Eta Compute

ECM3532 AI Sensor Board

User Guide

ECM3532-AISENSOR–UG

Revision 0.1

May 2020

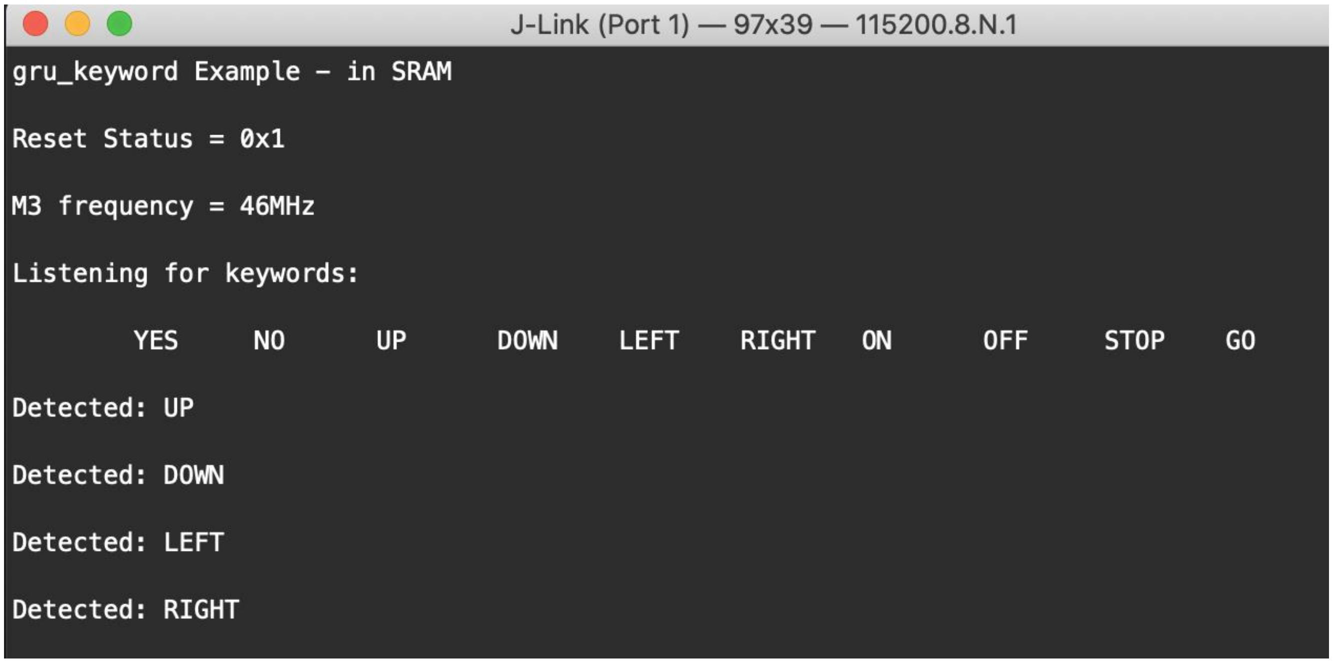
Table of Contents

1. Introduction

This document serves as the reference guide for working with the ECM3532 AI Sensor board (ASB).

Programmed Example

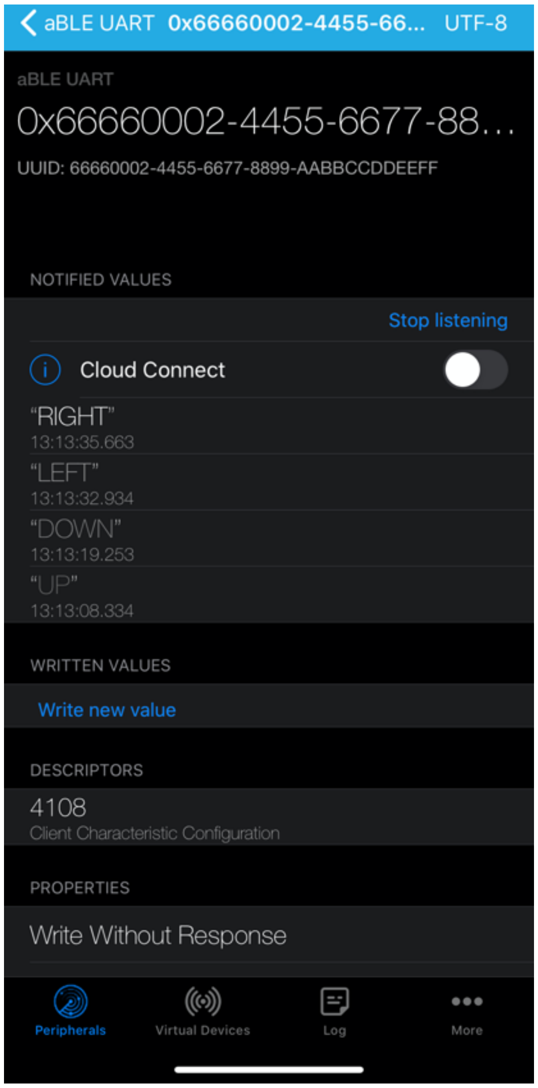
Each ECM3532 (ASB) is shipped with the 'gru_keyword' example programmed in Flash and detects the following keywords: Yes, No, Up, Down, Left, Right, On, Off, Stop and Go. It will also detect when an unknown word is spoken and when there is silence. This example copies itself into SRAM and executes from SRAM. Figure 1.1 shows the message printed to the screen upon application boot and detection of "UP", "DOWN, "LEFT", and "RIGHT".



```
gru_keyword Example - in SRAM

Reset Status = 0x1

M3 frequency = 46MHz

Listening for keywords:

        YES     NO      UP      DOWN    LEFT    RIGHT   ON      OFF     STOP    GO

Detected: UP

Detected: DOWN

Detected: LEFT

Detected: RIGHT
```

Figure 1.1. GRU Print Message

The detected keyword is also sent over Bluetooth through the BLE GATT profile as shown below in Figure 1.2.


aBLE UART 0x66660002-4455-66... UTF-8

aBLE UART

0x66660002-4455-6677-88...

UUID: 66660002-4455-6677-8899-AABBCCDDEEFF

NOTIFIED VALUES

Stop listening

Cloud Connect

"RIGHT"
13:13:35.663

"LEFT"
13:13:32.934

"DOWN"
13:13:19.253

"UP"
13:13:08.334

WRITTEN VALUES

Write new value

DESCRIPTORS

4108
Client Characteristic Configuration

PROPERTIES

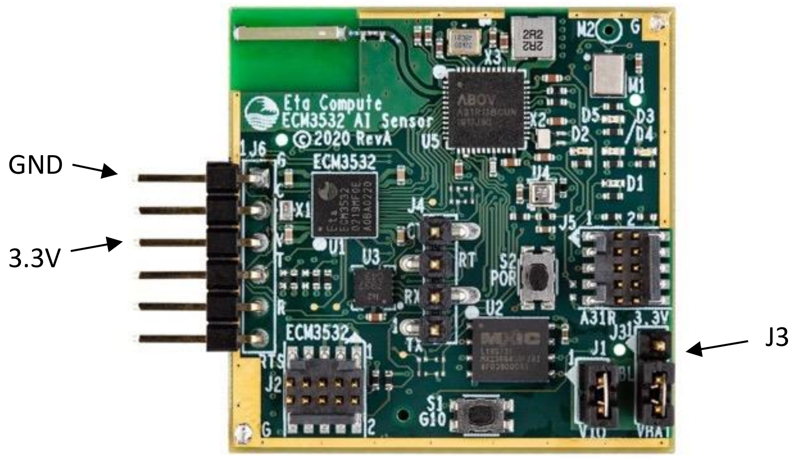
Write Without Response

Figure 1.2: Keyword Sent over BLE

For more information about the keyword spotting example and Bluetooth communication on this board, please contact Eta Compute.

2. *Quick Start*

1. **Install a CR2032 battery or apply 3.3V to J6 pin 3 and GND to J6 pin 1**



2. **Ensure J3 is selected for the correct power source proved per #1 above.**
3. **To observe UART prints, connect a serial to USB cable to J6. Only the RX (pin 5) and GND pins are needed as no flow control is used.**
4. **Bring up a serial terminal (PuTTY) on the correct COM port.**
    a. **Set for 115200, 1 stop, no parity**
    b. **Press/release SW2 (POR). This will reset the device and start the application.**
    c. **You should see an output that looks similar to Figure 1.**
5. **The example that is running is a GRU, keyword spotting, neural network developed by Eta Compute. It takes PDM data from a single mic channel and performs MRCG feature extraction on the DSP. It then DMAs the features to the M3 to run the GRU and provide a detected keyword.**
6. **Connect to the Bluetooth with a Bluetooth application for instance LightBlue:**
    a. **Select “aBLE UART”**
    b. **Select “Properties: Write Without Response Notify”**
    c. **To begin listening for notifications select *“Listen for notifications”***
    d. **To view the ASCII representation, click on *“Hex”* and select *“UTF-8 String”***
    e. **The detected keyword will appear as a string notification**
7. **For more examples and source code, the TENSAI® SDK contains example projects for:**
    a. **SEGGER’s Embedded Studio**
    b. **ARM’s KEIL MDK**
    c. **IAR’s Embedded Workbench**
    d. **Makefiles for compiling with GCC**

3. Overview

The ECM3532 ASB showcases the capabilities of the ECM3532 built with our self-timed continuous voltage and frequency scaling technology (CVFS).). The SoC includes an ARM Cortex-M3, CoolFlux Dual MAC DSP, an always-on block and numerous peripheral controllers. The board also contains an ABOV A31R118 Bluetooth device for RF connectivity over Bluetooth v4.2. The board has serial wire debug (SWD) CoreSight connectors for both the ECM3532 and for debugging. In addition to the ECM3532 and A31R118 there are a number of other supporting devices on board to aid with development. These devices include 2 PDM microphones (front and back), a 6-axis MEMS IMU with integrated temperature sensor, 64M-BIT serial flash, pressure sensor, SD card slot, CR2032 battery cradle, 5 LEDs, and a push button. See Fig.4.1 for a high-level block diagram.

4. Block Diagram

The ASB diagram is shown in Figure 4.1, below.

The board has a (2) 10-pin Coresight connectors, which allows a compatible debug probe to be easily connected to the ECM3532 and/or A31R118 for debugging.

The board contains two buttons:

A. S1 is a general-purpose button connected to GPIO10
B. S2 drives the POR (power on reset) input to the ECM3532

In addition, the ASB contains two general purpose buttons, S3 and S5, connected to GPIO pins on the ECM3532. The ASB contains 4 status LEDs for additional application/debug information.

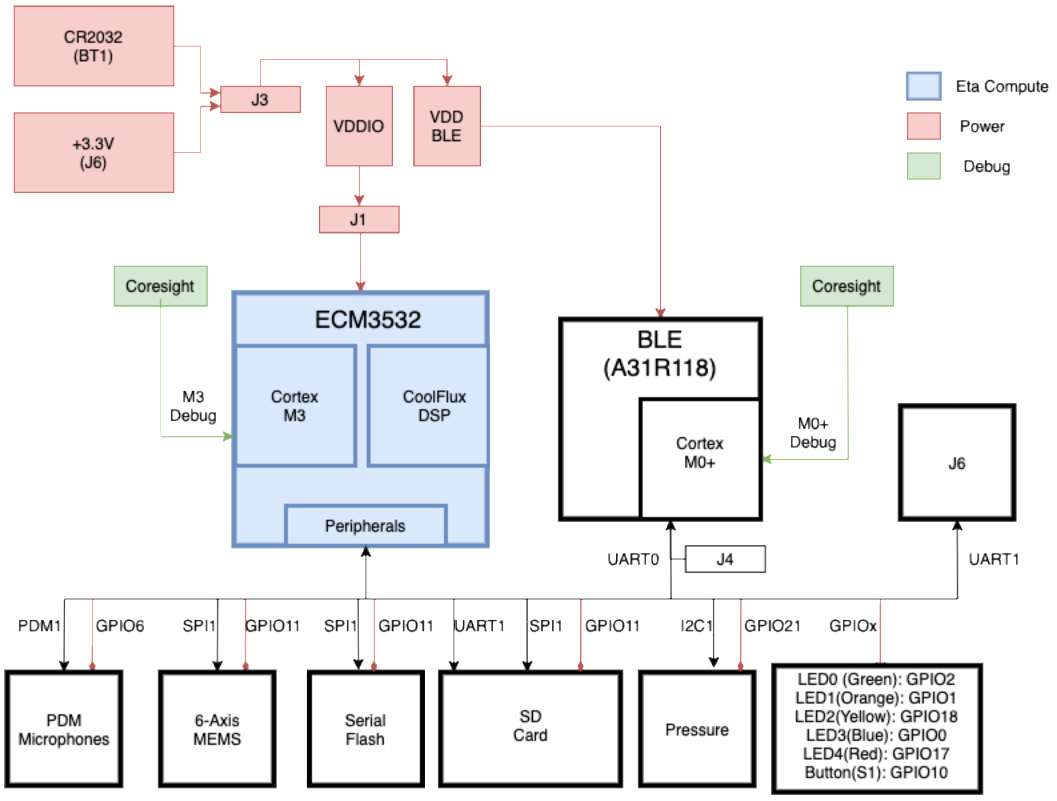


Figure 4.1. ECM3532 ASB Block Diagram

5. *Board Power*

The board can be powered through J6 pin 3 or through a CR2032 coin cell battery connector, BT1.

5.1 Measuring Power

The VDD_IO voltage plane on the board is fed directly from the input voltage supply (VBAT or J6 pin 3) through J1 into the ECM3532. As a result, J1-1 to J1-2 is the correct place to measure the current consumed by just the ECM3532. **Note that the ECM3532 does provide power via its GPIOs to other devices on the board**. Therefore, be sure you are aware which systems of the board are being powered by the ECM3532 so that an accurate measurement can be obtained.

6. Default Board Configuration

Please see Fig 6.1a, 6.1b below for the boards default configuration.

