Loading and Debugging Core8051s Application From External Flash Memory

Table of Contents

Purpose

This application note describes how to load and debug application code from external flash memory available on the Microsemi® Cortex-M1-enabled ProASIC3L Development Kit.

Introduction

A Core8051s based microcontroller system is implemented on the Microsemi M1 enabled ProASIC3L field programmable gate array (FPGA). The external flash memory is interfaced to the Core8051s microcontroller system to load and debug the application code.

References

The following references are used in this document:

- *Core8051s Based Hardware Tutorial*
- *Core8051s Based Software User Guide*

Design Requirements

Table 1 • **Design Requirements**

Design Requirements	Description
Hardware Requirements	
Cortex-M1-enabled ProASIC3L Development Kit	-
Host PC or Laptop	Any 64-bit Windows Operating System
Software Requirements	
Libero® System-on-Chip (SoC)	v11.3
SoftConsole	v3.4
One of the following serial terminal emulation programs: • HyperTerminal • TeraTerm • PuTTY	-

Design Overview

A Core8051s IP based microcontroller system is developed with peripheral IPs such as CoreGPIO, CoreUARTapb, CoreWatchdog, CoreTimer, and CoreAPB3 that are implemented on the *Microsemi Cortex-M1-enabled ProASIC3L Development Kit*. An external *Micron JS28F640J3D-75* flash memory is interfaced to the Core8051s microcontroller system. A simple application is loaded into the external Micron JS28F640J3D-75 flash memory to blink the on-board LEDs. Figure 1 shows the Core8051s microcontroller system.

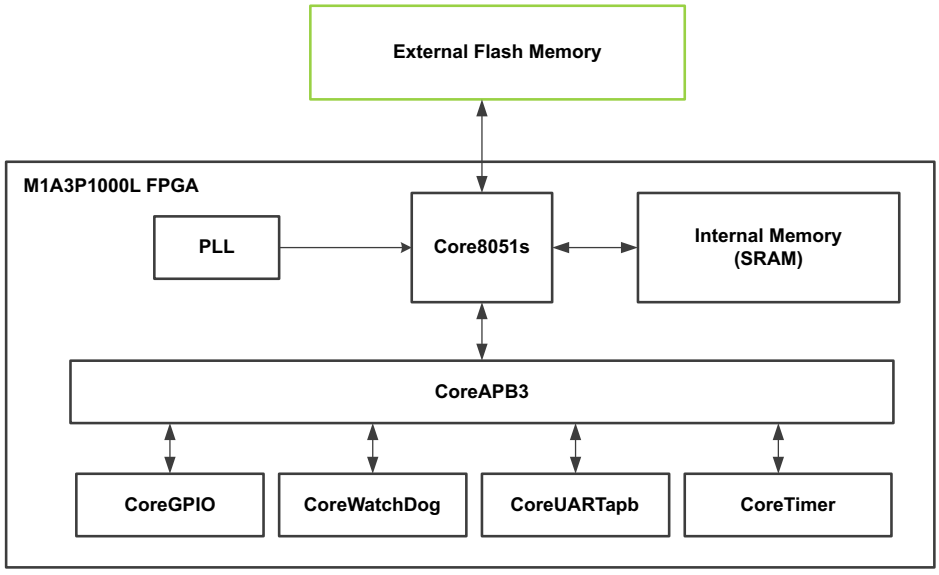

Figure 1 • **Core8051s Microcontroller System**

Design Description

This design example has the following IPs that are available in Libero SoC catalog:

- **Core8051s**: an 8-bit microcontroller IP core
- **CoreGPIO**: provides up to 32-bit inputs and 32-bit outputs for general purpose
- **CoreUARTapb**: a serial communication interface
- **CoreWatchdog**: provides a means of recovering from software crashes
- **CoreTimer**: for interrupt-generation and programmable counter
- **CoreAPB3**: a bus component that provides advanced microcontroller bus architecture (AMBA3) advanced peripheral bus (APB3) fabric supporting up to 16 APB slaves

The following sections provide a brief description of each IP and its configuration:

Core8051s Description

The Core8051s is a high-performance, 8-bit microcontroller IP core. It is an 8-bit embedded controller that executes all ASM51 instructions and has the same instruction set as 80C31. It provides software and hardware interrupts. It eliminates redundant bus states and implements parallel execution of fetch and execution phases. The Core8051s uses one clock per cycle, and most of the one byte instructions are performed in a single clock cycle. Figure 2 shows the Core8051s architecture.

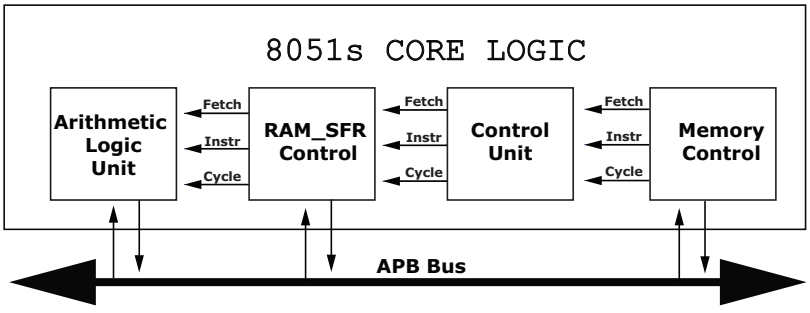

***Figure 2* • Core8051s Architecture**

Difference Between Core8051s and Core8051

The Core8051s is smaller and more flexible than the Core8051.The microcontroller-specific features such as SFR-mapped peripherals, power management circuitry, serial channel, I/O ports and timers of the original 8051 are not present in Core8051s. The Core8051s contains the main 8051 core logic, but it does not have peripheral logic. The Core8051s has an advanced peripheral bus interface that can be used like the SFR (special function register) bus to easily expand the functionality of the core by connecting it to the existing advanced peripheral bus IPs. The Core8051s allows to configure the core

with the peripheral functions (timers, UARTs, I/O ports, etc.) that are required for the application. Configure the Core8051s Configurator GUI as shown in Figure 3.

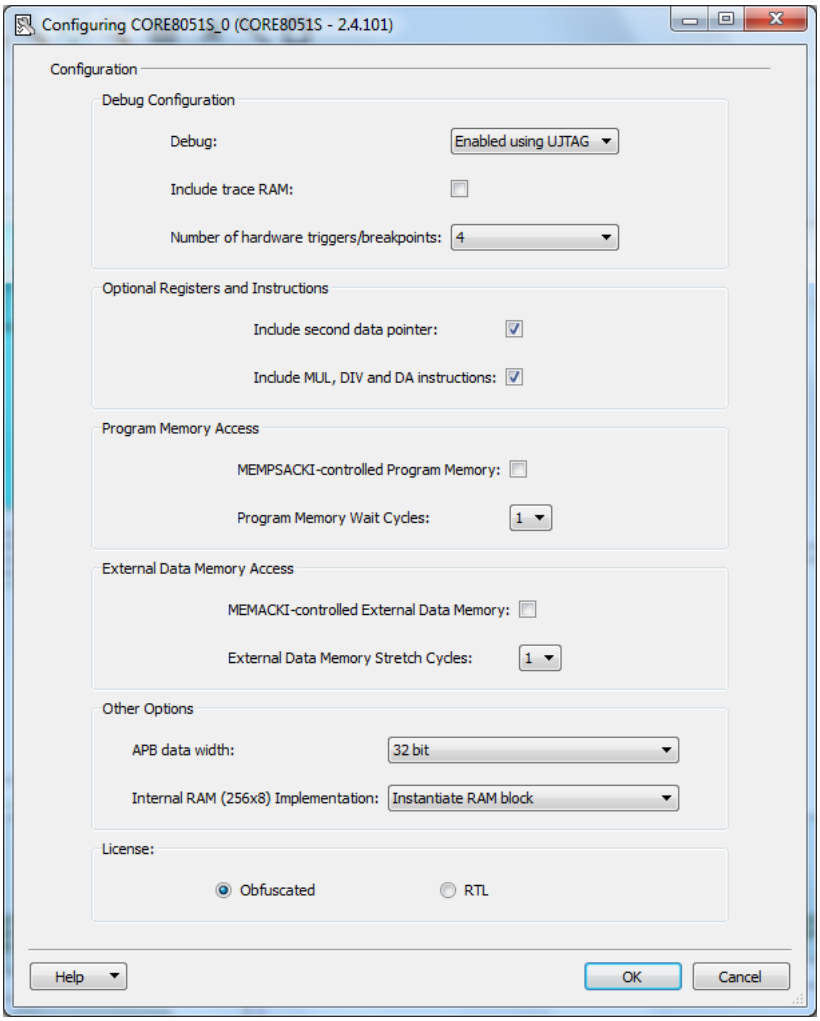

***Figure 3* • Core8051s Configurator GUI**

Refer to the *Core8051s Handbook* for more details.

