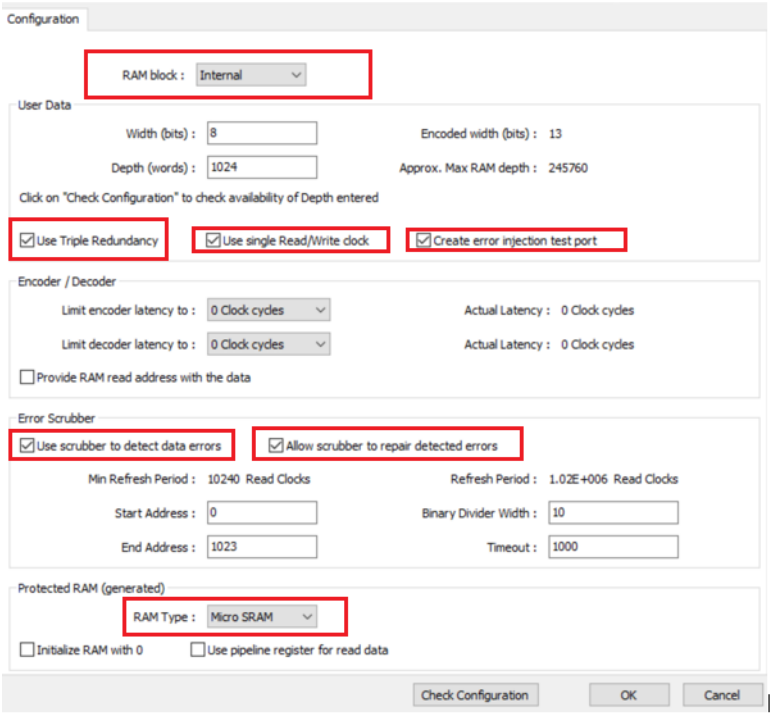
2.3.1.2 Manual Test

This method allows manual testing for enabling or disabling EDAC and write or read operation. Using this method, 1-bit or 2-bit errors can be introduced to any location within the µSRAM. Enable the EDAC and write data to the specified address using the GUI fields. Disable the EDAC and write 1-bit or 2-bit corrupted data to the same address location. Enable the EDAC and read the data from the same address location. The corresponding error counter is displayed on the GUI. The GUI Serial Console logs all the actions performed in PolarFire.

As shown in Figure 2, the CoreEDAC IP is configured with the following options:

- Internal µSRAM blocks are selected and the memory depth is set to 1Kb.
- Error injection is enabled for inducing errors into the memory.
- Scrubbing logic is enabled for correcting errors.
- Triple Redundancy is also enabled for the generation of three independent sets of the EDAC circuitry and the majority vote logic for protecting the EDAC from soft errors.
- Scrubber is enabled to detect data errors and repair them.

Figure 2 • **CoreEDAC IP Configuration**



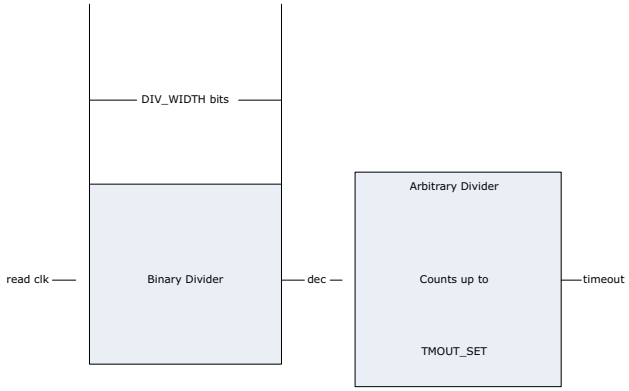
2.3.1.3 Scrubbing Refresh Period

The refresh period defines how often the scrubbing session runs. The block diagram of the refresh period timer is shown in Figure 3. The timer is driven by the RCLK signal. The binary divider that has a configurable bitwidth of DIV_WDTH, generates a relatively slow signal, *dec*. The frequency of the *dec* signal equals the frequency of the read clock divided by 2^{DIV_WDTH}. The *dec* signal serves as an input to the configurable arbitrary divider. It divides *dec* frequency by arbitrary number TMOUT_SET. As a result, the circuitry generates a timeout output signal once per TMOUT_SET × 2^{DIV_WDTH} RCLK periods.

The refresh period must be more than 10 times the scrubbing time that is DIV_WDTH, and TMOUT_SET parameters must satisfy the following condition: TMOUT_SET*2^{DIV_WDTH} > 10*(SCRUB_AMAX - SCRUB_AMIN). The timeout signal initiates another scrubbing session. As the user access takes priority over scrubbing, there might be an exception.

For more information about setting the refresh period timer, scrubbing, and scrubbing refresh period, see *HB0143:CoreEDAC IP Handbook*. Table 12 in this handbook explains DIV_WDTH, TMOUT_SET, SCRUB_AMAX, and SCRUB_AMIN parameters.

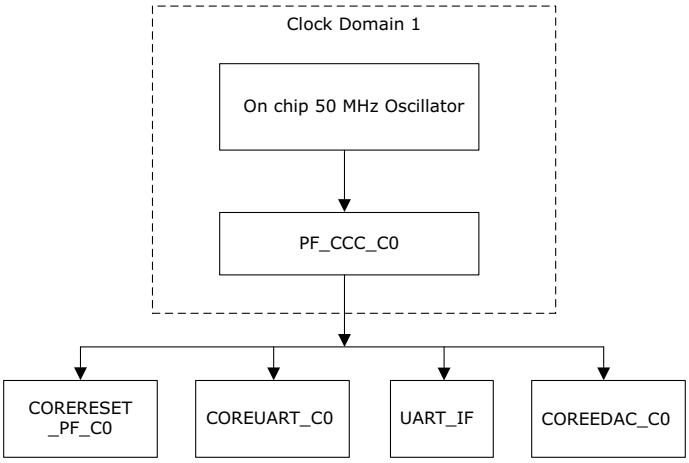
Figure 3 • **Refresh Period Timer**



2.4 Clocking Structure

In this design, there is one clock domain. The on-board 50 MHz crystal oscillator is connected to the PF_CCC block which generates 100 MHz clock that provides clock source to CORERESET_PF, COREUART, UART_IF, and COREEDAC modules. The following figure shows the clocking structure of the design.

