AN1497: SiWx917 SoC SWD Algorithm Programmer

This application note describes a Serial Wire Debug programmer can program the internal flash of an SiWx917 SoC.

KEY POINTS

- Overview – Hardware and Software
- Programming the SiWx917 SoC with JFlash
- SiWx917 SoC Generic Flash APIs

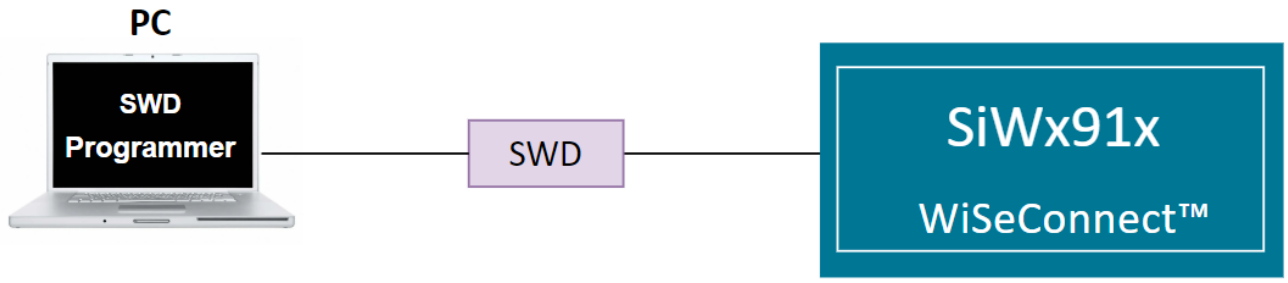


Table of Contents

1. Hardware Overview

1.1 Hardware Platform

The flash programmer supports two hardware topologies: one which is based on the SiWx917 Pro Kit and the other which is based on a custom hardware design. These are described in the following sections.

1.1.1 SiWx917 Pro Kit

The SiWx917 Pro Kit consists of a radio board and a mainboard.

- The radio board supports on-chip antenna and micro-coaxial connector for conducted RF measurements or external antenna connection
- Virtual COM port
- SEGGER J-link on-board debugger
- Breakout pads for Wireless SoC I/O

The flash programmer uses the SEGGER J-link on-board debugger for communicating with the SiWx917 SoC.

1.1.2 Customized Hardware

Alternatively, the SiWx917 SoC device can be used on a custom hardware platform to implement the programmer.

The SWDIO and SWCLK pins are used by SWD.

- The SWD pins can be used to communicate with the SWD Programmer.

2. Software Overview

This section covers the software drivers required for the programmer.

The generic flash programmer executes the flashing operation via the Network Wireless Processor (NWP) boot loader.

Description	Common Flash
Flash Memory	A single flash memory is shared between M4 (Host MCU) and NWP, and they access this flash memory over dedicated QSPI controllers.
Memory Access	An arbiter placed between the QSPIs and memory helps in arbitration for flash access.
Erase Operation	Uses the bootloader-based erase chip operation

The flash memory is shared between the M4 and NWP. Only the NWP can perform low-level operations on the flash memory, and the M4 can only support flash read operations. For low-level operations such as erase and write operations, the M4 can request the NWP bootloader to do the operations.

The steps required for flash programming are as follows.

Programming Stage	Description	APIs
Initialization	Device description needs to be provided in this stage.	In FlashDev.c file, we provided the device details in the "**struct FlashDevice_t**"
Program	Program the flash.	In FlashPrg.c file, we used the following two SiWx917 APIs in the ProgramPage() API. • **rsi_device_init(uint8_t select_option)** • **rsi_bl_upgrade_firmware(uint8_t *firmware_image , uint32_t fw_image_size, uint8_t flags)**
Erase	Erases the entire flash	In FlashPrg.c, we used the following SiWx917 API in the Erase-Chip() API. • **rsi_device_init(uint8_t select_option)**

Note: The SiWx917 APIs are explained in detail in Section 4. SiWx917 Generic Flash APIs.

3. Flash Loader – Example

This section shows the steps on how to use the SEGGER template files to program the SiWx917 using the SEGGER JFlash.

Note: The SiWx917 SoC is not SEGGER Licensed, we included the Device name locally in **JLinkDevices.xml** file to shows this example.

3.1 Prerequisites

The following hardware and software are required for programming the device.

3.1.1 Hardware

- SiWG917 SoC Kit (BRD4338A radio board + 4002A base board)
- USB to Type C cable for powering the kit and flashing the application
- A PC with USB ports
- SEGGER J-Trace Pro with power cable
- J-Link Adapter CortexM (You would get this with the J-Trace Pro when purchased)
- 20-position socket to socket connector cable

3.1.2 Software

- J-Link software (This example is verified with the version V7.96f.)
- Blinky example binary file can be downloaded here
- Source Code for Flash Loader – Download here
- Keil IDE (This example is verified with MDK version 5.29.0.0)
- Simplicity Commander: Download here

3.2 Setup

The image below illustrates the setup.

- The SiWx917 and the J-Trace are connected to a PC.
- The SiWx917 debug pins (on the 4002A board) and J-Trace J-Link Adapter CortexM – Target pins are connected using a socket-to-socket connector.
- Make sure the J-Trace Target power light is ON after connecting the socket-to-socket connector.

3.3 Programming the SiWx917 SoC with JFlash

To program the SiWx917 SoC using the JFlash, you need a .elf file which is generated in the Keil IDE by compiling the Si917_Flash-loader project.

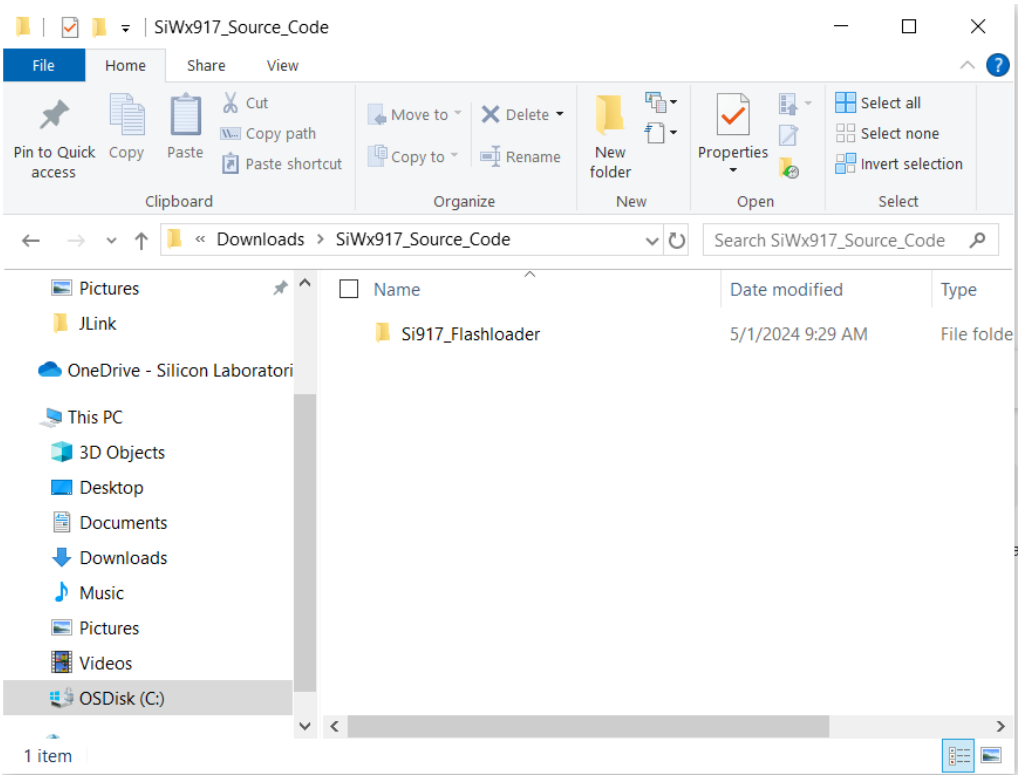
The flash programming involves the following steps, which are explained in detail in the sub-sections.

1. Generate the .ELF file in Keil IDE.
2. Add the SiWx917 device to SEGGER devices (locally).
3. Copy the .ELF file to the J-Link folder.
4. Erase the SiWx917 chip.
5. Program the SiWx917 using the JFlash.