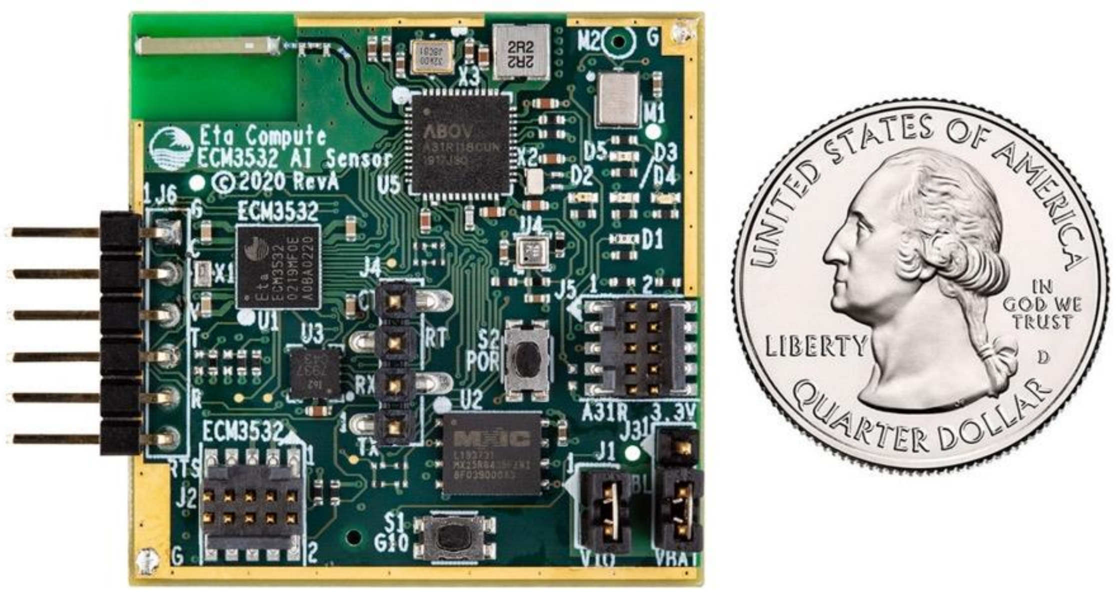


Figure 6.1a. Board Top

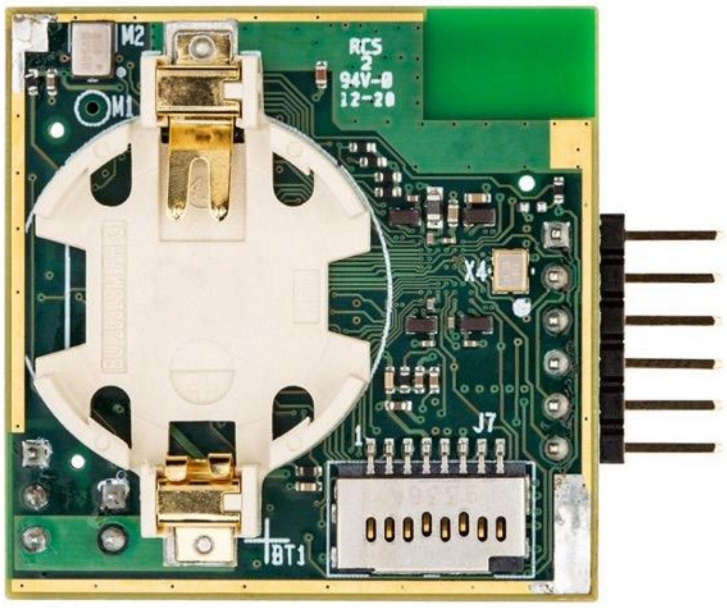


Figure 6.1b. Board Bottom

6.1 Jumpers

There are two configuration jumpers on the board. Please see the Table 6.1 below for their functions.

Table 6.1: Jumpers

J1	VDD_IO power jumper. Measure ECM3532 current here.
J3	SEL VBAT or J6 pin 3 as main power

6.2 Connectors

Table 6.2: Connectors

J2	ECM3532 CoreSight
J4	UART0 between ECM3532 and BLE
J5	A31R118 BLE CoreSight
J6	UART1 from ECM3532 and Power Pin 1: GND Pin 2: RTS Pin 3: 3.3V Pin 4: RX Pin 5: TX Pin 6: CTS
J7	MicroSD slot. NOTE: In the incorrect orientation.

6.3 LEDs

The board has 5 LEDs connected to 5 ECM3532 GPIOs. See Table 6.3 below for assignments.

Table 6.3: LED GPIO Assignment

LED0(Green)	GPIO2
LED1(Orange)	GPIO1
LED2(Yellow)	GPIO18
LED3(Blue)	GPIO0
LED4(Red)	GPIO17

6.4 Buttons

There are (5) push buttons on the ASB. Please see table 6.4 below for their functions.

Table 6.4: Buttons

S1	General purpose button on GPIO10
S2	Power on Reset (POR) to the ECM3532

6.5 UART0

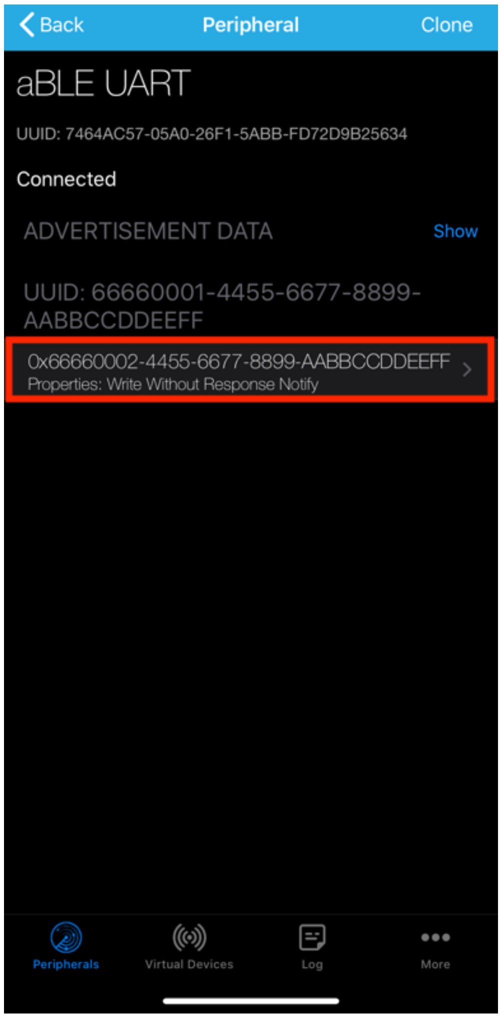
UART0 is the interface between the ECM3532 and A31R118 BLE. For debugging purposes, it is available on J4.

6.6 UART1

UART1 is the default debugging UART and is available on J6 and on the micro SD card connector. All provided examples use this UART to provide information on their status.

6.7 Bluetooth

By default, the A31R118 BLE comes preprogrammed with a BLE application that will automatically begin advertising as “aBLE UART” without any commands from the ECM3532. It also configures itself in a data mode in which anything sent over the UART interface is unconditionally sent over the BLE interface as it was received. To connect over BLE we will use the “LightBlue” iOS application. There are other applications that will work as well as long as they can display GATT notification and send GATT write without response. To connect to the device, find and select it as shown in Figure 6.2. To view notifications from the device, click on “*Properties: Write Without Response Notify*” (Fig 6.3). To begin listening for notifications, click *"Listen for notifications"* (Fig 6.4). To view the ASCII representation, click on "*Hex*" (Fig 6.5) and select *"UTF-8 String" (Fig.6.6)*. Once that is completed, you will see the notifications in their string representation (Fig 6.7). To exit data mode and put the BLE into a command mode the string sequence "///" must be sent.



aBLE UART

UUID: 7464AC57-05A0-26F1-5ABB-FD72D9B25634

Connected

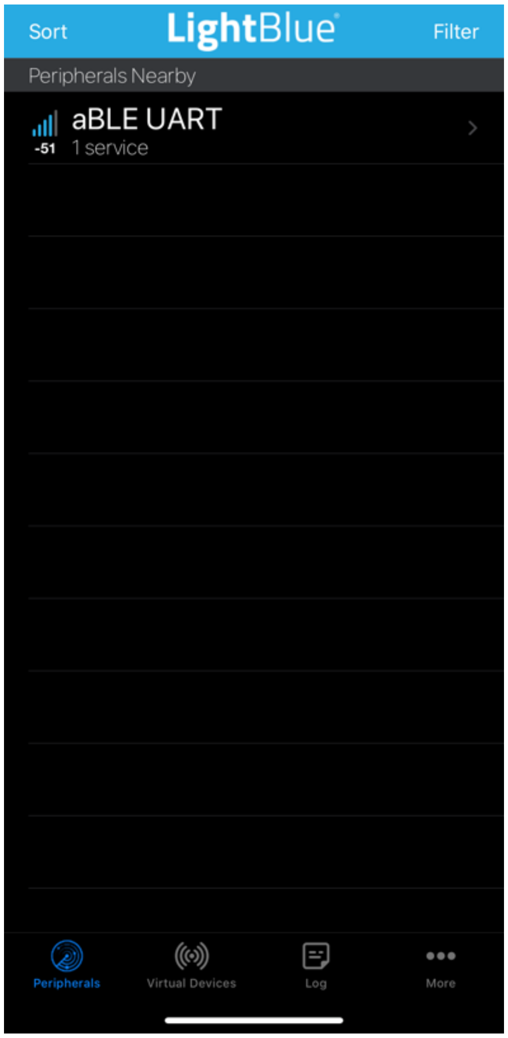
ADVERTISEMENT DATA Show

UUID: 66660001-4455-6677-8899-AABBCCDDEEFF

0x66660002-4455-6677-8899-AABBCCDDEEFF

Properties: Write Without Response Notify

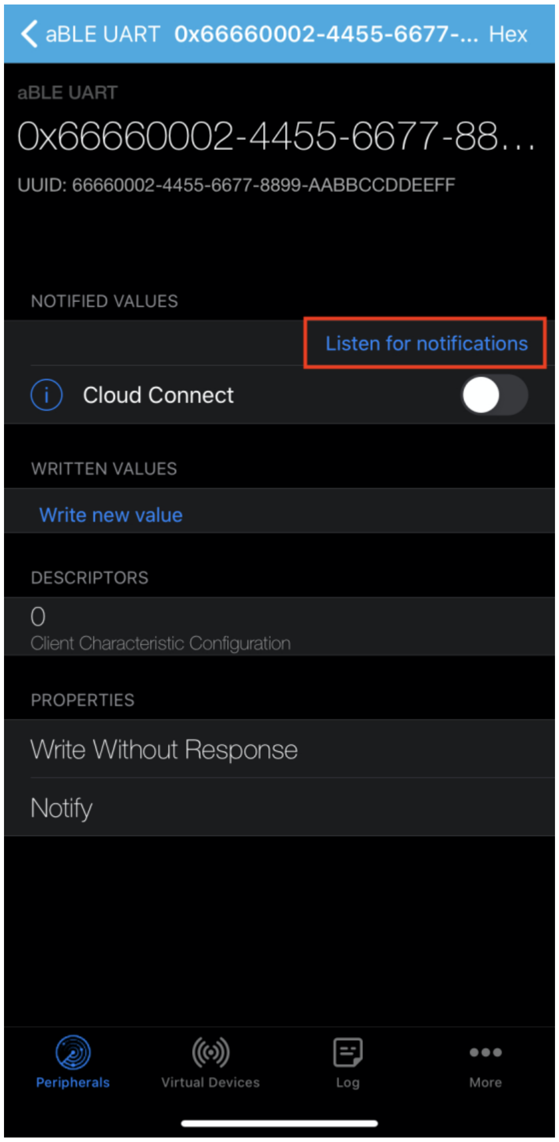
Figure 6.3: Click on Properties



aBLE UART

-51 1 service

Figure 6.2: Connect to BLE


aBLE UART 0x66660002-4455-6677-... Hex

aBLE UART
0x66660002-4455-6677-88...
UUID: 66660002-4455-6677-8899-AABBCCDDEEFF

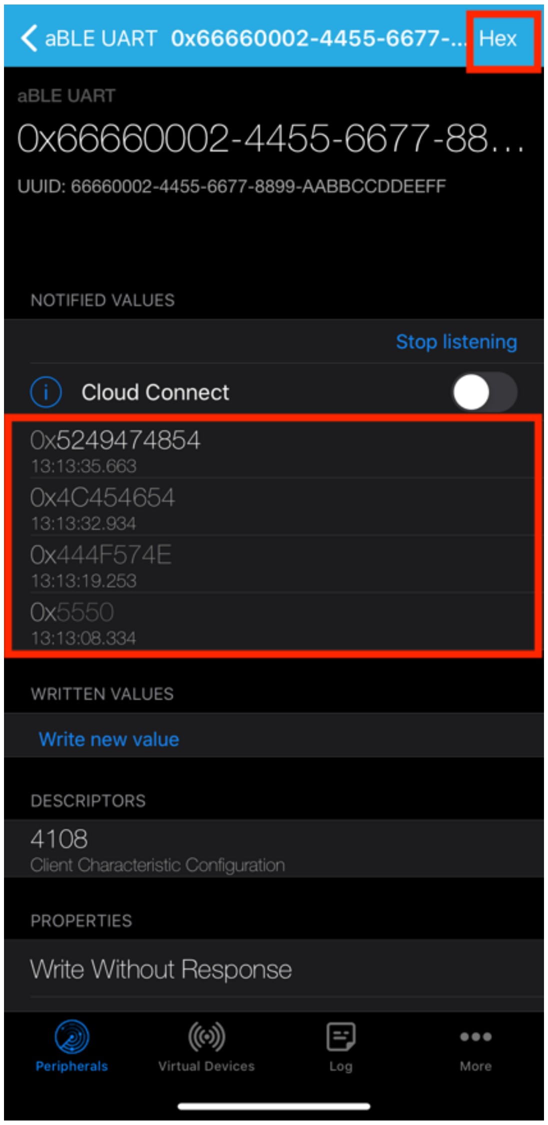
NOTIFIED VALUES
Listen for notifications
Cloud Connect