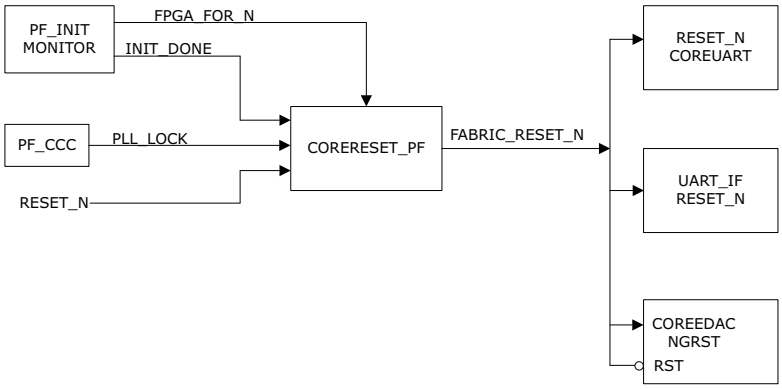
Figure 4 • **Clocking Structure**



2.5 Reset Structure

In this design, the reset signal of COREUART, UART_IF, and COREEDAC blocks are issued using the CORERESET_PF module. The CORERESET_PF module releases active low reset when the PF_CCC lock and PF_INIT_MONITOR INIT_DONE are asserted. The following figure shows the reset structure of the design.

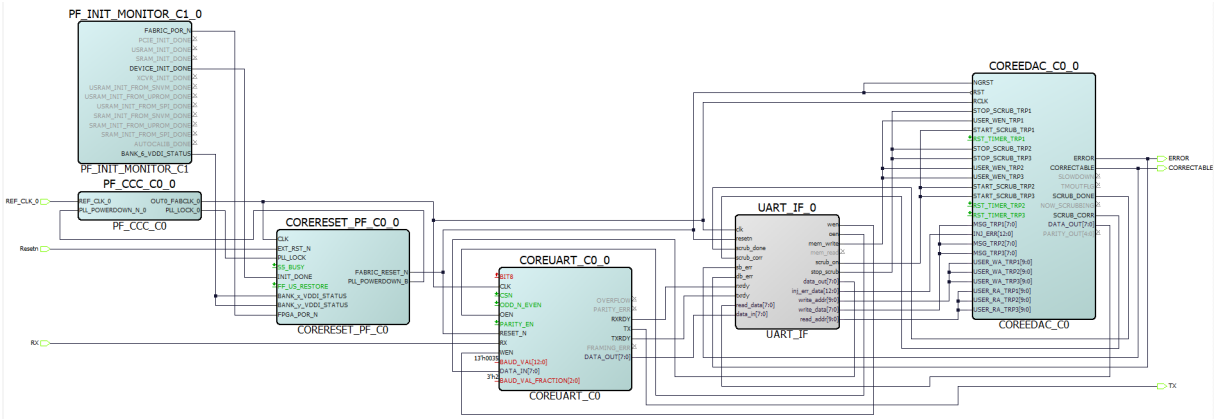
Figure 5 • **Reset Structure**



2.6 Hardware Implementation

Figure 6 shows the PolarFire EDAC and scrubbing design implemented in Libero SoC.

Figure 6 • **Hardware Implementation**



3 Setting Up the Hardware

The following steps describe how to setup the hardware.

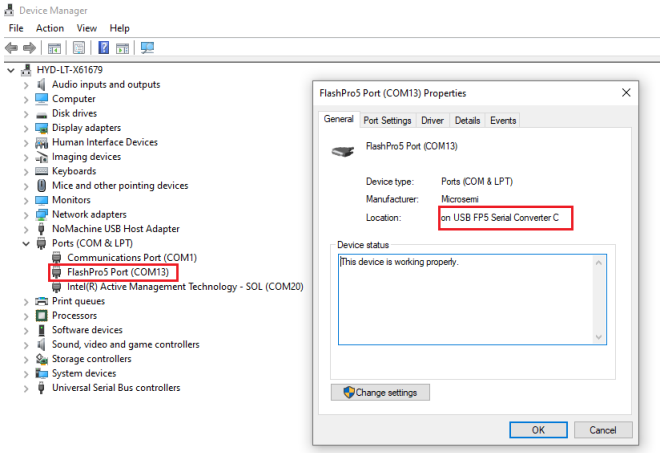
1. Ensure that the following Jumper Settings are set on the board.

Table 2 • **Jumper Settings**

Jumper	Description	Default
J18, J19, J20, J21, and J22	Short pin 2 and 3 for programming the PolarFire FPGA through FTDI	Closed
J28	Short pin 1 and 2 for programming through the on-board FlashPro5	Open
J26	Short pin 1 and 2 for programming through the FTDI SPI	Closed
J4	Short pin 1 and 2 for manual power switching using SW3	Closed
J12	Short pin 3 and 4 for 2.5 V	Closed

