

ECM3532 Evaluation Board

User Guide



ECM3532EVB-UG
Revision 0.3
April 2020

Table of Contents

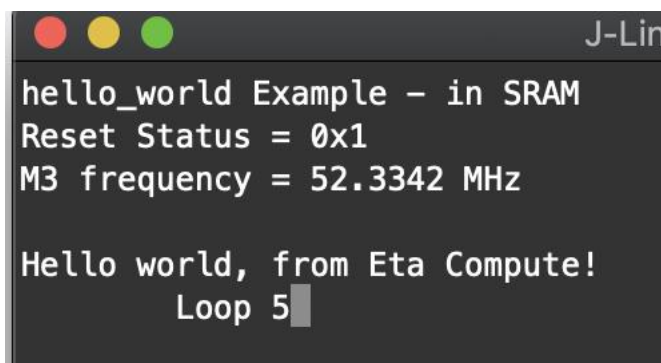
1. Introduction.....	3
1.1. Programmed Example	3
2. Quick Start.....	4
3. Overview	5
4. Block Diagram.....	6
5. Board Power.....	7
5.1 Measuring Power	7
6. Default Board Configuration.....	8
6.1 Jumpers	11
6.2 On-Board Regulators.....	11
6.3 Potentiometer	11
6.4 LEDs	11
6.5 Buttons	11
6.6 UART0/1 & DSP_UART.....	11
6.7 UART0	11
6.8 UART1	12
6.9 SPI0, SPI1	12
6.10 I2C0, I2C1	12
7. TensaiSDK.....	13
7.1 Installation.....	13
8. Supported IDEs & Compilers.....	14
8.1 Embedded Studio.....	14
8.2 GCC	24
APPENDIX A: Troubleshooting	29
APPENDIX B: Revision History.....	30

1. Introduction

This document serves as the reference guide for working with the ECM3532EVB.

1.1. Programmed Example

Each ECM3532EVB Evaluation board (EVB) is shipped with the 'hello_world' example programmed in Flash. However, this example copies itself into SRAM and then prints out a "Hello World" (Figure 1.1) message to the screen and increments a loop count ever 3 seconds.



```
hello_world Example - in SRAM
Reset Status = 0x1
M3 frequency = 52.3342 MHz

Hello world, from Eta Compute!
  Loop 5
```

Figure 1.1. Hello World print message

2. Quick Start

1. Install USB cable (standard A to micro B) into the left USB connector (J19: M3 Debug), applying power to the board.
2. The hello world example will clear the terminal and begin printing to UART0, which is the first available “/dev/tty*” or COM port (depending on your OS)
3. Bring up a serial terminal (PuTTY) on that port.
 - a. Set for 115200, 1 stop, no parity
 - b. Press/release SW4 (POR). This will reset the device and start the application.
 - c. You should see an output that looks similar to Figure 1.
4. The EVB contains a 2-line OLED display. Upon power up, you should see a hello world message on the display as well. If the display is blank, make sure J13 (DISP) is installed.
5. For programming, the EVB contains a J-Link OB which is supported natively in a variety of IDEs and tools. Out of the box the Tensi® 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 Evaluation Board (EVB) showcases the capabilities of the ECM3532 device with an Asynchronous ARM Cortex-M3 built with our Delay Insensitive Asynchronous Logic (DIAL). The SoC includes a CoolFlux Dual MAC DSP, an always-on block and numerous peripheral controllers. The board has serial wire debug (SWD) and a built in J-Link OB for serial communication and debugging, as well as USB power. A 2-line OLED is included as are GPIO headers, LEDs, and push buttons. See Fig.4.1

Block Diagram. The ECM3532EVB is also compatible with many Arduino shields.

While every ECM3532 digital pin is available on the two SIDECAR 2x20 headers (J9 and J10) on the extreme of either side of the board, a subset of ECM3532 pins are wired to the Arduino shield pins accessible from either the top or bottom of the EVB. The SHIELD pinout is intended to work with Arduino R3 compatible shields.

5. Board Power

The board can only be powered through the USB connector, J19. Onboard LDO regulators generate the required power for the ECM3532 and peripheral circuits.