WRITTEN VALUES
Write new value

DESCRIPTORS
0
Client Characteristic Configuration

PROPERTIES
Write Without Response
Notify

Figure 6.4: Click "Listen for notifications"


aBLE UART 0x66660002-4455-6677-... Hex

aBLE UART
0x66660002-4455-6677-88...
UUID: 66660002-4455-6677-8899-AABBCCDDEEFF

NOTIFIED VALUES
Stop listening
Cloud Connect

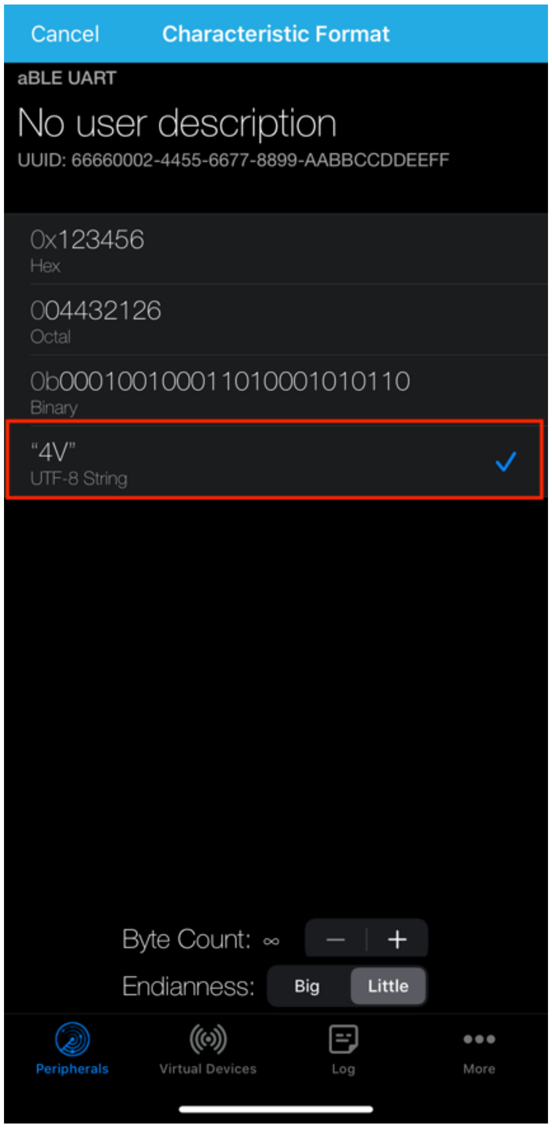
0x5249474854
13:13:35.663
0x4C454654
13:13:32.934
0x444F574E
13:13:19.253
0x5550
13:13:08.334

WRITTEN VALUES
Write new value

DESCRIPTORS
4108
Client Characteristic Configuration

PROPERTIES
Write Without Response

Figure 6.5: Click "Hex"



Cancel Characteristic Format

aBLE UART

No user description

UUID: 66660002-4455-6677-8899-AABBCCDDEEFF

0x123456
Hex

004432126
Octal

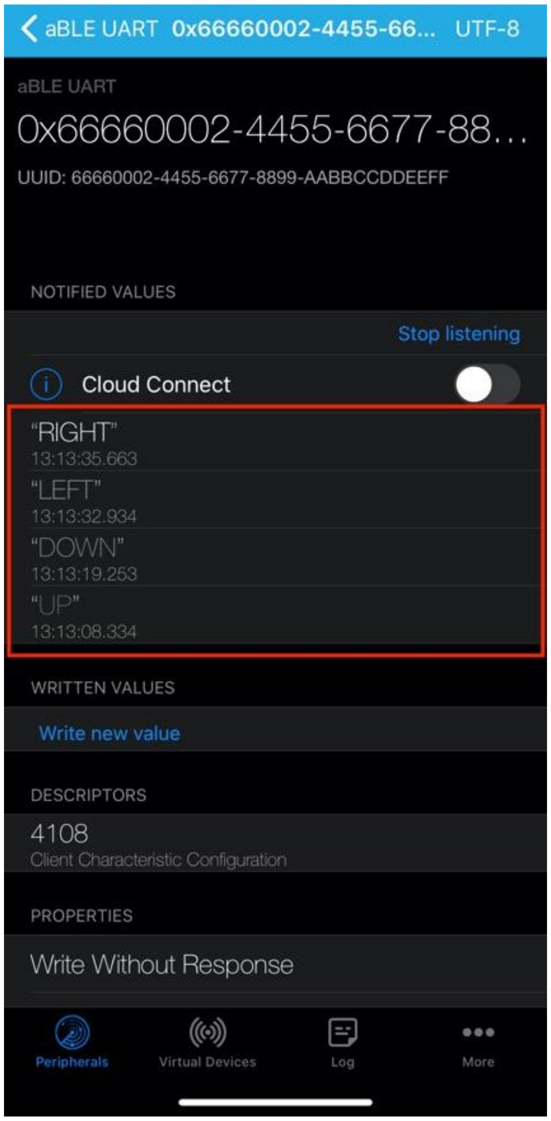
0b000100100011010001010110
Binary

"4V" ✓
UTF-8 String

Byte Count: ∞ − +

Endianness: Big [✓ Little]

Figure 6.6: Select "UTF-8 String"



aBLE UART 0x66660002-4455-66... UTF-8

aBLE UART

0x66660002-4455-6677-88...

UUID: 66660002-4455-6677-8899-AABBCCDDEEFF

NOTIFIED VALUES

Stop listening

Cloud Connect

"RIGHT"
13:13:35.663

"LEFT"
13:13:32.934

"DOWN"
13:13:19.253

"UP"
13:13:08.334

WRITTEN VALUES

Write new value

DESCRIPTORS

4108
Client Characteristic Configuration

PROPERTIES

Write Without Response

Figure 6.7: String notification format.

7. TENSAI SDK

The TENSAI SDK is a set of software examples, chip support libraries, register files, functions and data structures, and documentation to aid in the design and development, which are written in C, using Eta Compute devices. It is also openly licensed and allows for royalty-free use.

7.1 Installation

If you haven't already done so, download the SDK package for the ECM3532 and this board from here:

https://etacompute.com/document-library/

Unzip the contents into a folder of your choice. Depending on your operating system, its generally good practice to choose a folder and path without spaces in the name. Once extracted, the best place to start is the README.html at the root of the SDK.

It is also recommended to add the 'bin' folder at the root of the SDK to your system path. This will allow you to take advantage of our helper scripts during development.

8. Supported IDEs & Compilers

Eta Compute supports the ECM3532 ASB on all operating systems with the help of SEGGERs Embedded Studio and the GCC toolchain. In addition to that, there is out of the box support for both Keil's MDK and IAR's Embedded Workbench. However, both of these IDEs are Windows only. Note that not all provided examples are available on all IDEs. There are sometimes fundamental reasons as to why an example me be provided as a GCC only, Makefile example. When possible though, all example projects are available in Embedded Studio, Keil, IAR and GCC. Each version of the TENSAI SDK has a VERSIONS file, at the root, that contains the various compilers used to compile the various SDK components.

For the sections below, we will be working with the example project called "hello_world". It is located at:

<TensaiSDK_DIR>/soc/ecm3532/boards/eta_ai_sensor/examples/m3/hello_world

Note: This example is for the ECM3532EVB and is used purely for illustrative purposes.

For ECM3532 ASB examples please refer to the examples located under:

<TensaiSDK_DIR>/soc/ecm3532/boards/eta_ai_sensor/examples/

8.1 Embedded Studio

Embedded Studio is a powerful, cross-platform C/C++ IDE (Integrated Development Environment) for embedded systems. **Version 4.52 and later includes support for the ECM3532.**

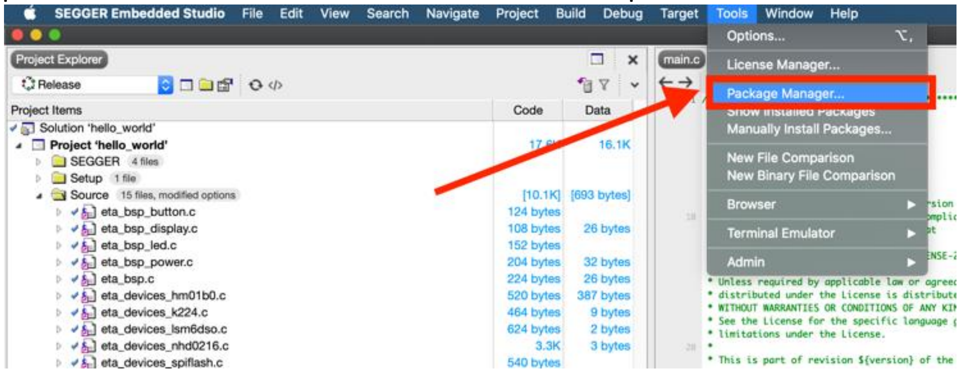
8.1.1 Setup and Installation

1. Download the Embedded Studio setup for your operating system and execute it. The graphical setup will guide you through the installation.
2. Download the J-Link Software and Documentation Package (v6.70 or later) for your operating system and install it.
3. [Optional] Add the path of the 'bin' folder inside the install directory to your system path.
4. Connect the ASB with the USB cable to your computer.

You are now ready to get started with Embedded Studio. In the next few sections the basic debug flow is outlined for more information you can visit: https://studio.segger.com/

8.1.2 Install Eta Compute Pack

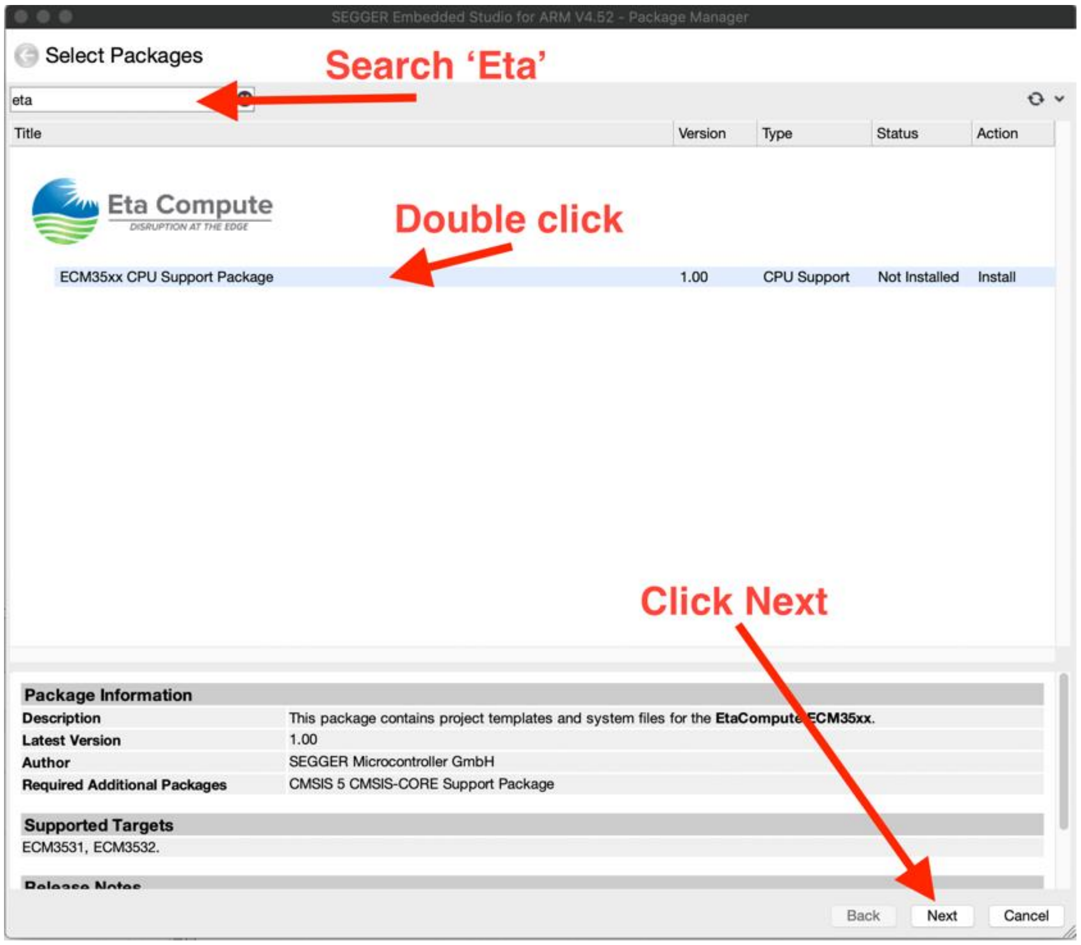
To install the Eta Compute pack inside embedded studio, navigate to the 'Tools' menu and select 'Package Manager' as seen in Fig 8.1a. Then in the dialog box that pops up, search for 'Eta', double click the pack to update the action to 'Install' and click 'Next'. The pack will then download and install.



Project Items	Code	Data
Solution 'hello_world'		
Project 'hello_world'	17.6K	16.1K
SEGGER 4 files		
Setup 1 file		
Source 15 files, modified options	[10.1K]	[693 bytes]
eta_bsp_button.c	124 bytes	
eta_bsp_display.c	108 bytes	26 bytes
eta_bsp_led.c	152 bytes	
eta_bsp_power.c	204 bytes	32 bytes
eta_bsp.c	224 bytes	26 bytes
eta_devices_hm01b0.c	520 bytes	387 bytes
eta_devices_k224.c	464 bytes	9 bytes
eta_devices_lsm6dso.c	624 bytes	2 bytes
eta_devices_nhd0216.c	3.3K	3 bytes
eta_devices_spiflash.c	540 bytes	

```
* Unless required by applicable law or agree
* distributed under the License is distribute
* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIN
* See the License for the specific language 
* limitations under the License.
*
* This is part of revision ${version} of the
```

Figure 8.1a: Select Package Manager



Title	Version	Type	Status	Action
ECM35xx CPU Support Package	1.00	CPU Support	Not Installed	Install

Package Information	
Description	This package contains project templates and system files for the **EtaCompute ECM35xx**.
Latest Version	1.00
Author	SEGGER Microcontroller GmbH
Required Additional Packages	CMSIS 5 CMSIS-CORE Support Package

Supported Targets
ECM3531, ECM3532.

