Igor Kuzmich

FreeRTOS on Metro m0 Express

Software Used:

- Atmel Studio 7
  - Used as IDE
  - Download Page: https://www.microchip.com/mplab/avr-support/atmel-studio-7
- BOSSA
  - Use to flash board through a USB connection:
  - Download Page: http://www.shumatech.com/web/products/bossa
  - bossac.exe , the terminal version was used for this lab
- Windows
  - This project was performed using windows
  - Since all tools are either open source or have a linux alternative, the build can easily be ported to a linux system
- FreeRTOS
  - Version 10.2.0 and lower used in this project
  - Product Page: https://www.freertos.org/
  - Product Download Page: https://www.freertos.org/a00104.html
- Adafruit Metro m0 Express Bootloader
  - Latest can be found at : https://github.com/adafruit/uf2-samdx1/releases/tag/v3.5.0

Hardware Used:

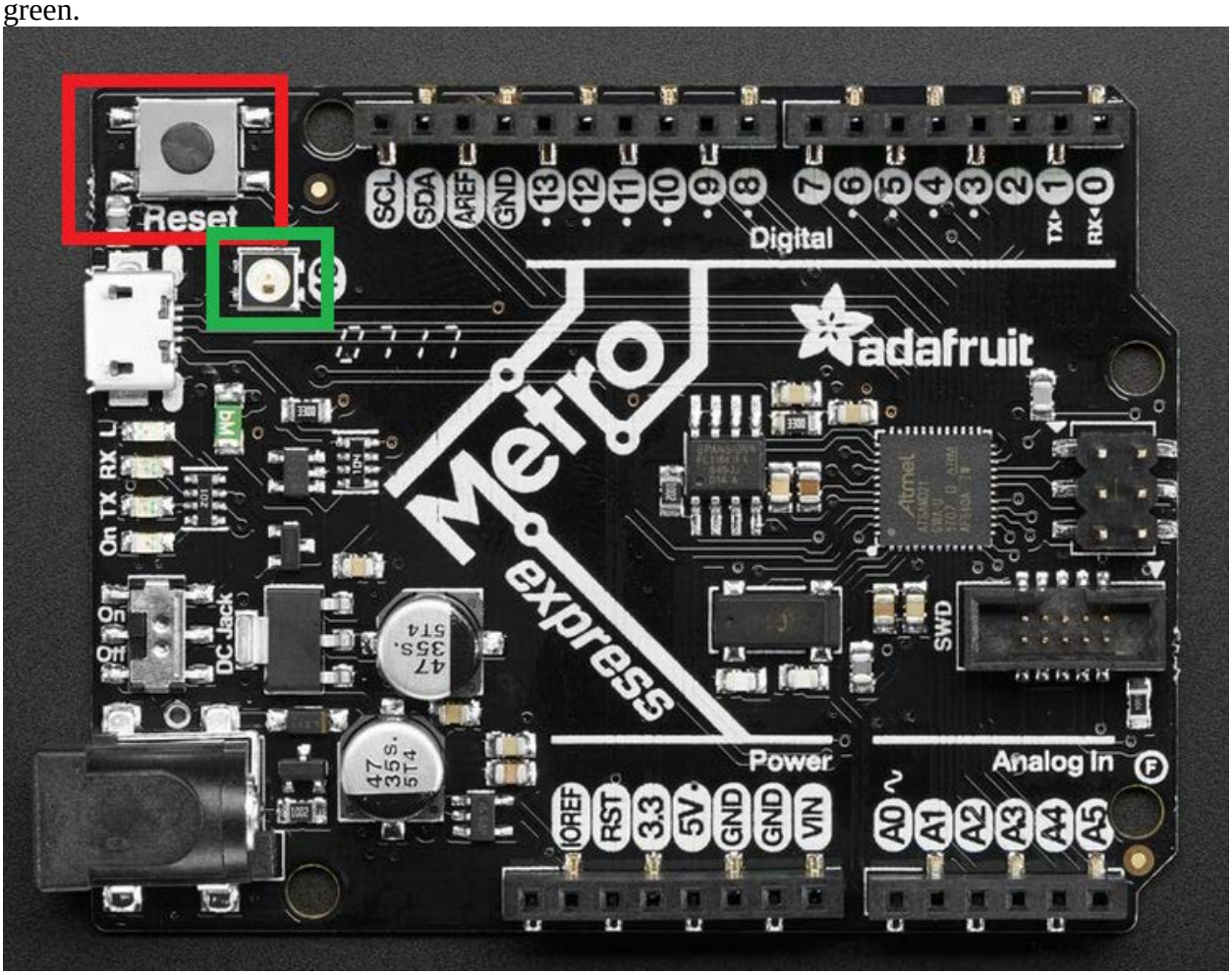
- Adafruit Metro m0 Express
  - Board Used
  - Product Page: https://www.adafruit.com/product/3505
- J-link EDU:
  - Product Page: https://www.segger.com/products/debug-probes/j-link/models/j-link-edu/
  - Used to reflash the bootloader whenever bug overwrites it, or bootloader is accidently overwritten by the user incorrectly using BOSSA
  - Can be also used for debugging purposes

Using BOSSA:

BOSSA is used to flash our program onto the board through a USB connection. To use BOSSA we must first figure out the port our board is located on. To do this we must run the “mode” command