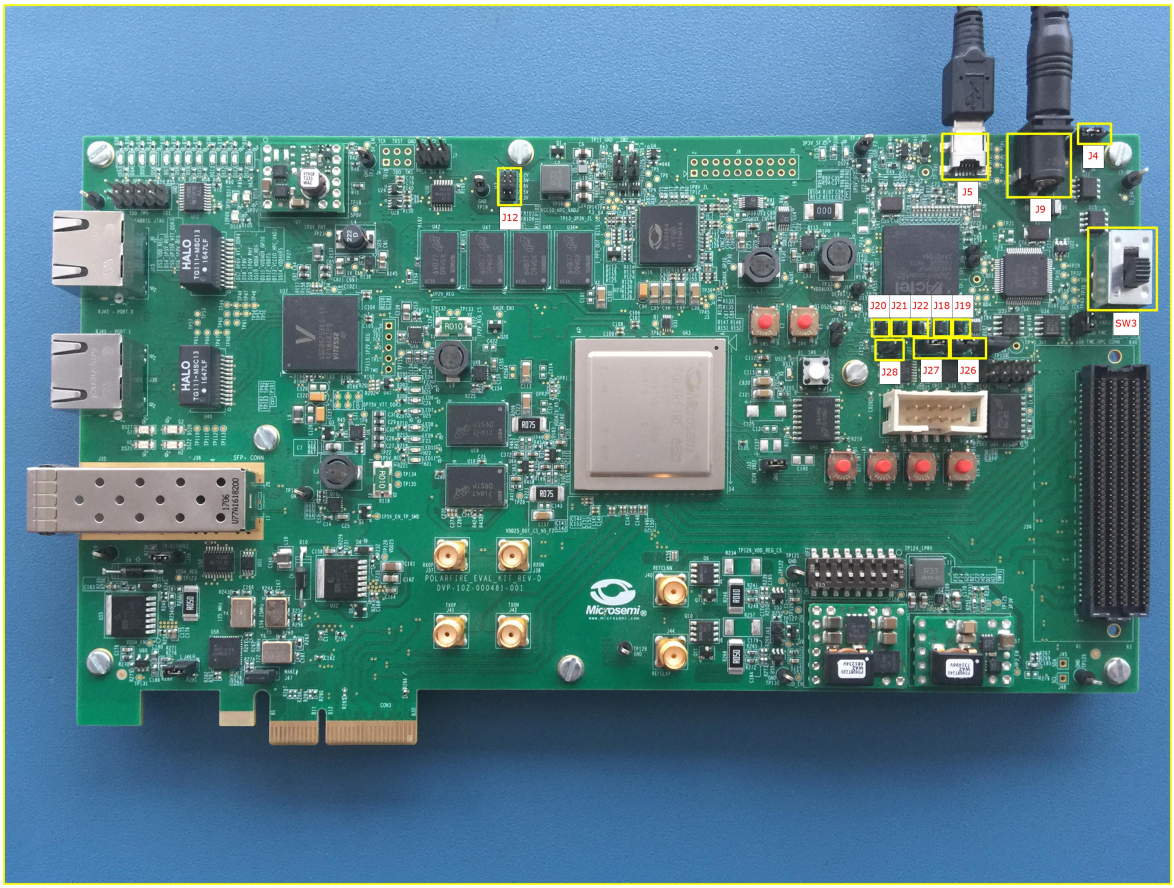
2. Connect the power supply cable to the **J9** connector on the board.
3. Connect the USB cable from the Host PC to **J5** (FTDI port) on the board.
4. Power on the board using the **SW3** slide switch.
5. Ensure that the USB to UART bridge drivers are automatically detected. This can be verified in the device manager of the host PC.
6. As shown in Figure 7, the port properties of COM13 show that it is connected to USB Serial Converter C. Hence, COM13 is selected in this example. The COM port number is system specific.

Figure 7 • **Device Manager**



The PolarFire board setup is shown in Figure 8.

Figure 8 • **PolarFire Board Setup**



4 Running the Demo

The PolarFire EDAC application is a simple Graphic User Interface (GUI) that runs on the host PC to communicate with the PolarFire Device. Before running the demo, ensure Setting Up the Hardware and Appendix 1: Programming the Device Using FlashPro Express.

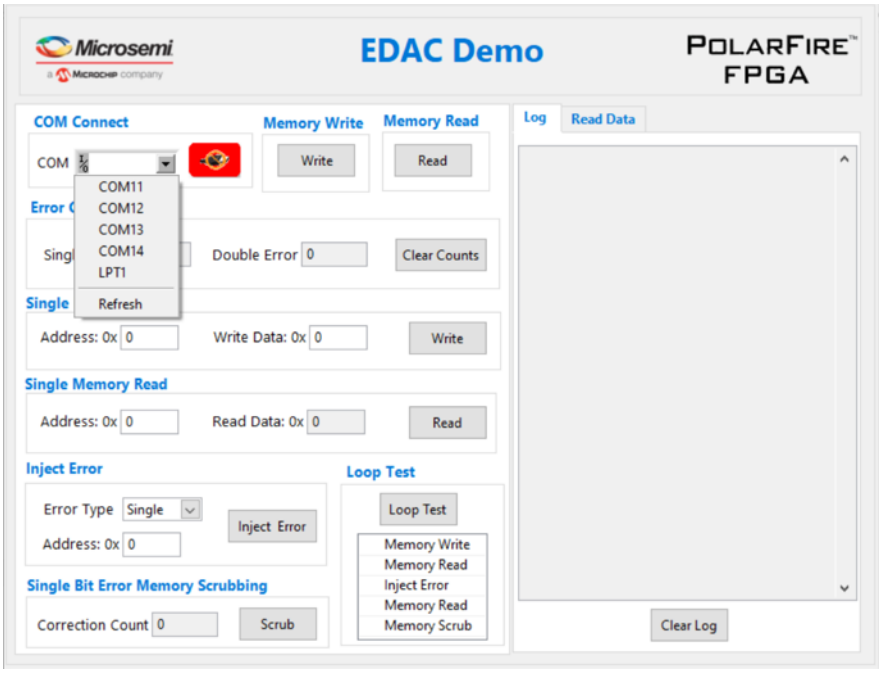
To run the EDAC demo:

1. Run the `setup.exe` file available at the following design files location:

   `<$Download_Directory>\mpf_ac491_df\GUI\EDAC_PF_GUI.exe`

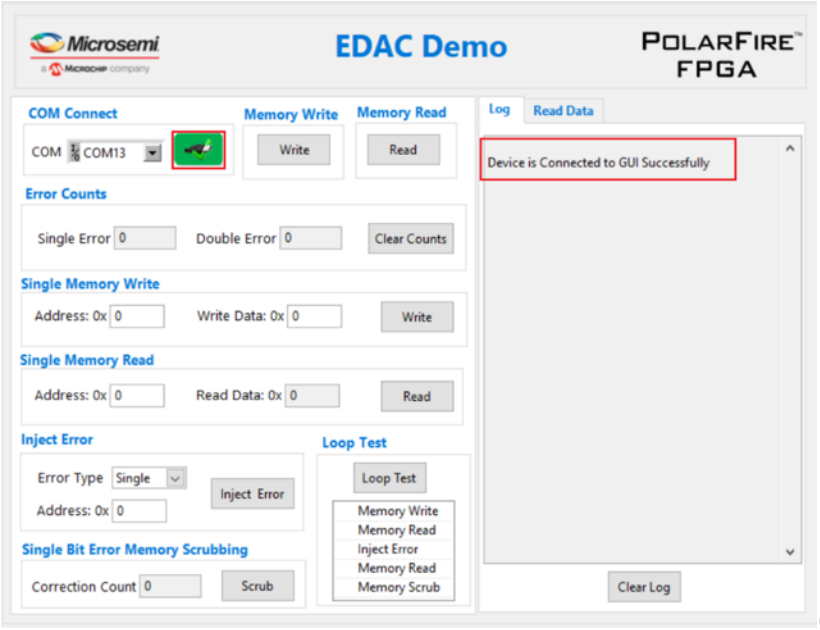
2. Follow the installation wizard to install the GUI application.
3. After successful GUI Installation. Invoke the EDAC GUI from **All Programs > EDAC Demo > EDAC PF GUI**
4. Open the PolarFire EDAC GUI and select the COM port.

Figure 9 • **Selecting the COM Port**



5. After selecting the COM port, click **Connect** icon as highlighted in Figure 10. The log window will display connection successful message.

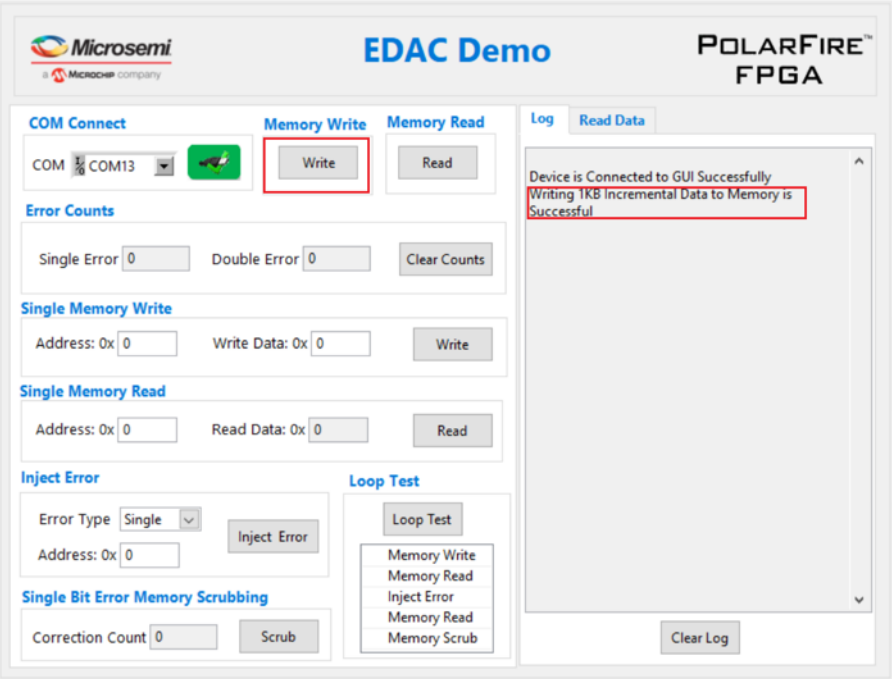
Figure 10 • **Device Connection Successful Message**



0

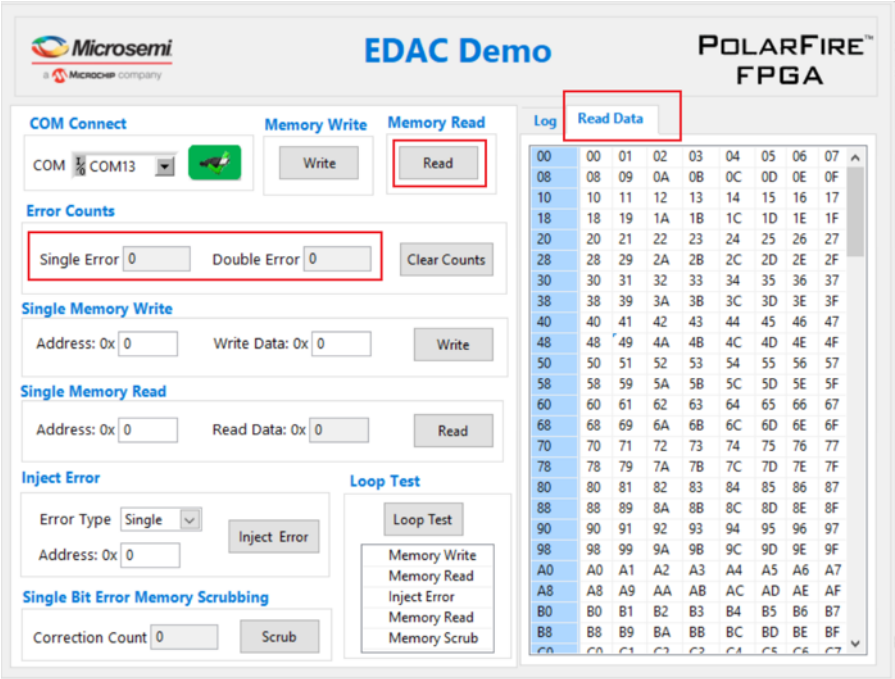
6. Click **Write** as highlighted in Figure 11 to write 1kb incremental data into the memory. The log window will display the write successful message.