CoreAPB3 Description

The CoreAPB3 is a bus component that provides advanced microcontroller bus architecture (AMBA3) advanced peripheral bus (APB3) fabric supporting up to 16 APB slaves, and a single APB master. The CoreAPB3 can be used with an APB3 master that does not have a built-in APB address decoding, such as Core8051s. A single APB3 master is connected to CoreAPB3. The master's PSEL and PADDR signals are used within the CoreAPB3 to decode the appropriate PSELS slave select signals, and only one signal can be active at a time. This address decoding depends on the RANGESIZE hardware parameter/generic. Refer to the *CoreAPB3 Handbook* for more information.

Configure the CoreAPB3 Configurator GUI as shown in Figure 4.

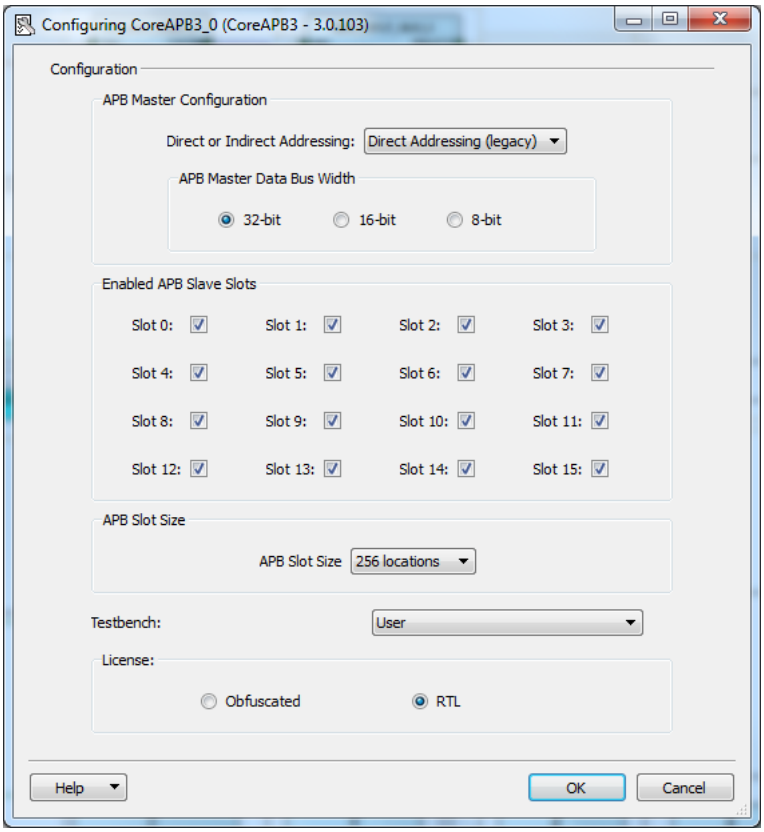

***Figure 4* • CoreAPB3 Configurator GUI**

External Flash Memory Description

Part Number:

- *Micron JS28F640J3D-75*

Architecture:

- 64 Mbit (64 blocks)

Performance:

- 75 ns Initial Access Speed, 25 ns 8-word and 4-word Asynchronous page-mode reads
- 32-Byte Write buffer (4 µs per Byte Effective programming time)

System voltage:

- VCC = 2.7 V to 3.6 V and VCCQ = 2.7 V to 3.6 V

Enhanced security options for code protection:

- 128-bit Protection Register (64-bits unique device identifier bits, 64-bits user-programmable OTP (one time programmable) bits)
- Absolute protection with VPEN = GND
- Individual block locking
- Block erase/program lockout during power transitions

Software:

- Program and erase suspend support
- Flash data integrator (FDI)
- Common flash interface (CFI) compatible

The external flash memory device can be accessed as 8- or 16-bit words. A command user interface (CUI) serves as the interface between the system processor and the internal operation of the device. A valid command sequence written to the CUI that initiates the device automation. An internal write state machine (WSM) automatically executes the algorithms and timings necessary for block erase, program, and lock-bit configuration operations.

Flash operations are command-based, where command codes are first issued to the flash memory, then the flash memory performs the required operation. Refer to the flash memory *Micron JS28F640J3D-75 datasheet* for a list of command codes and flowcharts. Flash memory has a read-only 8-bit status register that indicates the flash memory status and operational errors. Four types of data can be read from the flash memory: array data, device information, CFI data, and device status.

The flash memory is set to Read Array mode by default after power-up or reset. Executing the Read Array command sets the flash memory to Read Array mode and reads the output array data. The flash memory remains in Read Array mode until a different read command is executed. To change the flash memory to Read Array mode while it is programming or erasing, first issue the suspend command. After suspending the operation, run the Read Array command to set to Read Array mode. When the program or erase operation is subsequently resumed, the flash memory automatically sets to Read Status mode.

Issuing the Read Device Information command places the flash memory in Read Device Information mode and reads the output of the device information. The flash memory remains in Read Device Information mode until a different read command is issued. Also, performing a program, erase, or block-lock operation changes the flash memory to Read Status Register mode.

Array programming is performed by first issuing the single-word/byte program command. This is followed by writing the desired data at the desired array address. The read mode of the device is automatically changed to Read Status Register mode, which remains in effect until another read-mode command is issued.

Erasing a block changes zeros to ones. To change ones to zeros, a program operation must be performed. Erasing is performed on a block basis - an entire block is erased each time when an erase command sequence is issued. Once a block is fully erased, all addressable locations within that block read as logical ones (FFFFh). Only one block-erase operation can occur at a time, and it is not allowed during a program suspend. To perform a block-erase operation, issue the block erase command sequence at the required block address. An erase or programming operation can be suspended to perform other operations, and then subsequently resumed. To suspend an on-going erase or a program operation, issue the suspend command to any address.

All blocks are unlocked at the factory. Blocks can be locked individually by issuing the set block lock bit command sequence to any address within a block. Once locked, blocks remain locked when power cable is unplugged or when the device is reset. All locked blocks are unlocked simultaneously by issuing the clear block lock bits command sequence to any device address. The locked blocks cannot be erased or programmed.

The sequence of the commands that must be given to the flash memory are written in an XML file. The XML files are provided with the SoftConsole software for the JS28F640J3D-75 flash memory located at: *C:\Program Files (x86)\Microsemi\SoftConsole v3.4\Sourcery-G++\share\sprite\flash.*

CoreTimer Description

The CoreTimer is an APB slave that provides a functionality for the interrupt generations, and a programmable decrementing counter. It is configurable and programmable, and can be used in either continuous or one-shot modes. It is an essential element in many designs because it supports accurate generation of timing for precise application control. Refer to the *CoreTimer Handbook* for more information. Configure the CoreTimer Configurator GUI as shown in Figure 5.

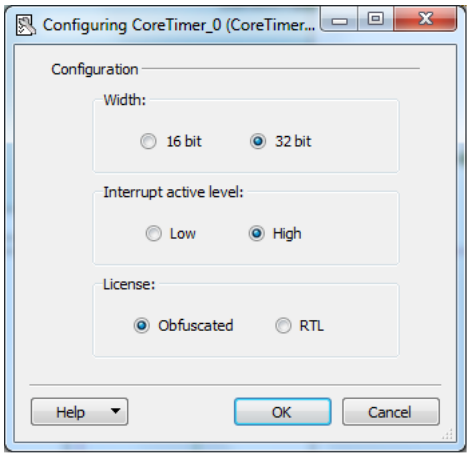

***Figure 5* • CoreTimer Configurator GUI**

CoreWatchdog Description

The CoreWatchdog is an APB slave that provides a means of recovering from software crashes. When the CoreWatchdog is enabled, the core generates a soft reset if the microprocessor fails to refresh it on a regular basis. The CoreWatchdog can be configured based on a decrementing counter, which asserts a reset signal if it is allowed to time out. The width of the decrementing counter can be configured as either 16 or 32-bits. The processor-accessible registers in CoreWatchdog provide a means to control and monitor the operation of the core. Refer to the *CoreWatchdog Handbook* for more information.

Configure the CoreWatchdog Configurator GUI as shown in Figure 6.