The ECM3532 has (4) integrated buck regulators:

1. AO: Always On regulator for the RTC, etc.
2. M3: Generate the core voltage for the M3
3. MEM: Generate the voltage for the SRAMs, etc.
4. DSP: Generate the core voltage for the DSP.

5.1 Measuring Power

The VDD_IO voltage plane on the EVB is divided into the portion that directly supplies the

ECM3532 (VDD_IO) and the portion that supplies the rest of the loads on the EVAL board (VDD_IO_TRK). As a result, J11-1 to J11-2 is the correct place to measure the current consumed by just the ECM3532. It's recommended to remove J12 (SWD), before measuring current, to disconnect the SWD from the J-Link so to be sure the DAP interface is powered down.

Also, the EVB is designed to integrate smoothly into the Arduino ecosystem. It can operate in several SHIELD based configurations:

- The EVB supplies +3.3V to the SHIELD stack
- The EVB supplies VDD_IO_LS to the SHIELD stack
- The EVB receives +3.3V from the stack. VDD_IO and VDD_IO_LS can be sourced from here.

6. Default Board Configuration

Please see Fig 5.1a, 5.1b below for the boards default configuration. Fig 5.1c is the EVb silkscreen top for easier identification of the reference designators referred to throughout the document.

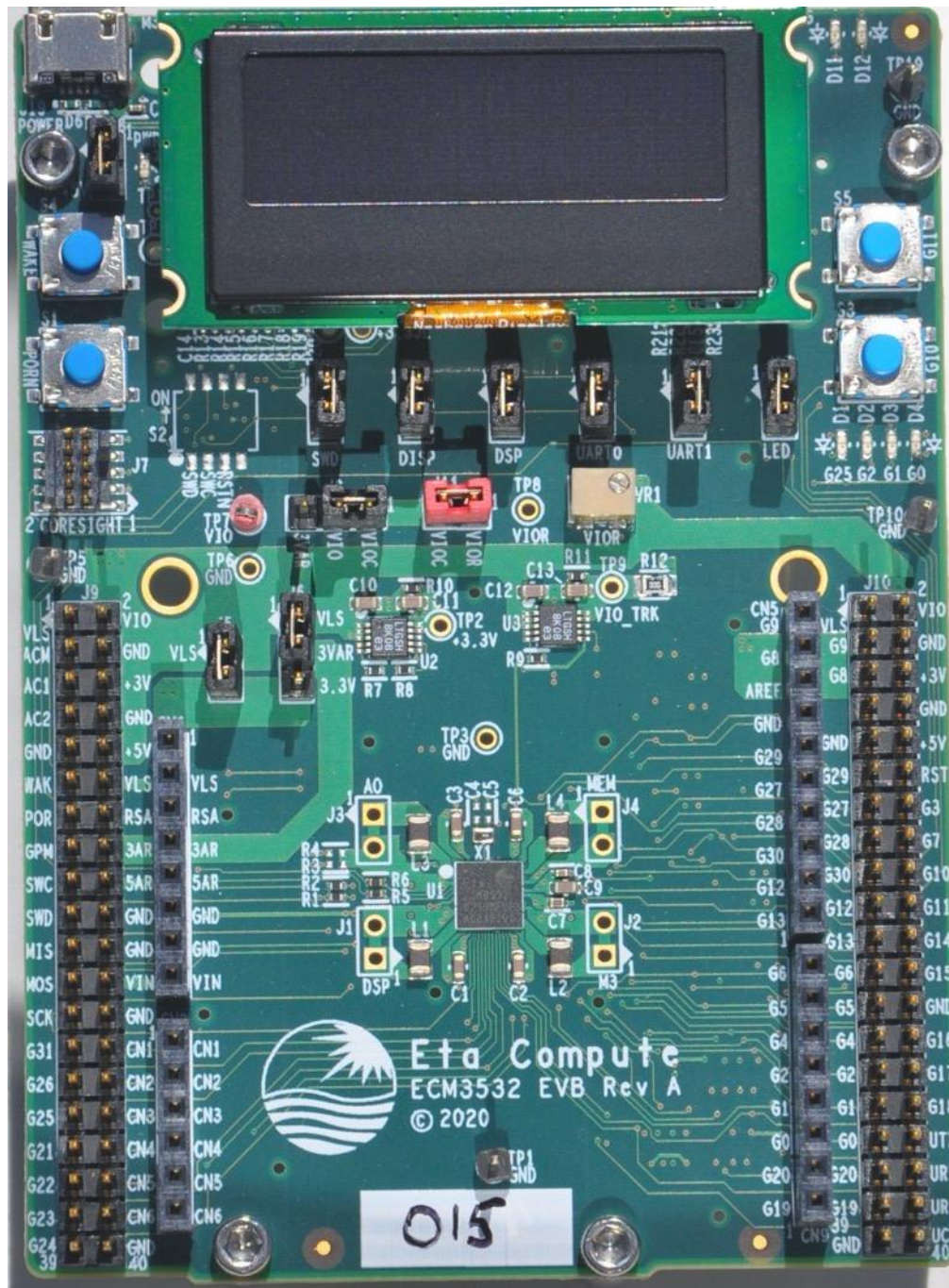


Figure 5.1a. Board Top

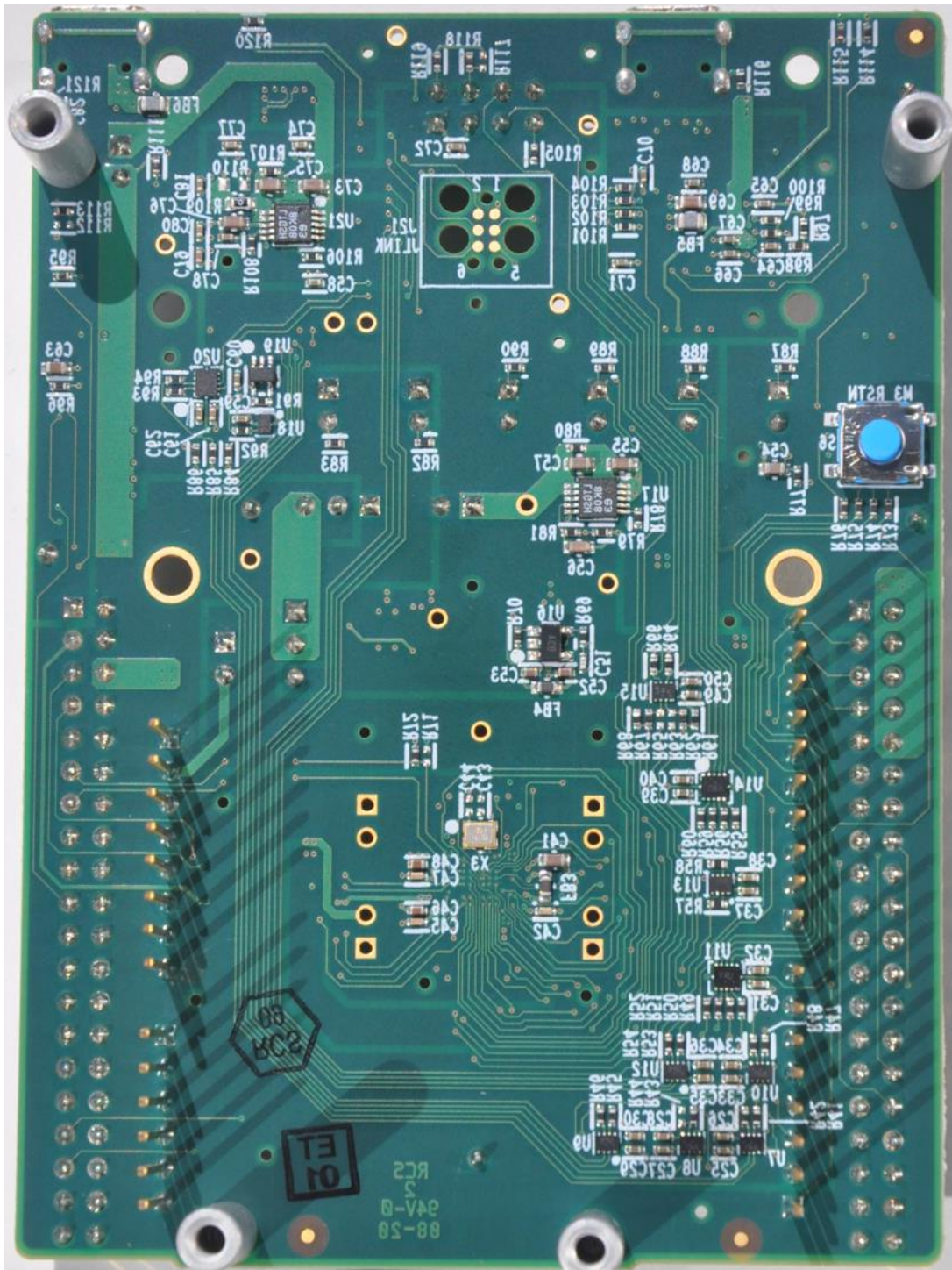


Figure 5.1b. Board Bottom



6.1 Jumpers

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

