AN784

SiMxxxxx Flash Programming through the Serial Wire Interface

1. Introduction

Erasing and writing the on-chip flash region of the Silicon Labs SiMxxxxx range of 32-bit microcontroller devices is accomplished using the Flash Controller module (FLASHCTRL) as described in the device family Reference Manual. The FLASHCTRL module is programmed by a set of registers that are accessed over the internal APB bus. These registers can be accessed by firmware running on the CM3 core or by a debug agent performing standard ARM CoreSight Debug Access Port (DAP) read and write accesses over the Serial Wire interface. Note that the DAP Bus cannot be accessed by the ARM core.

This document discusses the programming features and procedures unique to the SiMxxxxx devices. For more information on the Serial Wire interface and ARM CoreSight DAP, visit:

http://infocenter.arm.com/help/topic/com.arm.doc.prdc008772b/index.html

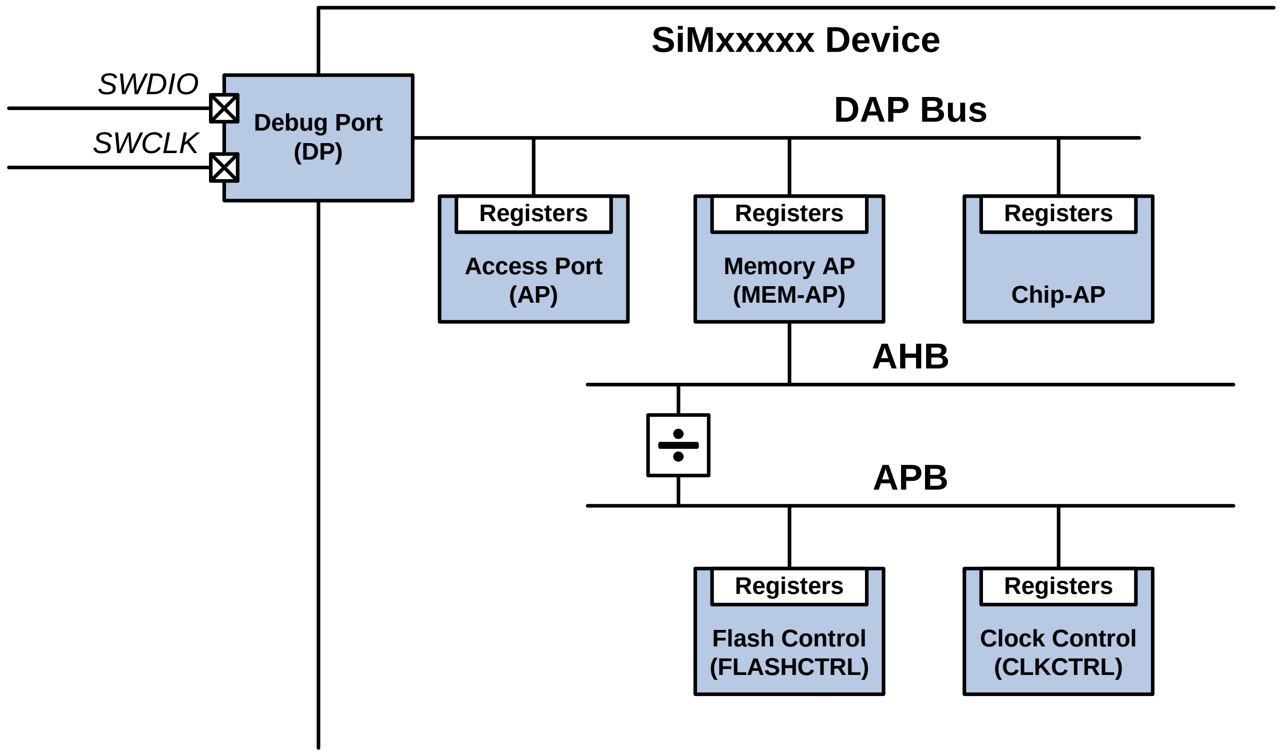


Figure 1. SiMxxxxx Debug Architecture

2. Relevant Documentation

Information in these documents is required to program the flash region of the SiMxxxxx devices.