Release Notes

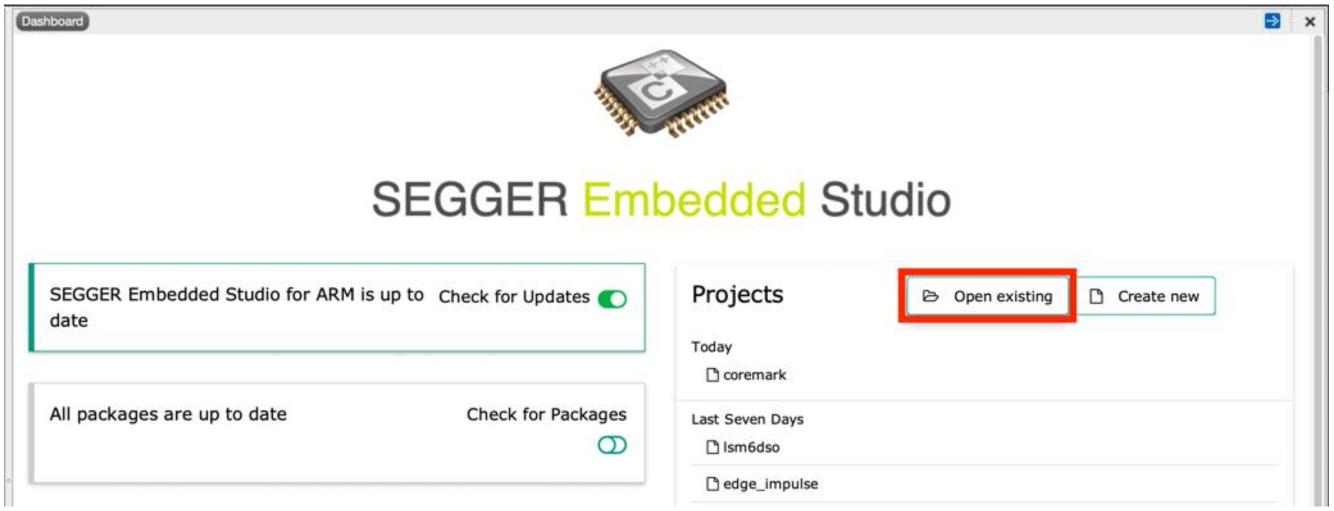
Figure 8.1b: Select and Install Eta Compute Pack

8.1.3 Import a project

When you open SEGGER's Embedded Studio (SES) for the first time you will see a screen similar to Fig 8.2. To import a project, click on the button that reads "Open Existing" as outlined in Fig 8.2. A dialog box will open and prompt you to navigate to the desired project to open. Inside the "hello_world" directory (using the path outlined in Section 8), there is an "embedded_studio" directory. Inside that are 3 options: "flash", "flash_boot_sram", and "sram". For this guide we will use the flash project. The different project types are described below:

flash	Stored in flash, executes from flash
flash_boot_sram	Stored in flash, copies itself into sram, executes from sram
sram	Stored in sram, executes from sram

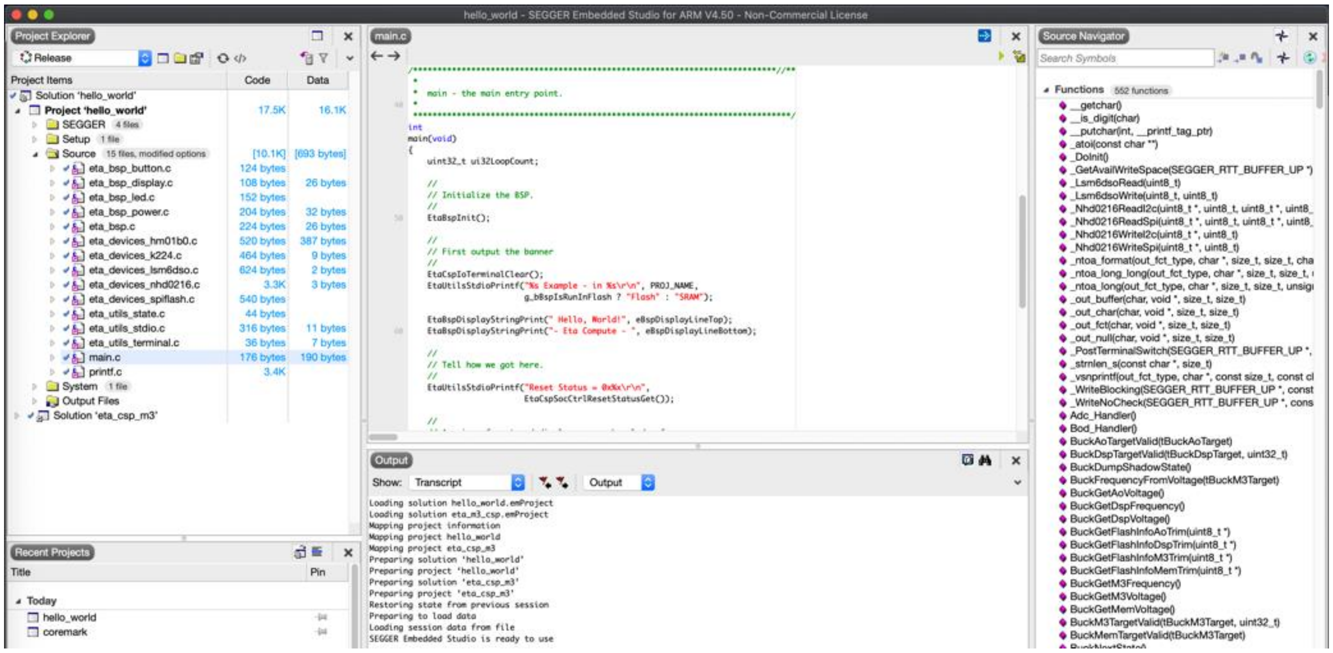
Inside the "flash" directory, there is a file called "hello_world.emProject". Select and open this file within the dialog. When opened successfully, you should see a window similar to Fig 8.3.



SEGGER Embedded Studio for ARM is up to date

All packages are up to date

Figure 8.2: Open Project from Dashboard



Project Items	Code	Data
Solution 'hello_world'		
Project 'hello_world'	17.5K	16.1K
SEGGER 4 files		
Setup 1 file		
Source 15 files, modified options	[10.1K]	[693 bytes]
eta_bsp_button.c	124 bytes	
eta_bsp_display.c	108 bytes	26 bytes
eta_bsp_led.c	152 bytes	
eta_bsp_power.c	204 bytes	32 bytes
eta_bsp.c	224 bytes	26 bytes
eta_devices_hm01b0.c	520 bytes	387 bytes
eta_devices_k224.c	464 bytes	9 bytes
eta_devices_lsm6dso.c	624 bytes	2 bytes
eta_devices_nhd0216.c	3.3K	3 bytes
eta_devices_spiflash.c	540 bytes	
eta_utils_state.c	44 bytes	
eta_utils_stdio.c	316 bytes	11 bytes
eta_utils_terminal.c	36 bytes	7 bytes
main.c	176 bytes	190 bytes
printf.c	3.4K	
System 1 file		
Output Files		
Solution 'eta_csp_m3'		

```
/***************************************************************************//**
 *
 *  main - the main entry point.
 *
 ******************************************************************************/
int
main(void)
{
    uint32_t ui32LoopCount;

    //
    // Initialize the BSP.
    //
    EtaBspInit();

    //
    // First output the banner
    //
    EtaCspIoTerminalClear();
    EtaUtilsStdioPrintf("%s Example - in %s\r\n", PROJ_NAME,
                        g_bBspIsRunInFlash ? "Flash" : "SRAM");

    EtaBspDisplayStringPrint(" Hello, World!", eBspDisplayLineTop);
    EtaBspDisplayStringPrint("- Eta Compute - ", eBspDisplayLineBottom);

    //
    // Tell how we got here.
    //
    EtaUtilsStdioPrintf("Reset Status = 0x%x\r\n",
                        EtaCspSocCtrlResetStatusGet());

    //
```

```
Loading solution hello_world.emProject
Loading solution eta_m3_csp.emProject
Mapping project information
Mapping project hello_world
Mapping project eta_csp_m3
Preparing solution 'hello_world'
Preparing project 'hello_world'
Preparing solution 'eta_csp_m3'
Preparing project 'eta_csp_m3'
Restoring state from previous session
Preparing to load data
Loading session data from file
SEGGER Embedded Studio is ready to use
```

Functions 552 functions
- __getchar()
- __is_digit(char)
- __putchar(int, __printf_tag_ptr)
- _atoi(const char **)
- _DoInit()
- _GetAvailWriteSpace(SEGGER_RTT_BUFFER_UP *)
- _Lsm6dsoRead(uint8_t)
- _Lsm6dsoWrite(uint8_t, uint8_t)
- _Nhd0216ReadI2c(uint8_t *, uint8_t, uint8_t *, uint8_
- _Nhd0216ReadSpi(uint8_t *, uint8_t, uint8_t *, uint8_
- _Nhd0216WriteI2c(uint8_t *, uint8_t)
- _Nhd0216WriteSpi(uint8_t *, uint8_t)
- _ntoa_format(out_fct_type, char *, size_t, size_t, cha
- _ntoa_long_long(out_fct_type, char *, size_t, size_t,
- _ntoa_long(out_fct_type, char *, size_t, size_t, unsig
- _out_buffer(char, void *, size_t, size_t)
- _out_char(char, void *, size_t, size_t)
- _out_fct(char, void *, size_t, size_t)
- _out_null(char, void *, size_t, size_t)
- _PostTerminalSwitch(SEGGER_RTT_BUFFER_UP *,
- _strnlen_s(const char *, size_t)
- _vsnprintf(out_fct_type, char *, const size_t, const cl
- _WriteBlocking(SEGGER_RTT_BUFFER_UP *, const
- _WriteNoCheck(SEGGER_RTT_BUFFER_UP *, cons
- Adc_Handler()
- Bod_Handler()
- BuckAoTargetValid(tBuckAoTarget)
- BuckDspTargetValid(tBuckDspTarget, uint32_t)
- BuckDumpShadowState()
- BuckFrequencyFromVoltage(tBuckM3Target)
- BuckGetAoVoltage()
- BuckGetDspFrequency()
- BuckGetDspVoltage()
- BuckGetFlashInfoAoTrim(uint8_t *)
- BuckGetFlashInfoDspTrim(uint8_t *)
- BuckGetFlashInfoM3Trim(uint8_t *)
- BuckGetFlashInfoMemTrim(uint8_t *)
- BuckGetM3Frequency()
- BuckGetM3Voltage()
- BuckGetMemVoltage()
- BuckM3TargetValid(tBuckM3Target, uint32_t)
- BuckMemTargetValid(tBuckM3Target)

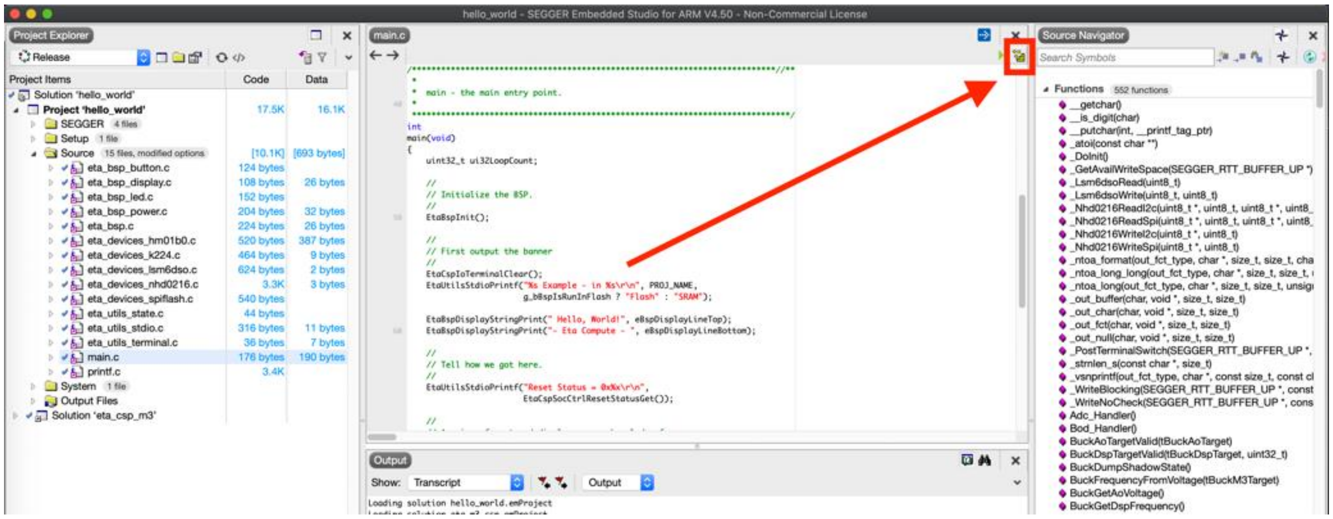
Figure 8.3: Project Opened

8.1.4 Compile and Flash

Now that the project is opened, we can compile it. Every TENSAI SDK release has the examples precompiled and tested before being packaged and distributed. Therefore, there is no need to recompile a fresh example. With that said, if you navigate to the button shown in Fig 8.4 and click, the project will attempt a compile and link. As mentioned, the example is already compiled so you should see an output that looks similar to Fig 8.5a that shows "Build up to date". If you change a source file and click build again you should see an output similar to Fig 8.5b. Under the hood, SES uses the GCC toolchain to compile and link. There are other options available though and you also have the option to use an external compiler should you choose but the project must be configured accordingly.

There is also a command line option for compiling. The J-Link tool "emBuild" needs to be in your path for this to work properly as seen in Fig 8.6. Under the "hello_world" directory one can execute "make es" and the projects under the "embedded_studio" directory will attempt a compile and link.