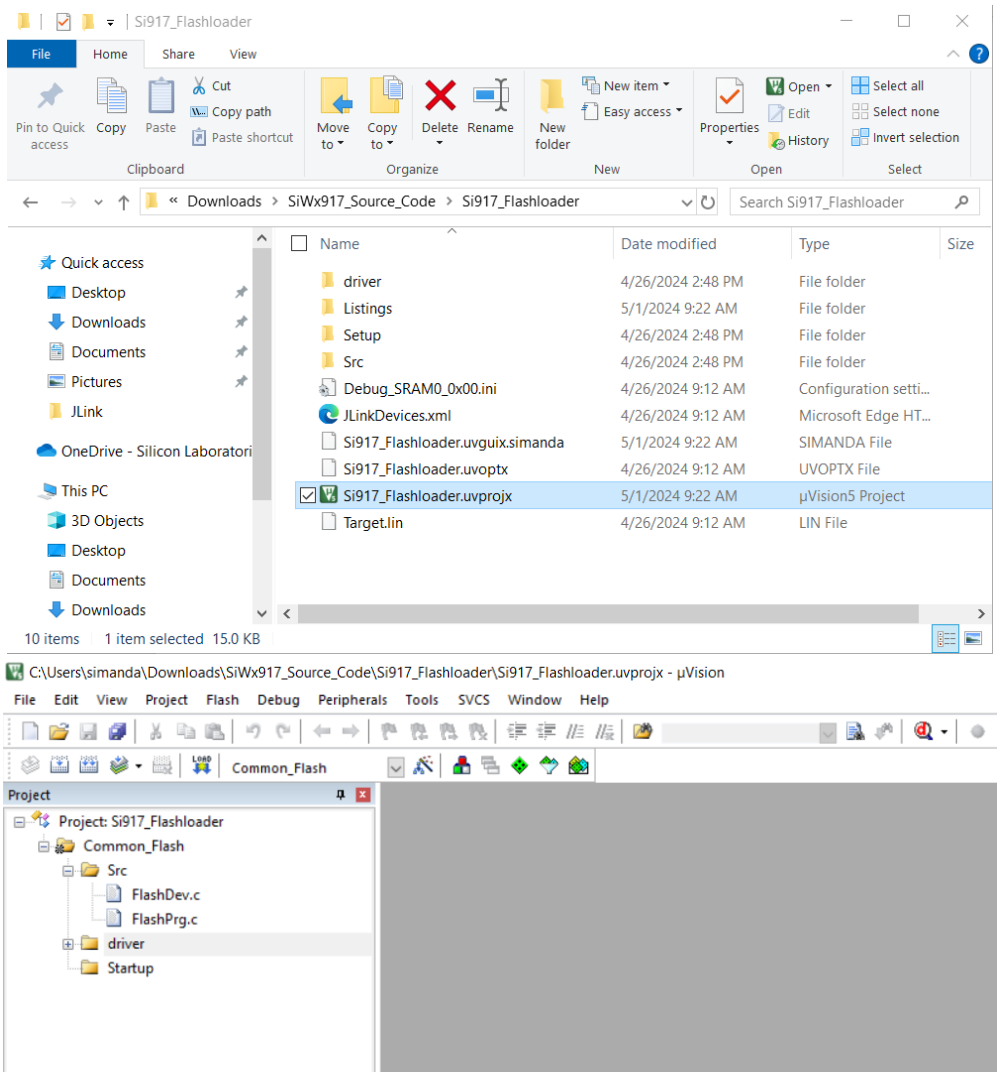
3.3.1 Generate .ELF File in Keil IDE

The following steps explain how to generate the Si917_Common_Flash.elf file.

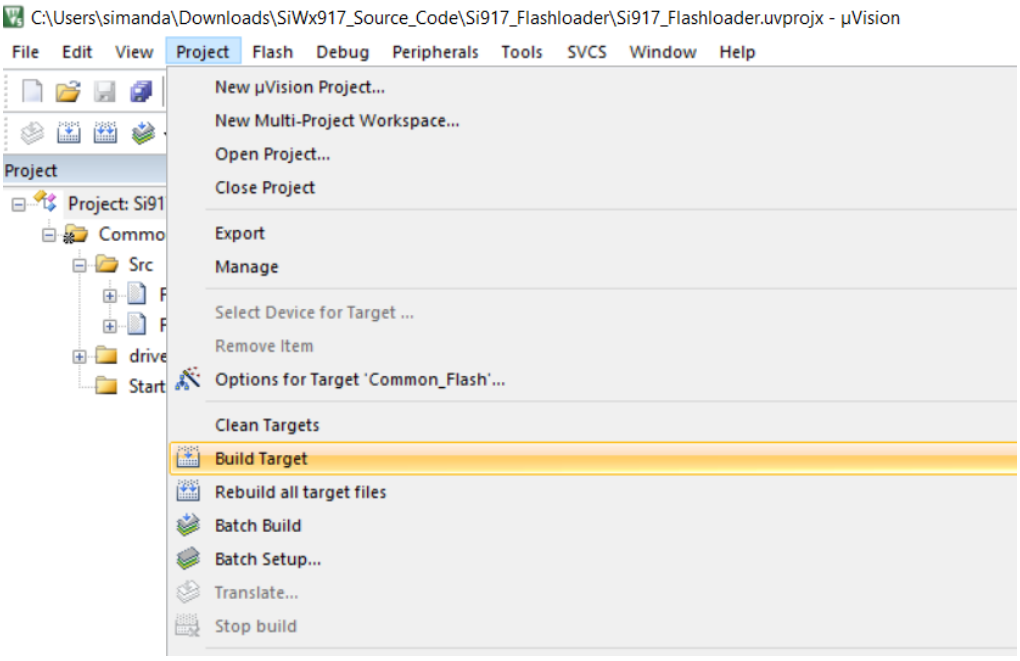
1. Download the SiWx917_Source_Code mentioned in the Section 3.1.2 Software.
2. Unzip the SiWx917_Source_Code. It should be as below.



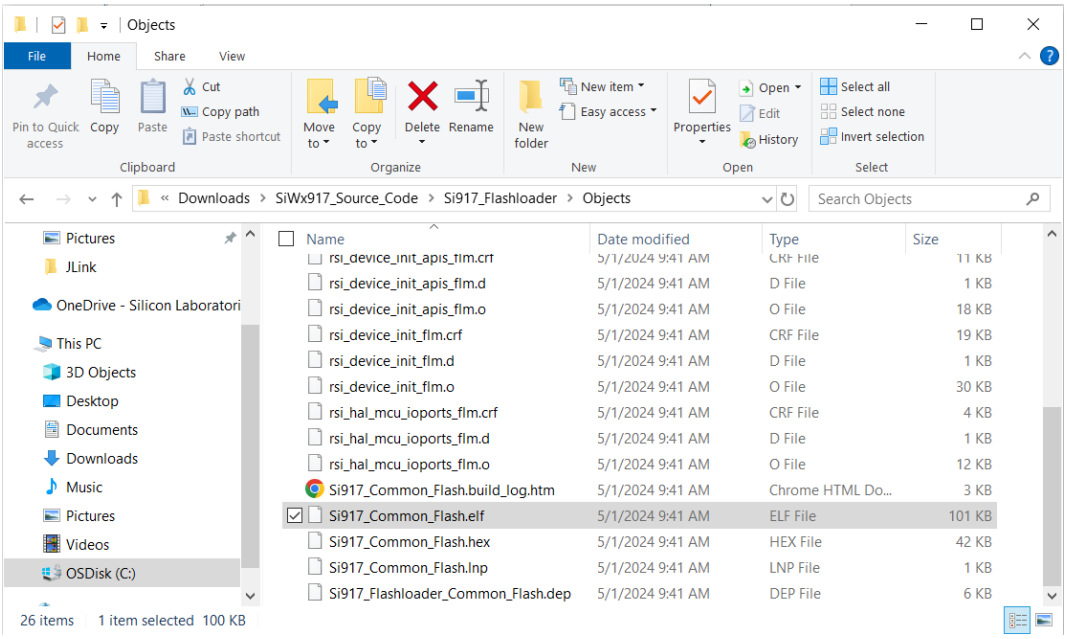
3. Got the Keil project path: **[Downloaded_Path]\SiWx917_Source_Code\Si917_Flashloader**. Open the Si917_Flashloader.uvprojx Keil project file which will open the Keil IDE with this project file.



4. Build the Project. This is generating the Si917_Common_Flash.elf file.



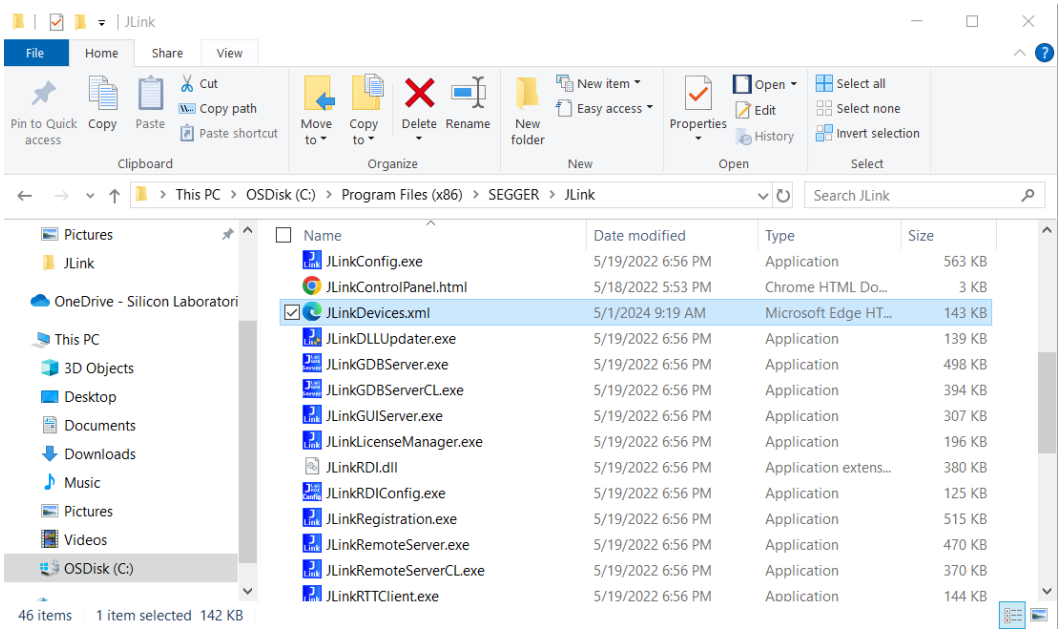
5. Go to the path: **[Downloaded_Path]\SiWx917_Source_Code\Si917_Flashloader\Objects**. You should see the Si917_Common_Flash.elf file.