- **ARM Serial Wire Interface and CoreSight DAP —** Describes the hardware programming protocol and Debug Access Port for SiMxxxxx devices: http://infocenter.arm.com/help/topic/com.arm.doc.prdc008772b/index.html.
- SiMxxxxx Reference Manual — Describes the flash controller (FLASHCTRL) and clock controller (CLKCTRL) modules for the particular device family: www.silabs.com/32bit-mcu→Device Family→Documentation tab.

3. Flash Programming Using the Debug Port

The SiM3U1xx/SiM3C1xx MCU includes standard ARM CoreSight DAP components. Specifically, the MCU provides a Serial Wire/JTAG Debug Port (SWJ-DP) and a Memory Access Port (MEM-AP). These DAP components allow an external debug agent to access all memory and peripheral registers residing on the internal AHB/APB buses. The erase and program operations for the on-chip flash region use the DAP to access FLASHCTRL registers, and the sequences for erasing and writing flash are the same as the operations from firmware described in the device family Reference Manual. When implementing these sequences for the DAP, there are some additional items to consider:

- While not necessary, it is a good idea to halt the core while programming the flash.
- The APB clock to the FLASHCTRL block must be enabled before the FLASHCTRL registers can be accessed using the DAP. This is done by writing to the clock control (CLKCTRL) module registers.
- The FLASHCTRL module programs the flash in 16-bit half-words. When writing to the FLASHCTRL WRDATA register, the lower 16-bits are written to flash and the upper 16-bits are ignored.
- When writing flash data to the FLASHCTRL WRDATA register, it is not necessary to poll the FLASHCTRL CONFIG.BUFSTS bit to determine if the previous write has completed. If FLASHCTRL is busy when a write to FLASHCTRL WRDATA is attempted, the write transaction will stall, causing the DP to return a WAIT acknowledgment. Continue to retry the transaction until FLASHCTRL is no longer busy with the previous write. If flash sequential writes have been enabled (FLASHCTRL.CONFIG.SQWEN = 1), an entire flash page can be written by repeatedly writing to the FLASHCTRL.WRDATA register, greatly speeding up the programming process.

4. Erasing All Non-Reserved Flash

The FLASHCTRL APB register interface only supports erasing flash pages one at a time. The SiMxxxxx devices include a feature to erase all non-reserved flash in a single operation. This erase-all feature is controlled by a special Access Port register that can only be accessed from the Debug Port, so this operation cannot be performed by firmware running on the core. There are two reasons why this erase-all operation should be supported with the debug port:

1. The erase-all operation is significantly faster than using the FLASHCTRL to erase each page one at a time.
2. The erase-all operation is the only way to unlock and erase a part that has been locked by writing the flash Lock Word address. A locked part disables the MEM-AP and prevents the DAP from accessing the FLASHCTRL module.

To perform the erase-all operation, complete the following DAP accesses:

1. Enable CSYSPWRUPREQ and CDBGPWRUPREQ in the DP.CTRLSTAT register by writing DP.CTRLSTAT = 0x50000000.
2. Select the special AP register by writing DP.SELECT with 0x0A000000.
3. Hold the core in reset by writing 0x8 to AP address 0x0.
4. Start the erase-all operation by writing 0x9 to AP address 0x0.
5. Bit 0 of AP address 0x0 clears when the erase operation completes. Poll AP address 0x0 and test Bit 0 to determine when the operation is finished.
6. Release the core reset by writing 0x0 to AP address 0x0. The part is now completely erased and ready to be programmed. Note that the erase-all operation enables all required clocks automatically.