Once compiled, you have the option to download the application without debugging it as shown in Fig 8.7. Otherwise, move to the next section for debugging.


hello_world - SEGGER Embedded Studio for ARM V4.50 - Non-Commercial License

Project Explorer — Release

Project Items	Code	Data
Solution 'hello_world'		
Project 'hello_world'	17.5K	16.1K
SEGGER 4 files		
Setup 1 file		
Source 15 files, modified options	[10.1K]	[693 bytes]
eta_bsp_button.c	124 bytes	
eta_bsp_display.c	108 bytes	26 bytes
eta_bsp_led.c	152 bytes	
eta_bsp_power.c	204 bytes	32 bytes
eta_bsp.c	224 bytes	26 bytes
eta_devices_hm01b0.c	520 bytes	387 bytes
eta_devices_k224.c	464 bytes	9 bytes
eta_devices_lsm6dso.c	624 bytes	2 bytes
eta_devices_nhd0216.c	3.3K	3 bytes
eta_devices_spiflash.c	540 bytes	
eta_utils_state.c	44 bytes	
eta_utils_stdio.c	316 bytes	11 bytes
eta_utils_terminal.c	36 bytes	7 bytes
main.c	176 bytes	190 bytes
printf.c	3.4K	
System 1 file		
Output Files		
Solution 'eta_csp_m3'		

main.c

```
/*****************************************************************************//**
 *
 *  main - the main entry point.
 *
 ******************************************************************************/
int
main(void)
{
    uint32_t ui32LoopCount;

    //
    // Initialize the BSP.
    //
    EtaBspInit();

    //
    // First output the banner
    //
    EtaCspIoTerminalClear();
    EtaUtilsStdioPrintf("%s Example - in %s\r\n", PROJ_NAME,
                        g_bBspIsRunInFlash ? "Flash" : "SRAM");

    EtaBspDisplayStringPrint(" Hello, World!", eBspDisplayLineTop);
    EtaBspDisplayStringPrint("- Eta Compute - ", eBspDisplayLineBottom);

    //
    // Tell how we got here.
    //
    EtaUtilsStdioPrintf("Reset Status = 0x%x\r\n",
                        EtaCspSocCtrlResetStatusGet());

    //
```

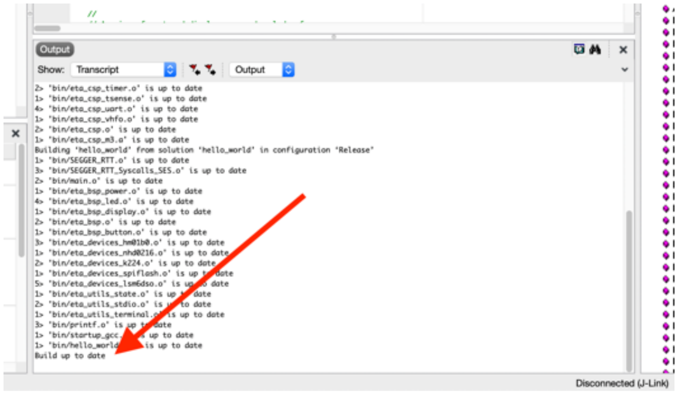
Output — Show: Transcript — Output

```
Loading solution hello_world.emProject
```

Source Navigator — Functions 552 functions

```
__getchar()
__is_digit(char)
__putchar(int, __printf_tag_ptr)
_atoi(const char **)
_DoInit()
_GetAvailWriteSpace(SEGGER_RTT_BUFFER_UP *)
_Lsm6dsoRead(uint8_t)
_Lsm6dsoWrite(uint8_t, uint8_t)
_Nhd0216ReadI2c(uint8_t *, uint8_t, uint8_t *, uint8_
_Nhd0216ReadSpi(uint8_t *, uint8_t, uint8_t *, uint8_
_Nhd0216WriteI2c(uint8_t *, uint8_t)
_Nhd0216WriteSpi(uint8_t *, uint8_t)
_ntoa_format(out_fct_type, char *, size_t, size_t, cha
_ntoa_long_long(out_fct_type, char *, size_t, size_t,
_ntoa_long(out_fct_type, char *, size_t, size_t, unsig
_out_buffer(char, void *, size_t, size_t)
_out_char(char, void *, size_t, size_t)
_out_fct(char, void *, size_t, size_t)
_out_null(char, void *, size_t, size_t)
_PostTerminalSwitch(SEGGER_RTT_BUFFER_UP *,
_strnlen_s(const char *, size_t)
_vsnprintf(out_fct_type, char *, const size_t, const c
_WriteBlocking(SEGGER_RTT_BUFFER_UP *, const
_WriteNoCheck(SEGGER_RTT_BUFFER_UP *, cons
Adc_Handler()
Bod_Handler()
BuckAoTargetValid(tBuckAoTarget)
BuckDspTargetValid(tBuckDspTarget, uint32_t)
BuckDumpShadowState()
BuckFrequencyFromVoltage(tBuckM3Target)
BuckGetAoVoltage()
BuckGetDspFrequency()
```

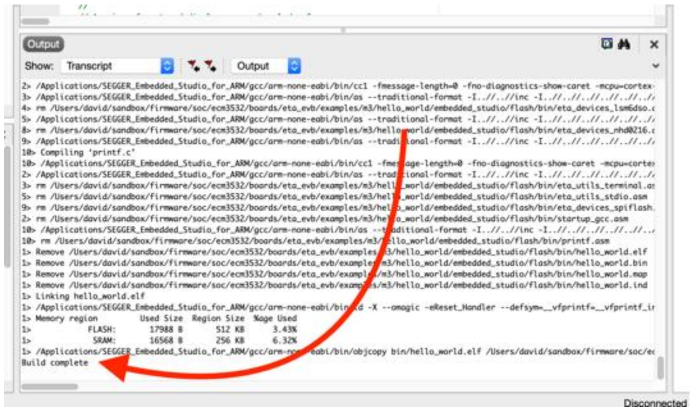
Figure 8.4: Build Project


Output — Show: Transcript — Output

```
2> 'bin/eta_csp_timer.o' is up to date
1> 'bin/eta_csp_tsense.o' is up to date
4> 'bin/eta_csp_uart.o' is up to date
1> 'bin/eta_csp_vhfo.o' is up to date
2> 'bin/eta_csp.o' is up to date
1> 'bin/eta_csp_m3.o' is up to date
Building 'hello_world' from solution 'hello_world' in configuration 'Release'
1> 'bin/SEGGER_RTT.o' is up to date
3> 'bin/SEGGER_RTT_Syscalls_SES.o' is up to date
2> 'bin/main.o' is up to date
1> 'bin/eta_bsp_power.o' is up to date
4> 'bin/eta_bsp_led.o' is up to date
1> 'bin/eta_bsp_display.o' is up to date
2> 'bin/eta_bsp.o' is up to date
1> 'bin/eta_bsp_button.o' is up to date
3> 'bin/eta_devices_hm01b0.o' is up to date
1> 'bin/eta_devices_nhd0216.o' is up to date
2> 'bin/eta_devices_k224.o' is up to date
1> 'bin/eta_devices_spiflash.o' is up to date
5> 'bin/eta_devices_lsm6dso.o' is up to date
1> 'bin/eta_utils_state.o' is up to date
2> 'bin/eta_utils_stdio.o' is up to date
1> 'bin/eta_utils_terminal.o' is up to date
3> 'bin/printf.o' is up to date
1> 'bin/startup_gcc.o' is up to date
1> 'bin/hello_world' is up to date
Build up to date
```

Disconnected (J-Link)

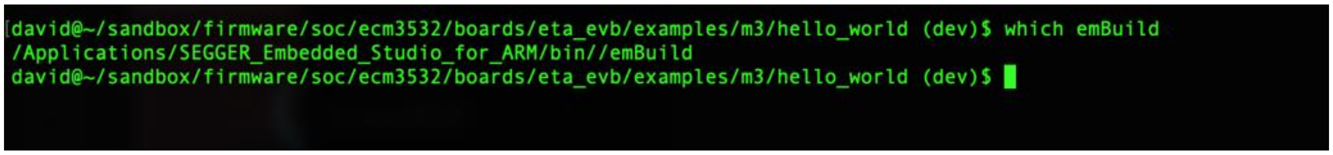
Figure 8.5a: Project is up-to-date


Output — Show: Transcript — Output

```
2> /Applications/SEGGER_Embedded_Studio_for_ARM/gcc/arm-none-eabi/bin/cc1 -fmessage-length=0 -fno-diagnostics-show-caret -mcpu=cortex-
3> /Applications/SEGGER_Embedded_Studio_for_ARM/gcc/arm-none-eabi/bin/as --traditional-format -I..//..//inc -I..//..//..//..//..//..
4> rm /Users/david/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world/embedded_studio/flash/bin/eta_devices_lsm6dso.
5> /Applications/SEGGER_Embedded_Studio_for_ARM/gcc/arm-none-eabi/bin/as --traditional-format -I..//..//inc -I..//..//..//..//..//..
8> rm /Users/david/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world/embedded_studio/flash/bin/eta_devices_nhd0216.
9> /Applications/SEGGER_Embedded_Studio_for_ARM/gcc/arm-none-eabi/bin/as --traditional-format -I..//..//inc -I..//..//..//..//..//..
10> Compiling 'printf.c'
10> /Applications/SEGGER_Embedded_Studio_for_ARM/gcc/arm-none-eabi/bin/cc1 -fmessage-length=0 -fno-diagnostics-show-caret -mcpu=cortex
2> /Applications/SEGGER_Embedded_Studio_for_ARM/gcc/arm-none-eabi/bin/as --traditional-format -I..//..//inc -I..//..//..//..//..//..
3> rm /Users/david/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world/embedded_studio/flash/bin/eta_utils_terminal.a
5> rm /Users/david/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world/embedded_studio/flash/bin/eta_utils_stdio.asm
9> rm /Users/david/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world/embedded_studio/flash/bin/eta_devices_spiflash.
2> rm /Users/david/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world/embedded_studio/flash/bin/startup_gcc.asm
10> /Applications/SEGGER_Embedded_Studio_for_ARM/gcc/arm-none-eabi/bin/as --traditional-format -I..//..//inc -I..//..//..//..//..//..
10> rm /Users/david/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world/embedded_studio/flash/bin/printf.asm
1> Remove /Users/david/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world/embedded_studio/flash/bin/hello_world.elf
1> Remove /Users/david/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world/embedded_studio/flash/bin/hello_world.bin
1> Remove /Users/david/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world/embedded_studio/flash/bin/hello_world.map
1> Remove /Users/david/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world/embedded_studio/flash/bin/hello_world.ind
1> Linking hello_world.elf
1> /Applications/SEGGER_Embedded_Studio_for_ARM/gcc/arm-none-eabi/bin/ld -X --omagic -eReset_Handler --defsym=__vfprintf=__vfprintf_i
1> Memory region         Used Size  Region Size  %age Used
1>            FLASH:       17988 B       512 KB      3.43%
1>             SRAM:       16568 B       256 KB      6.32%
1> /Applications/SEGGER_Embedded_Studio_for_ARM/gcc/arm-none-eabi/bin/objcopy bin/hello_world.elf /Users/david/sandbox/firmware/soc/ec
Build complete
```

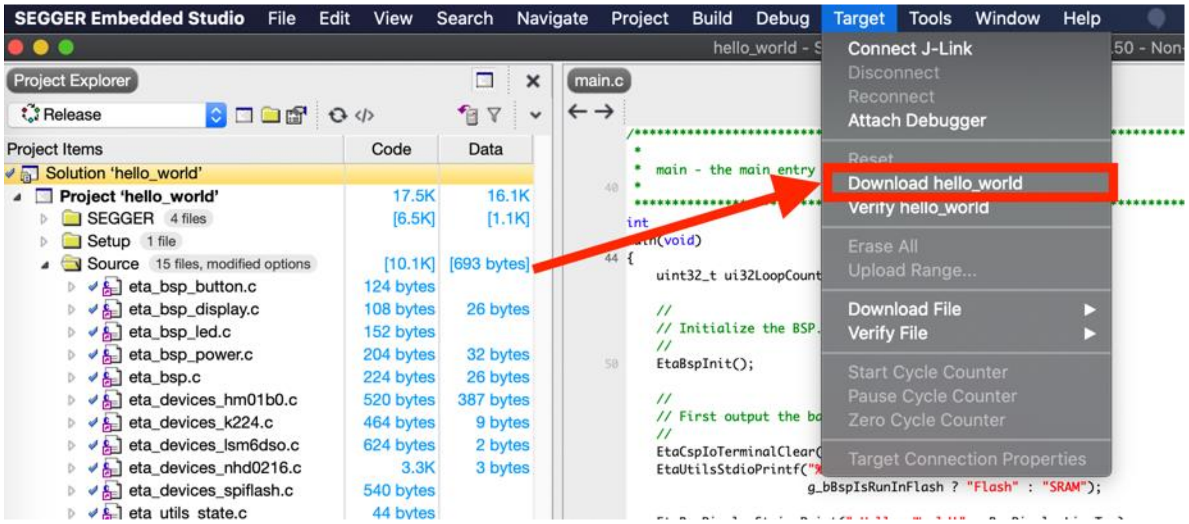
Disconnected

Figure 8.5b: Project is out-of-date and Compiles Successfully



```
[david@~/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world (dev)$ which emBuild
/Applications/SEGGER_Embedded_Studio_for_ARM/bin//emBuild
david@~/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world (dev)$
```

Fig 8.6: emBuild Check



Project Items	Code	Data
Solution 'hello_world'		
Project 'hello_world'	17.5K	16.1K
SEGGER 4 files	[6.5K]	[1.1K]
Setup 1 file		
Source 15 files, modified options	[10.1K]	[693 bytes]
eta_bsp_button.c	124 bytes	
eta_bsp_display.c	108 bytes	26 bytes
eta_bsp_led.c	152 bytes	
eta_bsp_power.c	204 bytes	32 bytes
eta_bsp.c	224 bytes	26 bytes
eta_devices_hm01b0.c	520 bytes	387 bytes
eta_devices_k224.c	464 bytes	9 bytes
eta_devices_lsm6dso.c	624 bytes	2 bytes
eta_devices_nhd0216.c	3.3K	3 bytes
eta_devices_spiflash.c	540 bytes	
eta_utils_state.c	44 bytes	

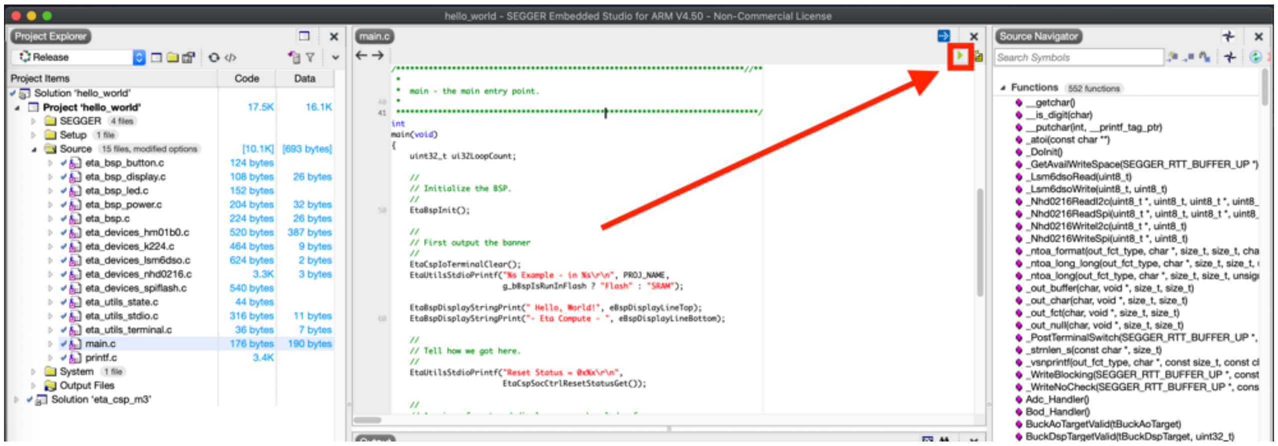
Figure 8.7: Download Application Without Debugging

8.1.5 Debug

Now that the application is compiled, you are ready to debug it. Navigate to the green play button as shown in Fig 8.8 and click it. This will launch a debug session. All SES projects are configured to use J-Link. If you would like to use your own debug probe you must update the project settings. Note that SES support debugging through J-Link or GDB connections only.

Once the debugger is started you should see a window similar to Fig 8.9. You may have slightly different windows. To add or remove debug windows, navigate to the menu shown in Fig 8.10. As you can see the application was loaded and was halted on the first source line in main(). The current line is highlighted for easy identification. To view peripheral registers, navigate to the menu as shown in Fig 8.11.