Table 6.1: Jumpers

J5	VDD_IO_LS to Arduino CN6_2
J6	VDD_IO_LS or +3.3V to Arduino CN6_4
J8	VDD_IO from Arduino or onboard
J11	VDD_IO to ECM3532. Connect an ammeter to measure ECM3532 current
J12	Enable SWD connection from J-Link. Remove before measuring current.
J13	Enable I2C connection to OLED display. Remove to disconnect I2C1 from the display
J14	Enable DSP debug path from FTDI. Remove to disconnect DSP GPIO pins from the FTDI.
J15	Enable UART0 connection to J-Link. Remove to disconnect UART0_RX and UART0_CTS from the J-Link
J16	Enable UART1 connection to J-Link. Remove to disconnect UART1_RX and UART1_CTS from the J-Link
J17	Enable the LED buffer. Remove to disconnect the LED buffer.

6.2 On-Board Regulators

VR1 - VDDIO LDO Adjustable (Probe TP21).
(DEFAULT = 3.3V)

6.3 Potentiometer

There is a potentiometer on board to configure the voltage for VDD_IO and VDD_IO_TRK (VDD_IO_LS). By default this is set to +3.3V.

6.4 LEDs

The EVB has 4 buffered LEDs connected through a mux to GPIOs[0:2] & GPIO25. These are buffered by U5 and U6.

6.5 Buttons

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

Table 6.2: Buttons

S1	Power on Reset (POR) to the ECM3532
S3	General purpose button on GPIO10
S4	Connected to WAKE pin on ECM3532
S5	General purpose button on GPIO11
S6	RESET_N to the ECM3532

6.6 UART0/1 & DSP_UART