5. Programming Examples

There are two examples included for programming the 32-bit SiMxxxxx devices:

- **High Level** — This example is a Python script that uses the USB Debug Adapter to demonstrate the sequences for a flash write, page erase, flash read, and device erase. The ADI DLL underneath this python script contains implementation details of the USB Debug Adapter, and knowledge of these implementation details is not needed to implement the 32-bit programming protocol.
- **SW Interface** — This is a firmware example for a C8051F38x device that demonstrates the low-level serial wire interface bit-banged with port pins.

5.1. Using the High Level Example

To run the High Level example:

1. Download and install Python 3.x (www.python.org). Note that the **si32FlashProgrammer** script is only compatible with v3.x of Python.
2. Connect a 32-bit USB Debug Adapter to the PC.
3. Connect the 32-bit USB Debug Adapter to the 10-pin CoreSight connector on a SiMxxxxx board.
4. Power the board with the SiMxxxxx device.
5. Run the **si32FlashProgrammer** Python script by typing **python si32FlashProgrammer.py** in a console from the **High_Level\src** directory. The output in the console will indicate the status of each flash operation.

Note: Python may need to be added to the Windows path to run scripts from the command line.

Project files are also included with the High Level example to enable debugging and running with Eclipse (www.eclipse.org) and an Eclipse Python plugin like PyDev (www.pydev.org).

The files in this example are as follows:

- **si32FlashProgrammer.py** — Defines high-level functions to read flash, write flash, page erase, and perform the erase-all operation. The main script at the bottom of the file calls these functions.
- **adi.py** — Implements a low-level interface to the 32-bit USB Debug Adapter libraries.
- **SLAB_ADI.dll** and **SLABHIDDevice.dll** — Low-level 32-bit USB Debug Adapter libraries.

5.2. Using the Serial Wire Interface Example

To run the Serial Wire (SW) Interface example:

1. Download and install the Silicon Labs IDE and Keil C51 tools (www.silabs.com/8bit-software).
2. Open the IDE and open the **si32FlashProgrammer.wsp** project by going to **Project→Open Project...**.
3. If necessary, update the path for the Keil tools by going to **Project→Tool Chain Integration...**. The default path of the project is the default installation path for the tools.
4. Build the project by clicking the **Rebuild All** button or going to **Project→Rebuild Project**. The two warnings that will appear can be ignored.
5. Connect an 8-bit USB Debug Adapter to the PC.
6. Inside the IDE, go to **Options→Connection Options...** to ensure the USB Debug Adapter is recognized and selected. Select the **C2** interface.
7. Connect the 8-bit USB Debug Adapter to the 10-pin debug connector on a C8051F380 board.
8. If using the C8051F380 Target Board, remove J15.
9. Connect the C8051F380 pins to the 10-pin CoreSight connector of an SiMxxxxx device:

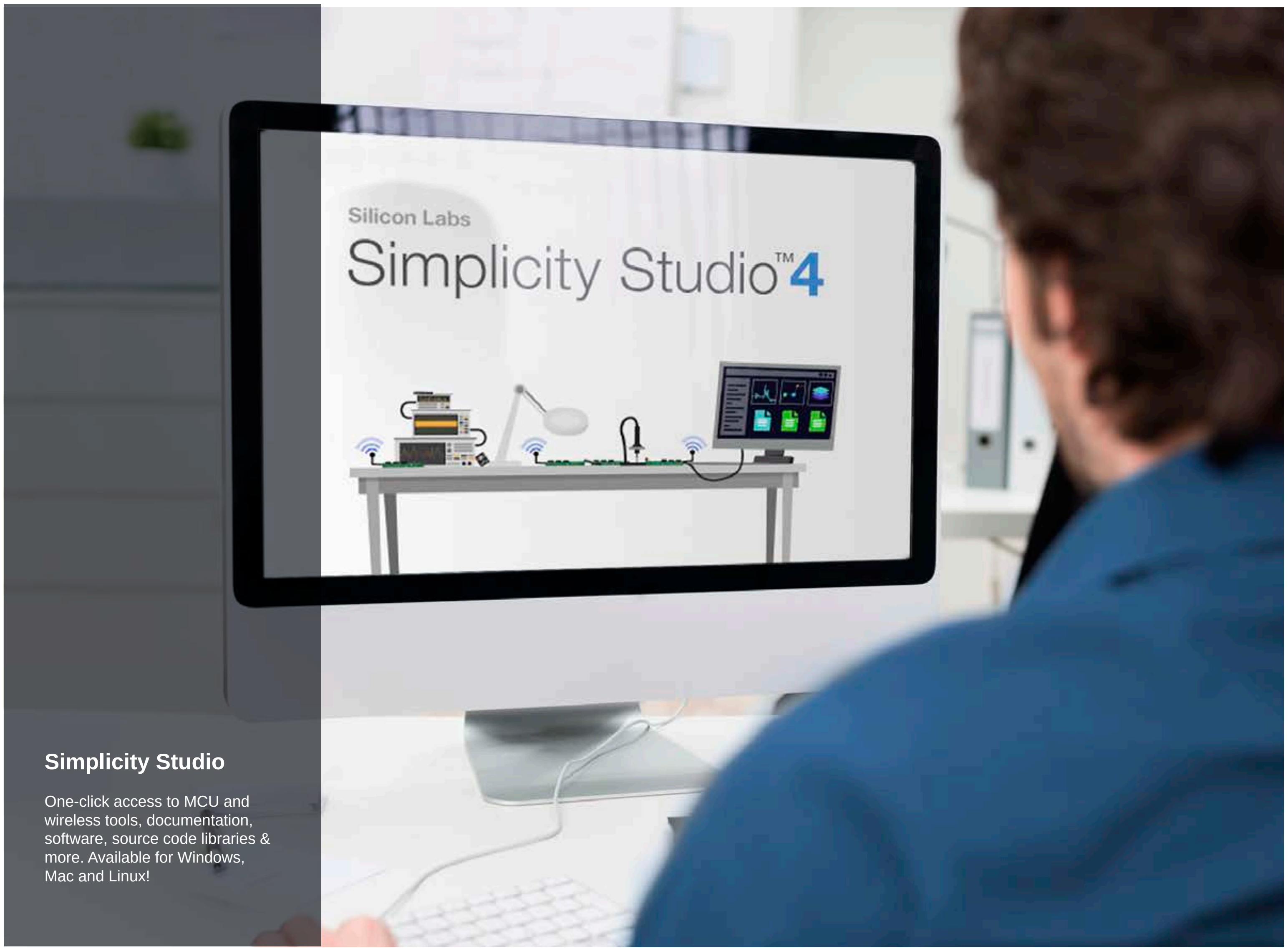
Table 1. C8051F380 Programmer Connections

Signal	C8051F380 MCU Pin	CoreSight 10-pin Connector Pin
SWDIO	P1.1	2
SWCLK	P1.3	4
ground	ground	3, 5, or 9

10. Power the board with the C8051F380 device.
11. Download the code to the C8051F380 by clicking the **Download** button in the IDE.
12. Set a breakpoint at **line 61** in the **main.c** file.
13. Run the code by pressing the **Go** button or going to **Debug→Go**.
14. Add **transfer_data** to the Watch Window by right-clicking on it and selecting **Add transfer_data to Watch as Default Type**.
15. Observe that **transfer_data** contains the correct IDCODE for the device.

The files in this example are as follows:

- **Init.c/.h** — Initialization functions specific for the C8051F380 device.
- **32bit_prog_defs.h** — Specific timing and value definitions for the SiMxxxxx 32-bit programming interface.
- **dp_swd.c** — Bit-bang implementation of the Serial Wire interface.
- **main.c** — Initializes the device and reads the IDCODE of the connected SiMxxxxx device.



Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio *www.silabs.com/IoT*	**SW/HW** *www.silabs.com/simplicity*	**Quality** *www.silabs.com/quality*	**Support and Community** *community.silabs.com*

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com