To set a breakpoint, click on the margin next to the source line you would like to halt at. You have successfully placed a break point if you see a red circle as shown in Fig 8.12. Once your break point is set you can free run the application which will run until manually halted or a breakpoint is hit. To free run and continue execution click the green play button as shown in Fig 8.13. The application will run and halt at the breakpoint as shown in Fig 8.14. To exit debug mode and return to editor mode, click the red square as shown in Fig 8.15.

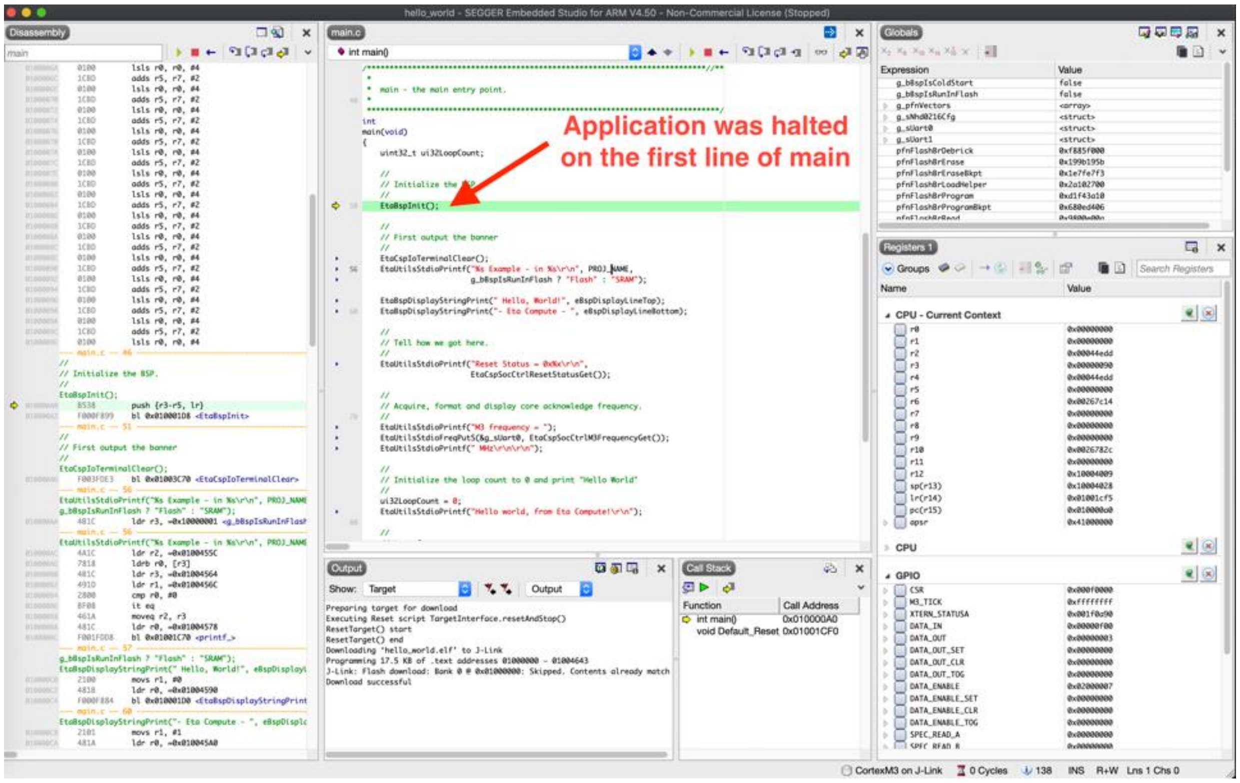


hello_world - SEGGER Embedded Studio for ARM V4.50 - Non-Commercial License

Project Items	Code	Data
Solution 'hello_world'		
Project 'hello_world'	17.5K	16.1K
SEGGER 4 files		
Setup 1 file		
Source 15 files, modified options	[10.1K]	[693 bytes]
eta_bsp_button.c	124 bytes	
eta_bsp_display.c	108 bytes	26 bytes
eta_bsp_led.c	152 bytes	
eta_bsp_power.c	204 bytes	32 bytes
eta_bsp.c	224 bytes	26 bytes
eta_devices_hm01b0.c	520 bytes	387 bytes
eta_devices_k224.c	464 bytes	9 bytes
eta_devices_lsm6dso.c	624 bytes	2 bytes
eta_devices_nhd0216.c	3.3K	3 bytes
eta_devices_spiflash.c	540 bytes	
eta_utils_state.c	44 bytes	
eta_utils_stdio.c	316 bytes	11 bytes
eta_utils_terminal.c	36 bytes	7 bytes
main.c	176 bytes	190 bytes
printf.c	3.4K	
System 1 file		
Output Files		
Solution 'eta_csp_m3'		

```
/*****************************************************************************//**
 *
 *  main - the main entry point.
 *
 ******************************************************************************/
int
main(void)
{
    uint32_t ui32LoopCount;

    //
    // Initialize the BSP.
    //
    EtaBspInit();

    //
    // First output the banner
    //
    EtaCspIoTerminalClear();
    EtaUtilsStdioPrintf("%s Example - in %s\r\n", PROJ_NAME,
                        g_bBspIsRunInFlash ? "Flash" : "SRAM");

    EtaBspDisplayStringPrint(" Hello, World!", eBspDisplayLineTop);
    EtaBspDisplayStringPrint("- Eta Compute - ", eBspDisplayLineBottom);

    //
    // Tell how we got here.
    //
    EtaUtilsStdioPrintf("Reset Status = 0x%x\r\n",
                        EtaCspSocCtrlResetStatusGet());

    //
```

Figure 8.8: Start Execution



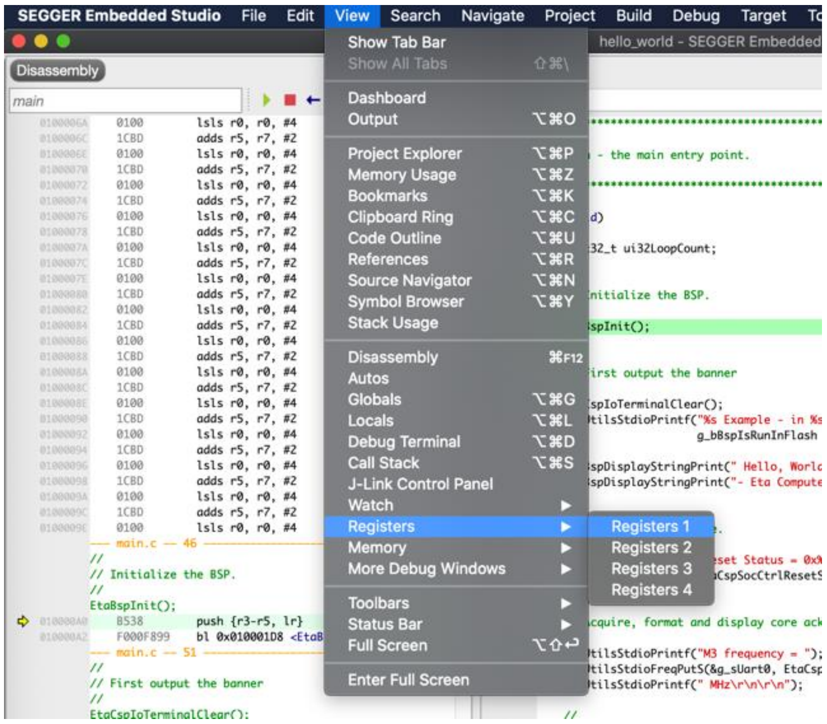
hello_world - SEGGER Embedded Studio for ARM V4.50 - Non-Commercial License (Stopped)

```
/*****************************************************************************//**
 *
 *  main - the main entry point.
 *
 ******************************************************************************/
int
main(void)
{
    uint32_t ui32LoopCount;

    //
    // Initialize the BSP.
    //
    EtaBspInit();

    //
    // First output the banner
    //
    EtaCspIoTerminalClear();
    EtaUtilsStdioPrintf("%s Example - in %s\r\n", PROJ_NAME,
                        g_bBspIsRunInFlash ? "Flash" : "SRAM");

    EtaBspDisplayStringPrint(" Hello, World!", eBspDisplayLineTop);
    EtaBspDisplayStringPrint("- Eta Compute - ", eBspDisplayLineBottom);

    //
    // Tell how we got here.
    //
    EtaUtilsStdioPrintf("Reset Status = 0x%x\r\n",
                        EtaCspSocCtrlResetStatusGet());

    //
    // Acquire, format and display core acknowledge frequency.
    //
    EtaUtilsStdioPrintf("M3 frequency = ");
    EtaUtilsStdioFreqPutS(&g_sUart0, EtaCspSocCtrlM3FrequencyGet());
    EtaUtilsStdioPrintf(" MHz\r\n\r\n");

    //
    // Initialize the loop count to 0 and print "Hello World"
    //
    ui32LoopCount = 0;
    EtaUtilsStdioPrintf("Hello world, from Eta Compute!\r\n");

    //
```

```
Preparing target for download
Executing Reset script TargetInterface.resetAndStop()
ResetTarget() start
ResetTarget() end
Downloading 'hello_world.elf' to J-Link
Programming 17.5 KB of .text addresses 01000000 - 01004643
J-Link: Flash download: Bank 0 @ 0x01000000: Skipped. Contents already match
Download successful
```

Function	Call Address
int main()	0x010000A0
void Default_Reset	0x01001CF0

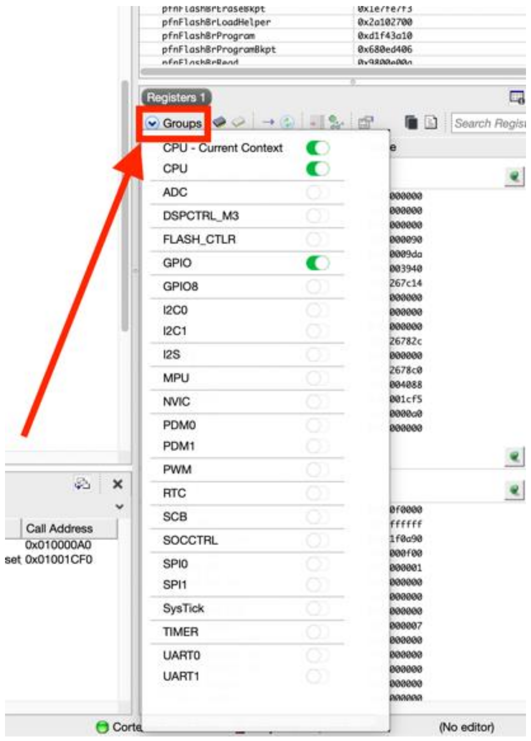
Figure 8.9: Debug View



```
0100006A  0100   lsls r0, r0, #4
0100006C  1CBD   adds r5, r7, #2
0100006E  0100   lsls r0, r0, #4
01000070  1CBD   adds r5, r7, #2
01000072  0100   lsls r0, r0, #4
01000074  1CBD   adds r5, r7, #2
01000076  0100   lsls r0, r0, #4
01000078  1CBD   adds r5, r7, #2
0100007A  0100   lsls r0, r0, #4
0100007C  1CBD   adds r5, r7, #2
0100007E  0100   lsls r0, r0, #4
01000080  1CBD   adds r5, r7, #2
01000082  0100   lsls r0, r0, #4
01000084  1CBD   adds r5, r7, #2
01000086  0100   lsls r0, r0, #4
01000088  1CBD   adds r5, r7, #2
0100008A  0100   lsls r0, r0, #4
0100008C  1CBD   adds r5, r7, #2
0100008E  0100   lsls r0, r0, #4
01000090  1CBD   adds r5, r7, #2
01000092  0100   lsls r0, r0, #4
01000094  1CBD   adds r5, r7, #2
01000096  0100   lsls r0, r0, #4
01000098  1CBD   adds r5, r7, #2
0100009A  0100   lsls r0, r0, #4
0100009C  1CBD   adds r5, r7, #2
0100009E  0100   lsls r0, r0, #4
—— main.c — 46 ——————————
//
// Initialize the BSP.
//
EtaBspInit();
010000A0  B538      push {r3-r5, lr}
010000A2  F000F899  bl 0x010001D8 <EtaB
—— main.c — 51 ——————————
//
// First output the banner
//
EtaCspIoTerminalClear();
```

```
**********************************
 - the main entry point.
**********************************
d)

32_t ui32LoopCount;

nitialize the BSP.

spInit();

irst output the banner

spIoTerminalClear();
tilsStdioPrintf("%s Example - in %s
        g_bBspIsRunInFlash

spDisplayStringPrint(" Hello, Worlc
spDisplayStringPrint("- Eta Compute

set Status = 0x%
CspSocCtrlResetS

cquire, format and display core ack

tilsStdioPrintf("M3 frequency = ");
tilsStdioFreqPutS(&g_sUart0, EtaCsp
tilsStdioPrintf(" MHz\r\n\r\n");

//
```

Figure 8.10: Add Different View Windows

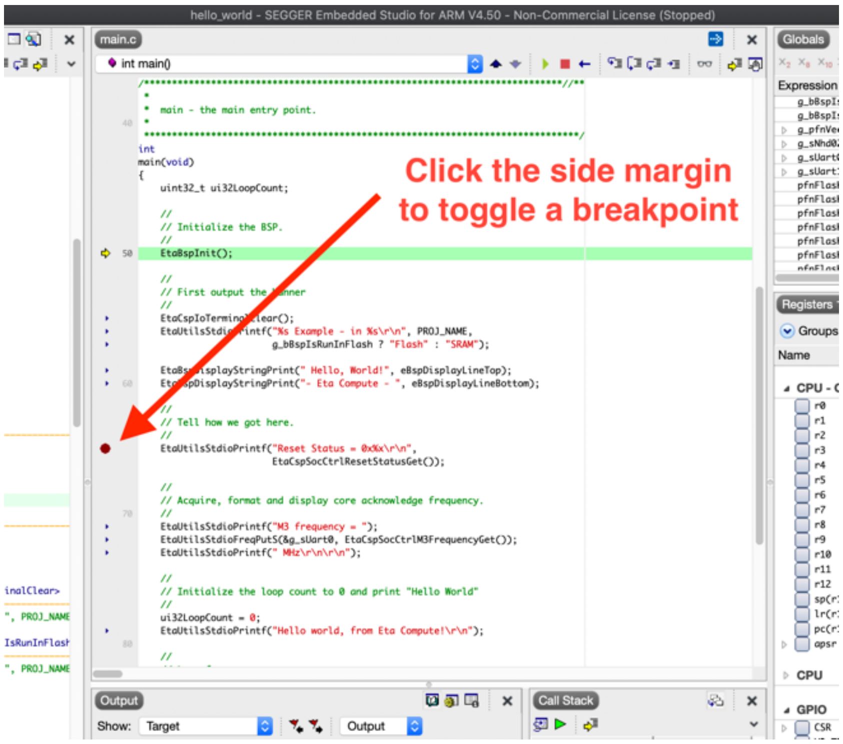


```
pfnFlashBrEraseBkpt       0x1e7fe7f3
pfnFlashBrLoadHelper      0x2a102700
pfnFlashBrProgram         0xd1f43a10
pfnFlashBrProgramBkpt     0x680ed406
```

```
0x010000A0
set 0x01001CF0
```

