

HB0764

**CoreJTAGDebug v3.1 Handbook
May 2019**



a  **MICROCHIP** company



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

©2019 Microsemi, a wholly owned subsidiary of Microchip Technology Inc. All rights reserved. Microsemi and the Microsemi logo are registered trademarks of Microsemi Corporation. All other trademarks and service marks are the property of their respective owners.

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

About Microsemi

Microsemi, 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.

Contents

1	Revision History	1
1.1	Revision 3.0	1
1.2	Revision 2.0	1
1.3	Revision 1.0	1
2	Introduction	2
2.1	Overview	2
2.2	Features	2
2.3	Core Version	2
2.4	Supported Families	2
2.5	Device Utilization and Performance	3
3	Functional Description	4
3.1	Device Chaining	6
3.1.1	Through FlashPro Header	6
3.1.2	Through GPIO	7
4	Interface	10
4.1	Configuration Parameters	10
4.1.1	Signal Descriptions	11
5	Register Map and Descriptions	12
6	Tool Flow	13
6.1	License	13
6.1.1	RTL	13
6.1.2	SmartDesign	13
6.2	Configuring CoreJTAGDebug in SmartDesign	14
6.3	Simulation Flows	15
6.4	Synthesis in Libero	15
6.5	Place-and-Route in Libero	15
7	System Integration	16
7.1	System Level Design	16
7.1.1	IGLOO2 / RTG4	16
7.2	SmartFusion2	17
8	Design Constraints	18

Figures

Figure 1	CoreJTAGDebug Block Diagram	4
Figure 2	Tunnel Packet Protocol	5
Figure 3	CoreJTAGDebug Serial Data and Clocking	5
Figure 4	Multiple Processors in a Single Device	6
Figure 5	Multiple Processors Across Two Devices	6
Figure 6	Debugger Configuration UJ_JTAG_IRCODE	7
Figure 7	Debugger Configuration GPIO	7
Figure 8	Debugging over GPIO pins	7
Figure 9	Device Chaining through GPIO pins	8
Figure 10	Debug Configuration	8
Figure 11	MIV Configuration File	8
Figure 12	Example Debug System	9
Figure 13	SmartDesign CoreJTAGDebug Instance View using JTAG Header	13
Figure 14	SmartDesign CoreJTAGDebug Instance using GPIO pins	13
Figure 15	Configuring CoreJTAGDebug in SmartDesign	14
Figure 16	RTG4/IGLOO2 JTAG Debug Design	16
Figure 17	SmartFusion2 JTAG Debug Design	17

Tables

Table 1	Device Utilization and Performance	3
Table 2	CoreJTAGDebug Configuration Options	10
Table 3	CoreJTAGDebug I/O Signals	11

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 3.0

Updated for CoreJTAGDebug v3.1 release.

The major changes made in this release are:

- Added BUFD macros to improve performance and Place-And-Route (PAR) timing for PolarFire family.
- Removed CLKINT macros around uj_jtag module since these are now added automatically in Libero SoC v12.1.

1.2 Revision 2.0

Updated for CoreJTAGDebug v3.0 release.

The major changes made in this release are:

- Support added for multiple JTAG devices through the FlashPro header and through GPIO. For more information about Device Chaining, refer to Functional Description section.
- Support added for multiple UJ_JTAG devices.

1.3 Revision 1.0

Revision 1.0 is released for CoreJTAGDebug v2.0.

2 Introduction

2.1 Overview

CoreJTAGDebug v3.1 facilitates the connection of JTAG (Joint Test Action Group) compatible soft core processors to the JTAG header or GPIO pins for debugging. This core processors facilitates the debugging of maximum 16 soft core processors within a single device, and also provides the debugging support of four devices over GPIO.

2.2 Features

Following are the key features of CoreJTAGDebug:

- Provides the fabric access to the JTAG interface through the JTAG header.
- Configures the IR Code support for the JTAG tunneling.
- Supports the linking of multiple devices through the dedicated JTAG header.
- Supports the multi-processor debugging.
- Promotes the s clock and reset signals to the low-skew routing resources.
- Supports active-high target resetting.
- Provides the fabric access to the JTAG interfaces through the GPIO pins.
- Supports the multiple UJTAG devices.

2.3 Core Version

This Handbook applies to CoreJTAGDebug v3.1.

2.4 Supported Families

- PolarFire®
- RTG4™
- IGLOO®2
- SmartFusion®2
- SmartFusion
- ProASIC3/3E/3L
- IGLOO
- IGLOOe/+

2.5 Device Utilization and Performance

Utilization and performance data is listed in Table 1 for the supported device families. The data listed in this table is only indicative. The overall device utilization and performance of the core is system dependent.