UART0 and UART1 offer hardware flow control (RTS, CTS). All 4 pins for each UART are available on the Side 2x20 headers. The DSP UART is a single wire TX output only. All three UARTs are connected to the J-Link OB and available as a virtual COM port on your operating system of choice (Windows, MacOS, Linux). They are connected as follows:

1. UART0 -> COM0
2. UART1 -> COM1
3. DSP_UART -> COM2

6.7 UART0

UART0 by default is configured to operate through the J-Link OB USB interface on the first channel. This channel is operated as a UART (serial port) from the PC connected to the J-Link USB connector. UART0 has dedicated (non-

shared) signal pins on the ECM3532. If you want to connect UART0 to an off-board device with fly wires or if you want to utilize it with an Arduino shield, then be sure to disconnect it from the J-Link OB using the jumper J15 (UART0).

6.8 UART1

UART1 is muxed with other GPIOs. It is configured by default (J16 UART1) to connect it to the second channel on the J-Link. If you want to use these as GPIO pins, then remove J16 (UART1) to disconnect from the J-Link.

6.9 SPI0, SPI1

The ECM3532 supports (2) SPI busses, SPI0 and SPI1. For SPI0, the SPI clock, MISO and MOSI are on dedicated ECM3532 pins. The chip selects for use with SPI0 are shared with GPIO pins. Refer to the pin mux table in the data sheet.

6.10 I2C0, I2C1

The ECM3532 supports (2) I2C busses, I2C0 and I2C1. I2C0 SCL and SDA share pins with SPI0. Further note that I2C1 shares pins with SPI1 and the GPIO.