3.3.2 Add SiWx917 Device to SEGGER Devices [Locally]

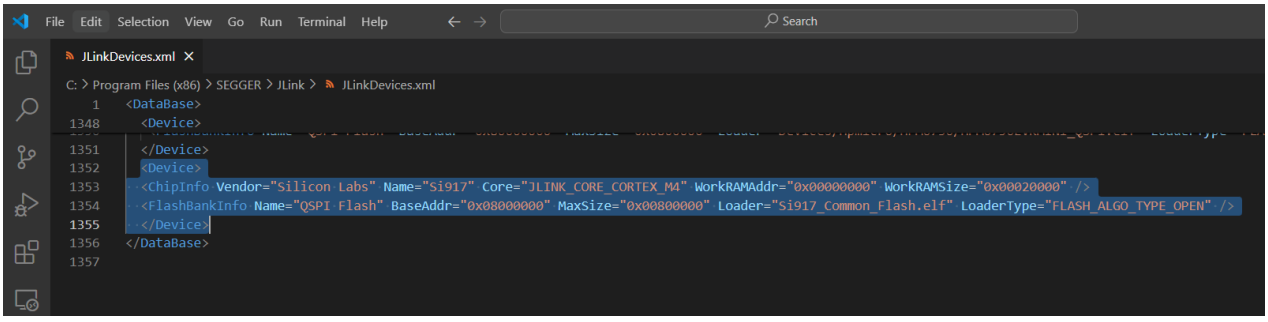
The following steps explain how to add the SiWx917 device to the SEGGER device locally.

1. Download and Install the SEGGER J-Link software mentioned in Section 3.1.2 Software.
2. Once installed, go to the path: **[Installed_Path]\ SEGGER\JLink**. Example: Here it is installed in the path: (C:\Program Files (x86)\SEGGER\JLink). Open the JLinkDevices.xml file with an editor of your choice.



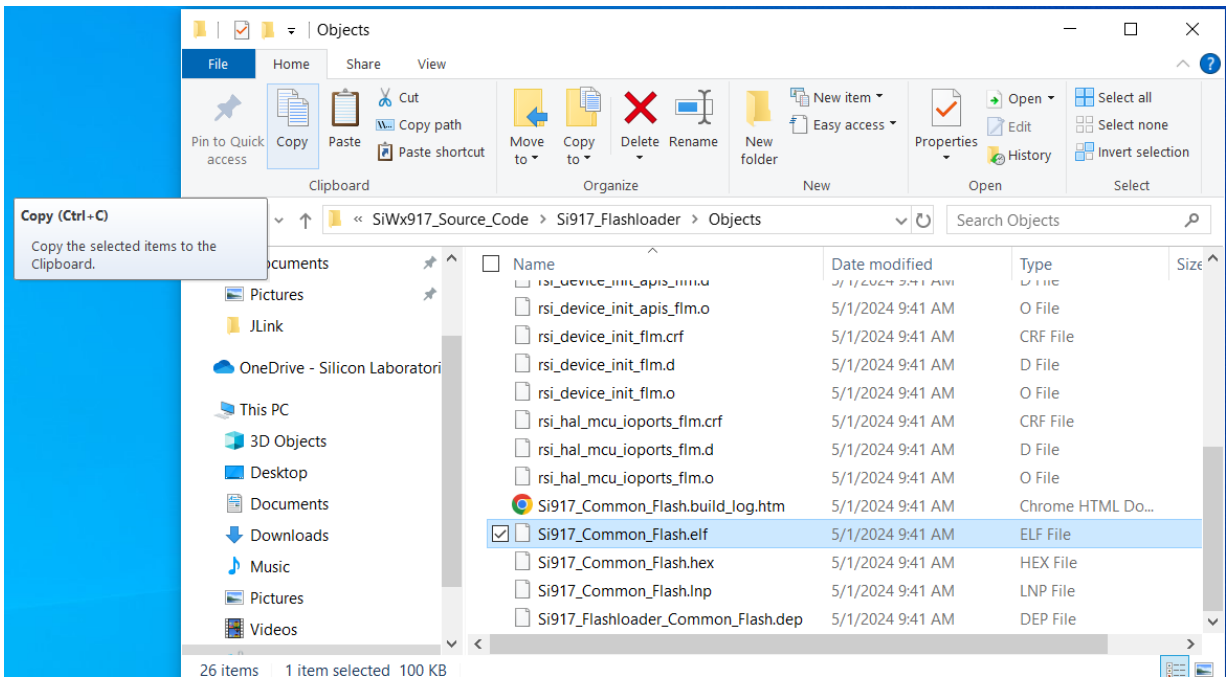
3. In the JLinkDevices.xml file, go to the end of the file and add the following text and then save the file as shown below.

```
<Device>

<ChipInfo Vendor="Silicon Labs" Name="Si917" Core="JLINK_CORE_CORTEX_M4" WorkRAMAddr="0x00000000"
WorkRAMSize="0x00020000" />

<FlashBankInfo Name="QSPI Flash" BaseAddr="0x08000000" MaxSize="0x00800000"
Loader="Si917_Common_Flash.elf" LoaderType="FLASH_ALGO_TYPE_OPEN" />

</Device>
```

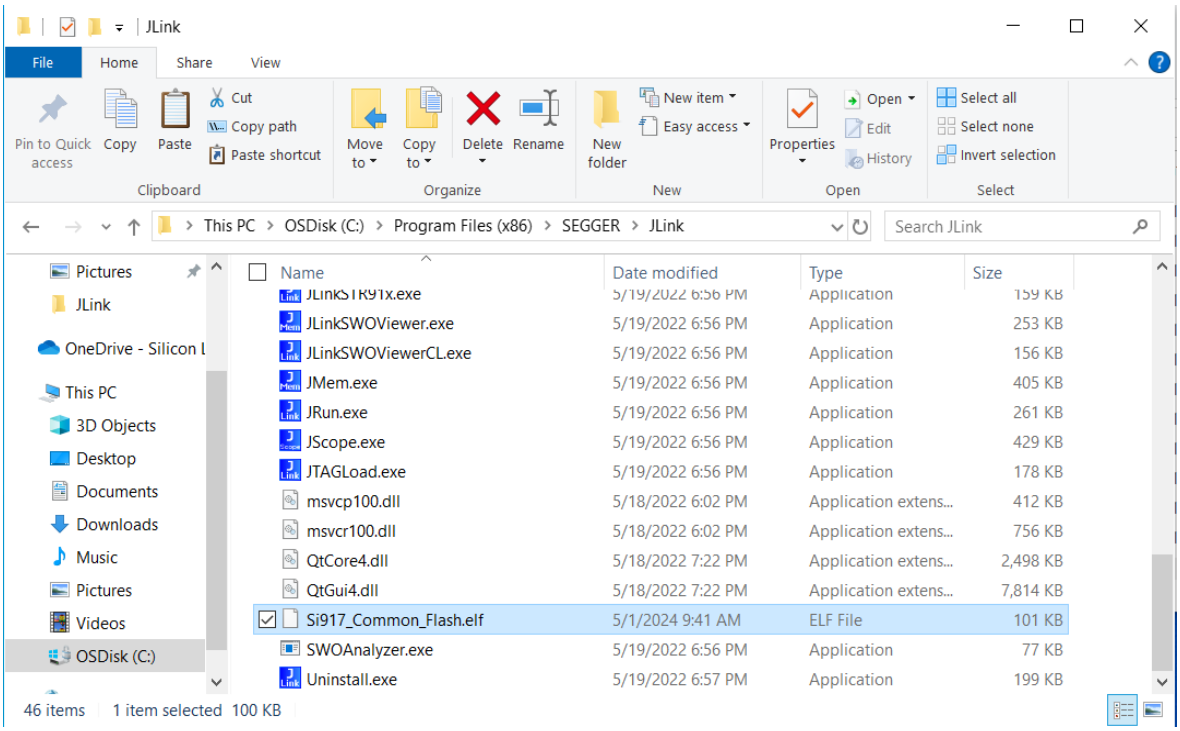


3.3.3 Copy the .ELF file to the J-Link Folder

1. Copy the Si917_Common_Flash.elf file generated in Step 5 in Section 3.3.1 Generate .ELF File in Keil IDE as shown below.



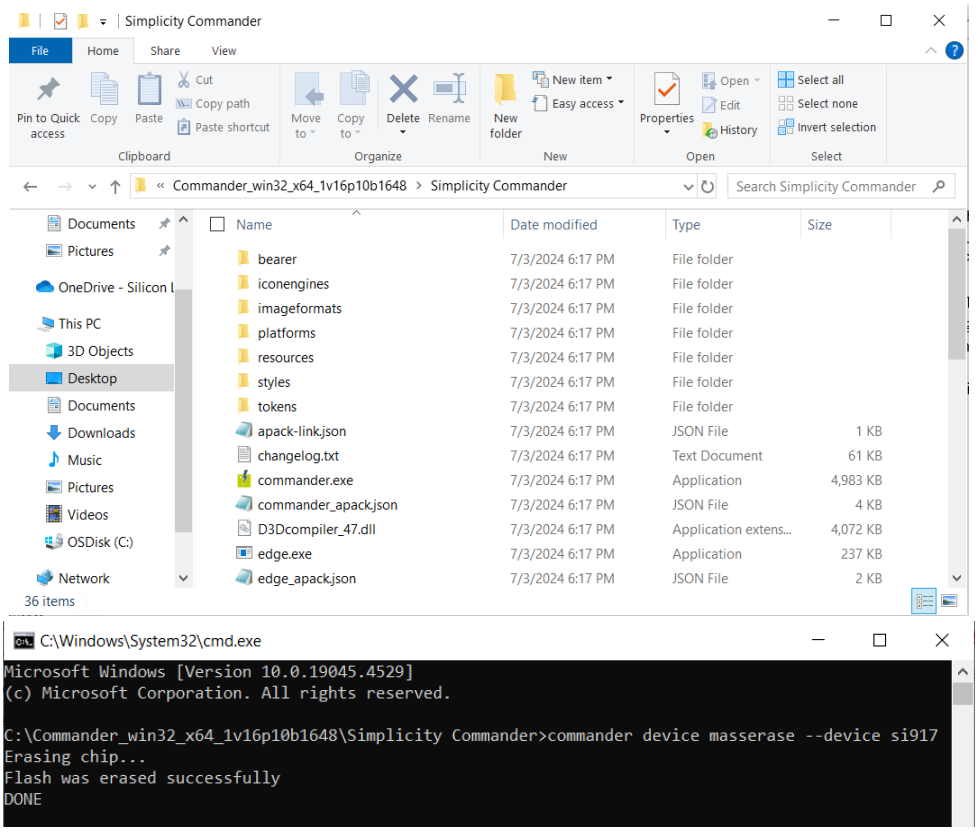
2. Paste the copied Si917_Common_Flash.elf file in the path: **[Installed_Path]/SEGGER/JLink** as shown below.