Figure 11 • **Write Incremental Data to Memory**



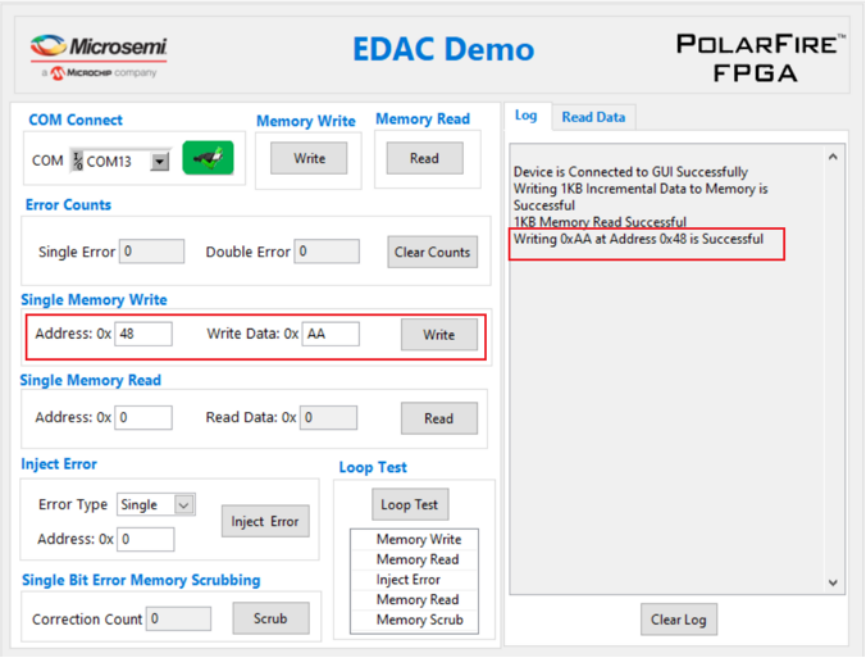
7. Click **Read** as highlighted in Figure 12 to read 1kb data. The read data will be displayed in the **Read Data** tab and the single bit errors and double bit errors will be displayed in their respective fields. Also, the log window will display the read successful message.

Figure 12 • **Single Bit Errors and Double Bit Errors Count**



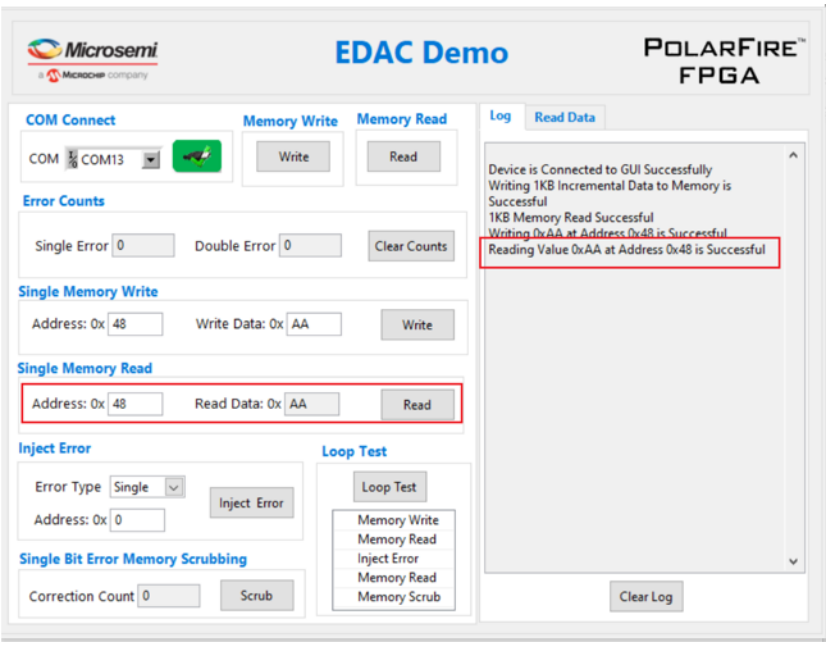
8. For a single memory location, enter the address and data to be written and click **Write** as highlighted in Figure 13.

Figure 13 • **Writing 0xAA at Address 0x48**



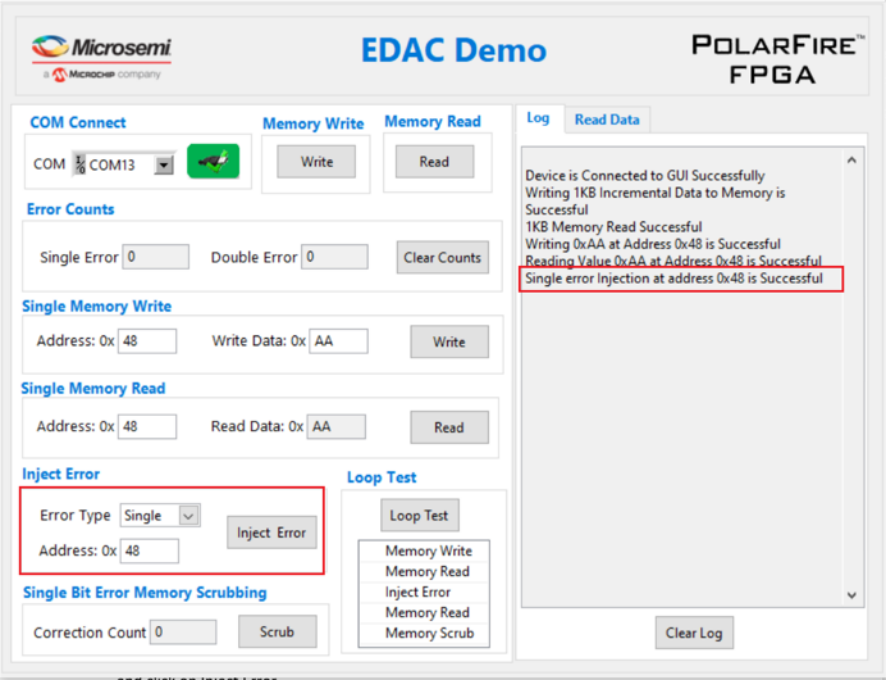
9. For reading a single location, provide the memory address and click **Read** as highlighted in Figure 14. The corresponding data will be displayed in the GUI.