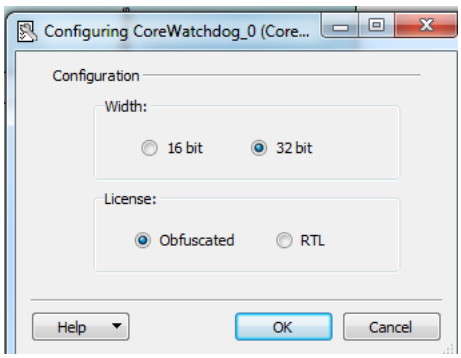

***Figure 6* • CoreWatchdog Configurator GUI**

CoreUARTapb Description

The CoreUARTapb is a serial communications interface that is primarily used in the embedded systems. The controller can operate in either an asynchronous (UART) or a synchronous mode. In asynchronous mode, the CoreUARTapb can be used to interface directly to industry standard UARTs. The CoreUARTapb has an APB-wrapper that adds an APB interface allowing the core to be connected to the APB bus and controlled by an APB bus master. Unlike a standard 8051 UART, the CoreUARTapb includes a baud rate generator and so does not need a separate timer for the baud rate. Refer to the *CoreUARTapb Handbook* for more information.

Configure the CoreUARTapb Configurator GUI as shown in Figure 7.

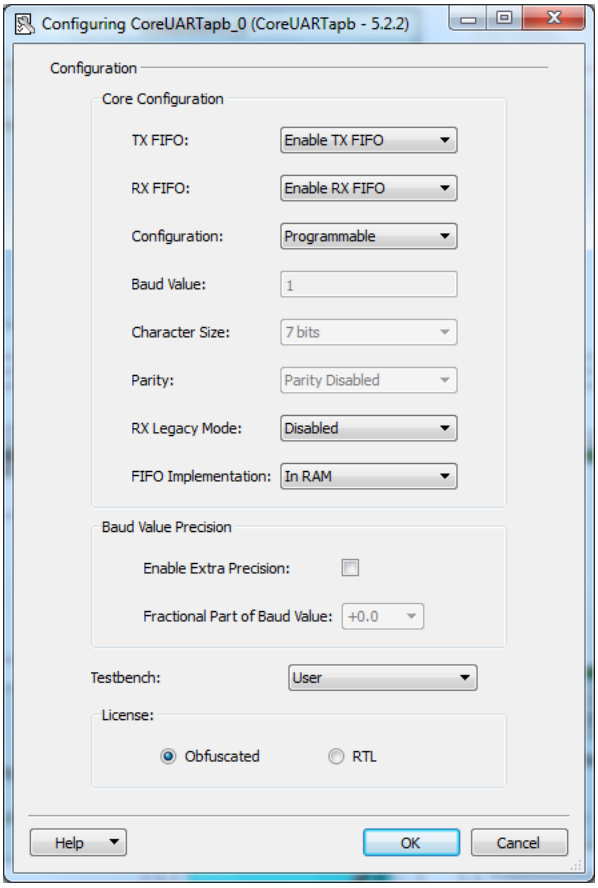

***Figure 7* • CoreUARTapb Configurator GUI**

CoreGPIO Description

The CoreGPIO is an APB bus peripheral that provides up to 32-bit inputs and 32-bit outputs for general purpose. Refer to the *CoreGPIO Handbook* for more information.

Configure the CoreGPIO Configurator GUI as shown in Figure 8.

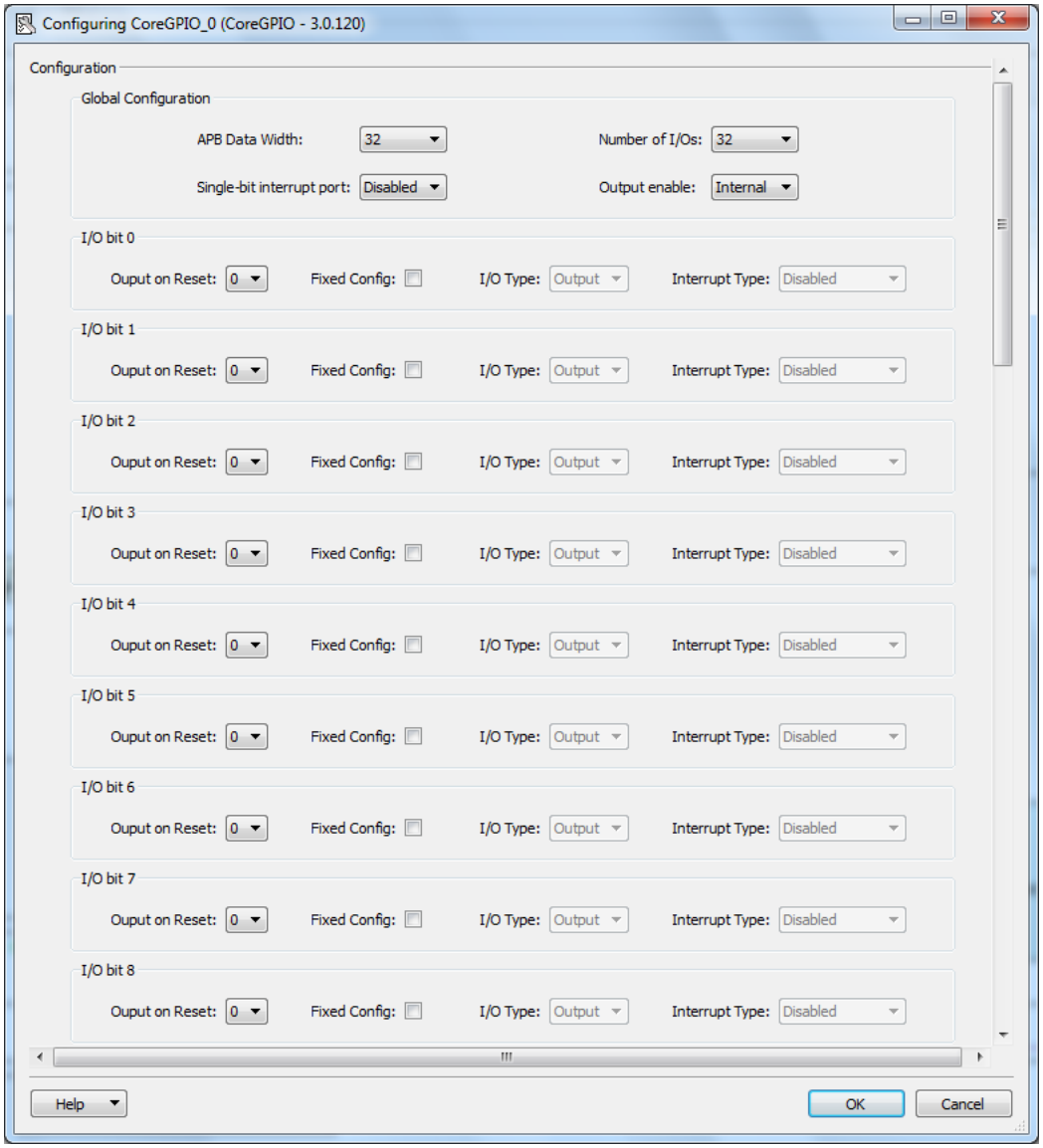

***Figure 8* • CoreGPIO Configurator GUI**

Description of Core8051s based Microcontroller System

All the peripherals are interfaced to the Core8051s as shown in Figure 9.