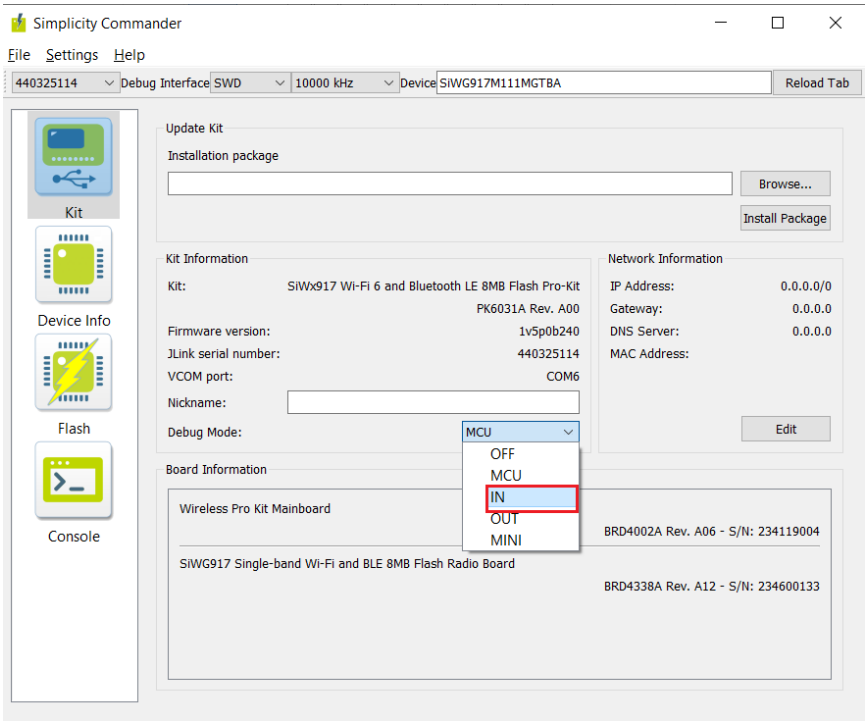
3.3.4 Erase Chip - SiWx917

To confirm an example is loaded through the flash loader, first, erase the SiWx917 chip and keep it in IN mode using the Simplicity Commander by following the steps below.

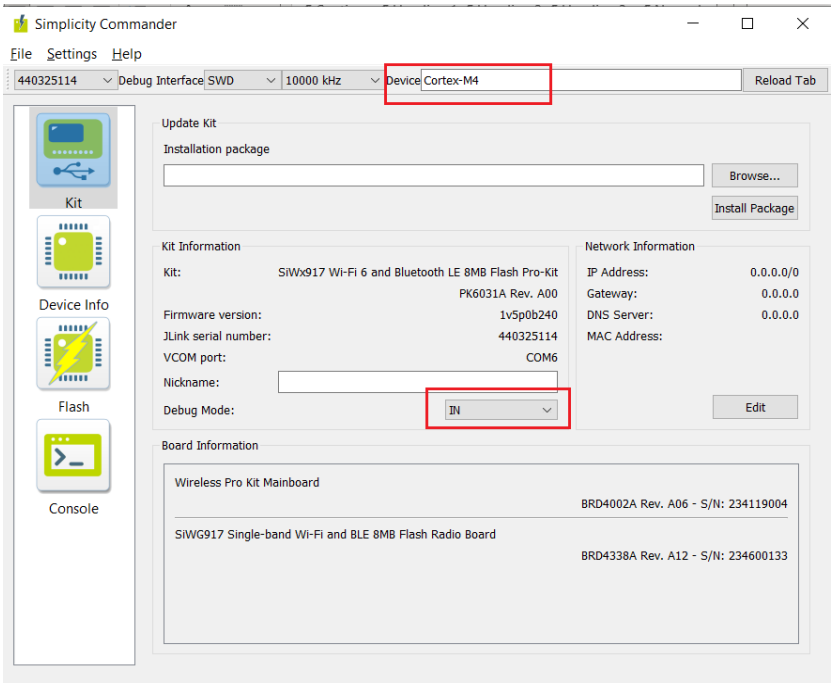
1. Download and Install Simplicity Commander mentioned in Section 3.1.2 Software.
2. Go to the path: **[Installed_Path]\ Simplicity Commander**. Open Terminal and give the command: **commander device masserase --device si917**



3. Go to the path: **[Installed_Path]\ Simplicity Commander**. Double click the **commander.exe**.
4. Click on Kit, change the **Debug Mode** from MCU to **IN**.



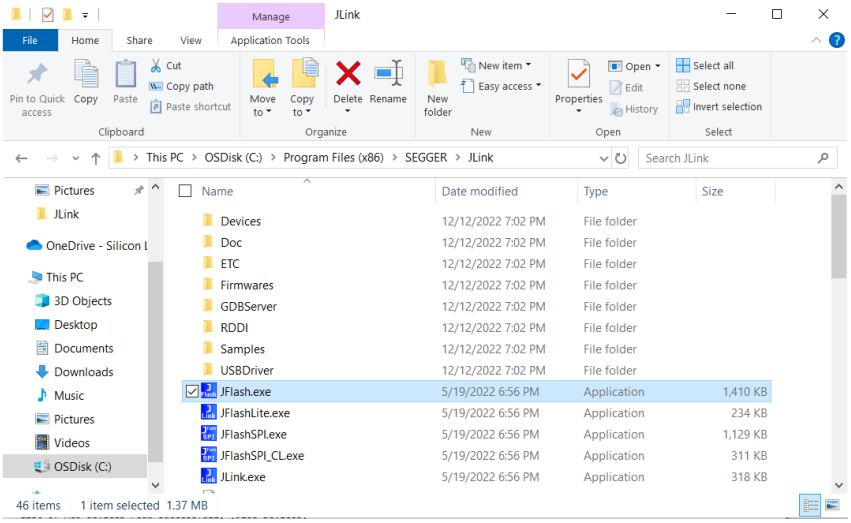
5. Upon changing the Debug Mode, the Device will be shown as **Cortex M4** .



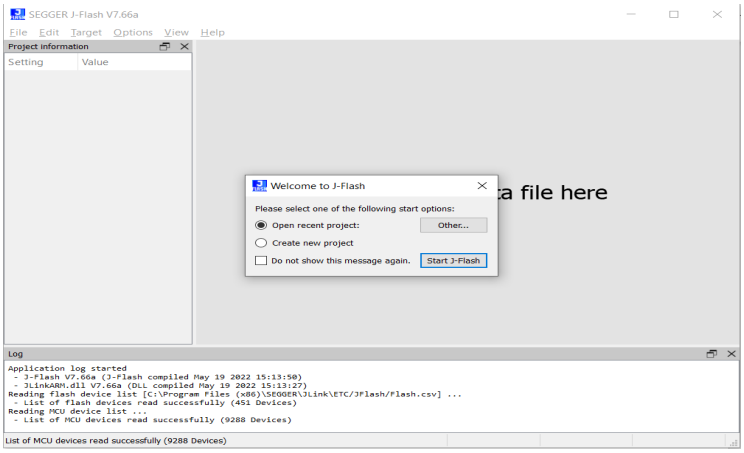
3.3.5 Program Using the JFlash

This section provides the steps to program the SiWx917 SoC using the JFlash.

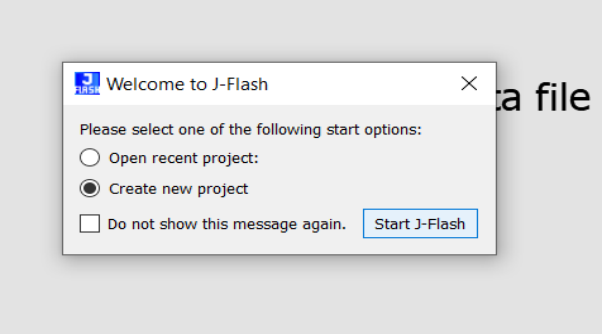
1. Go to the Path: **[Installed_Path]\SEGGER\JLink**. You will find JFlash.exe in it.