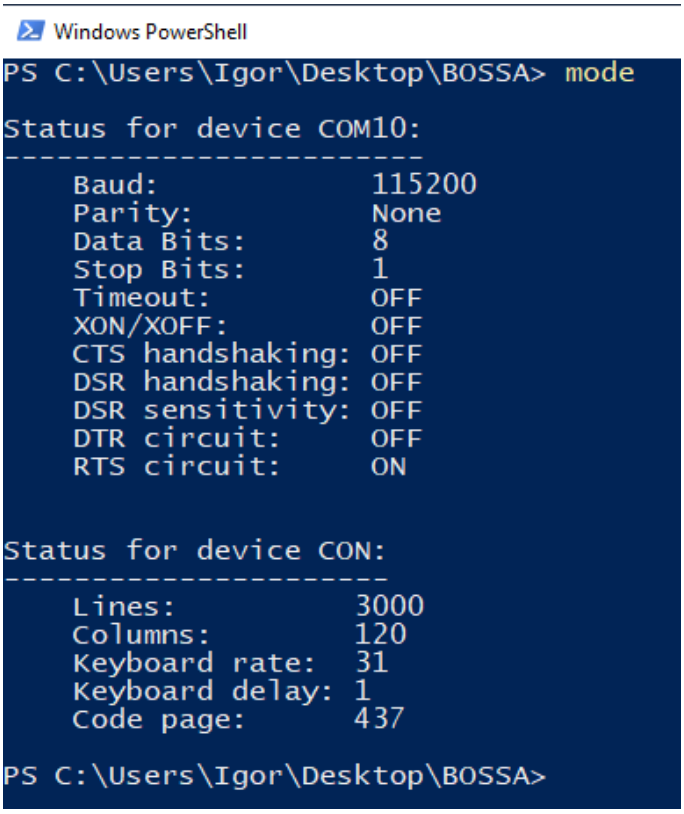
```
Windows PowerShell
PS C:\Users\Igor\Desktop\BOSSA> mode

Status for device CON:
----------------------
    Lines:          3000
    Columns:        134
    Keyboard rate:  31
    Keyboard delay: 1
    Code page:      437

PS C:\Users\Igor\Desktop\BOSSA>
```

However, we must first double tap the reset button for our board to show up. Doing so puts our board in programming Mode. This can be seen by the RGB on the board lighting up green.

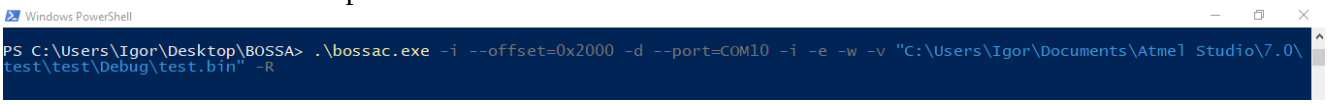


The reset button in highlighted in red, and the pixel in green.

Once the board is in programming mode, the command “mode” should be able to tell you what port to use:



Once we know the port we can run the command: “bossac.exe” to flash the board



.\bossac.exe -i --offset=0x2000 -d --port=COM10 -i -e -w -v "C:\Users\Igor\Documents\Atmel Studio\7.0\test\test\Debug\test.bin" -R

Be sure to change the location in in quotes and port number

These are the possible parameters:

Windows PowerShell

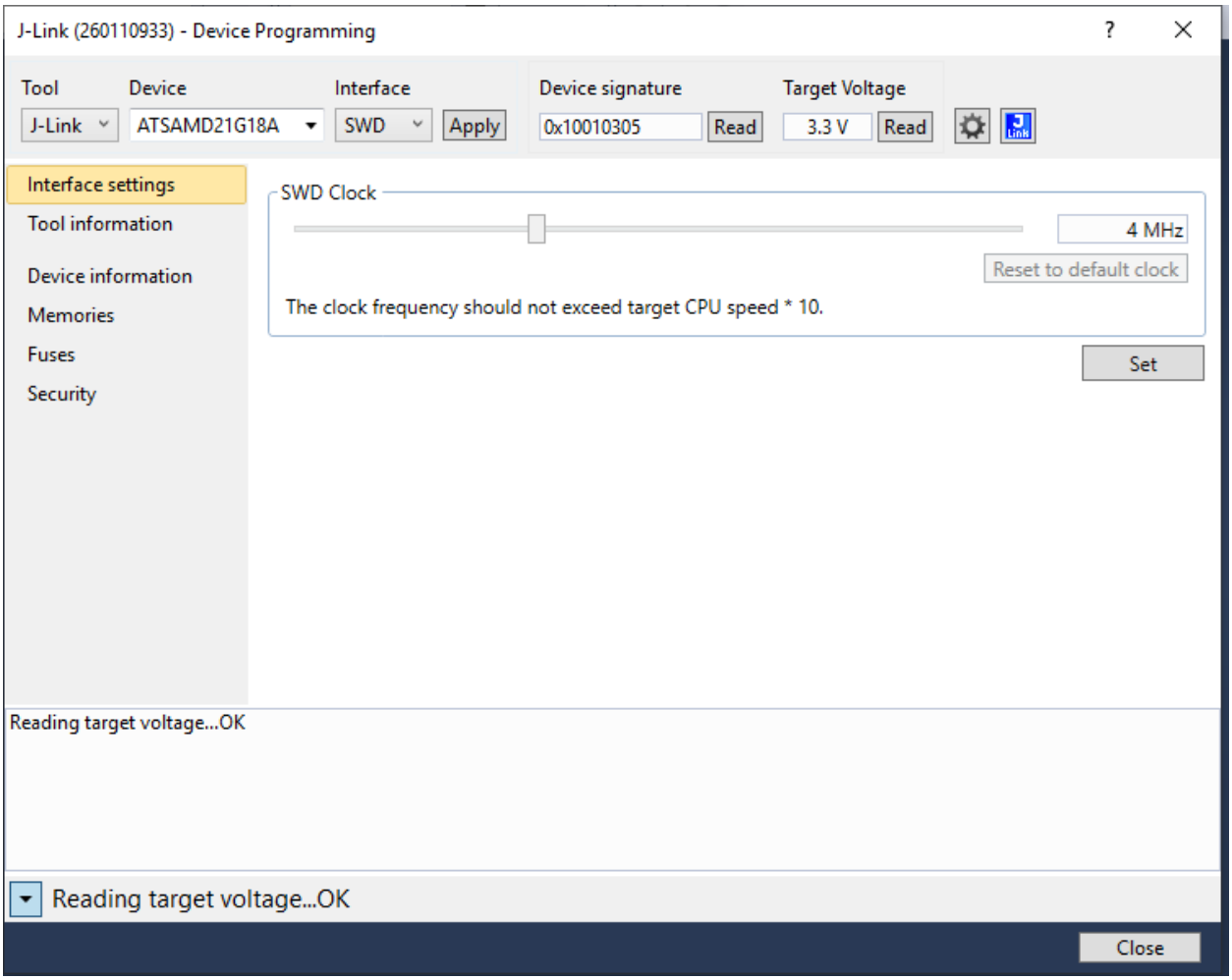
```
PS C:\Users\Igor\Desktop\BOSSA> .\bossac.exe -h
Usage: bossac.exe [OPTION...] [FILE]
Basic Open Source SAM-BA Application (BOSSA) Version 1.9.1
Flash programmer for Atmel SAM devices.
Copyright (c) 2011-2018 ShumaTech (http://www.shumatech.com)

Examples:
  bossac -e -w -v -b image.bin   # Erase flash, write flash with image.bin,
                                 # verify the write, and set boot from flash
  bossac -r0x10000 image.bin     # Read 64KB from flash and store in image.bin

Options:
  -e, --erase           erase the entire flash starting at the offset
  -w, --write           write FILE to the flash; accelerated when
                        combined with erase option
  -r, --read[=SIZE]     read SIZE from flash and store in FILE;
                        read entire flash if SIZE not specified
  -v, --verify          verify FILE matches flash contents
  -o, --offset=OFFSET   start erase/write/read/verify operation at flash OFFSET;
                        OFFSET must be aligned to a flash page boundary
  -p, --port=PORT       use serial PORT to communicate to device;
                        default behavior is to use first serial port
  -b, --boot[=BOOL]     boot from ROM if BOOL is 0;
                        boot from FLASH if BOOL is 1 [default];
                        option is ignored on unsupported devices
  -c, --bod[=BOOL]      no brownout detection if BOOL is 0;
                        brownout detection is on if BOOL is 1 [default]
  -t, --bor[=BOOL]      no brownout reset if BOOL is 0;
                        brownout reset is on if BOOL is 1 [default]
  -l, --lock[=REGION]   lock the flash REGION as a comma-separated list;
                        lock all if not given [default]
  -u, --unlock[=REGION] unlock the flash REGION as a comma-separated list;
                        unlock all if not given [default]
  -s, --security        set the flash security flag
  -i, --info            display device information
  -d, --debug           print debug messages
  -h, --help            display this help text
  -U, --usb-port[=BOOL] force serial port detection to USB if BOOL is 1 [default]
                        or to RS-232 if BOOL is 0
  -R, --reset           reset CPU (if supported)
  -a, --arduino-erase   erase and reset via Arduino 1200 baud hack
```

The main ones to keep track of is that “—offset=” must be set to 0x2000 as to not overwrite the bootloader. “—port=” must be set to the port the board is on, in this case it is set to “COM10”.

After running the command our program should be flashed to the board and running. The board remaining in programming mode, or unable to be put into programming mode by double tapping reset, is usually a sign that there is a bug in the program, or that the board was incorrectly flashed.