Table 1 • Device Utilization and Performance

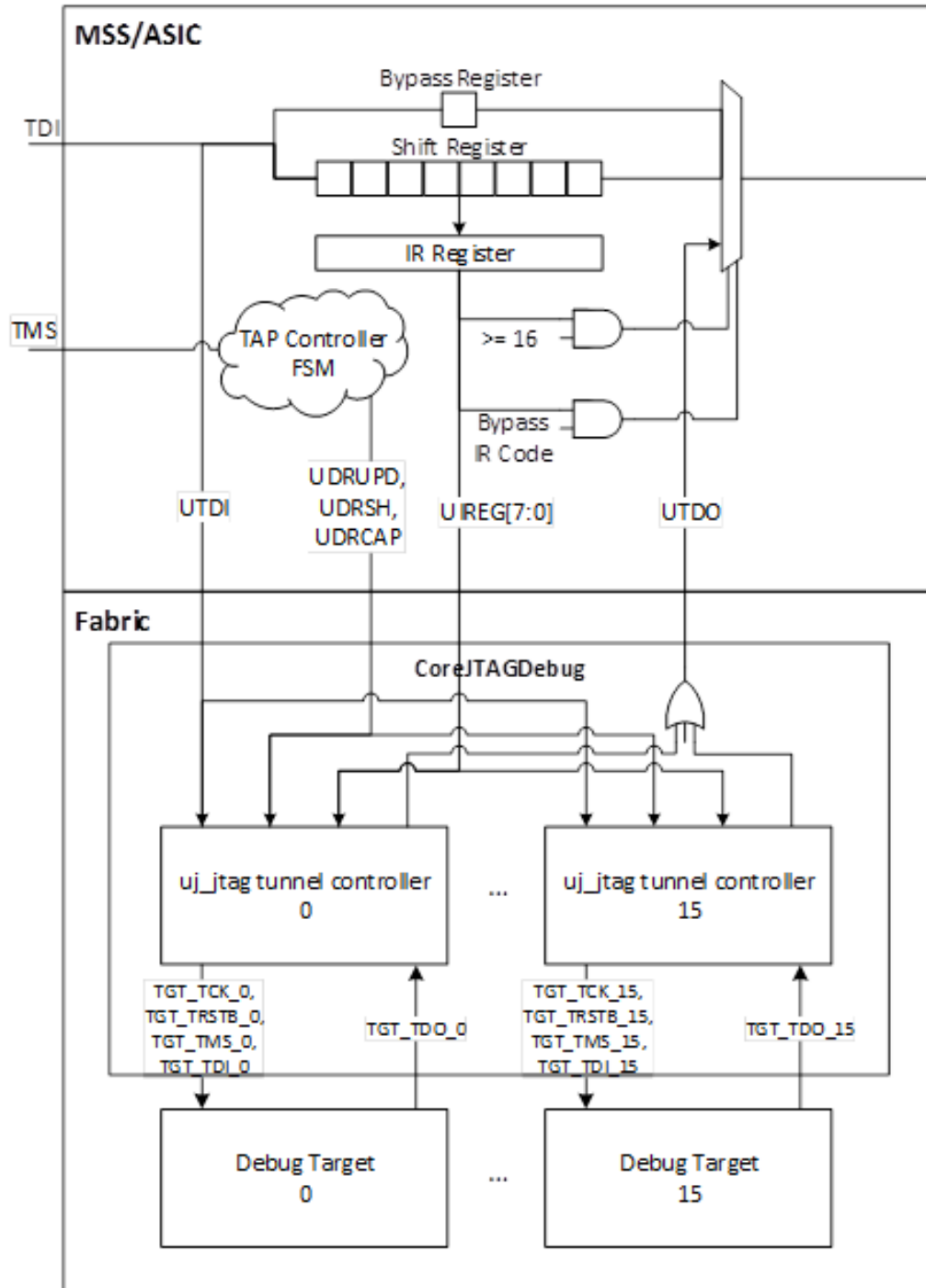
Family	Tiles			Utilization		Performance (MHz)
	Sequential	Combinatorial	Total	Device	Total %	
PolarFire	17	116	299554	MPF300TS	0.04	111.111
RTG4	19	121	151824	RT4G150	0.09	50.00
SmartFusion2	17	120	56340	M2S050	0.24	69.47
IGLOO2	17	120	56340	M2GL050	0.24	68.76
SmartFusion	17	151	4608	A2F200M3F	3.65	63.53
IGLOO	17	172	3072	AFL125V5	6.15	69.34
ProASIC3	17	157	13824	A3P600	1.26	50.00

Note: Data in this table was achieved using the Verilog RTL with typical synthesis and layout settings on -1 parts. Top-level parameters or generics were left at default settings.

3 Functional Description

CoreJTAGDebug uses the UJTAG hard macro to provide fabric access to the JTAG interface. The UJTAG hard macro facilitates connecting to the output of the MSS or ASIC TAP controller from the fabric. Only, one instance of the UJTAG macro is allowed in the fabric.

Figure 1 • CoreJTAGDebug Block Diagram



CoreJTAGDebug contains an instantiation of the `uj_jtag` tunnel controller, which implements a JTAG tunnel controller to facilitate JTAG tunneling between a FlashPro programmer and a target softcore processor. The softcore processor is connected through the dedicated JTAG header pins. IR scans from the JTAG interface are inaccessible in the FPGA fabric. Hence, the tunnel protocol is required to facilitate IR and DR scans to the debug target, which supports the industry standard JTAG interface. The tunnel controller decodes the tunnel packet transferred as a DR scan and generates a resultant IR or DR scan, based on the contents of the tunnel packet and the contents of the IR register provided through UIREG. The tunnel controller also decodes the tunnel packet, when the contents of the IR register matches its IR code.

Figure 2 • Tunnel Packet Protocol

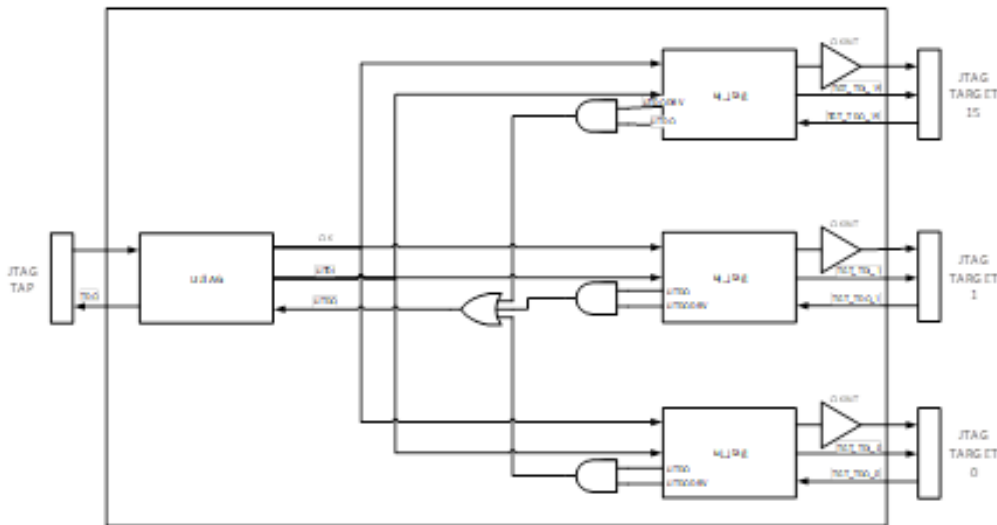
Entry TMS Length [2:0]	Entry TMS Data (0-7)	Payload Data Length [5:0]	Payload Data Data (0-63)	Exit TMS Length [2:0]	Exit TMS Data (0-7)
------------------------	----------------------	---------------------------	--------------------------	-----------------------	---------------------

A configuration parameter provides configuration of the IR code used by the tunnel controller. To facilitate the debugging of multiple softcore processors inside a single design, the number of tunnel controllers instantiated are configurable in the range 1-16, providing a JTAG compliant interface to each target processor. These target processors are each addressable through a unique IR code set at instantiation time.