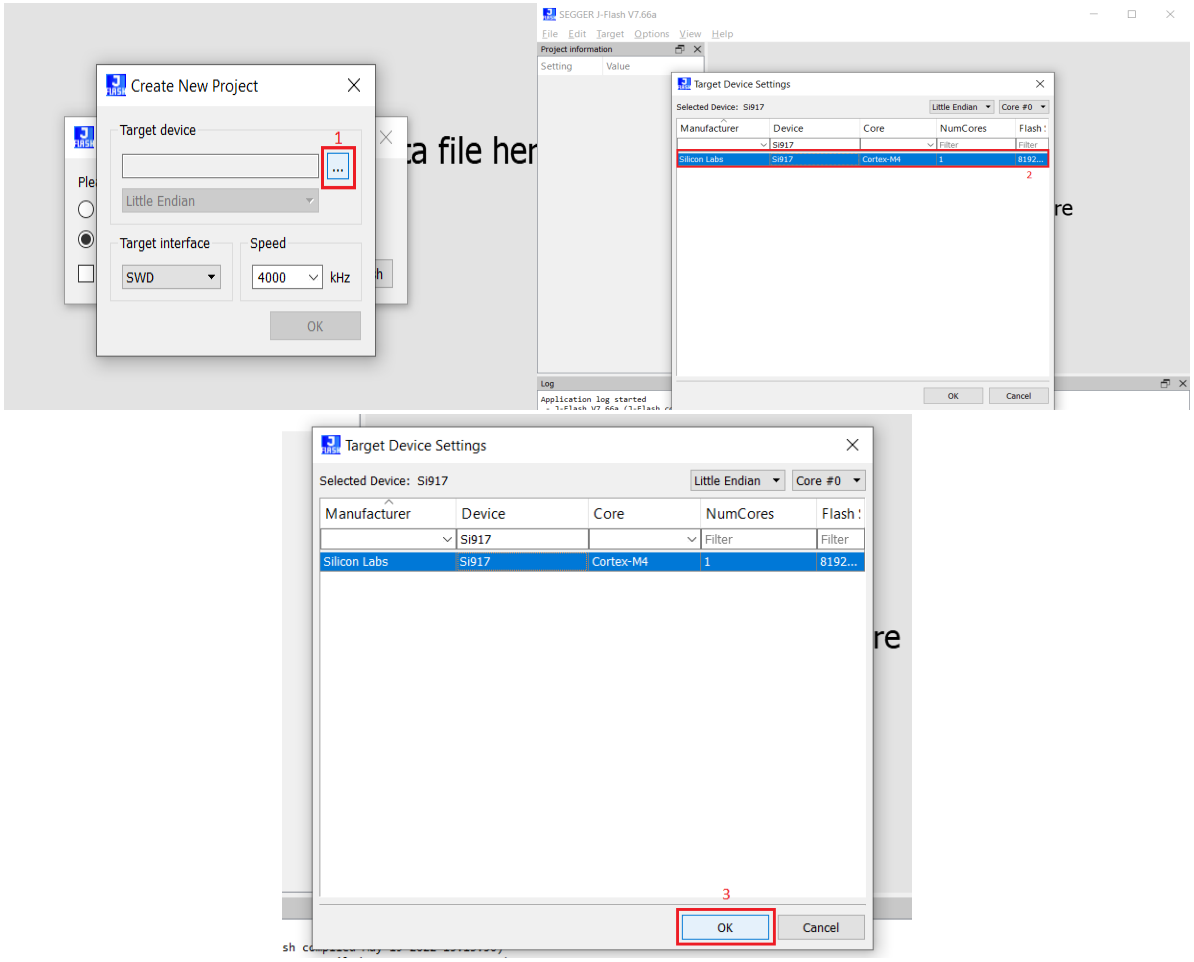
2. Double click on **JFlash.exe**. This will open the SEGGER JFlash.



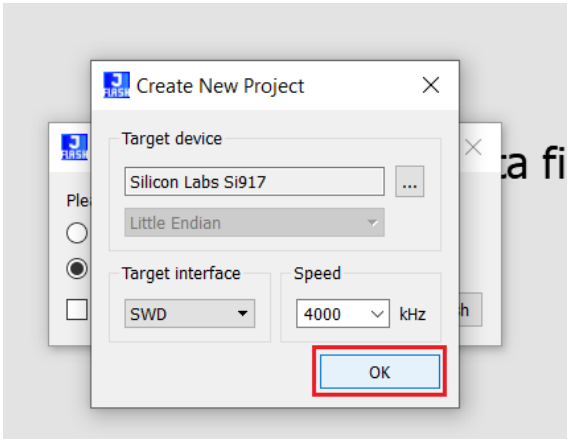
3. Select **"Create new project"** and click on **Start J-Flash**.



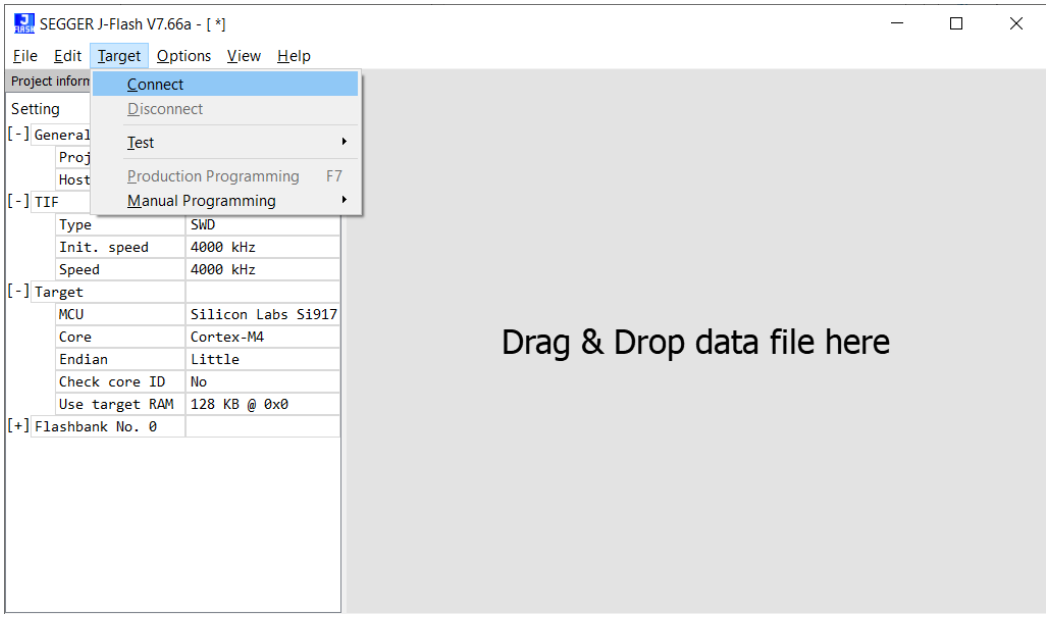
4. Click on the **"..."** under the Target device. You will be re-directed to Target Device Settings. Search for **Si917** under Device column and select it. Next, click on **OK**.



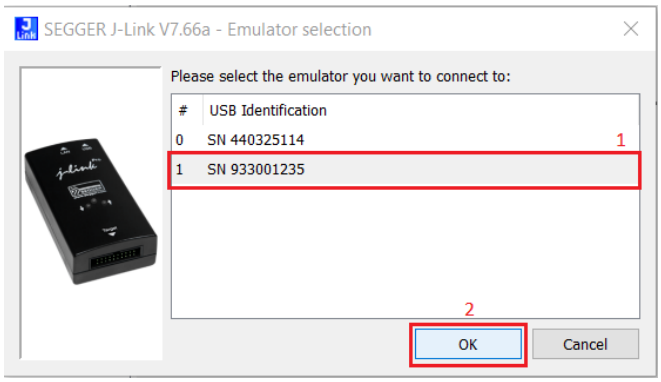
5. In the Create New Project Window, click on **OK**.



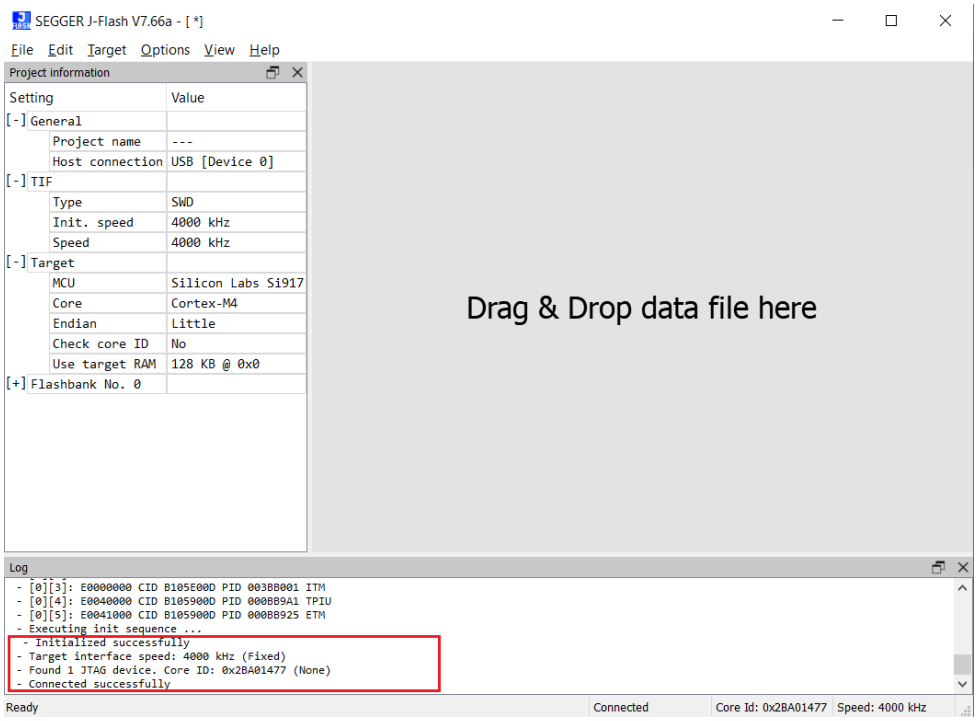
6. Next, click on **Target** → **Connect**.