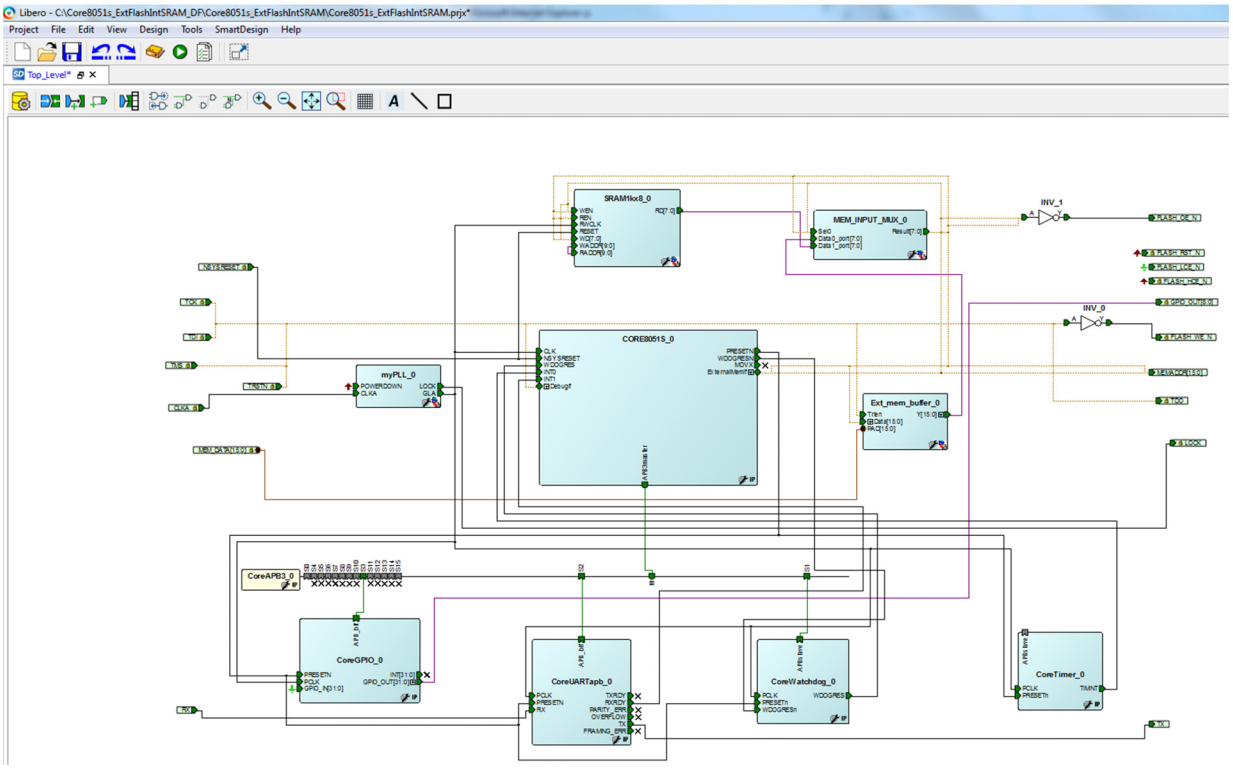

***Figure 9* • SmartDesign Top-Level Block Diagram**

Refer to the *Core8051s Based Hardware Tutorial* for more information.

Instantiate a two port RAM on the SmartDesign top-level and configure it as shown in Figure 10.

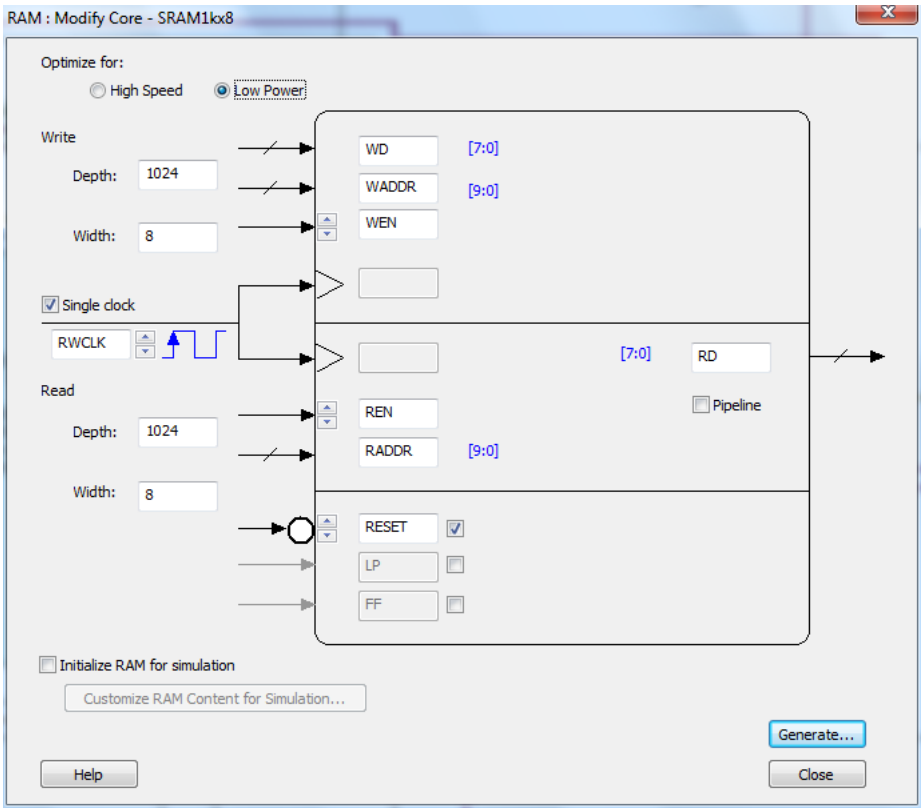

***Figure 10* • SRAM Configuration**

External memory buffer and multiplexer are configured as shown in Figure 11 and Figure 12.

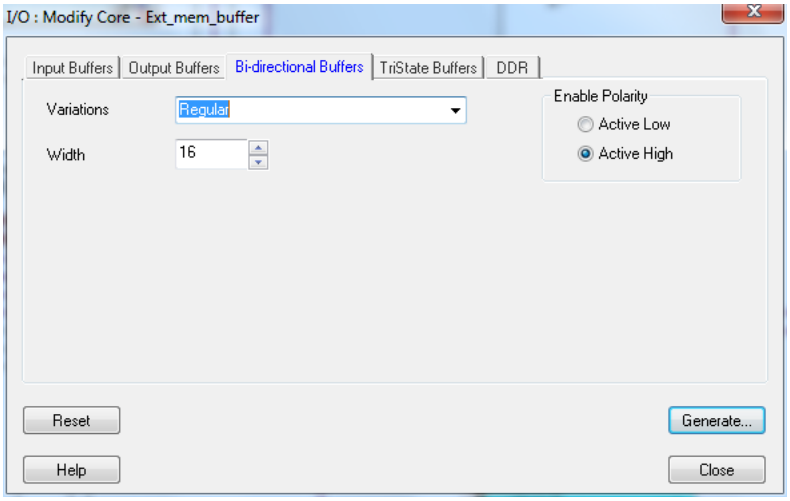

***Figure 11* • External Memory Buffer Configuration**

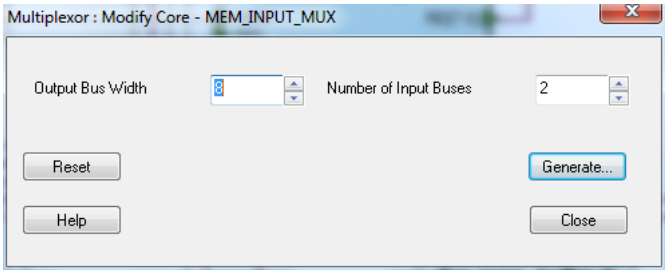

Figure 12 • **Multiplexer Configuration**

Memory Map

Right-click the **Modify Memory Map** to see the memory map as shown in Figure 13.

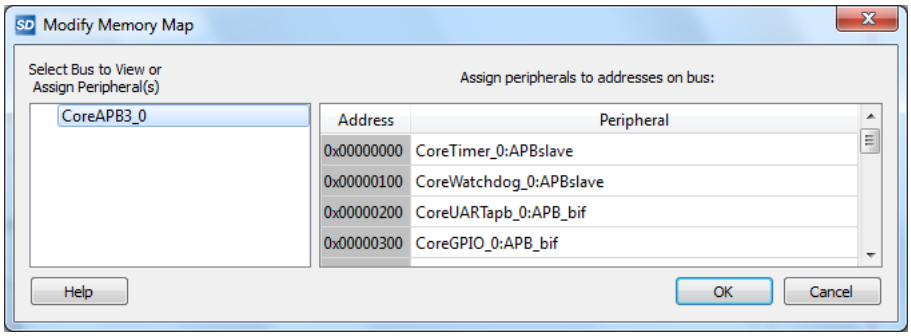

Figure 13 • **Memory Map**

Software Development Description

The drivers are generated from firmware catalog for CoreTimer, CoreGPIO, CoreWatchdog, CoreTimer, and hardware abstraction layer (HAL). The HAL is used by drivers to access the hardware and also allows the control of interrupts.Refer to the *Core8051s Based Software User Guide* for more information.

The Core8051s hardware design provides access to the external flash memory and internal SRAM. The Core8051s flash programming flow for Core8051s program memory is similar to the existing programming flow for Cortex-M1 flash program memory. The principal difference is, instead of specifying the location, size and the type of the program memory in a linker script, the program memory details are given in a text file (a memory-region-file) which uses the same syntax as the memory command section of a GCC linker script. The SoftConsole project configuration must be modified to specify the memory-region-file as an argument to the *actel-map.exe* helper program. Application code is written in *main.c* of the SoftConsole project to blink the on-board LED's.

Running the Design Example

To run the design example,

1. Download the design example at,
   http://soc.microsemi.com/download/rsc/?f=Core8051s_ExtFlashIntSRAM_DF
2. Double-click the **Program Device** under **Program Design** to program the Cortex-M1-enabled ProASIC3L Development Kit in the **Design Flow** window, as shown in Figure 14.