7. TensaiSDK

The TensaiSDK 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 ECM3532EVB 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 may 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 TensaiSDK 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_evb/examples/m3/hello_world`

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 EVB 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.

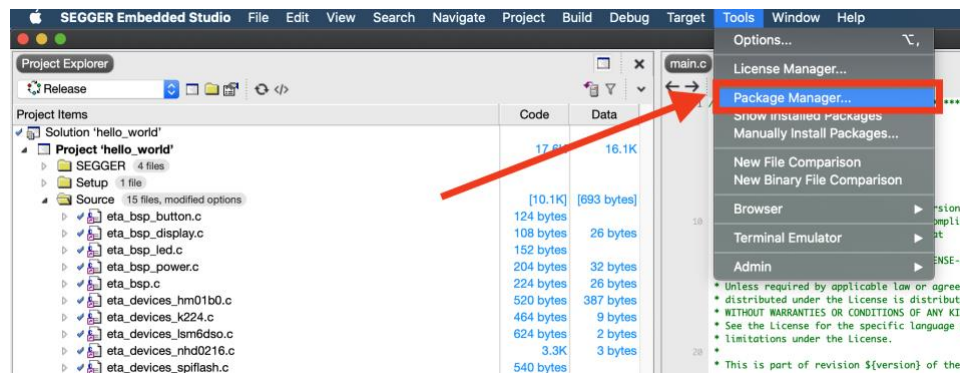


Figure 8.1a: Select Package Manager

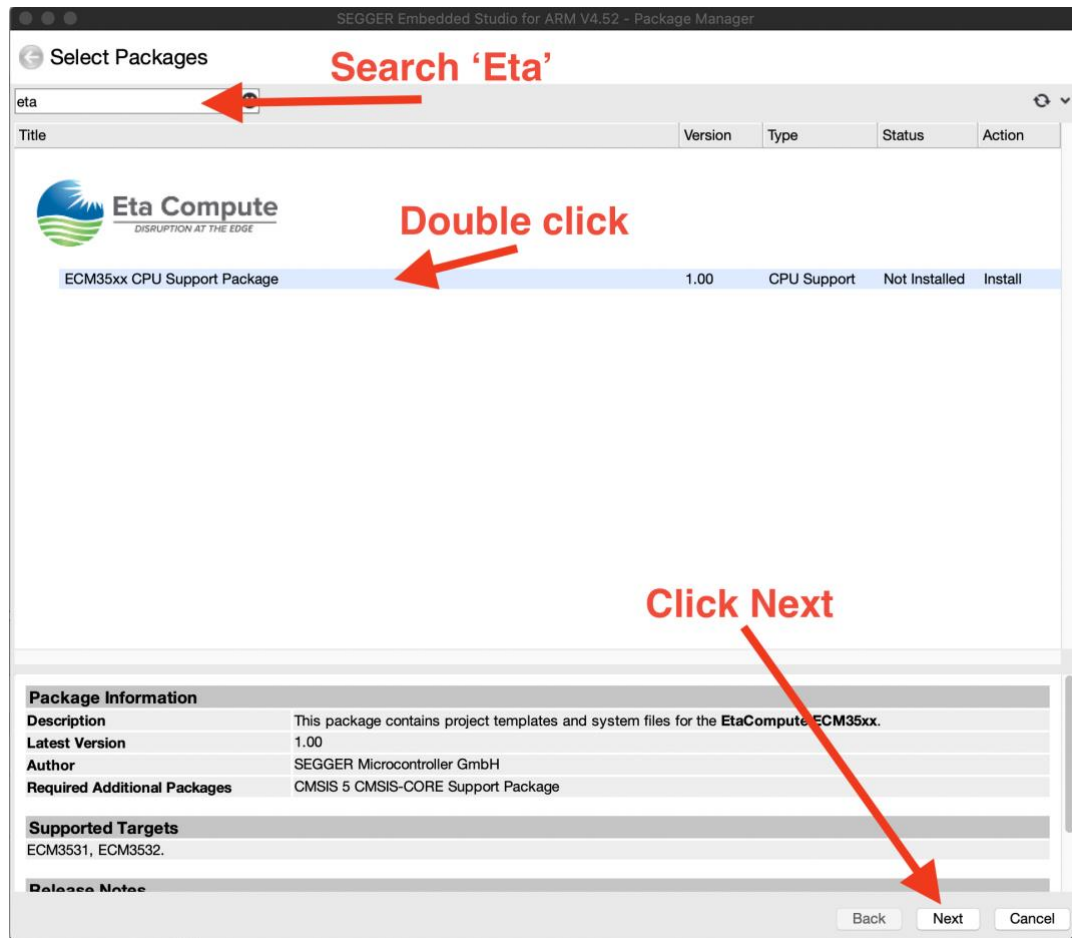


Figure 8.1b: Select and Install Eta Compute Pack

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.