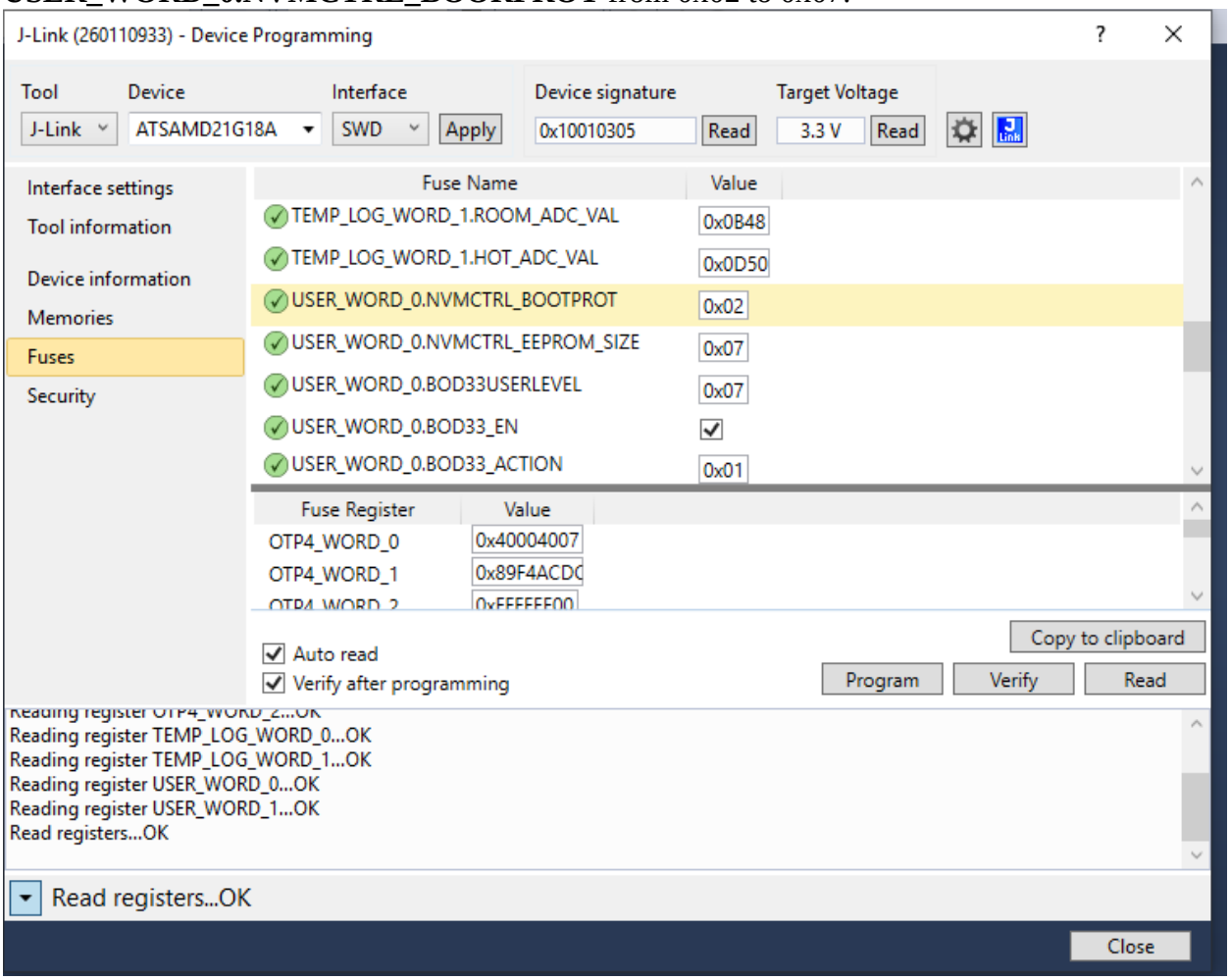
Using J-link to Flash Boot-Loader:

In Atmel Studio, go to **Tools** and then **Device Programming**

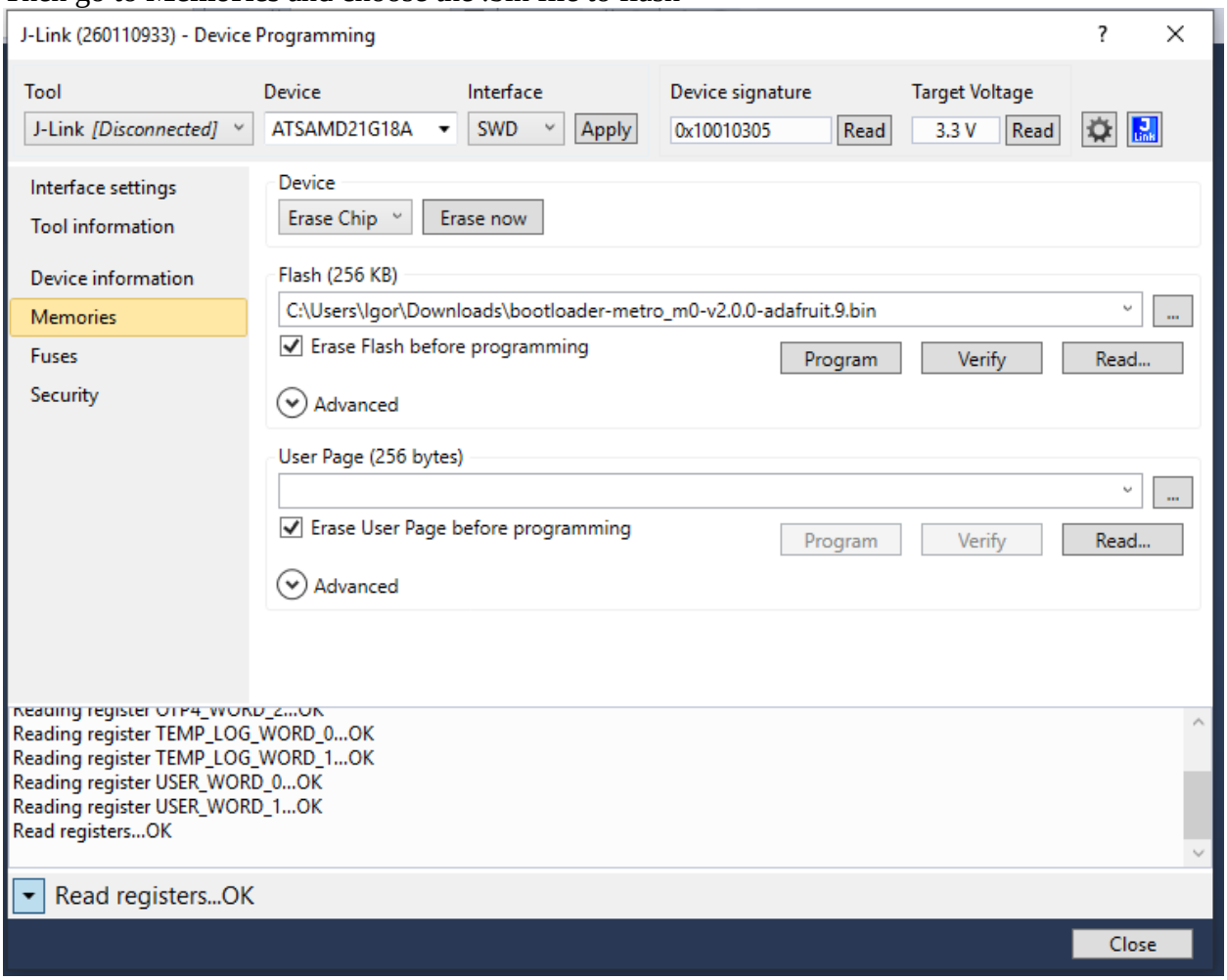


Set the correct tool (j-link), device, interface, hit apply, read, read. If the device is correctly configured and connected, values should be displayed in device signature and target voltage.

In order to load a program, or bootloader go to **Fuses** and change **USER_WORD_0.NVMCTRL_BOORPROT** from 0x02 to 0x07.



Then go to **Memories** and choose the .bin file to flash

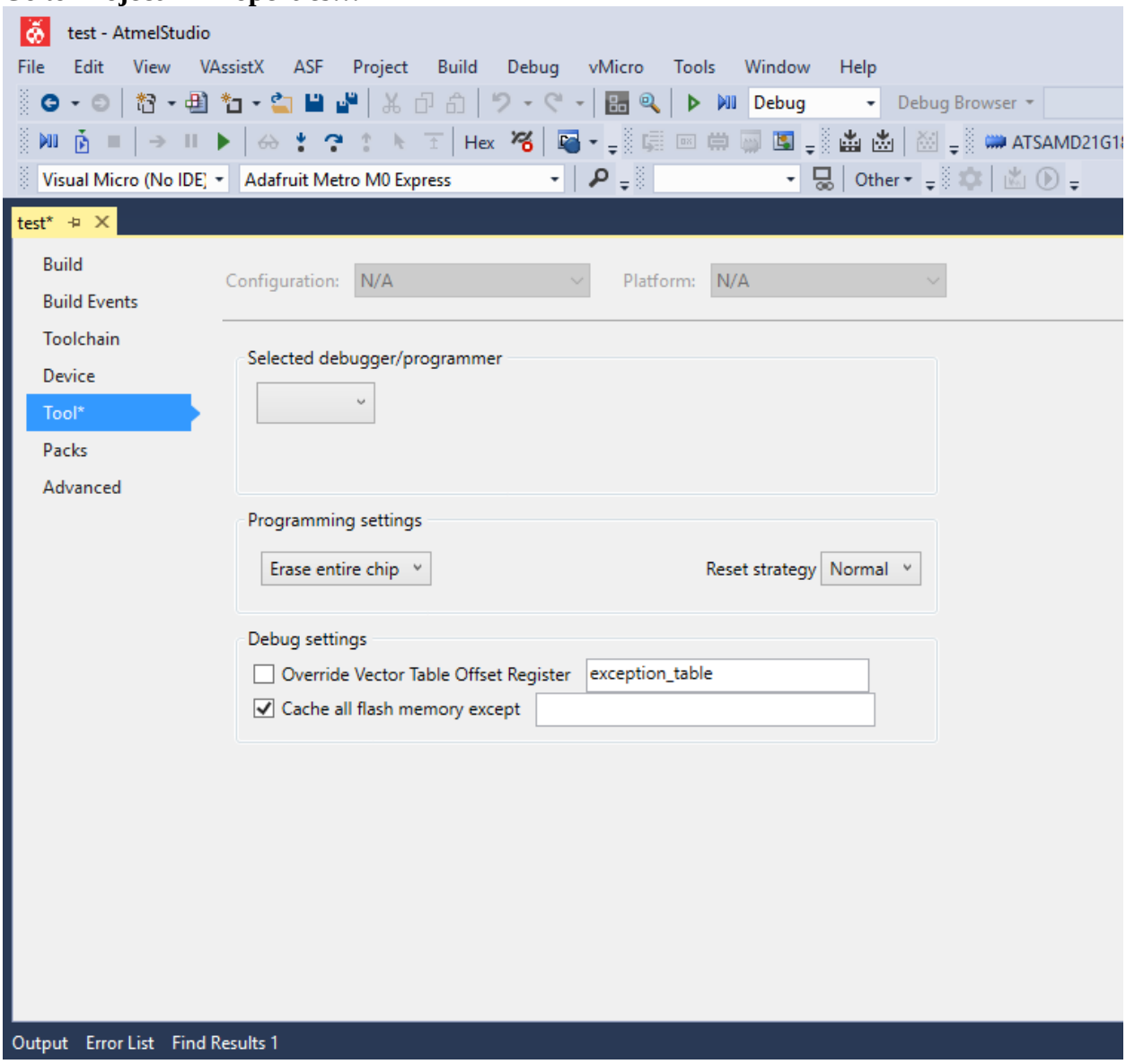


Then hit Program and then Verify.