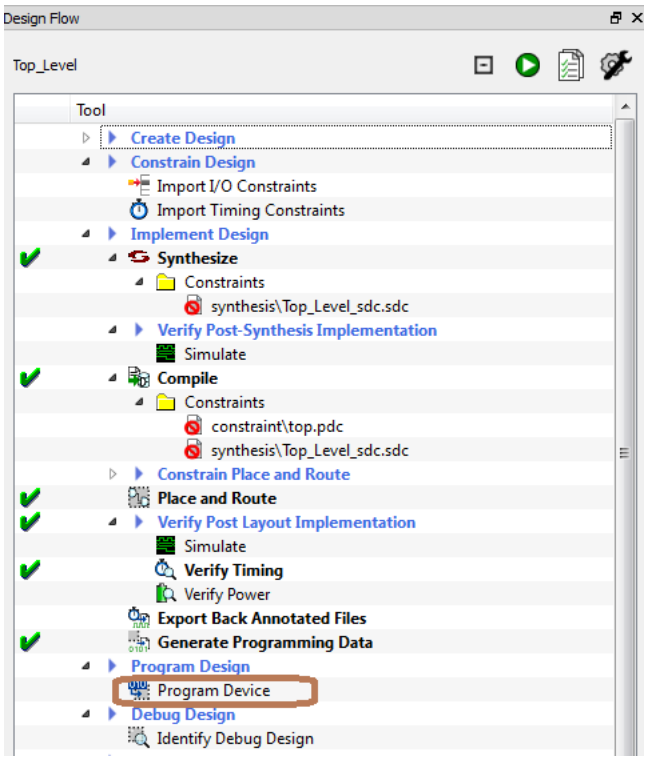

***Figure 14* • Program Device**

3. Open the **SoftConsole** project after successfully programming the device, as shown in Figure 15.

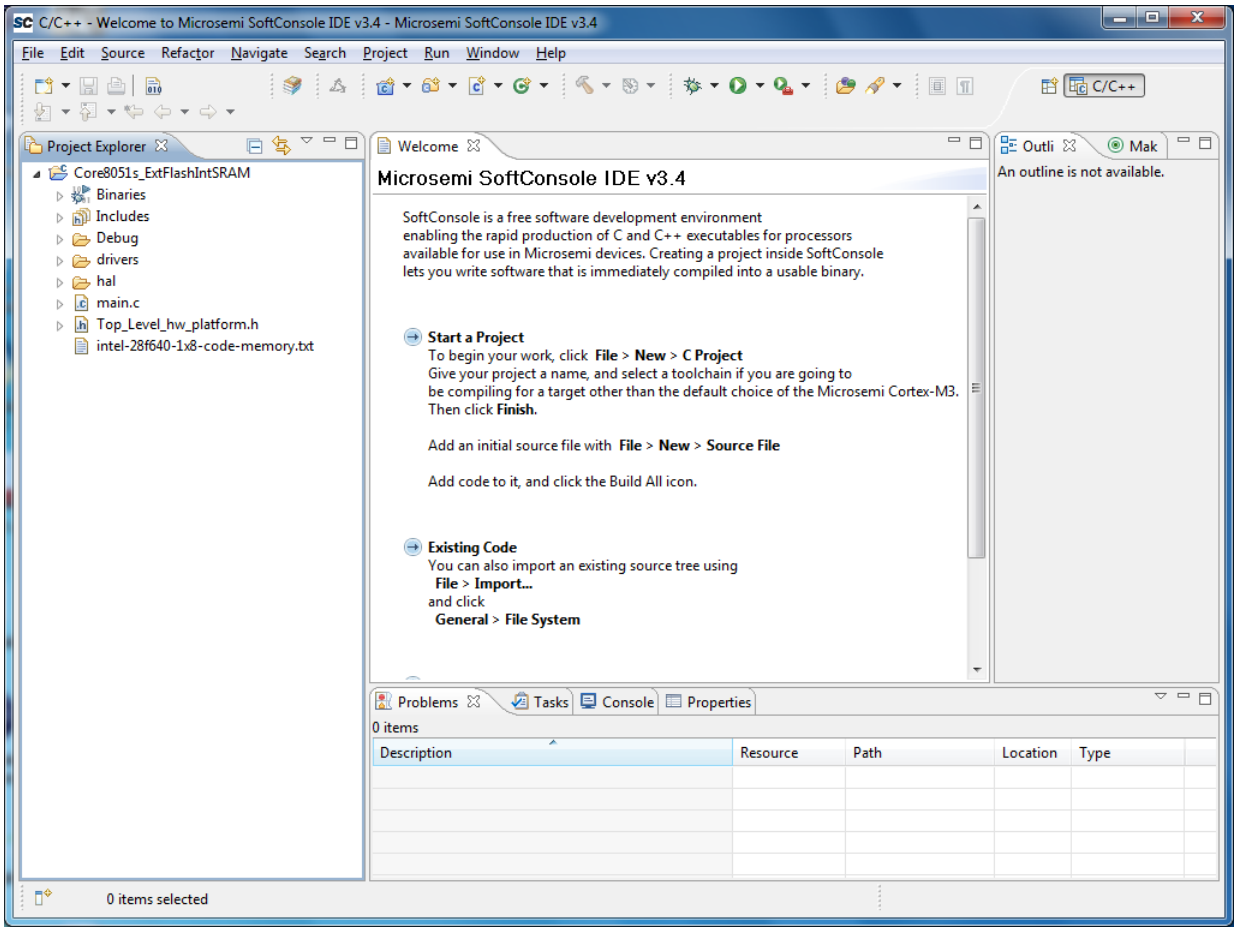

***Figure 15* • SoftConsole Project Window**

4. Right-click the **Core8051s_ExtFlashIntSRAM** on the left pane and click **Properties**, as shown in Figure 16. The **Properties** window is displayed as shown in Figure 16.