A CLKINT or BFR buffer is instantiated on the TGT_TCK line of each target processor debug interface.

The URSTB line from the UJTAG macro (TRSTB) is promoted to a global resource within CoreJTAGDebug. An optional inverter is placed on the TGT_TRST line within CoreJTAGDebug for connection to a debug target, which is then expected to be connected to an active-high reset source. It is configured when it is assumed that the incoming TRSTB signal from the JTAG TAP is active low. If this configuration requires one or more debug targets, an additional global routing resource will be consumed.

Figure 3 • CoreJTAGDebug Serial Data and Clocking



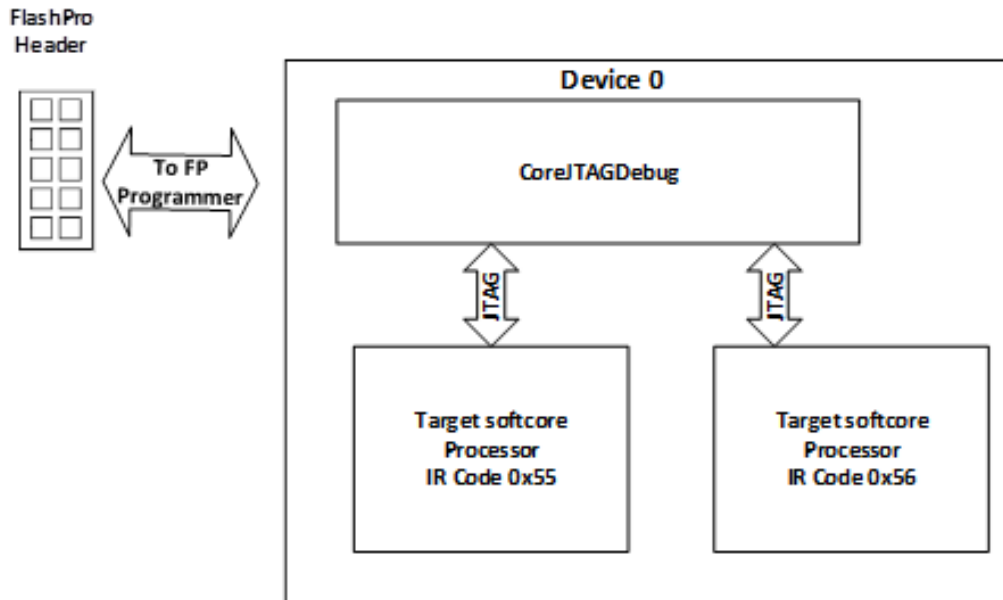
3.1 Device Chaining

Read the **FPGA Programming User Guides** for the specific development board or family. Each development board may operate at different voltages, and you may choose to verify if it is possible with their development platforms. Also, if you are using multiple development boards, ensure that, they share a common ground.

3.1.1 Through FlashPro Header

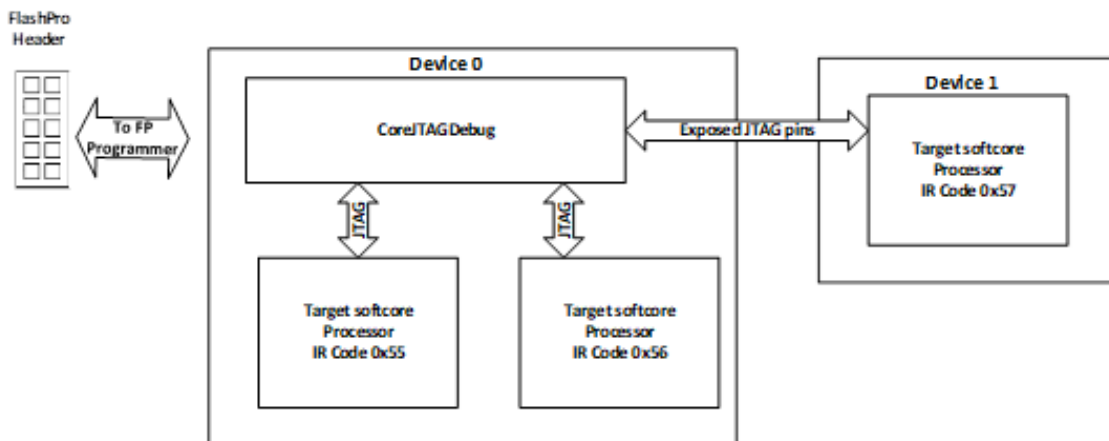
To support the chaining of multiple devices in the Fabric using the FlashPro header, multiple instances of `uj_jtag` are required. This version of the core provides users with access of maximum 16 cores without the need for manually instantiating `uj_jtag`. Each core has a unique IR Code (from 0x55 to 0x64) that will provide access to the specific core matching the ID code.

Figure 4 • Multiple Processors in a Single Device



To use CoreJTAGDebug across multiple devices, one of the devices needs to become the master. This device contains the CoreJTAGDebug core. Each processor is then connected as follows:

Figure 5 • Multiple Processors Across Two Devices

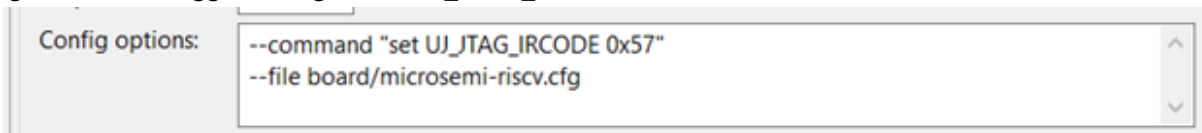


To debug a core on another board, the JTAG signals from CoreJTAGDebug are promoted to top level pins in the SmartDesign. These are then connected to the JTAG signals directly on the processor. A CoreJTAGDebug, in the second board design, is optional. Note that the `UJ_JTAG` macro and the FlashPro header are unused in the second board design.