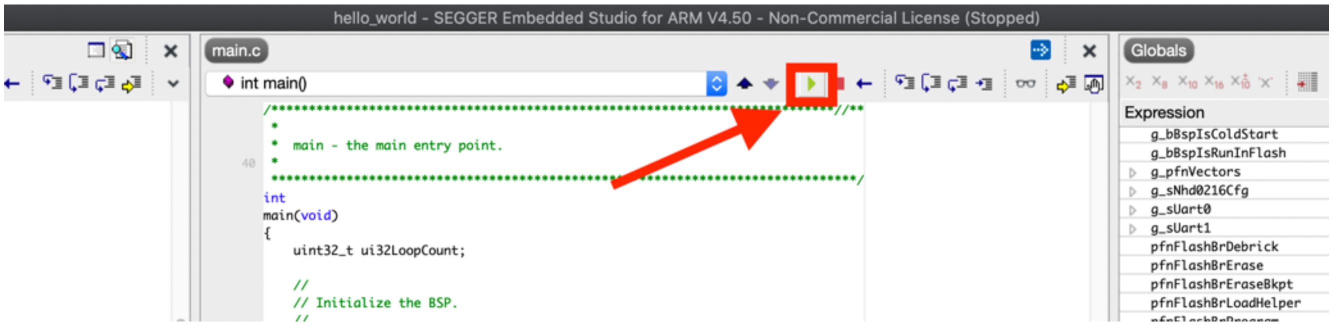
CPU - Current Context	✓
CPU	✓
ADC	
DSPCTRL_M3	
FLASH_CTLR	
GPIO	✓
GPIO8	
I2C0	
I2C1	
I2S	
MPU	
NVIC	
PDM0	
PDM1	
PWM	
RTC	
SCB	
SOCCTRL	
SPI0	
SPI1	
SysTick	
TIMER	
UART0	
UART1	

Figure 8.11: Add Different Peripheral Registers


hello_world - SEGGER Embedded Studio for ARM V4.50 - Non-Commercial License (Stopped)

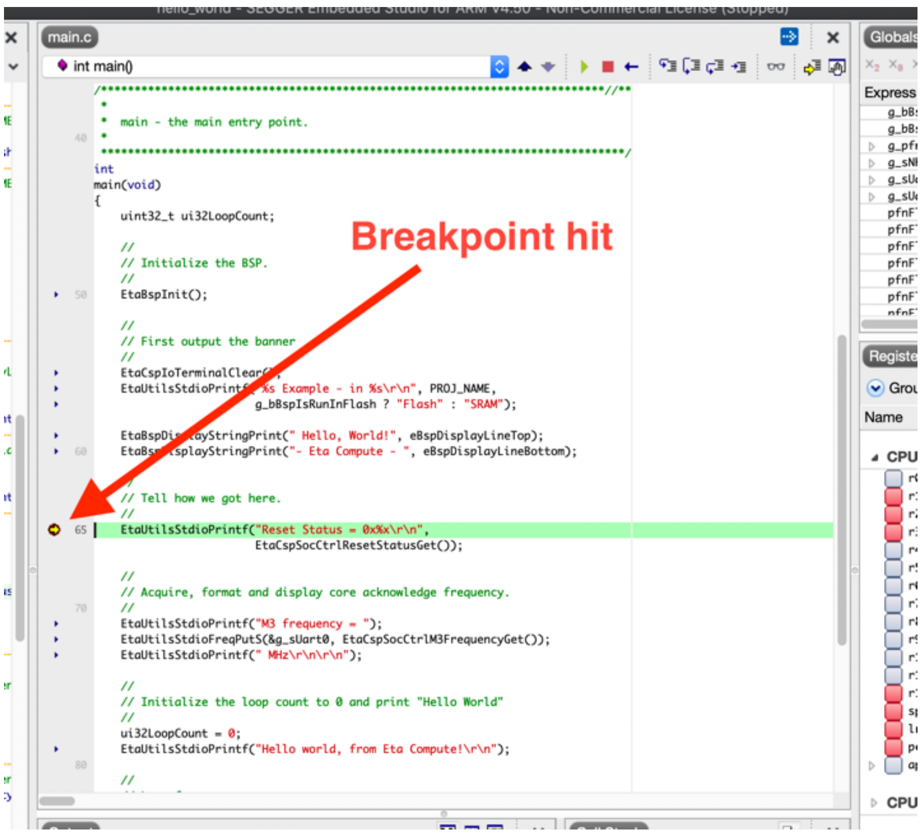
```
/*******************************************************************//**
 *
 *  main - the main entry point.
 *
 ********************************************************************/
int
main(void)
{
    uint32_t ui32LoopCount;

    //
    // Initialize the BSP.
    //
    EtaBspInit();

    //
    // First output the banner
    //
    EtaCspIoTerminalClear();
    EtaUtilsStdioPrintf("%s Example - in %s\r\n", PROJ_NAME,
                        g_bBspIsRunInFlash ? "Flash" : "SRAM");

    EtaBspDisplayStringPrint(" Hello, World!", eBspDisplayLineTop);
    EtaBspDisplayStringPrint("- Eta Compute - ", eBspDisplayLineBottom);

    //
    // Tell how we got here.
    //
    EtaUtilsStdioPrintf("Reset Status = 0x%x\r\n",
                        EtaCspSocCtrlResetStatusGet());

    //
    // Acquire, format and display core acknowledge frequency.
    //
    EtaUtilsStdioPrintf("M3 frequency = ");
    EtaUtilsStdioFreqPutS(&g_sUart0, EtaCspSocCtrlM3FrequencyGet());
    EtaUtilsStdioPrintf(" MHz\r\n\r\n");

    //
    // Initialize the loop count to 0 and print "Hello World"
    //
    ui32LoopCount = 0;
    EtaUtilsStdioPrintf("Hello world, from Eta Compute!\r\n");

    //
```

Figure 8.12: Toggle a Breakpoint


hello_world - SEGGER Embedded Studio for ARM V4.50 - Non-Commercial License (Stopped)

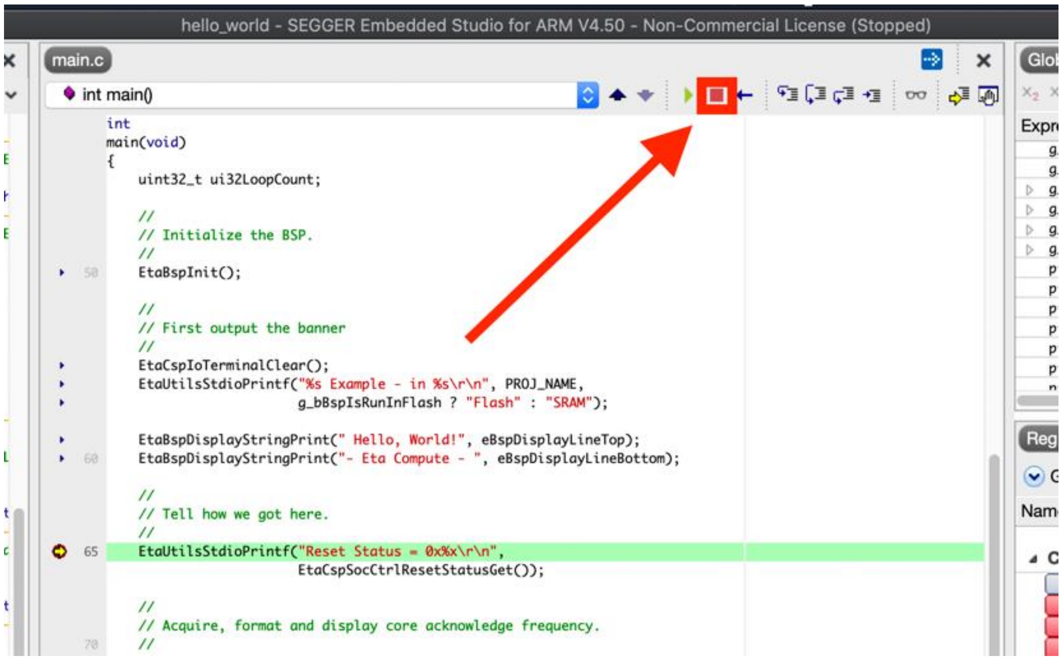
```
/*******************************************************************//**
 *
 *  main - the main entry point.
 *
 ********************************************************************/
int
main(void)
{
    uint32_t ui32LoopCount;

    //
    // Initialize the BSP.
    //
```

Figure 8.13: Free Run


hello_world - SEGGER Embedded Studio for ARM V4.50 - Non-Commercial License (Stopped)
main.c
int main()
Globals
Breakpoint hit

```
/*********************************************************************//**
 *
 *  main - the main entry point.
 *
 ************************************************************************/
int
main(void)
{
    uint32_t ui32LoopCount;

    //
    // Initialize the BSP.
    //
    EtaBspInit();

    //
    // First output the banner
    //
    EtaCspIoTerminalClear();
    EtaUtilsStdioPrintf("%s Example - in %s\r\n", PROJ_NAME,
                        g_bBspIsRunInFlash ? "Flash" : "SRAM");

    EtaBspDisplayStringPrint(" Hello, World!", eBspDisplayLineTop);
    EtaBspDisplayStringPrint("- Eta Compute - ", eBspDisplayLineBottom);

    //
    // Tell how we got here.
    //
    EtaUtilsStdioPrintf("Reset Status = 0x%x\r\n",
                        EtaCspSocCtrlResetStatusGet());

    //
    // Acquire, format and display core acknowledge frequency.
    //
    EtaUtilsStdioPrintf("M3 frequency = ");
    EtaUtilsStdioFreqPutS(&g_sUart0, EtaCspSocCtrlM3FrequencyGet());
    EtaUtilsStdioPrintf(" MHz\r\n\r\n");

    //
    // Initialize the loop count to 0 and print "Hello World"
    //
    ui32LoopCount = 0;
    EtaUtilsStdioPrintf("Hello world, from Eta Compute!\r\n");

    //
```

Figure 8.14: Breakpoint Hit


hello_world - SEGGER Embedded Studio for ARM V4.50 - Non-Commercial License (Stopped)
main.c
int main()

```
int
main(void)
{
    uint32_t ui32LoopCount;

    //
    // Initialize the BSP.
    //
    EtaBspInit();

    //
    // First output the banner
    //
    EtaCspIoTerminalClear();
    EtaUtilsStdioPrintf("%s Example - in %s\r\n", PROJ_NAME,
                        g_bBspIsRunInFlash ? "Flash" : "SRAM");

    EtaBspDisplayStringPrint(" Hello, World!", eBspDisplayLineTop);
    EtaBspDisplayStringPrint("- Eta Compute - ", eBspDisplayLineBottom);

    //
    // Tell how we got here.
    //
    EtaUtilsStdioPrintf("Reset Status = 0x%x\r\n",
                        EtaCspSocCtrlResetStatusGet());

    //
    // Acquire, format and display core acknowledge frequency.
    //
```

Figure 8.15: Stop Debugging

8.2 GCC

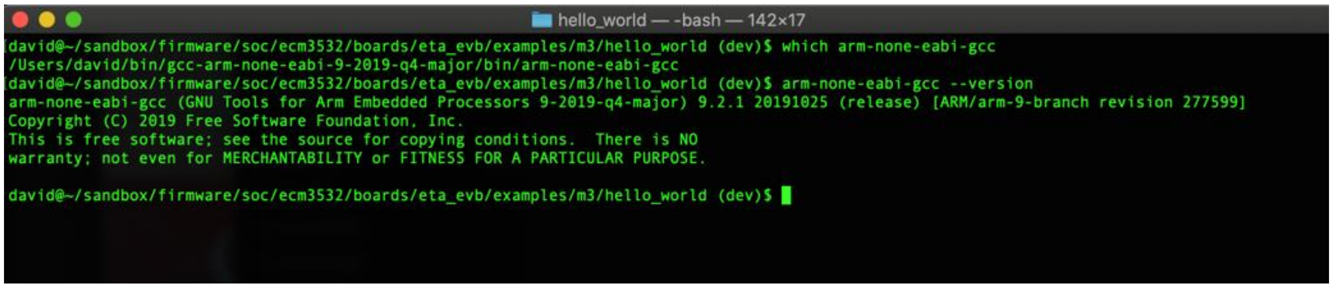
GCC is a cross-compilation compiler for devices based on the Arm Cortex-M and Cortex-R processors. GNU Arm Embedded toolchain contains the Arm Embedded GCC compiler, libraries and other GNU tools necessary for bare-metal software development.

8.2.2 Setup and Installation

1. Download the ARM GCC toolchain for your operating system.
2. Install or extract the download to your PC
3. Download the J-Link Software and Documentation Package (v6.70 or later) for your operating system and install it.
4. Add the path to the arm-none-eabi-gcc and JLink executables to your system path.

8.2.3 Compile and Flash

To compile the GCC 'hello_world' navigate to the project directory shown in Section 8 and seen below in Fig 8.15. You can also see in Fig 8.15 that we have verified we have the GCC toolchain in our path as well as double checking its version. If you chose to install the SEGGER J-Link tools as well, check that they are in your path, as shown in Fig 8.6 above.