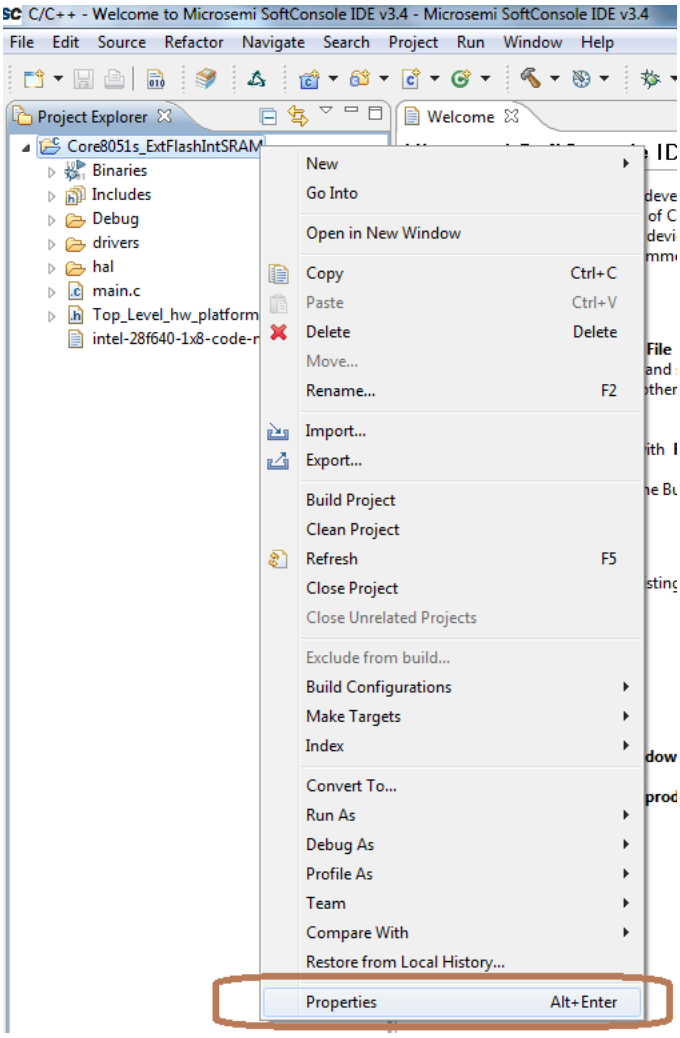

***Figure 16* • Project Properties**

5. Double-click **Settings** under **C/C++ Build** on the left pane of **Properties** window.

6. Click **Tools Settings** tab on the right pane and select the **Memory map generator,** as shown in Figure 17.

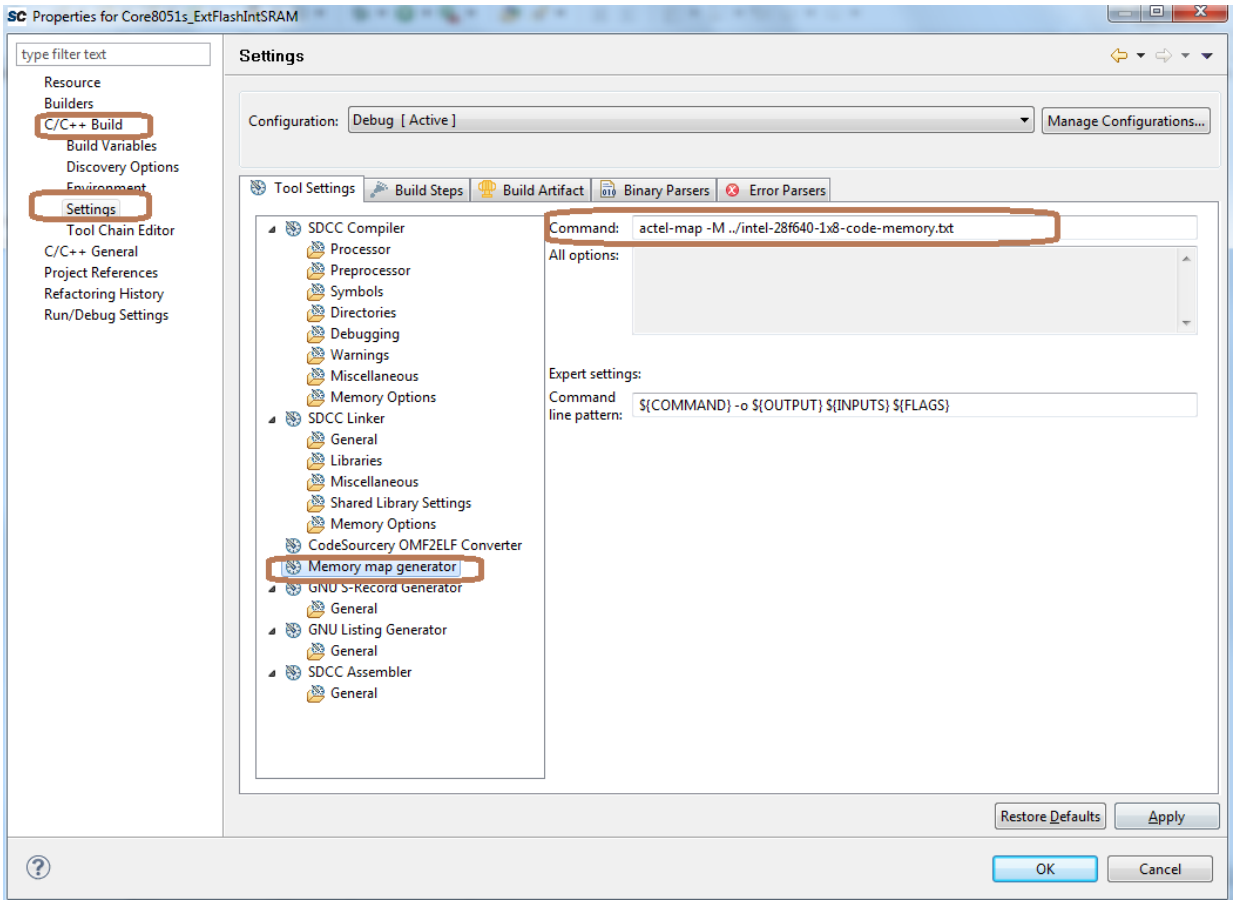

Figure 17 • **Memory Map Generator**

7. Enter **actel-map -M../intel-28f640-1x8-code-memory.txt** text in the **Command** field.

Note: The “intel-28f640-1x8” XML file, which is at *C:\Program Files (x86)\Microsemi\SoftConsole v3.4\Sourcery-G++\share\sprite\flash* is used for loading and debugging the JS28F640J3D-75 flash memory.

8. Right-click **Core8051s_ExtFlashIntSRAM** on the left pane and click **Debug As > Debug Configurations…**, as shown in Figure 18. The **Debug Configurations** window is displayed.

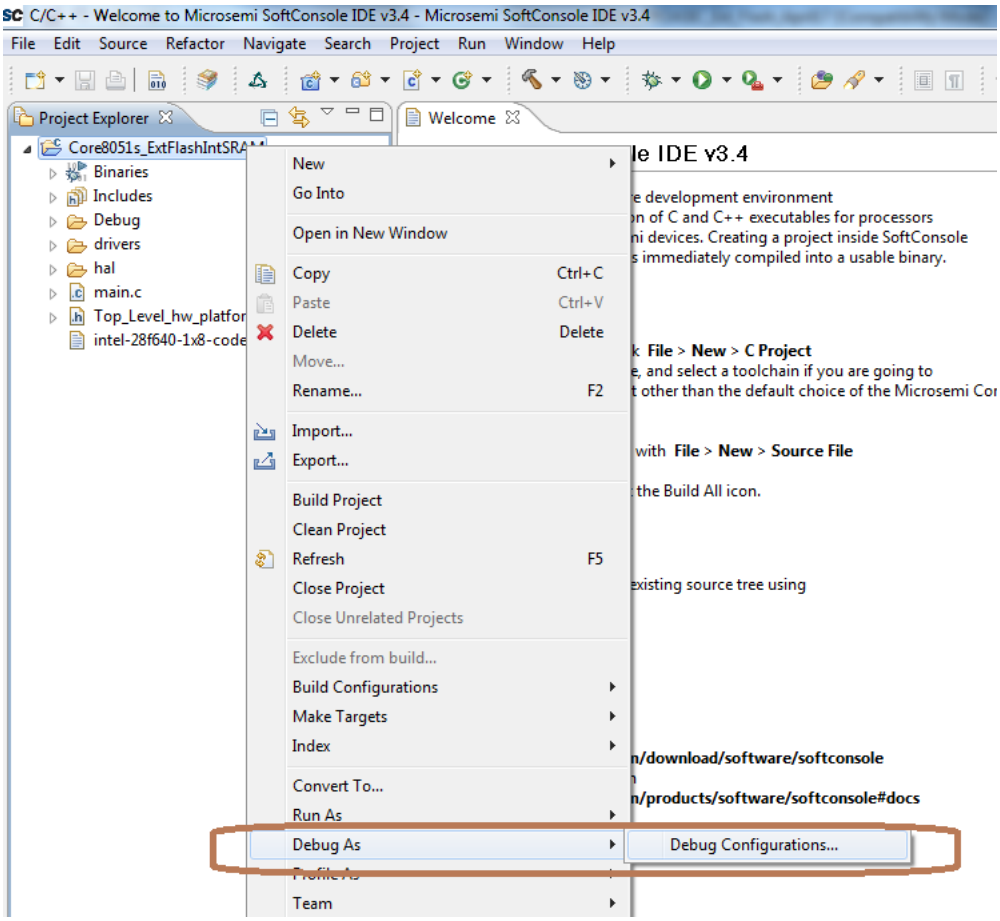

***Figure 18* • Debug Configurations**

9. Right-click **Microsemi Core8051s Target** and click **New** to create a new debug configuration, as shown in Figure 19.