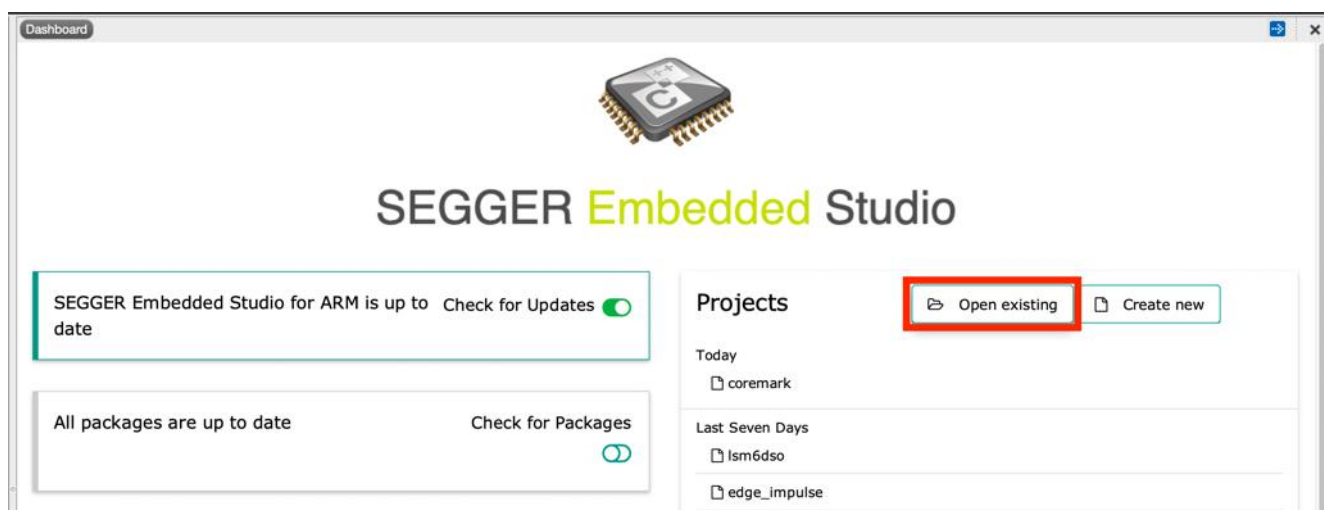


Figure 8.2: Open Project from Dashboard

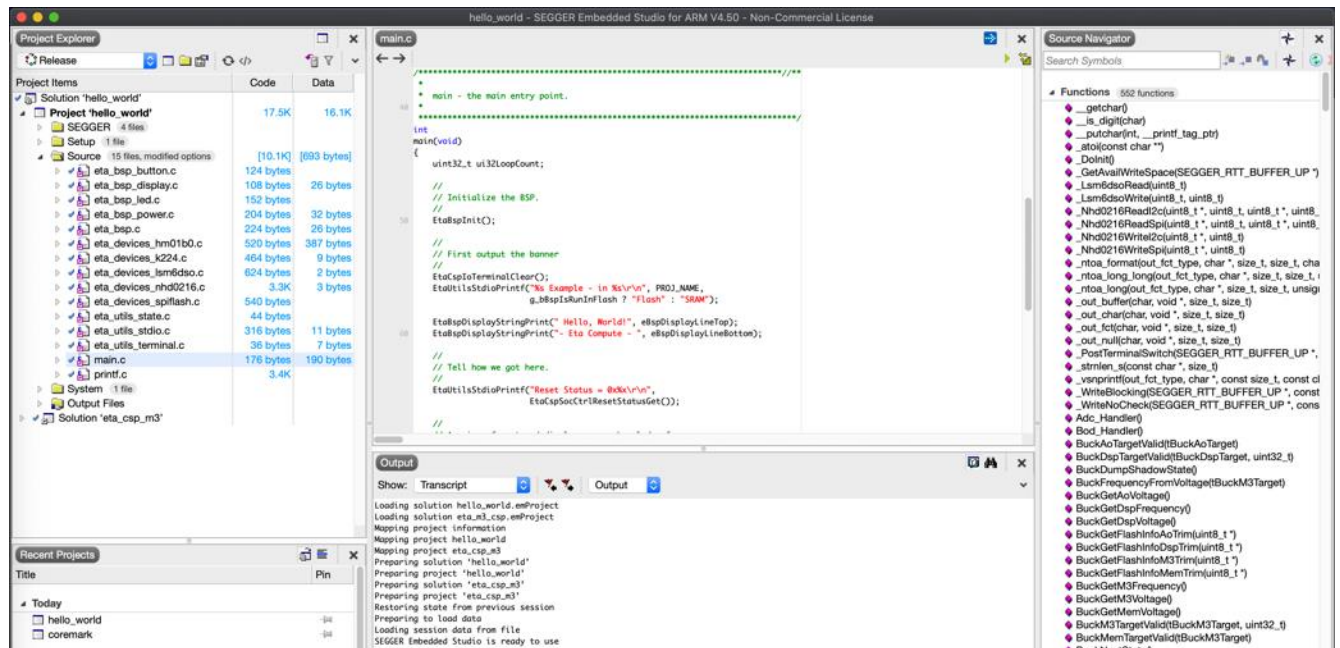


Figure 8.3: Project Opened

8.1.4 Compile and Flash

Now that the project is opened, we can compile it. Every TensiSDK 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.