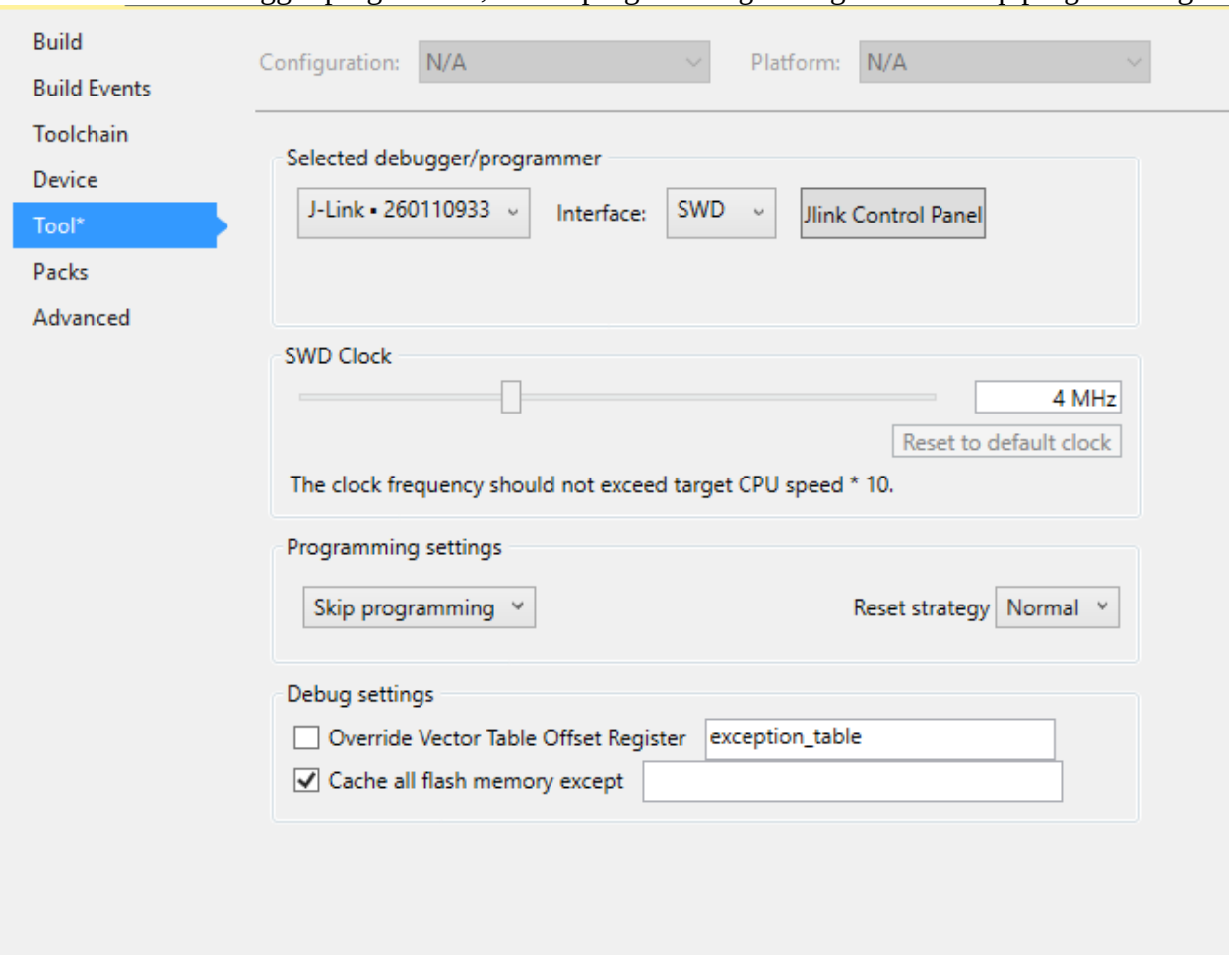
Finally Restore **USER_WORD_0.NVMCTRL_BOORPROT** back to 0x02

Using J-Link as debugger:

Go to **Project -> Properties…**



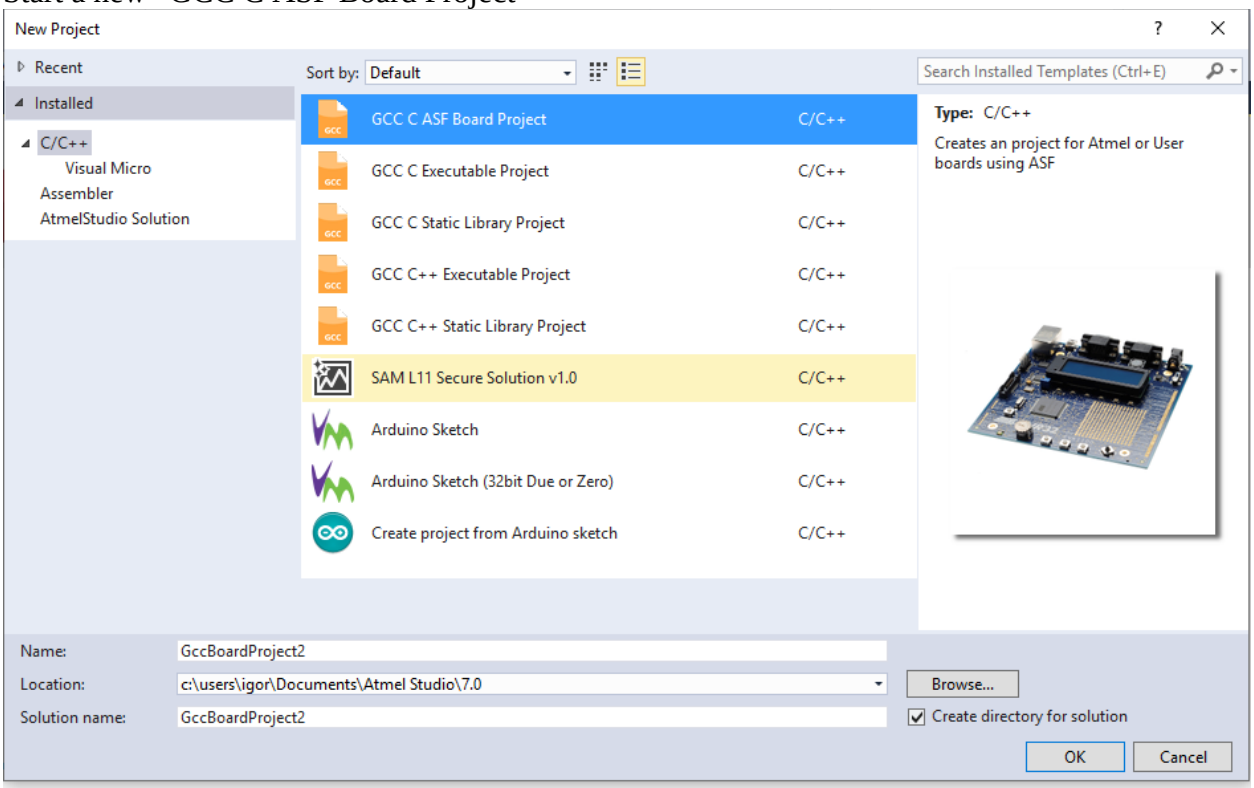
Select J-link as debugger/programmer, and in programming settings choose skip programming