Figure 14 • **Reading Value at 0xAA at Address 0x48**



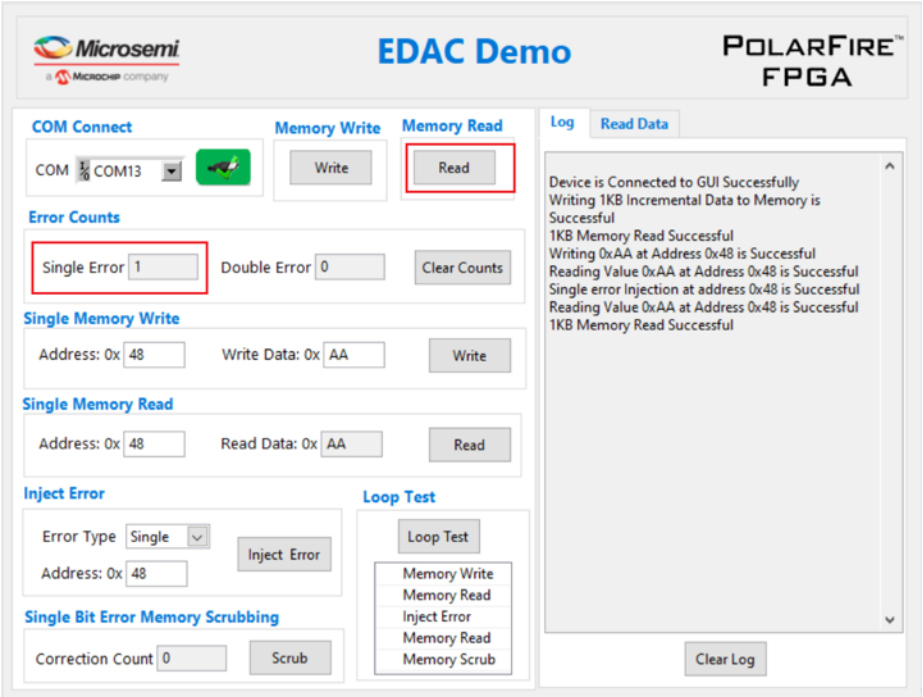
10. For injecting a single bit error, provide the memory address and click **Inject Error** as highlighted in Figure 15.

Figure 15 • **Single Error Injection at Address 0x48**



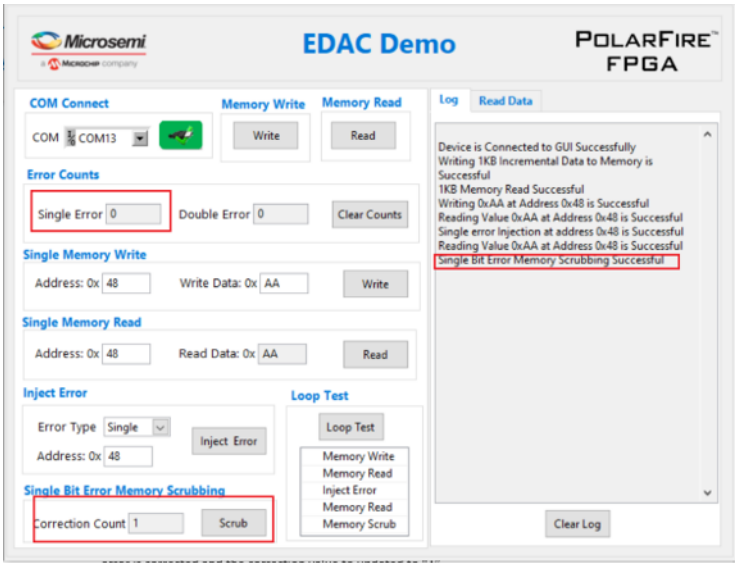
11. After Injecting a single bit error, click **Read** as highlighted in Figure 16. The Single Error count will be updated to 1.

Figure 16 • **Single Error count**



12. The Memory Scrubbing detects and corrects single bit errors. Click **Scrub** as highlighted in Figure 17 to perform scrubbing on the memory. After the scrubbing is completed, the **Correction Count** will display the number of single bit errors corrected during scrubbing.

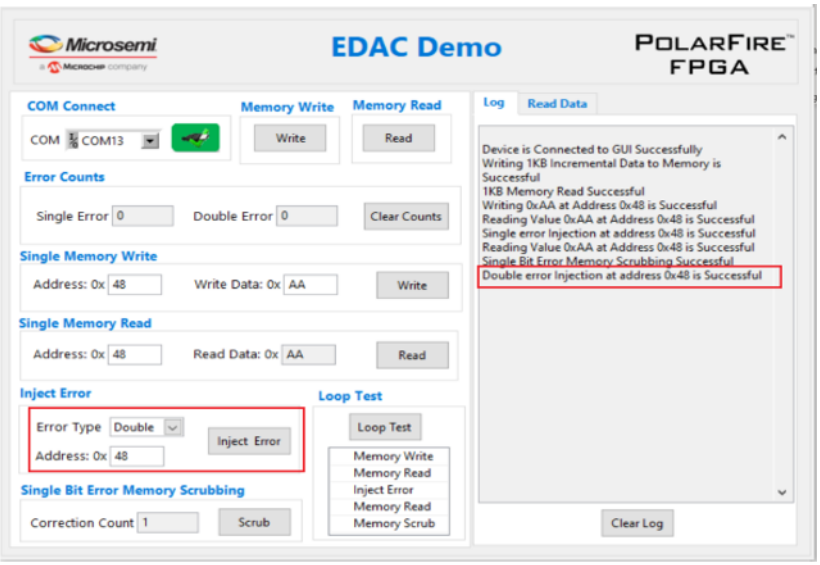
Figure 17 • **Single Bit Error Memory Scrubbing**



Note: One single bit error is introduced in step 11. Hence, during scrubbing, the single bit error is corrected, and the correction value is updated to "**1**".