To select a processor for debugging in SoftConsole, click the debug configurations, and then click the **Debugger** tab.

The following command is used.

Figure 6 • Debugger Configuration UJ_JTAG_IRCODE

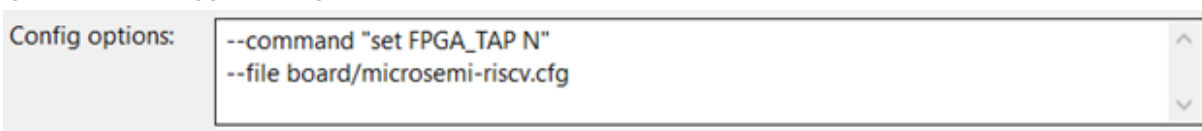


The UJ_JTAG_IRCODE can be changed depending on which processor you are debugging. For example: to debug a processor in Device 0, the UJ_JTAG_IRCODE can be set to **0x55** or **0x56**.

3.1.2 Through GPIO

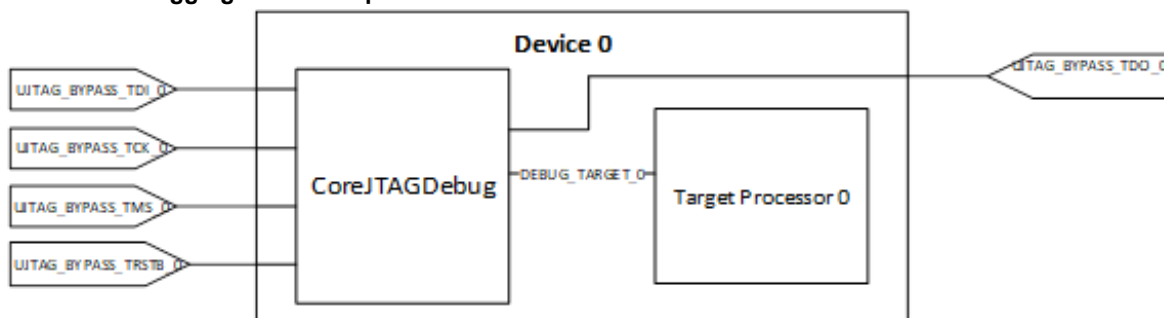
To debug over GPIO, the parameter **UJTAG_BYPASS** is selected. One and four cores can be debugged over GPIO headers or pins. To run a debug session using GPIOs from SoftConsole v5.3 or higher, the **Debug Configuration** must be set up as follows:

Figure 7 • Debugger Configuration GPIO



Note: If you are debugging over GPIO, you cannot concurrently debug the processor through the FlashPro Header or the Embedded FlashPro5, on the development boards. For example: FlashPro Header or Embedded FlashPro 5 are available to facilitate debug using Identify or SmartDebug.

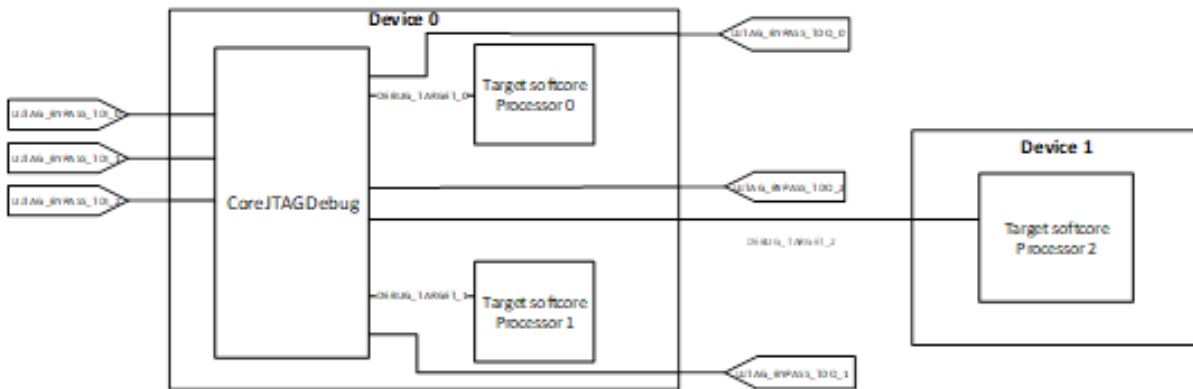
Figure 8 • Debugging over GPIO pins



Device Chaining via GPIO Pins

To support the chaining of multiple devices through GPIO, the **BYPASS_UJTAG** parameter needs to be selected. Then the TCK, TMS, and TRSTb signals can be promoted to top level ports. All target processors have TCK, TMS, and TRSTb. These are not shown below.

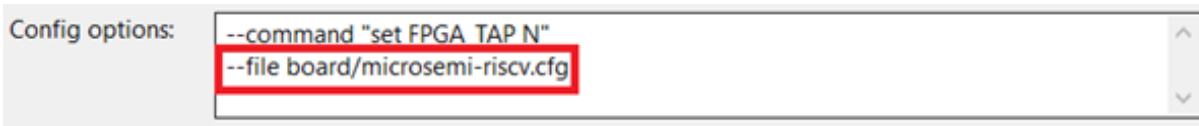
Figure 9 • Device Chaining through GPIO pins



In a basic JTAG chain, the TDO of a processor connects to the TDI of another processor, and it continues until all processors are chained, in this manner. The TDI of the first processor and the TDO of the last processor connects to the JTAG programmer chaining all the processors. The JTAG signals from the processors are routed to CoreJTAGDebug, where they can be chained. If the chaining across multiple devices is completed, the device with CoreJTAGDebug becomes the master device.

In a GPIO debug scenario, where an IRCODE is unallocated to each processor, a modified OpenOCD script is used to select, which device is being debugged. An OpenOCD script is modified to select, which device is debugged. For an Mi-V design, the file is found in the SoftConsole install location, under the openocd/scripts/board/microsemi-riscv.cfg. For the other processors, the files are found in the same openocd location. Note that, the Debug Configuration options also needs to be updated, if the file is renamed.