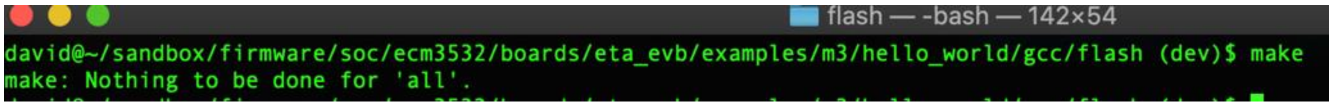
```
david@~/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world (dev)$ which arm-none-eabi-gcc
/Users/david/bin/gcc-arm-none-eabi-9-2019-q4-major/bin/arm-none-eabi-gcc
david@~/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world (dev)$ arm-none-eabi-gcc --version
arm-none-eabi-gcc (GNU Tools for Arm Embedded Processors 9-2019-q4-major) 9.2.1 20191025 (release) [ARM/arm-9-branch revision 277599]
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

david@~/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world (dev)$ 
```

Fig 8.15: Check GCC Path and Version

Now we can attempt a compile and link via 'make gcc'. Also, if you also have the SEGGER tools in your path you can build from the command line using 'make es'. A simple 'make' will attempt to compile everything (SES, GCC, Keil, IAR), if the correct executables are found in your path. For now, we will just attempt a GCC compile. Out of the box the software examples have already been compiled and tested. Therefore, the compilation will be quick and benign as there is nothing out of date. Inside the 'hello_world' directory, there is a 'gcc' directory. Inside that are 3 directories, flash, flash_boot_sram, and sram. For this guide we will use the flash project. The different project types are described below:

flash	Stored in flash, executes from flash
flash_boot_sram	Stored in flash, copies itself into sram, executes from sram
sram	Stored in sram, executes from sram

If we execute ‘make’ in the ‘flash’ directory we can see that there is nothing to do as everything has already been compiled as mentioned. This can be seen in Fig 8.16. If a source file is updated, the source file that has changed will be recompiled. The command ‘make clean’ is also available which will clean the compiled objects and remake everything using the version of the toolchain you just installed.



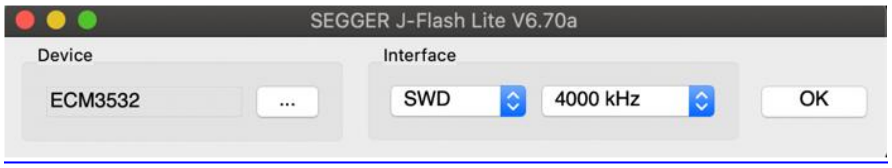
```
david@~/sandbox/firmware/soc/ecm3532/boards/eta_evb/examples/m3/hello_world/gcc/flash (dev)$ make
make: Nothing to be done for 'all'.
```

Fig 8.16: GCC Compile up to date.

To flash the newly compiled binary, we will use the SEGGER JFlashLite tool.

Note: If you prefer a command line option to program, jump to Section ‘8.2.4 Debug’.

Navigate to the location you installed your SEGGER tools and start JFlashLite. If launched correctly, you should see something similar to Fig 8.17. From this window, select the correct device. In this case, that is the ECM3532 and click “OK”.



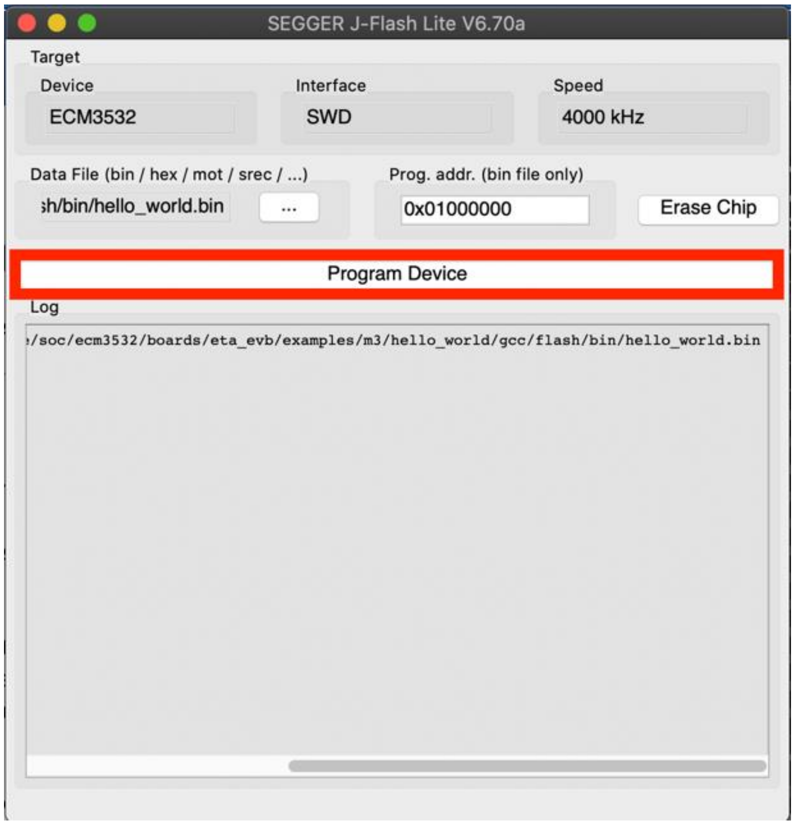
SEGGER J-Flash Lite V6.70a

Device: ECM3532

Interface: SWD, 4000 kHz

Fig 8.17: JFlashLite Device Selection.

Once the device is selected you can select the .bin file to program. You will also need to update the “Prog addr.” field as the ECM3532 user flash begins at address 0x01000000. Once you have your bin file selected and program address updated you can click “Program Device” as shown in Fig 8.18.


SEGGER J-Flash Lite V6.70a
Target
Device
Interface
Speed
Data File (bin / hex / mot / srec / ...)
Prog. addr. (bin file only)
...
Erase Chip
Program Device
Log

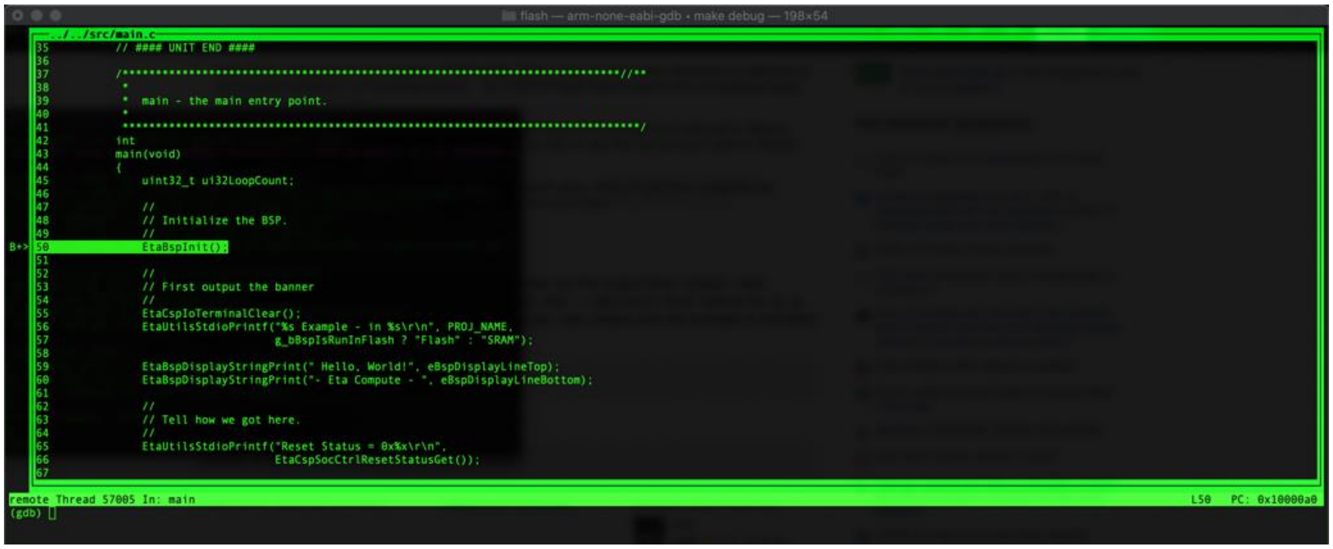
Device: ECM3532
Interface: SWD
Speed: 4000 kHz
Data File (bin / hex / mot / srec / ...): sh/bin/hello_world.bin
Prog. addr. (bin file only): 0x01000000

```
/soc/ecm3532/boards/eta_evb/examples/m3/hello_world/gcc/flash/bin/hello_world.bin
```

Figure 8.18: Program Device

8.2.4 Debug

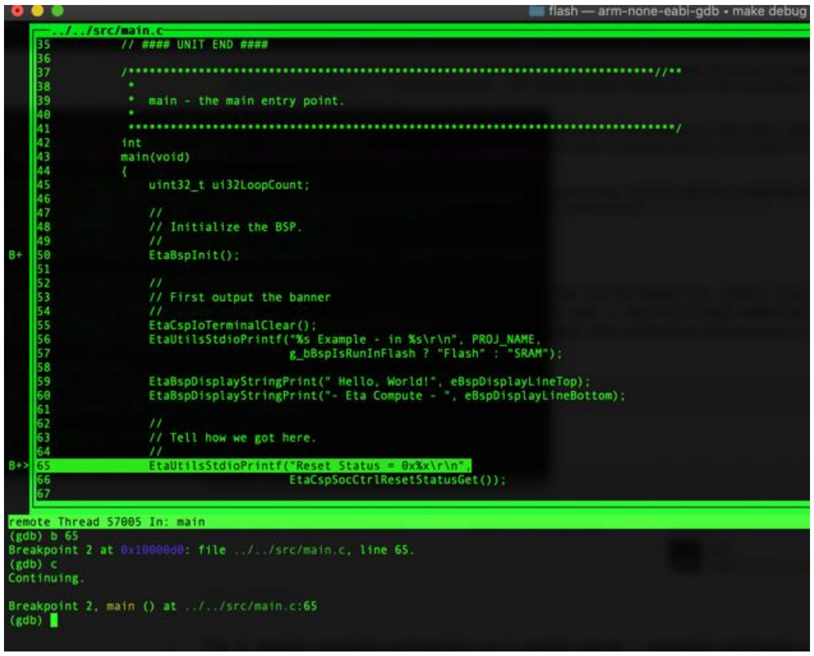
There are many ways one might chose to debug a GCC binary but a common way is to use GDB. SEGGER includes a GDB server with their download so we will use that to debug this application. First, we need to start the GDB server and then connect to the running server using the GDB executable. Once connected we need to configure GDB to program the binary. Luckily, all of the hard work has been taken care of for you. To begin a GCC/GDB debug session just execute 'make debug' inside the project you are working on. This will automatically start the JLinkGDBServer, connect to the running server's GDB port, load the application, load the debug symbol information, bring up a source window, set a break point at main, and run to that breakpoint. The bulk of the work is done using the gdb_server.sh (**under Windows, there is a condition in the Makefile to call a Windows PowerShell script that mimics the functionality of the shell script. You must have the toolchain, JLinkGDBServer, and 'make' available in your PowerShell terminal)** script under the "bin" directory at the root of the TensiSDK and the '.gdbinit' script inside the project. The result of this can be seen in Figure 8.19. If you do not see the source lines then you most likely do not have "tui" support in the version of GDB you have. This allows you to see source information within the command terminal.



```
../../src/main.c
        // #### UNIT END ####

        /******************************************************************************//**
         *
         *  main - the main entry point.
         *
         ******************************************************************************/
        int
        main(void)
        {
            uint32_t ui32LoopCount;

            //
            // Initialize the BSP.
            //
B+>         EtaBspInit();

            //
            // First output the banner
            //
            EtaCspIoTerminalClear();
            EtaUtilsStdioPrintf("%s Example - in %s\r\n", PROJ_NAME,
                                g_bBspIsRunInFlash ? "Flash" : "SRAM");

            EtaBspDisplayStringPrint(" Hello, World!", eBspDisplayLineTop);
            EtaBspDisplayStringPrint("- Eta Compute - ", eBspDisplayLineBottom);

            //
            // Tell how we got here.
            //
            EtaUtilsStdioPrintf("Reset Status = 0x%x\r\n",
                                EtaCspSocCtrlResetStatusGet());

remote Thread 57005 In: main                                                    L50   PC: 0x10000a0
(gdb)
```

Figure 8.19: GDB Debugging

From here is a standard GDB debug process. We can set breakpoints and run/halt the core from this interface. Figure 8.20 shows a breakpoint and run to illustrate this. To exit the debug session type "quit" and confirm with "y" that you would like to exit.



```
flash — arm-none-eabi-gdb • make debug
   ┌──../../src/main.c──────────────────────────────────────────────────────
   │35          // #### UNIT END ####
   │36
   │37          /**************************************************************************//**
   │38           *
   │39           *  main - the main entry point.
   │40           *
   │41           *****************************************************************************/
   │42          int
   │43          main(void)
   │44          {
   │45              uint32_t ui32LoopCount;
   │46
   │47              //
   │48              // Initialize the BSP.
   │49              //
B+ │50              EtaBspInit();
   │51
   │52              //
   │53              // First output the banner
   │54              //
   │55              EtaCspIoTerminalClear();
   │56              EtaUtilsStdioPrintf("%s Example - in %s\r\n", PROJ_NAME,
   │57                                  g_bBspIsRunInFlash ? "Flash" : "SRAM");
   │58
   │59              EtaBspDisplayStringPrint(" Hello, World!", eBspDisplayLineTop);
   │60              EtaBspDisplayStringPrint("- Eta Compute - ", eBspDisplayLineBottom);
   │61
   │62              //
   │63              // Tell how we got here.
   │64              //
B+>│65              EtaUtilsStdioPrintf("Reset Status = 0x%x\r\n",
   │66                                  EtaCspSocCtrlResetStatusGet());
   │67
   └────────────────────────────────────────────────────────────────────────
remote Thread 57005 In: main
(gdb) b 65
Breakpoint 2 at 0x10000d0: file ../../src/main.c, line 65.
(gdb) c
Continuing.

Breakpoint 2, main () at ../../src/main.c:65
(gdb) █
```

Figure 8.20: GDB Breakpoint and Run

APPENDIX A: Troubleshooting

- **I don't see and serial output on my terminal**

  Double check:

  1. The board is powered over J6 pin 3 or VBAT and that the J3 jumper is set for the supplied source.
  2. Terminal settings match what is configured on the device.
     a. The default is 115,200 baud, 8N1 with no flow control.
  3. Press and release POR (S2)

- **I can't connect to the device or flash the device with a new example**

  Double check:

  1. The board is powered over J6 pin 3 or VBAT and that the J3 jumper is set for the supplied source.
  2. The device is not in sleep or stall mode. To ensure this, press the POR button (S2) before attempting to connect to the ECM3532.
  3. If using a debug adapter on Windows, be sure that the drivers have installed correctly on your target PC.

APPENDIX B: Revision History

Date	Version	Changes
05-06-2020	0.1	Alpha release of ECM3532 AI Sensor User Guide

Contact Information

Address	Eta Compute, Inc. 310 N Westlake Blvd. Suite 110 Westlake Village, California 91362
Website	https://etacompute.com/
Support	support@etacompute.com

Legal Information and Disclaimers