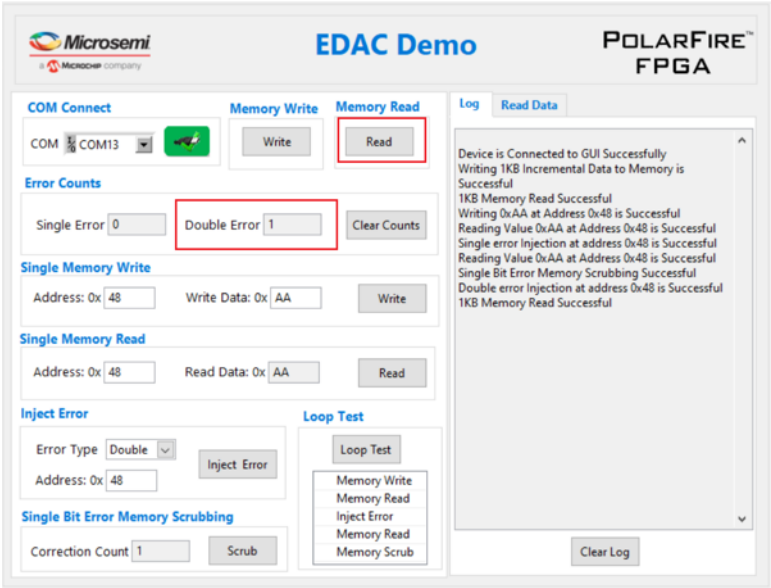
13. For injecting Double bit error, select the **Error Type** to **Double,** provide the memory address and click **Inject Error** as highlighted in Figure 18.

Figure 18 • **Double Error Injection at Address 0x45**



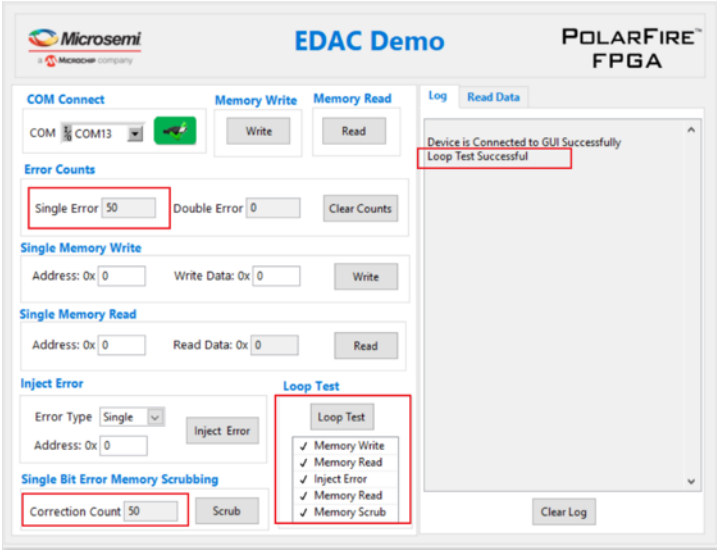
14. Click **Read** as highlighted in Figure 19. The **Double Error** will be updated to "**1**", indicating a double bit error exists in the memory. The **Single Error** is zero because the single bit error is corrected during scrubbing as mentioned in the previous step.

Figure 19 • **Double Error count**



15. Click **Loop Test** as highlighted in Figure 20. The loop test performs the following operations.
    - Writes 1kb incremental data into the memory.
    - Reads 1kb data from the memory.
    - Injects 50 single bit errors into the memory.
    - Reads the memory and displays the error count.
    - Performs scrubbing on the memory and displays the scrubbing correction count.

Figure 20 • **Loop Test**



This concludes the PolarFire EDAC Demo.

5 Conclusion

This demo highlights the EDAC capabilities of the PolarFire µSRAM memories. The 1-bit error or 2-bit error are introduced manually. 1-bit error correction and 2-bit error detection is observed using a GUI.

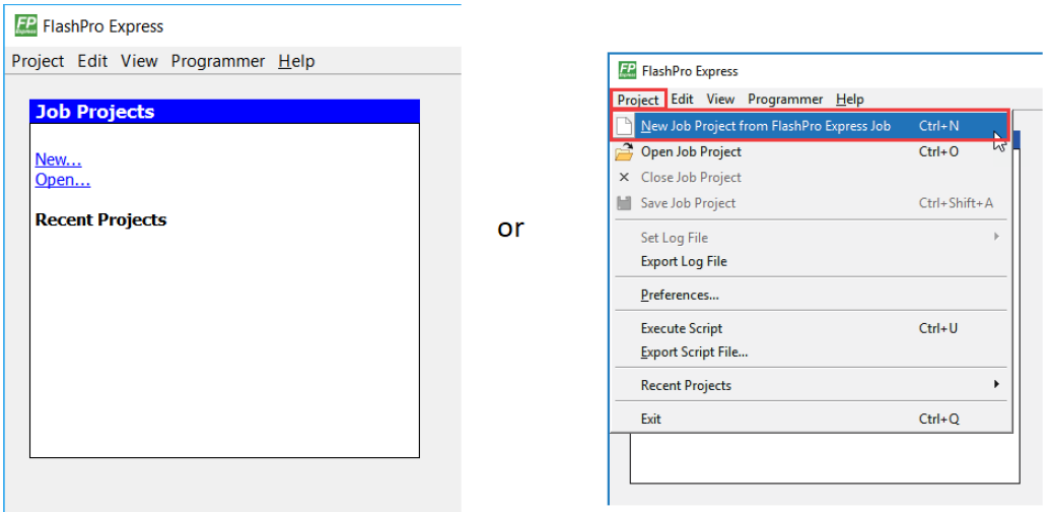
6 Appendix 1: Programming the Device Using FlashPro Express

This chapter describes how to program the PolarFire device with the job file using Flashpro Express. The job file is available at the following design files folder location:
<$Download_Directory>\mpf_ac491_df/Programming_Job

Follow these steps:

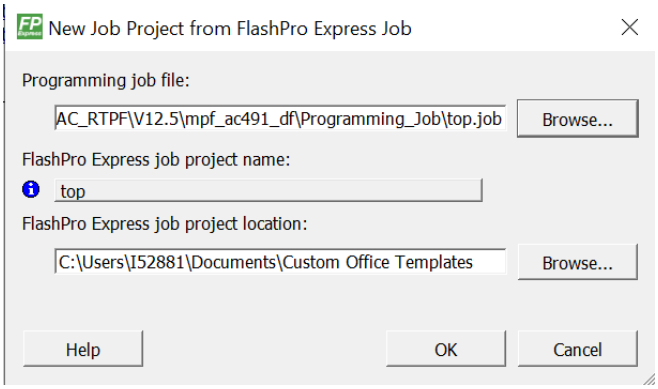
1. On the host PC, start the FlashPro Express software from its installation directory.
2. Select **New or New Job Project from FlashPro Express Job** from Project menu to create a new job project, as shown in Figure 21, page 18.