Figure 10 • Debug Configuration



Open username-riscv-gpio-chain.cfg, following is an example of what must be seen:

Figure 11 • MIV Configuration File

```

#-----
# Microsemi RISC-V board
#-----

# FlashPro
source [find interface/microsemi-flashpro.cfg]

# Device
source [find target/microsemi-riscv.cfg]

# Board specific initialization
proc do_board_reset_init () {
}
    
```

The following settings works for a single device debugging over GPIO. For debugging a chain, additional commands need to be added,so that the devicesthat are not debugged are put in the bypass mode.

For the MIV core, the following command is used:

```
jtag newtap IGNORE tap -irlen 5 -expected-id 0 -ignore-version
```

For a two processor in a chain, the following command in the example shown is used:

```
# Device
jtag newtap IGNORE tap -irlen 5 -expected-id 0 -ignore-version
source [find target/microsemi-riscv.cfg]
```

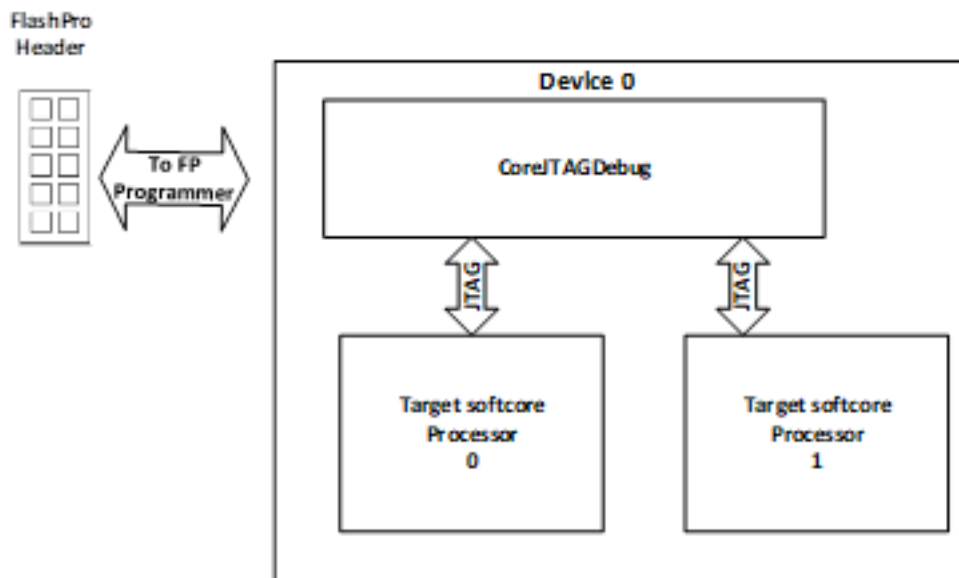
This allows debugging of Target softcore Processor 1 by putting Target softcore Processor 0 into the bypass mode. To debug the Target softcore Processor 0, the following command is used:

```
# Device
source [find target/microsemi-riscv.cfg]
jtag newtap IGNORE tap -irlen 5 -expected-id 0 -ignore-version
```

Note: The only difference between these two configurations is that the source, which is calling the Microsemi RISCv configuration file (microsemi-riscv.cfg) either comes first, when debugging Target softcore Processor 0, or second, when debugging Target Softcore Processor 1. For more than two devices in the chain, additional “jtag newtaps” is added. For example, if there are three processors in a chain, then the following command is used:

```
# Device
source [find target/microsemi-riscv.cfg]
jtag newtap IGNORE tap -irlen 5 -expected-id 0 -ignore-version
jtag newtap IGNORE tap -irlen 5 -expected-id 0 -ignore-version
```

Figure 12 • Example Debug System



4 Interface

4.1 Configuration Parameters

The configurable options applies to CoreJTAGDebug are in Table 2. If a configuration except default is required, use the **Configuration** dialog box in SmartDesign, to select the appropriate values for the configurable options.

Table 2 • CoreJTAGDebug Configuration Options

Name	Valid Range	Default	Description
NUM_DEBUG_TGTS	1-16	1	The number of available debug targets through Flash Pro (UJTAG_DEBUG = 0) is 1 -16. The number of available debug targets through GPIO (UJTAG_DEBUG = 1) is 1 -4.
IR_CODE_TGT_x	0X55-0X64	0X55	JTAG IR Code, one per debug target. The value specified must be unique to this debug target. The tunnel controller associated with this debug target interface only drives TDO and drives the target debug interface, when the contents of the IR register matches this IR code.
TGT_ACTIVE_HIGH _RESET_x	0-1	0	0: TGT_TRST_x output is connected to a global form of the active-low URSTB output of the UJTAG macro. 1: TGT_TRST output is internally connected to a global inverted form of the active-low URSTB output of the UJTAG macro. An extra global routing resource is consumed if this parameter is set to 1 for any debug target.
UJTAG_BYPASS	0-1	0	0: GPIO Debug is disabled, Debug is available through the FlashPro Header or Embedded FlashPro5. 1: GPIO Debug is enabled, Debug is available through auser selected GPIO pins on the board. Note: When you are Debugging through GPIO, the following debug command is used: “—command “set FPGA_TAP N”” must be used in SoftConsole debug options.

4.1.1 Signal Descriptions

Signal descriptions for CoreJTAGDebug are listed in Table 3.

Table 3 • CoreJTAGDebug I/O Signals

Port Name	Width	Direction	Description
JTAG TAG Ports			
TDI	1	Input	Test Data In. Serial data input from TAP.
TCK	1	Input	Test Clock. Clock source to all sequential elements within CoreJTAGDebug.
TMS	1	Input	Test Mode Select.
TDO	1	Output	Test Data out. Serial data output to TAP.
TRSTB	1	Input	Test Reset. Active low reset input from TAP.
JTAG Target X Ports			
TGT_TDO_x	1	Input	Test data out from debug target x to the TAP. Connect to the target TDO port.
TGT_TCK_x	1	Output	Test Clock output to debug target x. TCK is promoted to a global, low skew net internally within CoreJTAGDebug.
TGT_TRSTB_x	1	Output	Test Reset. Configurable active-high or active-low reset output to debug target x.
TGT_TMS_x	1	Output	Test Mode Select output to debug target x.
TGT_TDI_x	1	Output	Test Data In. Serial data input from debug target x.
UJTAG_BYPASS_TCK_x	1	Input	Test Clock input to debug target x from GPIO pin.
UJTAG_BYPASS_TMS_x	1	Input	Test Mode Select to debug target x from GPIO pin.
UJTAG_BYPASS_TDI_x	1	Input	Test Data In, Serial data to debug target x from GPIO pin.
UJTAG_BYPASS_TRSTB_x	1	Input	Test Reset. Reset input to debug target x from GPIO pin.
UJTAG_BYPASS_TDO_x	1	Output	Test Data Out, Serial data from debug target x from GPIO pin.