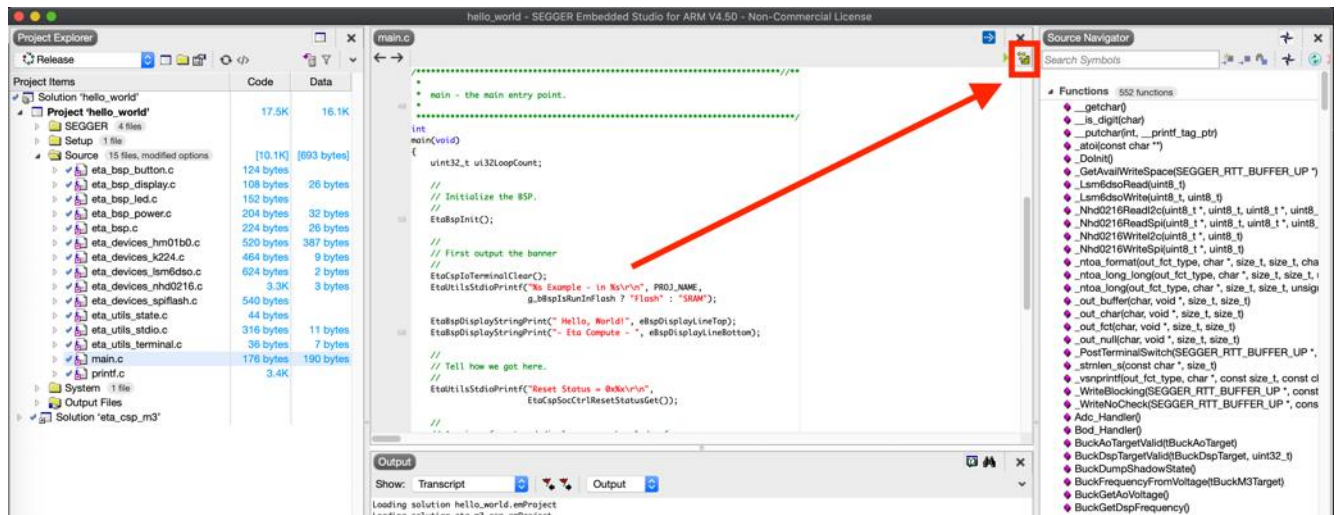


Figure 8.4: Build Project

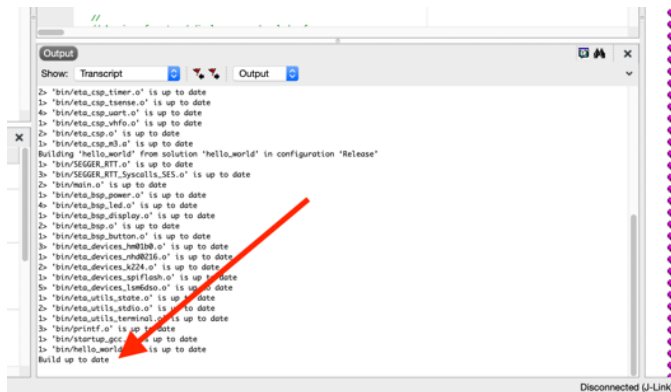


Figure 8.5a: Project is up-to-date

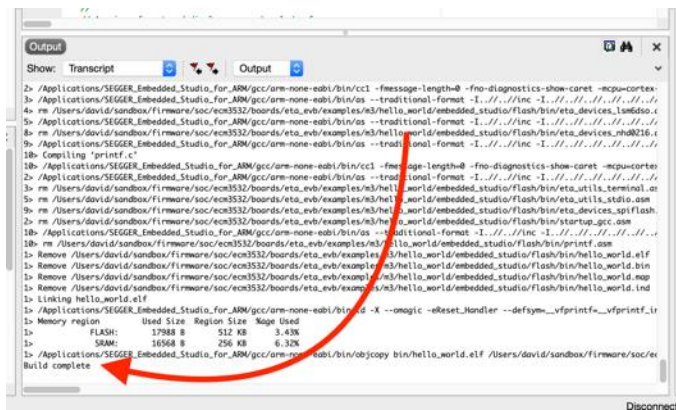