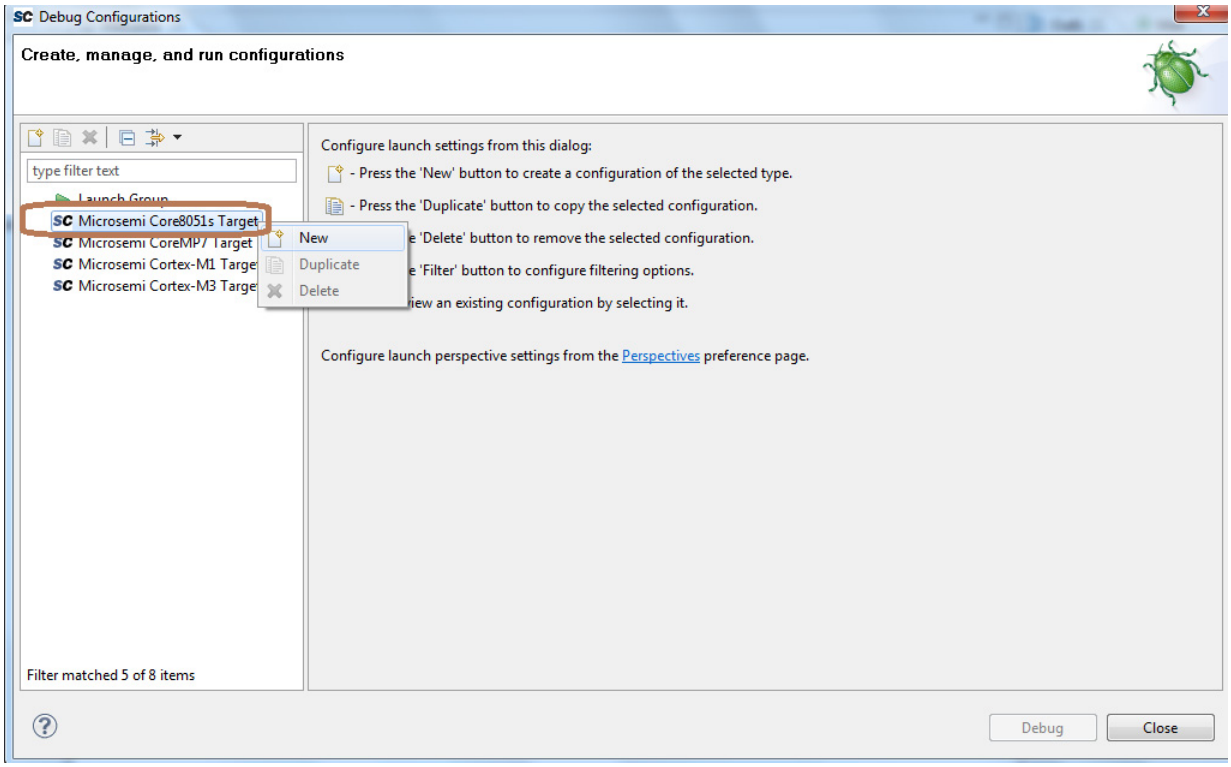

***Figure 19* • New Debug Configuration**

10. Click **Debug**.

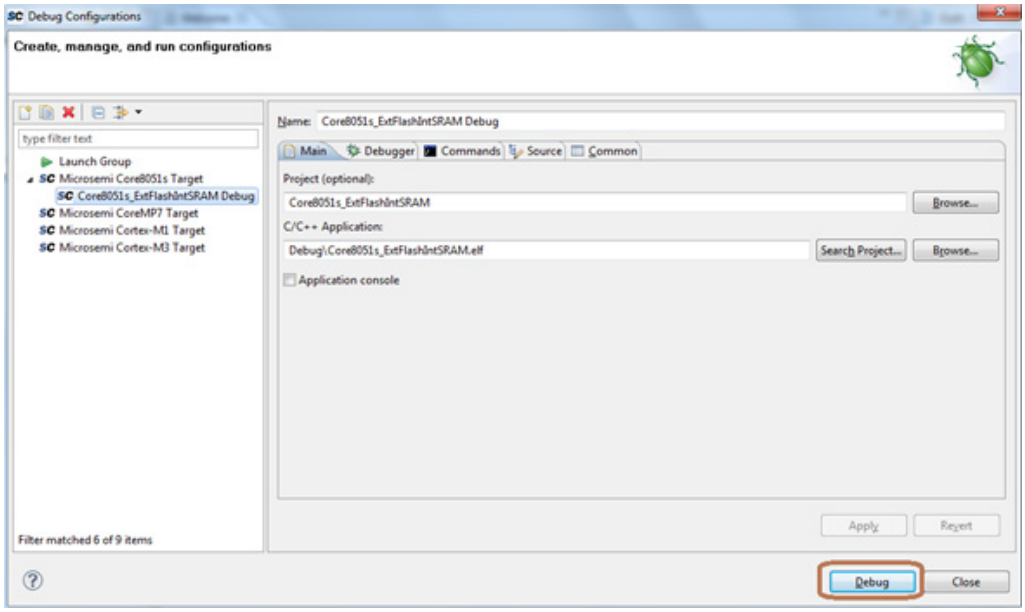

***Figure 20* • Debug Configurations**

After launching the debug session, the flash programming operation starts. The erase and write operations are shown in Figure 21.

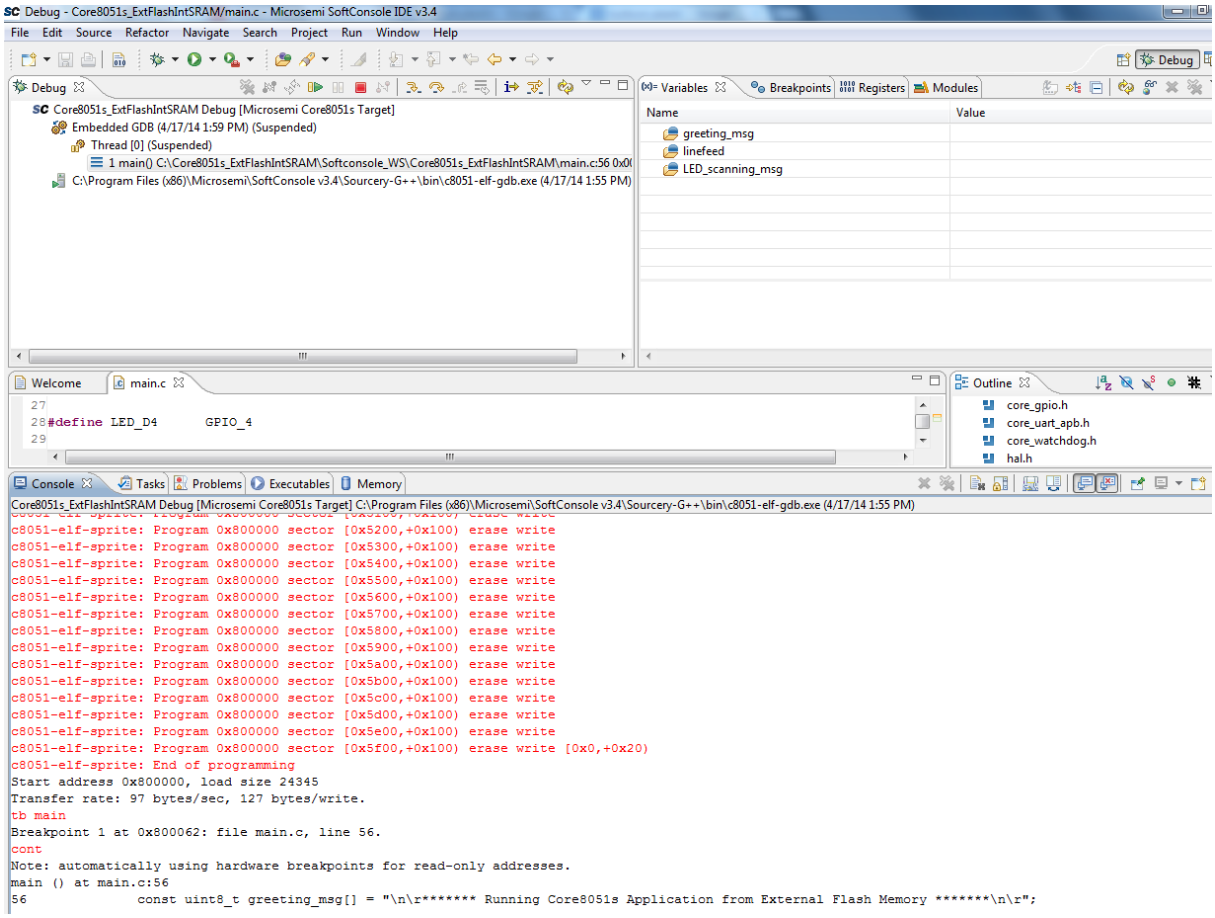

Figure 21 • **Flash Programming**

11. Start **PuTTY** (with settings 57600 baud rate, 8 data bits, and No parity), and choose **Resume** from the **Run** menu. The LEDs are scanned on the Cortex-M1-enabled ProASIC3L Development Kit in the forward and reverse direction. The messages are displayed as shown in Figure 22.

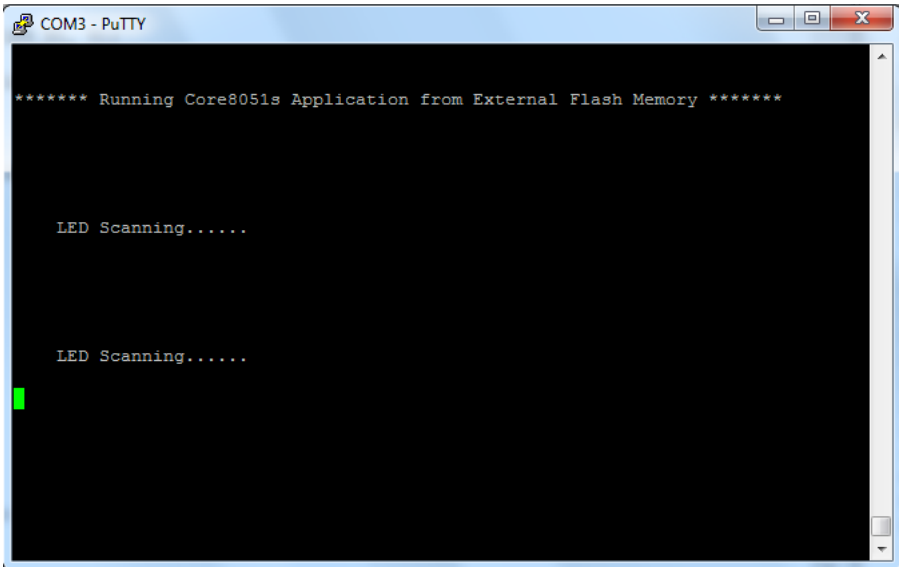

***Figure 22* • Application Running From External Flash Memory**

12. Terminate and relaunch the debug session.
13. Set break points at 60, 115 and 149 lines of main.c.
14. Choose **Resume** from the **Run** menu.
15. Choose **Step Over** from the **Run** menu until it reaches the 115 line of main.c. The “Running Core8051s Application from External Flash Memory” message is displayed as shown in Figure 23.