Note: All signals in the JTAG TAP ports list above must be promoted to top-level ports in SmartDesign.

5 Register Map and Descriptions

There are no registers for CoreJTAGDebug.

6 Tool Flow

6.1 License

A license is not required to use this IP Core with Libero SOC.

6.1.1 RTL

Complete RTL code is provided for the core and test benches, allowing the core to be instantiated with SmartDesign. Simulation, Synthesis, and Layout can be performed within Libero SoC.

6.1.2 SmartDesign

An example instantiated view of CoreJTAGDebug is shown in Figure 13.

For more information on using SmartDesign to instantiate and generate cores, refer to the **Using DirectCore in Libero® SoC User Guide**.

Figure 13 • SmartDesign CoreJTAGDebug Instance View using JTAG Header

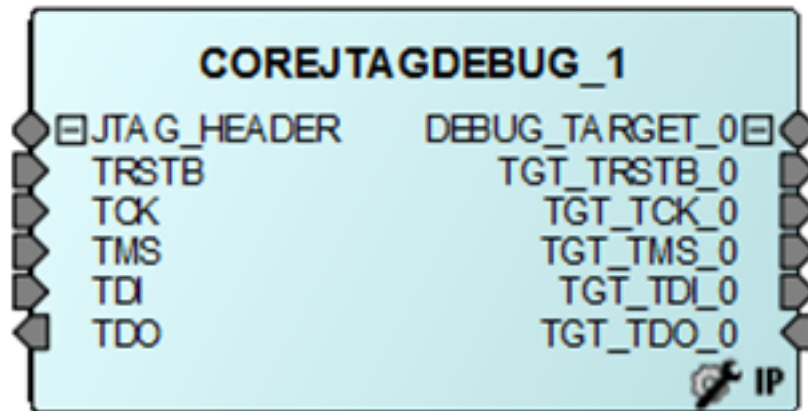
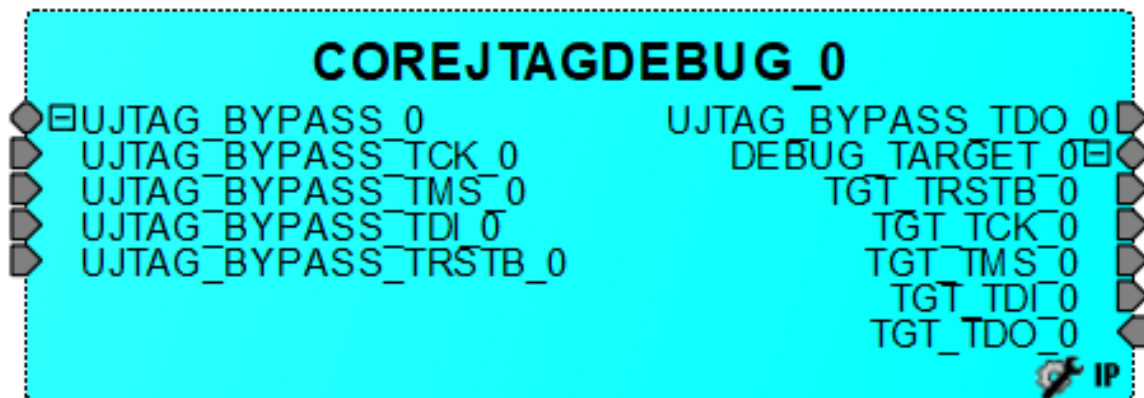


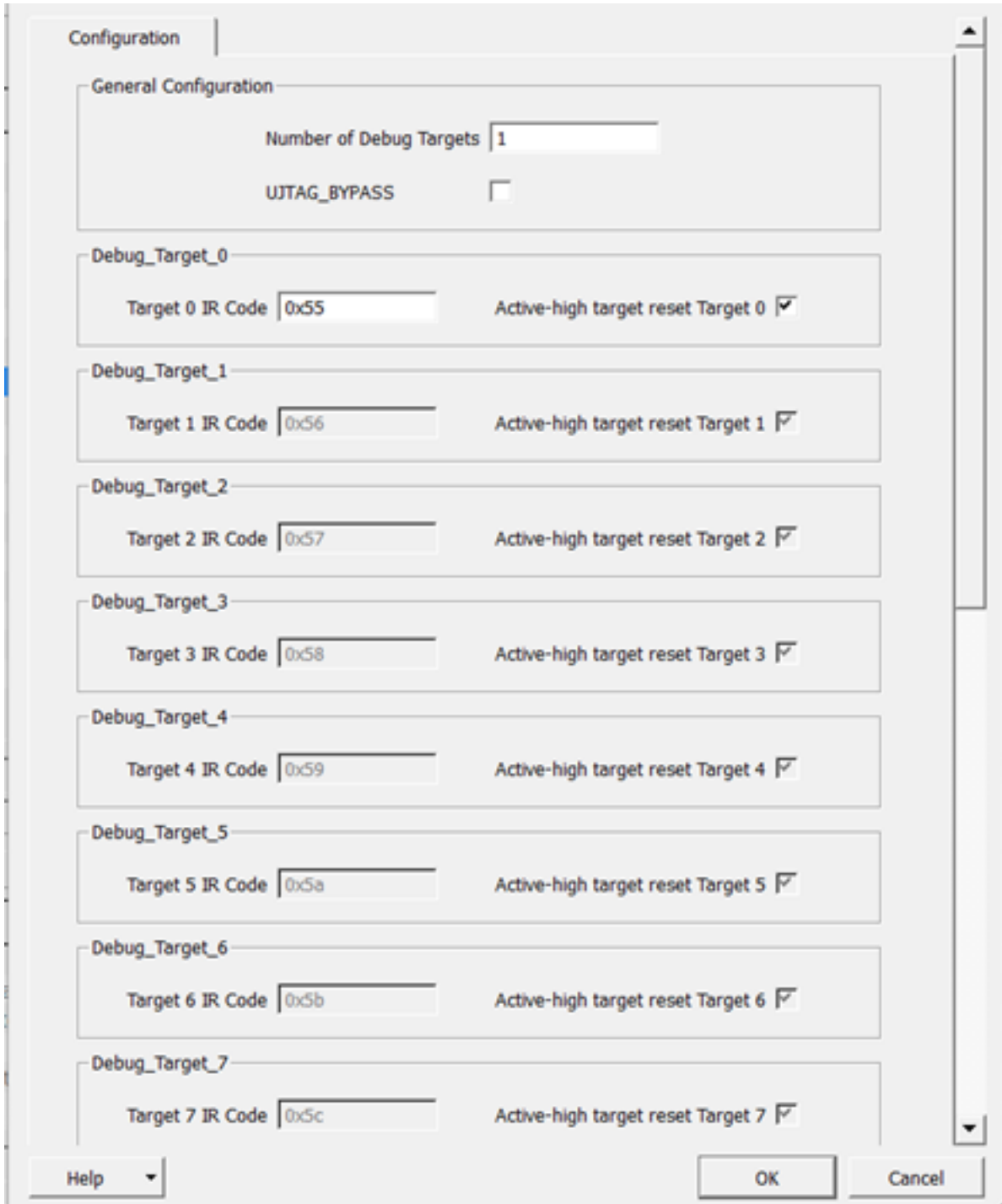
Figure 14 • SmartDesign CoreJTAGDebug Instance using GPIO pins