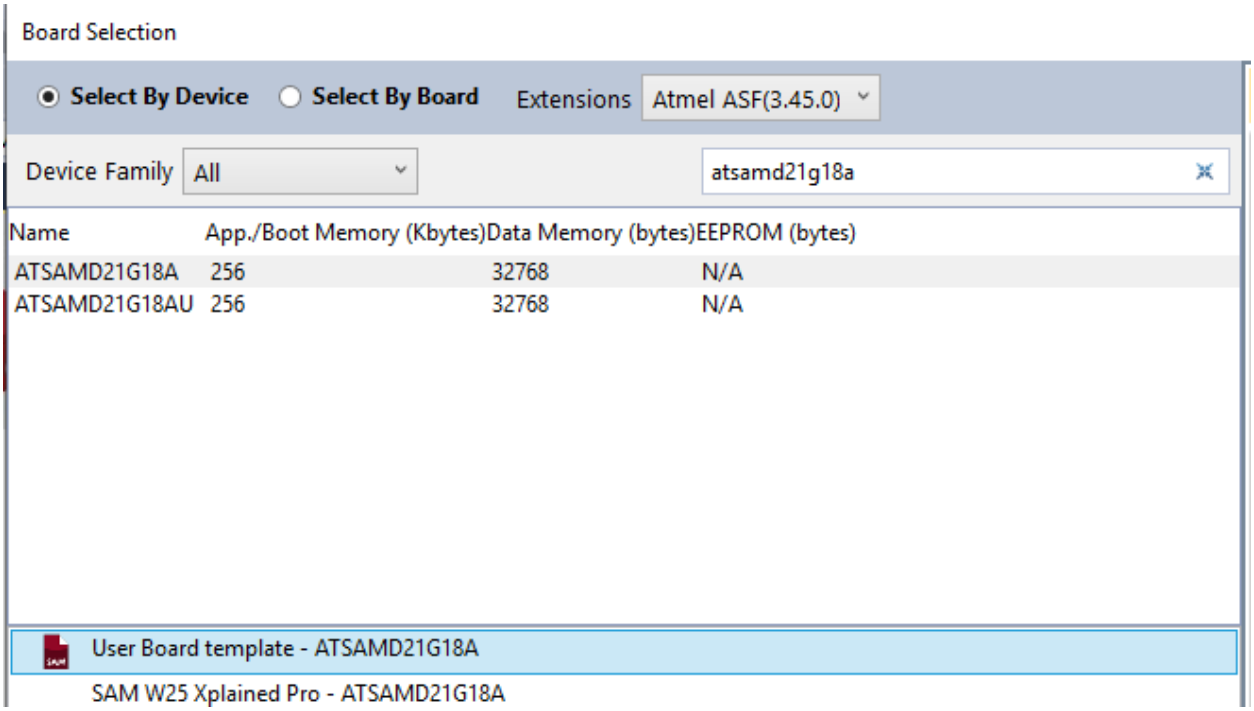
You can now debug by running **Start Debugging and Break.** 

To Start New Project for Adafruit Metro m0 Express using Atmel Studio:

Start a new “GCC C ASF Board Project”



Choose the “atsamd21g18a” processor and “User Board Template – ATSAMD21G18A” for the board.



In Solution Explorer go to ..\src\ASF\sam0\utils\linker_scripts\samd21\gcc\samd21g18a_flash.ld

Add 0x2000 to rom ORIGIN and subtract 0x2000 for rom Length for the 8kb bootloader