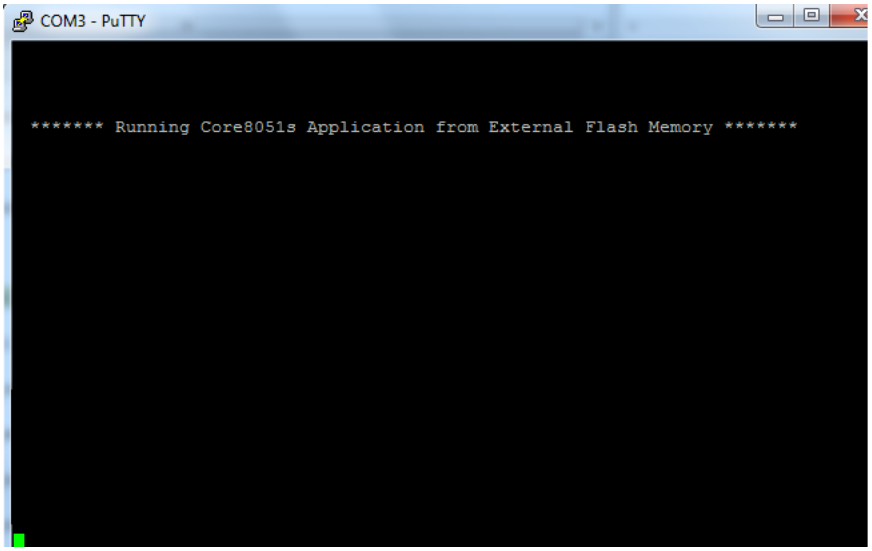

***Figure 23* • Debug Code**

16. Choose **Step Over** from the **Run** menu. While stepping over the code, the LEDs blinks on the *Microsemi Cortex-M1-enabled ProASIC3L Development Kit*. The message is displayed as shown in Figure 24.

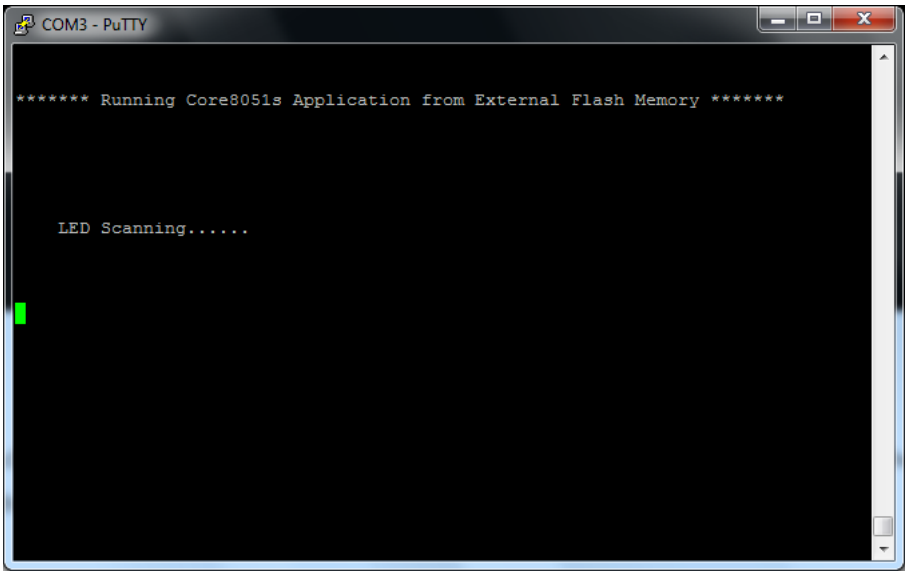

Figure 24 • **Step Over**

17. Right-click **Core8051s_ExtFlashIntSRAM Debug [Microsemi Core8051 Target]** and click **Terminate and Remove** the debug session as shown in Figure 25.

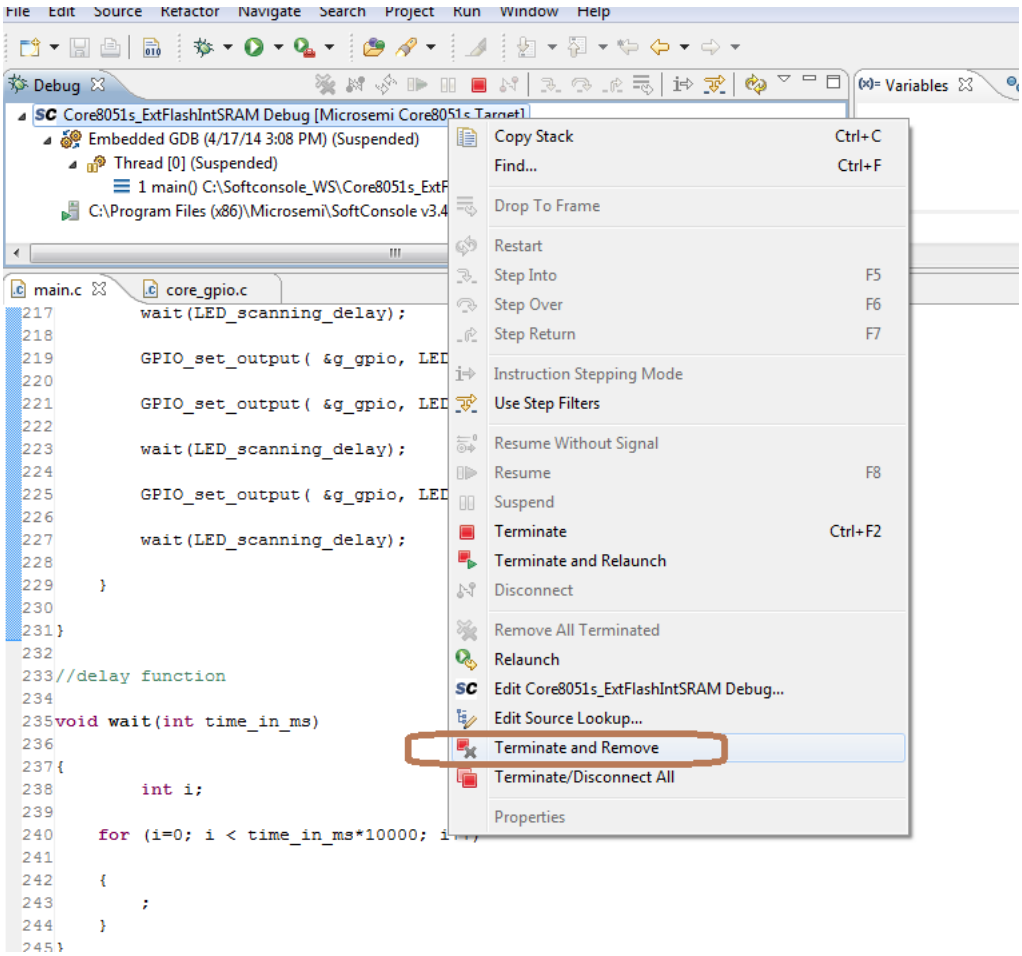

***Figure 25* • Terminate and Remove Debug Session**

18. Choose **Exit** from the **File** menu to close the SoftConsole project.
19. Unplug the USB cables and power supply cable and plug-in the power supply cable. The same LED scanning application runs from the non-volatile external flash memory.

Conclusion

This application note describes how to load and debug the Core8051s application from the external flash memory using SoftConsole. The example design serves as a starting point to other Core8051s designs. It includes a Core8051s based system, firmware drivers, and a sample LED scanning application that runs from the external flash memory.

Appendix A – Design and Programming Files

You can download the design files from the Microsemi SoC Products Group website:

http://soc.microsemi.com/download/rsc/?f=Core8051s_ExtFlashIntSRAM_DF

The design file consists of Libero project and programming file. Refer to the Readme.txt file included in the design file for directory structure and description.

List of Changes

The following table lists the critical changes that were made in the current version of the application note.

Date	Changes	Page
Revision 1 (July 2014)	Initial Release.	NA

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

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for communications, defense and 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; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, Calif. and has approximately 3,400 employees globally. Learn more at **www.microsemi.com**.

51900295-1/7.14