6.2 Configuring CoreJTAGDebug in SmartDesign

The core is configured using the configuration GUI in SmartDesign. An example of the GUI is shown in Figure 15.

Figure 15 • Configuring CoreJTAGDebug in SmartDesign



The screenshot shows the 'Configuration' dialog box for CoreJTAGDebug. It is divided into several sections:

- General Configuration:**
 - Number of Debug Targets: 1
 - UJTAG_BYPASS:
- Debug_Target_0:** Target 0 IR Code: 0x55, Active-high target reset Target 0:
- Debug_Target_1:** Target 1 IR Code: 0x56, Active-high target reset Target 1:
- Debug_Target_2:** Target 2 IR Code: 0x57, Active-high target reset Target 2:
- Debug_Target_3:** Target 3 IR Code: 0x58, Active-high target reset Target 3:
- Debug_Target_4:** Target 4 IR Code: 0x59, Active-high target reset Target 4:
- Debug_Target_5:** Target 5 IR Code: 0x5a, Active-high target reset Target 5:
- Debug_Target_6:** Target 6 IR Code: 0x5b, Active-high target reset Target 6:
- Debug_Target_7:** Target 7 IR Code: 0x5c, Active-high target reset Target 7:

At the bottom of the dialog are three buttons: 'Help', 'OK', and 'Cancel'.

The **Number of Debug Targets** is configurable up to 16 debug targets, with **UJTAG_BYPASS** disabled and up to 4 debug targets, with **UJTAG_BYPASS** enabled.

UJTAG_BYPASS selects debugging through UJTAG and the FlashPro header, and debugging through GPIO pins.

The Target # IR Code is the JTAG IR Code given to the debug target. This must be a unique value within the range specified in Table 2.

6.3 Simulation Flows

A user testbench is provided with CoreJTAGDebug.

To run simulations:

1. Select the user testbench flow within the SmartDesign.
2. Click **Save and Generate** in the **Generate** pane. Select the **user testbench** from the **Core Configuration** GUI.

When SmartDesign generates the Libero project, it installs the user testbench files.

To run the user testbench:

1. Set the design root to the CoreJTAGDebug instantiation in the **Libero design** hierarchy pane.
2. Click **Verify Pre-Synthesized Design ->Simulate** in the **Libero Design Flow** window. This starts ModelSim and automatically runs the simulation.

6.4 Synthesis in Libero

To run Synthesis:

1. Click the **Synthesize** icon in the **Libero SoC Design Flow** window to synthesize the core. Alternatively, right-click on the Synthesize option in the Design Flow window, and select **Open Interactively**. The **Synthesis** window displays the Synplify® project.
2. Click the **Run** icon.

6.5 Place-and-Route in Libero

Once the synthesis stage is completed, click the **Place and Route** icon in Libero SoC to start the placement process.