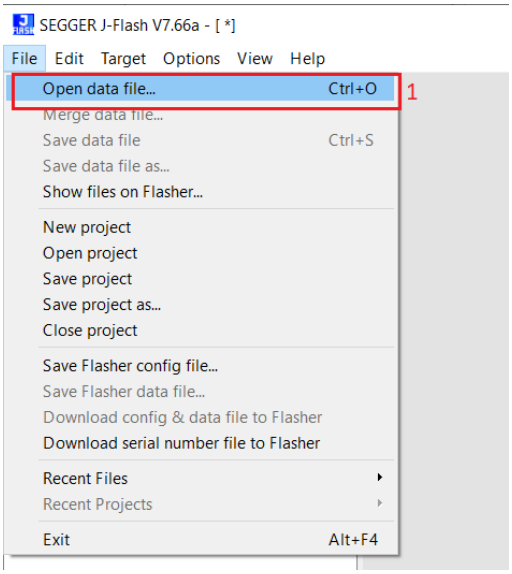
7. In the **Emulator selection** pop-up window, select the J-Trace S/N. You can see the S/N on the backside of the J-Trace. Select the correct SN in the Emulator selection window. (Example: In the image below, the J-Trace S/N is 933001235, so SN 933001235 is selected). Next, click on **OK**.

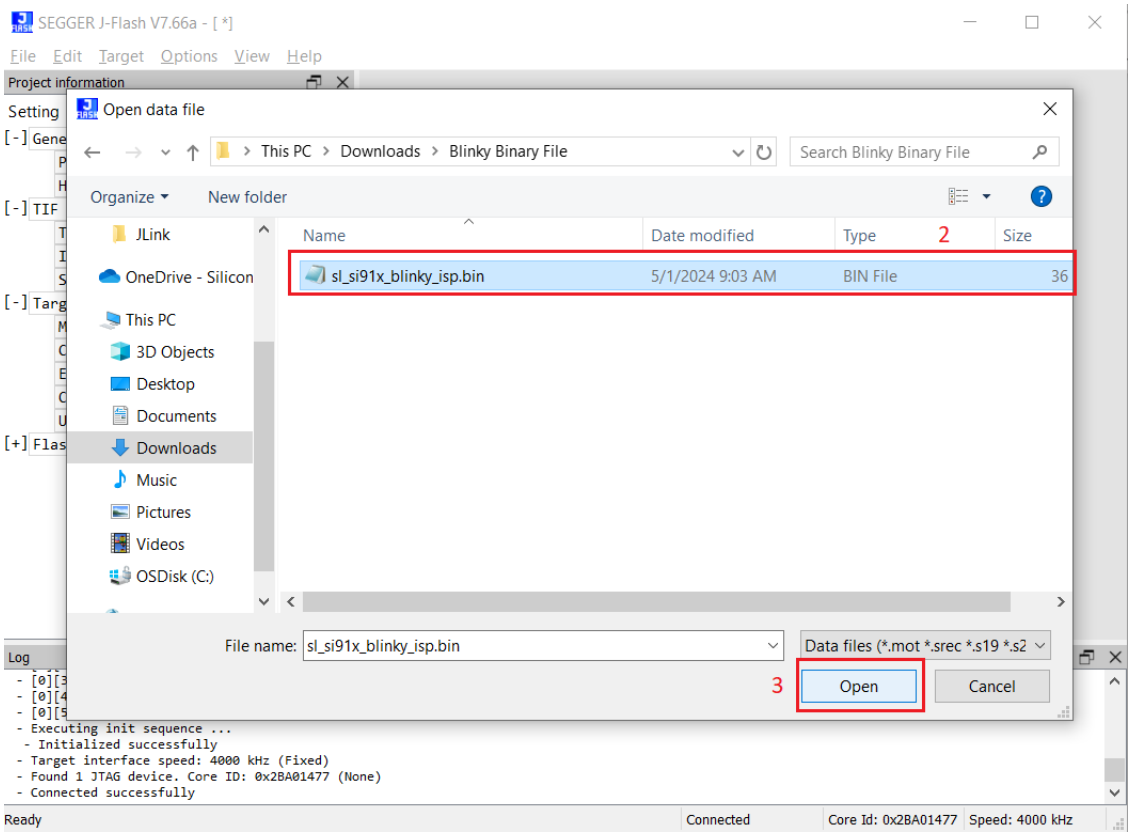


8. Upon successful connection, you will see **"Connected successfully"** in the log window as shown below.

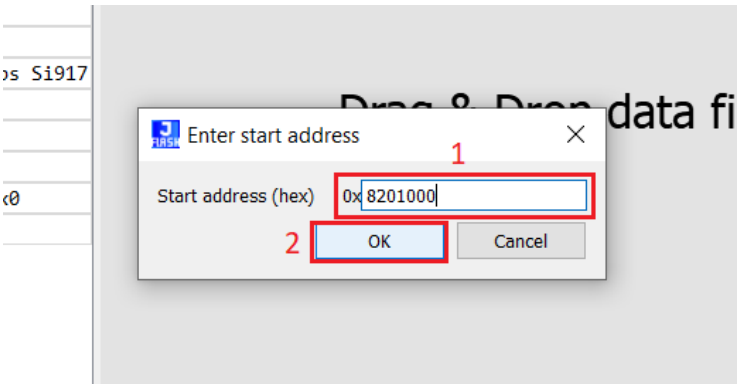


9. Click on **File** → **Open data file…**, and then go to the path where you downloaded the Blinky Binary file mentioned in Section 3.1.2 Software. Next, select the **sl_si91x_blinky_isp.bin** file and click on **Open**.