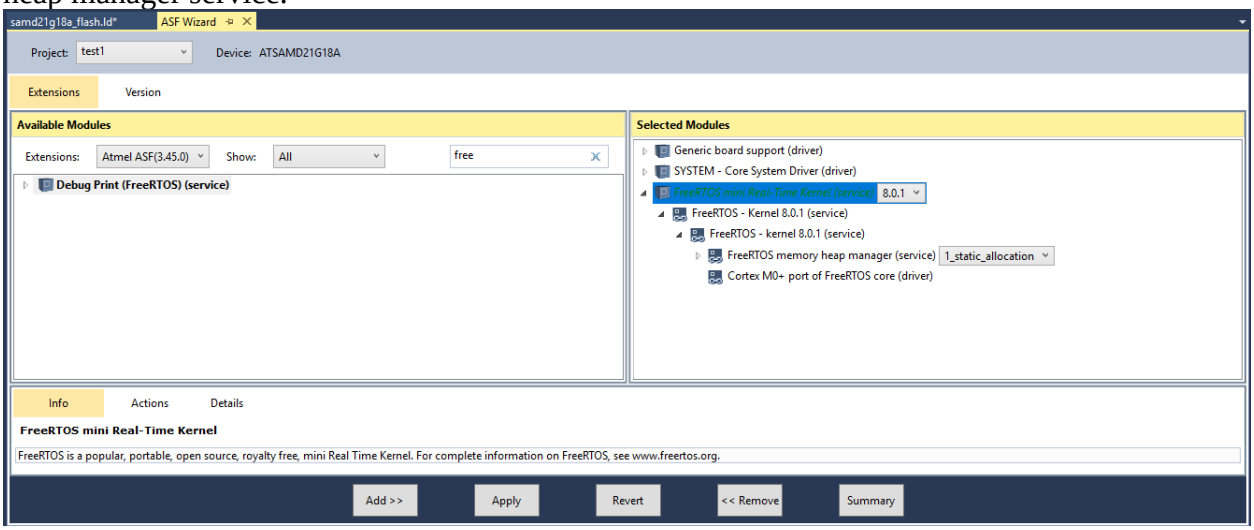
```
/* Memory Spaces Definitions */
MEMORY
{
  rom      (rx)  : ORIGIN = 0x00000000+0x2000, LENGTH = 0x00040000-0x2000
  ram      (rwx) : ORIGIN = 0x20000000, LENGTH = 0x00008000
}
```

You are now ready to program the board. Build with Atmel Studio then flash with BOSSA.

FreeRTOS 7.4.2 – 8.0.1

Higher versions from Atmel Studio run into memory problems.

Go to ASF Wizard, search and add FreeRTOS mini Real-Time Kernel. Can choose any heap manager service.



Apply.

You can now program using FreeRTOS.

To implement my project.

Go to ASF Wizard and Add “IOPORT – General purpose I/O service (service)”

Import and replace your main.c with main.c from
“Cortex_m0+_ATSAMD21G18A_metro_m0_Express” project

Delete the following declaration and definitions:

```
void vApplicationMallocFailedHook( void );
void vApplicationIdleHook( void );
void vApplicationStackOverflowHook( TaskHandle_t pxTask, char *pcTaskName );
void vApplicationTickHook( void );
unsigned long ulMainGetRunTimeCounterValue( void );
void vMainConfigureTimerForRunTimeStats( void );
/* Used in the run time stats calculations. */
static unsigned long ulClocksPer10thOfAMilliSecond = 0UL;
```

You can now build and flash the system onto the board.

FreeRTOS 10.2.0

To implement a new FreeRTOS version, FreeRTOS suggests using an implemented project and building off that. In this case "CORTEX_M0+_Atmel_SAMD20_XPlained" was used to implement FreeRTOS on the adafruit metro m0 board. If implementing newer version of FreeRTOS onto the metro m0 board, it might be useful to again reimplement another project other than building off of this one.

To create a new project, follow "**To Start New Project for Adafruit Metro m0 Express using Atmel Studio"** describe previously.

After creating a new project, download the FreeRTOS source code and import everything into the project. For 10.2.0 this includes (files with /* mean include everything in the folder

- ../source/include/*
- ../source/portable/GCC/ARM_CM0/*
- ../source/portable/GCC/MemMang/heap_4.c

Go to **Projects -> Properties -> Toolchain -> ARM/GNU C Compiler** and add the previously added directories to the compiler directories

FreeRTOS 10.2.0 was only tested with heap_4.c, however I don't see why others shouldn't work. https://www.freertos.org/a00111.html describes what each heap file does.

Go to "..\FreeRTOS\Demo\CORTEX_M0+_Atmel_SAMD20_XPlained\RTOSDemo\src\main.c" and "..\FreeRTOS\Demo\CORTEX_M0+_Atmel_SAMD20_XPlained\RTOSDemo\src\config\FreeRTOSConfig.h"if you've downloaded FreeRTOS 10.2.0 with the demos.

Import FreeRTOSConfig.h, and copy these function declarations and definitions from main.c to your own main.c

```
void vApplicationMallocFailedHook( void );
void vApplicationIdleHook( void );
void vApplicationStackOverflowHook( TaskHandle_t pxTask, char *pcTaskName );
void vApplicationTickHook( void );
unsigned long ulMainGetRunTimeCounterValue( void );
void vMainConfigureTimerForRunTimeStats( void );
/* Used in the run time stats calculations. */
static unsigned long ulClocksPer10thOfAMilliSecond = 0UL;
```

Finally add

```
#ifndef portHAS_STACK_OVERFLOW_CHECKING
    #define portHAS_STACK_OVERFLOW_CHECKING 0
#endif
```

To FreeRTOS.h and portable.h

You should now be able to program using FreeRTOS 10.2.0.

To implement my program, go to ASF Wizard and Add “IOPORT – General purpose I/O service (service)”. Then simply delete main.c and import main.c from “Cortex_m0+_ATSAMD21G18A_metro_m0_Express” project.

To configure FreeRTOS:

FreeRTOSConfig - https://www.freertos.org/a00110.html

Other:

The FreeRTOS was implemented at the bare minimum to get the LEDs to blink using multitasking. It is unknown whether other services need to be reimplemented or work.