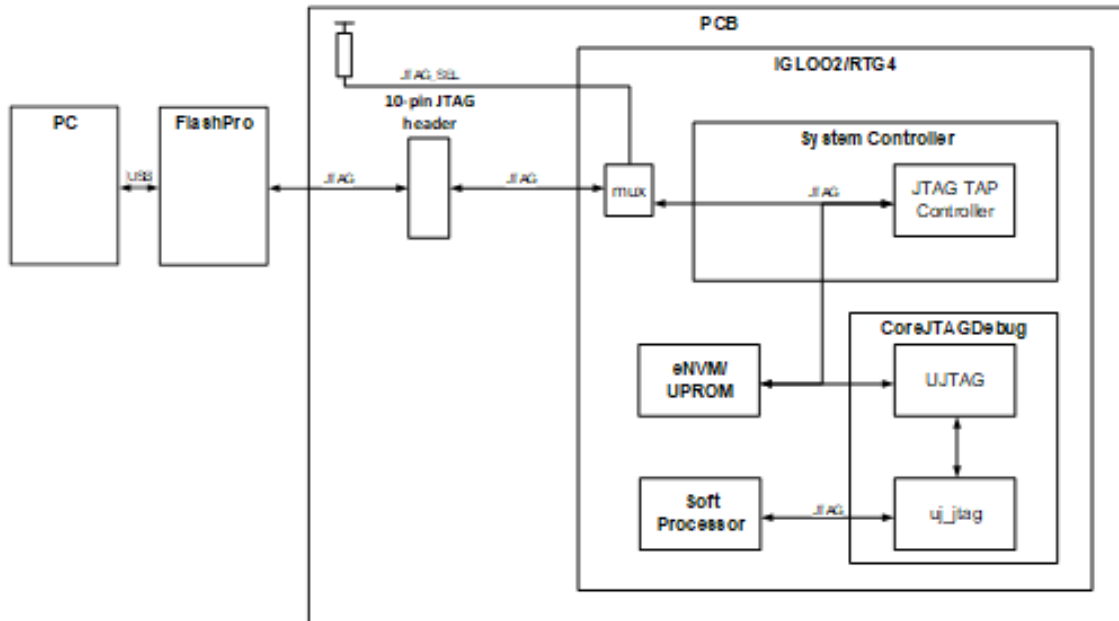
7 System Integration

7.1 System Level Design

7.1.1 IGLOO2 / RTG4

The Figure 16 shows the design requirements to perform JTAG debugging of a softcore processor, located in the fabric from SoftConsole to the JTAG interface for IGLOO2 and RTG4 devices.

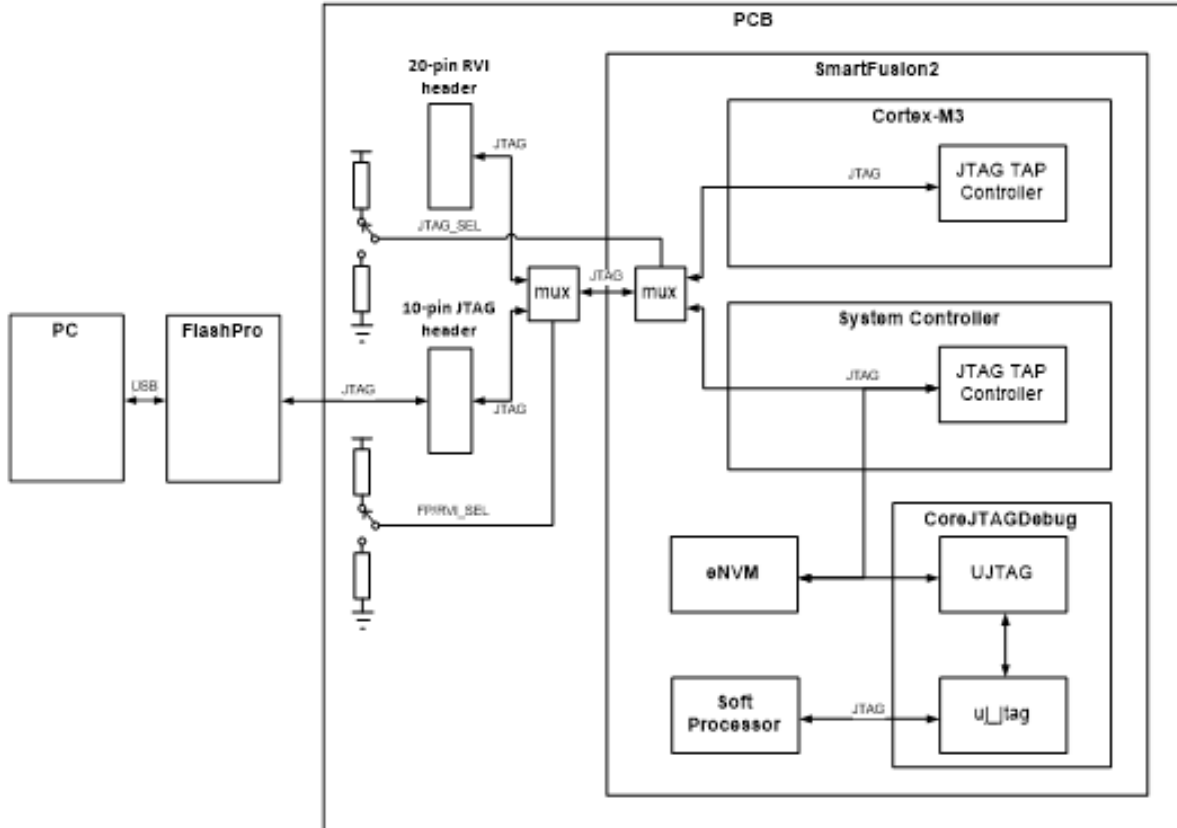
Figure 16 • RTG4/IGLOO2 JTAG Debug Design



7.2 SmartFusion2

The Figure 17 shows the design requirements to perform JTAG debugging of a softcore processor, located in fabric from SoftConsole to the JTAG interface for SmartFusion2 devices.

Figure 17 • SmartFusion2 JTAG Debug Design



8 Design Constraints

The designs with CoreJTAGDebug require the application to follow the constraints, in the design flow, for allowing timing analysis to be used on the TCK clock domain.

To add the constraints:

1. If the Enhanced Constraint flow in Libero v11.7 or higher is used, double-click **Constraints > Manage Constraints** in the **Design Flow** window and click **Timing** tab.
2. In the **Timing** tab of the **Constraint Manager** window, click **New** to create a new SDC file, and name the file. The Design constraints include the clock source constraints that can be entered in this blank SDC file.
3. If the Classic Constraint flows in Libero v11.7 or higher is used, right-click **Create Constraints > Timing Constraint**, in the **Design Flow** window, and then click **Create New Constraint**. It creates a new SDC file. The design constraints includes the clock source constraints, which is entered in this blank SDC file.
4. Calculate the TCK period and half period. TCK is set to 6 MHz when debugging is done with FlashPro, and is set to a maximum frequency of 30 MHz when debugging is supported by FlashPro5. After you have completed this step, enter the following constraints in the SDC file:

```
create_clock -name { TCK } \
    -period TCK_PERIOD \
    -waveform { 0 TCK_HALF_PERIOD } \
    [ get_ports { TCK } ]
```

For example, the following constraints is applied for a design that uses a TCK frequency of 6 MHz:

```
create_clock -name { TCK } \
    -period 166.67 \
    -waveform { 0 83.33 } \
    [ get_ports { TCK } ]
```

5. Associate all the constraints files with the Synthesis, Place-and-Route, and Timing Verification stages in the **Constraint Manager > Timing** tab. This is completed by selecting the related check boxes for the SDC files in which the constraints were entered in.