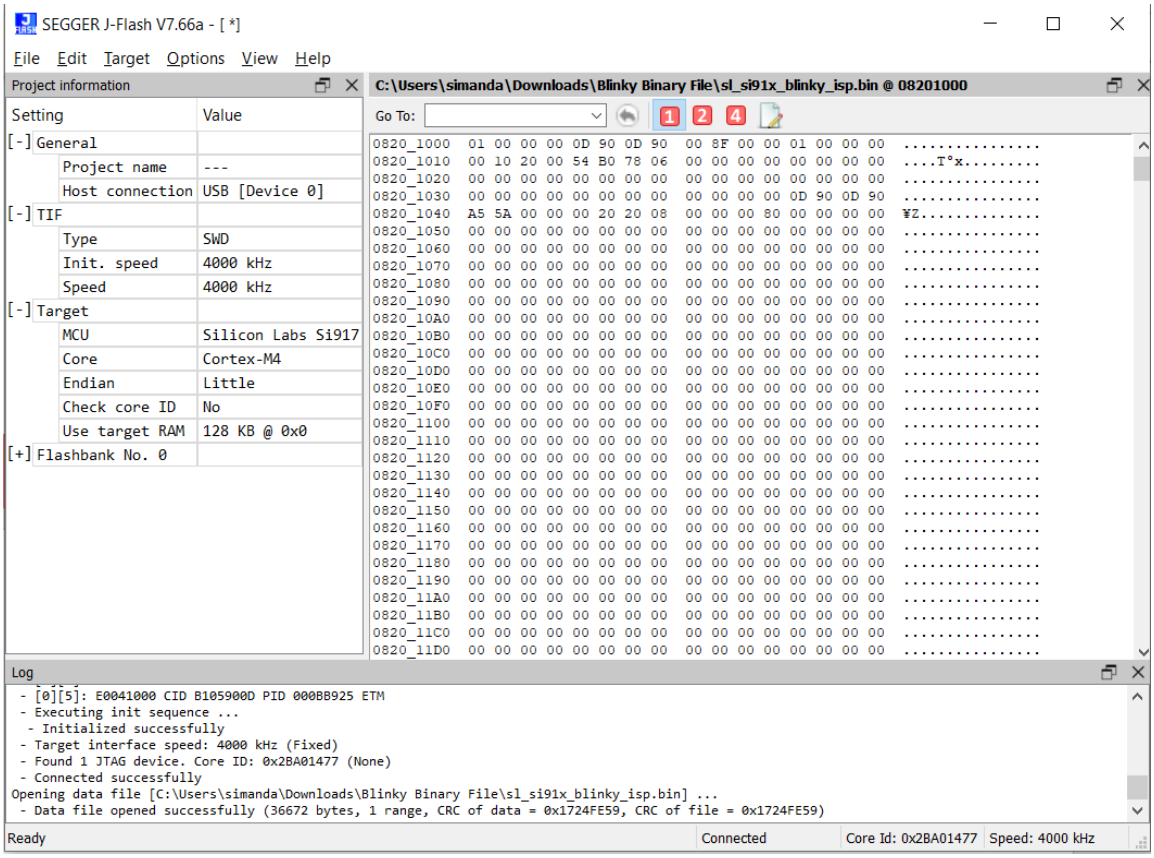




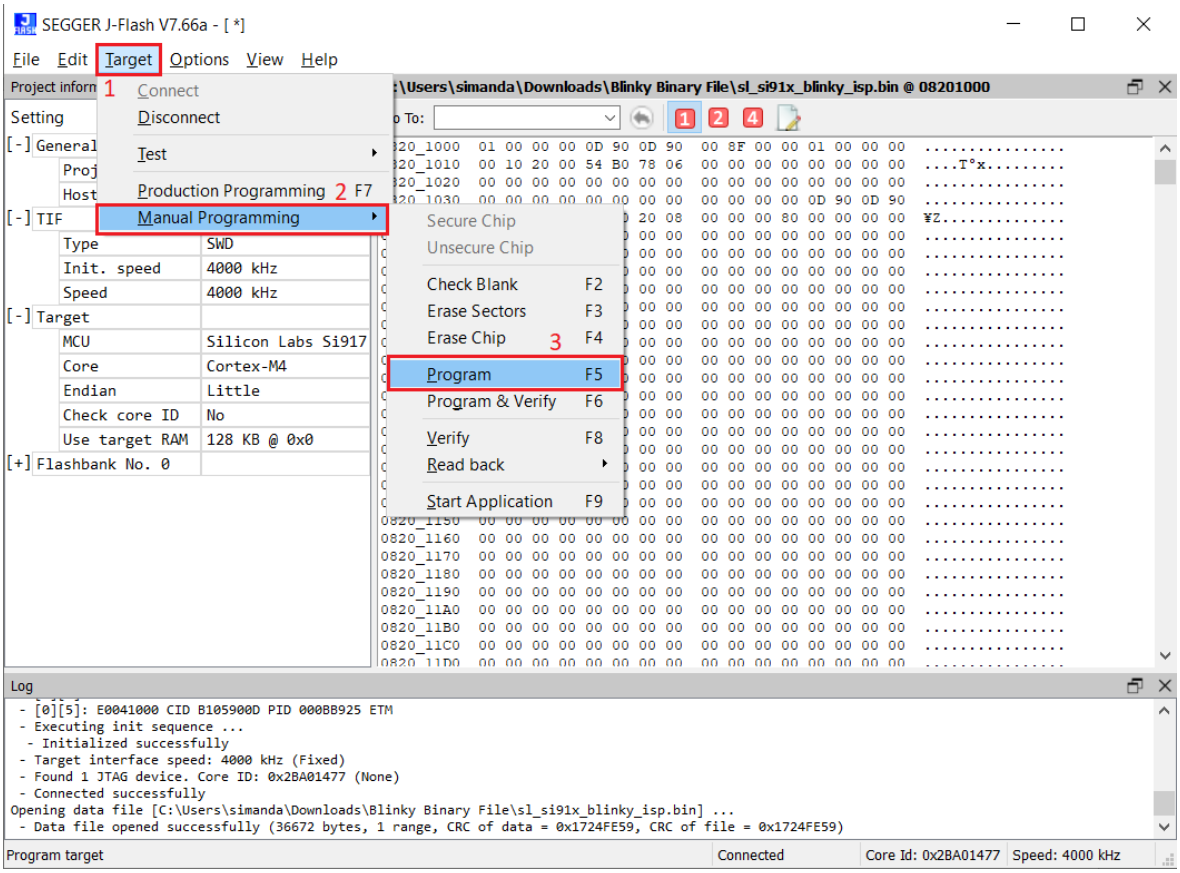
10. Next, in the Enter start address pop-up window, give the Start address as **0x8201000** (This is the M4 MBR Start address). Then, click on **OK**.



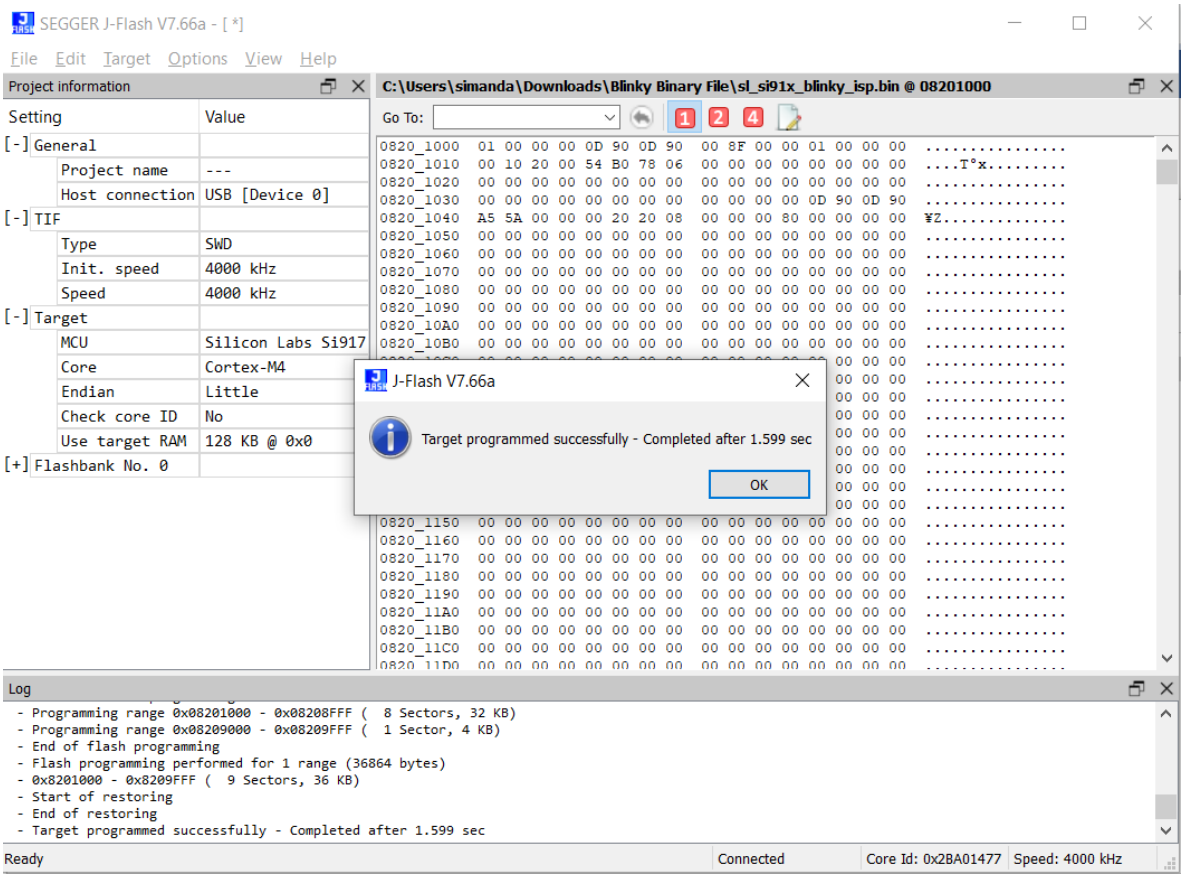
11. Upon successful opening of the binary file, the JFlash screen will be shown as below.



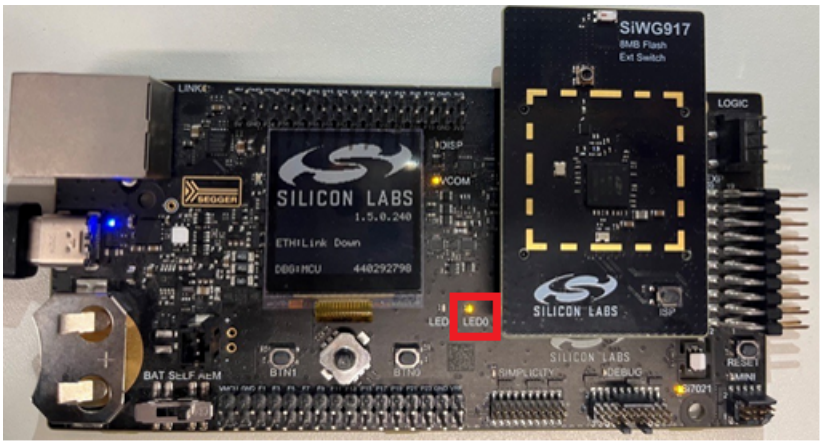
12. To flash the blinky application onto the SiWx917 device, click on **Target → Manual Programming → Program**.



13. Upon successful programming, you will see **"Target programmed successfully"**.



14. After programming successfully, you will see the LED0 on the board blinking continuously. This proves that the application is flashed through the JFlash and is running as expected.



4. SiWx917 Generic Flash APIs

This section explains how to configure and use flash APIs for the generic flash programmer.

The related software modules are found at **SiWx917_SoC_Flash_Loader\Source_Code\Si917_Flashloader\Src**. (This source code is provided in Section 3.1.2 Software.)

There are two main files:

- FlashDev.c
- FlashPrg.c

These two files are SEGGER template files. One can port these files to their system to make a flash programmer of their own.

The flash APIs provide complete low-level access to the flash memory and help to modify the flash memory contents. The APIs support flash memory erase, program, and read operations.

The code blocks (shown in the table below) play a major role in the flash programmer.

Code Block	Description
FlashOS.h	Contains all defines and prototypes of public functions
FlashDev.c	Flash device Description
FlashPrg.c	Implementation of RAM Code

4.1 Initialization

The initialization is done in the **FlashDev.c** file. The SEGGER template is used.

Though this is dummy, it is included for device description.

```
File    : FlashDev.c
Purpose : Flash device Description Template
*/
 #include "FlashOS.h"
struct FlashDevice_t const FlashDevice __attribute__ ((section ("DevDscr"))) = {
  FLASH_DRV_VERS,                 // Algo version. Must be == 0x0101
  { "Si917_Generic_Flash" },        // Flash device name
  ONCHIP,                                 // Flash device type. Must be == 1
  0x08000000,                             // Flash base address
  0x00800000,                             // Total flash device erase size in Bytes
  4096,                                        // Page Size (Will be passed as <NumBytes> to ProgramPage().
                                                          A multiple of this is passed as <NumBytes> to
SEGGER_OPEN_Program() to
                                                      program more than 1 page in 1 RAMCode call, speeding up
programming).
  0,                                         // Reserved, should be 0
  0xFF,                                   // Flash erased value
  50000000,                             // Program page timeout in ms
  50000000,                            // Erase sector timeout in ms
  {
    { 0x00001000, 0x00000000 },
    { 0xFFFFFFFF, 0xFFFFFFFF }    // Indicates the end of the flash sector layout. Must be present.
  }
};
```

4.2 Program the Device

In **FlashPrg.c**, implementation of the RAM code template is done. The file contains SEGGER template APIs.

Among all the APIs, the **ProgramPage()** and **EraseChip()** play a major role in programming the common flash SiWx917 device.