Figure 21 • **FlashPro Express Job Project**



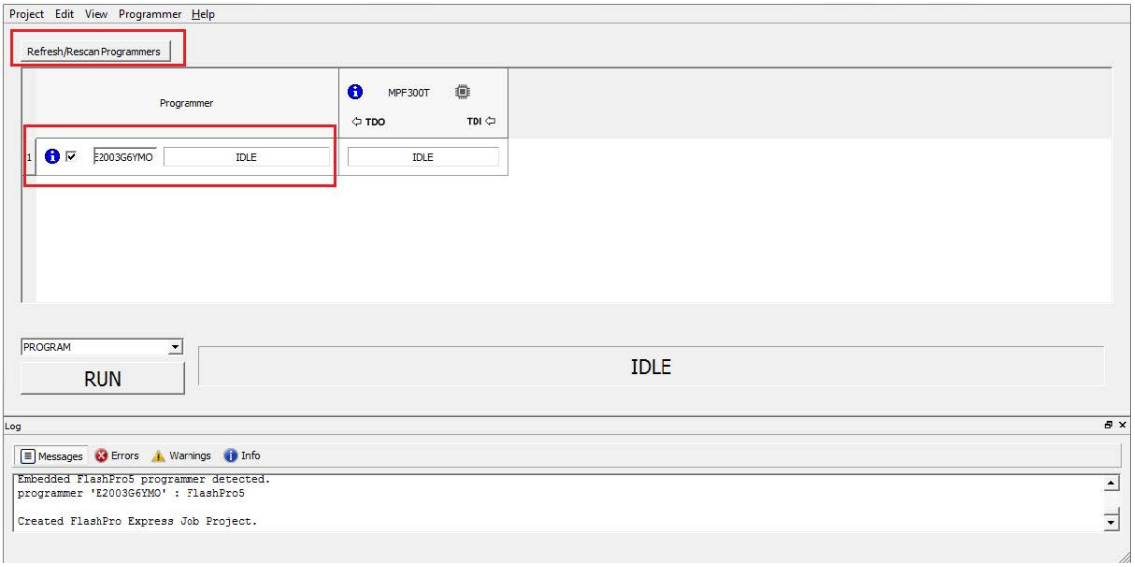
3. Enter the following in the New Job Project from FlashPro Express Job dialog box:
   - Programming job file: Click **Browse** and navigate to the location where the job file is located and select the file. The default location is: *<$Download_Directory>\mpf_ac491_liberosoc_jb*
   - **FlashPro Express job project location**: Select **Browse** and navigate to the location where you want to save the project.

Figure 22 • **New Job Project from FlashPro Express Job**



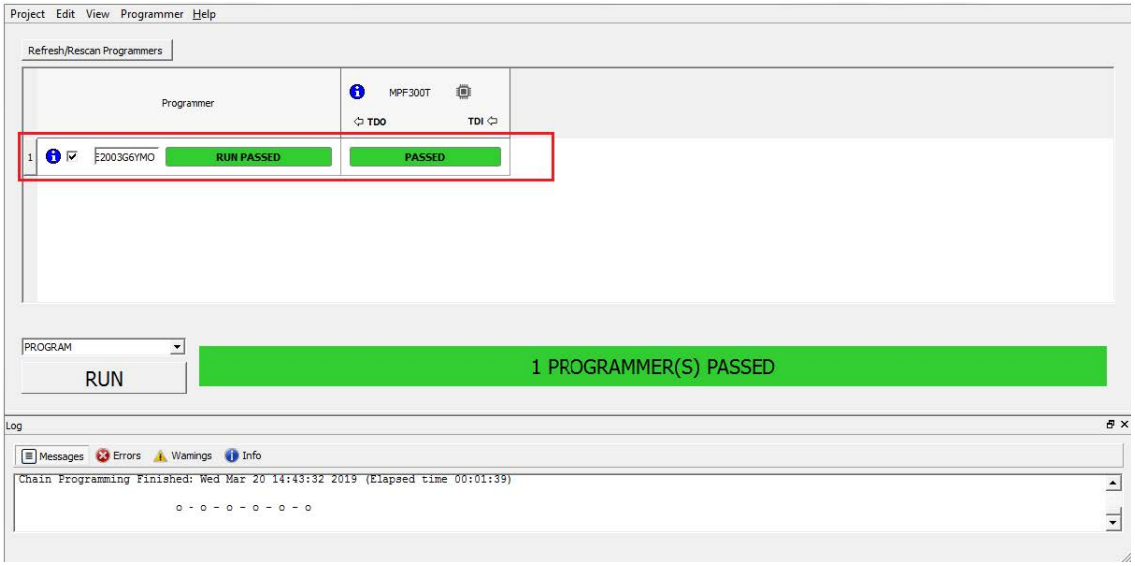
4. Click **OK**. The required programming file is selected and ready to be programmed in the device.
5. The FlashPro Express window appears as shown in Figure 23, page 19. Confirm that a programmer number appears in the Programmer field. If it does not, confirm the board connections and click **Refresh/Rescan Programmers**.

Figure 23 • **Programming the Device**



6. Click **RUN** to program the device. When the device is programmed successfully, a RUN PASSED status is displayed as shown in Figure 24, page 19. See Running the Demo, page 10.

Figure 24 • **FlashPro Express—RUN PASSED**



7. Close FlashPro Express (**Project > Exit**).

The PolarFire device is programmed.

7 Appendix 2: Running the TCL Script

TCL scripts are provided in the design files folder under directory TCL_Scripts. If required, the design flow can be reproduced from Design Implementation till generation of job file.

To run the TCL, follow the steps below:

1. Launch the Libero software
2. Select **Project > Execute Script....**
3. Click Browse and select `script.tcl` from the downloaded TCL_Scripts directory.
4. Click **Run**.

After successful execution of TCL script, Libero project is created within TCL_Scripts directory.

For more information about TCL scripts, refer to **mpf_ac491_df/TCL_Scripts/readme.txt**

Refer to *Libero® SoC TCL Command Reference Guide* for more details on TCL commands. Contact Technical Support for any queries encountered when running the TCL script.