4.2.1 ProgramPage API

The code block shown below explains the ProgramPage API.

```
   *  ProgramPage
   *
   *  Function description: Programs one flash page.
   *  Parameters
   *    DestAddr    - Address to start programming on
   *    NumBytes  - Number of bytes to program. Guaranteed to be == <FlashDevice.PageSize>
   *    pSrcBuff     - Pointer to data to be programmed
   *  Return value
   *    == 0  O.K.
   *    == 1  Error
   */
   #define TA_RESET_ADDR      0x22000004

   int ProgramPage(U32 DestAddr, U32 NumBytes, U8 *pSrcBuff)
{
    int32_t status = 1;
      static int32_t x   = 0, size;
      uint32_t Imageheader[HEADER_LENGTH];
    *(uint32_t *)(TA_RESET_ADDR) = 0x0; //put TA in reset
   #ifdef DEBUG_OFL
  *(volatile uint32_t *)0x24048624 |= (1<<5);
   #endif
   memset(Imageheader,'\0', HEADER_LENGTH);
     if (x == 0)
    {
        memcpy(Imageheader, pSrcBuff, HEADER_LENGTH);
       if (!board_ready) {
            if((uint32_t)(Imageheader[0] & IMAGE_TYPE_MASK) == (uint32_t)TA_IMAGE)
            {
                status = rsi_device_init(BURN_NWP_FW);
            } else
            {
                status = rsi_device_init(RSI_UPGRADE_IMAGE_I_FW);
            }
         if (status != RSI_SUCCESS) {
           return status;
         }
       }
             size = Imageheader[2];
           size = (size) / CHUNK_SIZE;
       status = rsi_bl_upgrade_firmware(pSrcBuff, NumBytes, 1);
        size--;
        x = 1;
      #ifdef DEBUG_OFL
       *(volatile uint32_t *)0x24048624 &= ~(1<<5);
      #endif
       return status;
     }
     if (size == 0) {
       status = rsi_bl_upgrade_firmware(pSrcBuff, NumBytes, 2);
        x  = 0;
     } else {
       status = rsi_bl_upgrade_firmware(pSrcBuff, NumBytes, 0);
       size--;
     }
   #ifdef DEBUG_OFL
       *(volatile uint32_t *)0x24048624 &= ~(1<<5);
   #endif
     return status;
   }
```

Function Description

- Mandatory function. Must be present to make OFL(Open Flash Loader) detected as valid.
- Programs flash. The block passed to this function is always a multiple of what is indicated as page size by FlashDevice.PageSize.
- This function can rely on only being called with destination addresses and NumBytes that are aligned to FlashDevice.PageSize.
- In this function, we call the boot loader APIs, to load the M4 and TA firmware for the common flash radio board.

API: rsi_device_init

Source File: rsi_device_init_apis_flm.c

Prototype:

```
int32_t rsi_device_init(uint8_t select_option);
```

Description

This API power cycles the module and sets the boot up option for module features. This API also initializes the module SPI.

Parameter

Parameter	Description
select_option	RSI_LOAD_IMAGE_I_FW : To load Firmware image RSI_LOAD_IMAGE_I_ACTIVE_LOW_FW : To load active low Firmware image. Active low firmware will generate active low interrupts to indicate that packets are pending on the module, instead of the default active high. RSI_UPGRADE_IMAGE_I_FW : To upgrade firmware file ERASE_COMMON_FLASH : To Erase the Common flash region.

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	-1: Failure

Example

```
if (!board_ready)
{
     status = rsi_device_init(RSI_UPGRADE_IMAGE_I_FW);
     if (status != RSI_SUCCESS) {
     return status;
}
```

API: rsi_bl_select_option

Source File: rsi_device_init_flm.c

Prototype:

```
int16_t rsi_bl_select_option(uint8_t cmd);
```

Description

This API is used to send firmware load requests to TA or update default configurations.

Parameter

Parameter	Description
cmd	Type of configuration to be loaded

Return Values

Value	Description
0	Success
<0	Failure

Example

```
uint8_t cmd = LOAD_NWP_FW;

status = rsi_bl_select_option(cmd);
```

API: rsi_bl_upgrade_firmware

Source File: rsi_device_init_flm.c

Prototype:

```
int16_t rsi_bl_upgrade_firmware(uint8_t *firmware_image, uint32_t fw_image_size, uint8_t flags);
```

Description

- This API upgrades the firmware in the module device from the host. The firmware file is given in chunks to this API.
- Each chunk must be a multiple of 4096 bytes unless it is the last chunk.
- For the first chunk, set RSI_FW_START_OF_FILE in flags.
- For the last chunk set RSI_FW_END_OF_FILE in flags.

Parameters

Parameter	Description
firmware_image	This is a pointer to firmware image buffer
flags	1 - RSI_FW_START_OF_FILE 2 - RSI_FW_END_OF_FILE Set flags to 1 - If it is the first chunk 2 - If it is last chunk, 0 - For all other chunks
fw_image_size	This is the size of firmware image

Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	-1: Failure

Example

```
rsi_bl_upgrade_firmware(fw_image, FW_IMG_SIZE, 1);
```

API: rsi_bootloader_instructions

Source File: rsi_device_init_flm.c

Prototype:

```
int16_t rsi_bootloader_instructions(uint8_t type, uint16_t *data);
```

Description

This API is used to send boot instructions to TA.

Parameters

Parameter	Description
type	Type of the instruction to perform
data	Pointer to data which is to be read/write

Return Values

Value	Description
0	Success
Non Zero Value	Failure

Example

```
status = rsi_bootloader_instructions(RSI_REG_READ, data);
```

4.2.2 EraseChip APIs

The below code block explains the EraseChip operation.

```
/*       EraseChip
*
*  Function description: Erases the entire flash.
*
*  Return value
*    == 0  O.K.
*    == 1  Error
*/
#if SUPPORT_ERASE_CHIP
 int EraseChip(void) {
    int status =RSI_SUCCESS;
        //Send the common flash erase command to the TA
        status = rsi_device_init(ERASE_COMMON_FLASH);
        if(status==RSI_SUCCESS)
            return RSI_OK;
        else
            return 0;
}
#endif
```

Function description:

- This function is used to Erases the entire flash for the common flash radio board.

API: rsi_device_init

Source File: rsi_device_init_apis_flm.c

Prototype :

```
int32_t rsi_device_init(uint8_t select_option);
```

Description

This API power cycles the module and sets the boot up option for module features. This API also initializes the module SPI.

Parameter

Parameter	Description
select_option	**ERASE_COMMON_FLASH**: To Erase the Common flash region.

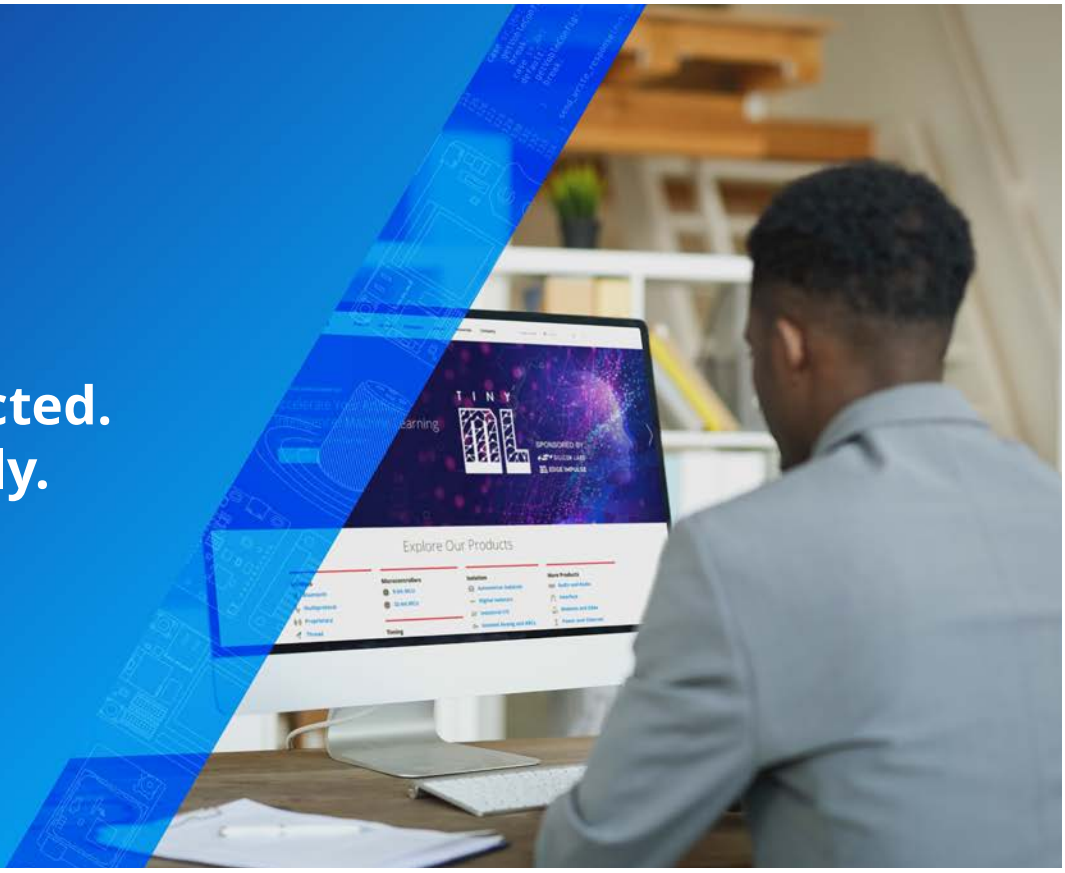
Return Values

Value	Description
0	Successful execution of the command
Non Zero Value	-1: Failure

Example

```
status = rsi_device_init(ERASE_COMMON_FLASH);
         if(status==RSI_SUCCESS)
            return RSI_OK;
```

Smart. Connected. Energy-Friendly.



IoT Portfolio
www.silabs.com/products

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

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

www.silabs.com