Figure 8.5b: Project is out-of-date and Compiles Successfully

```
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

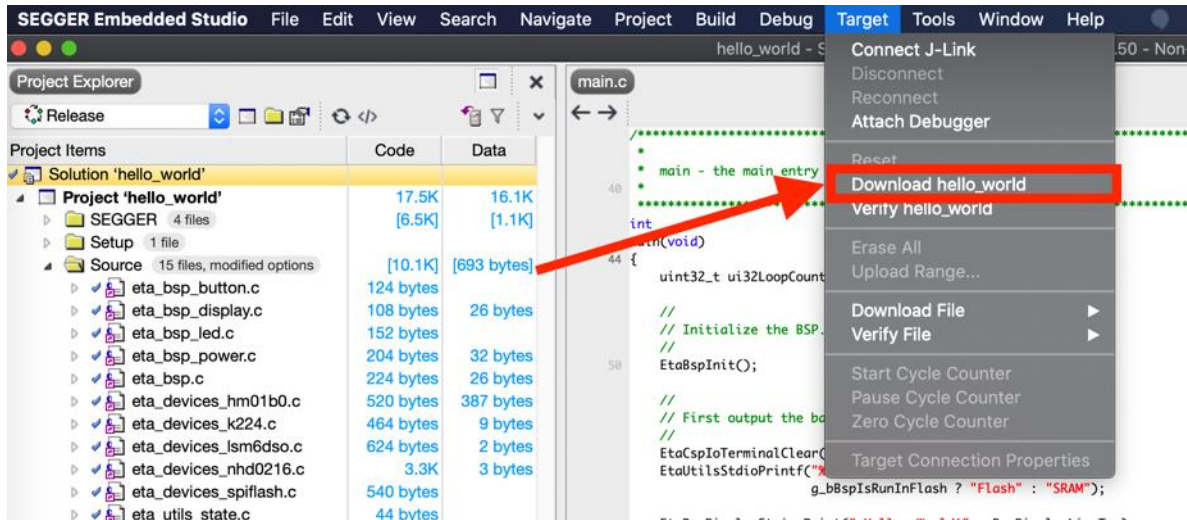


Figure 8.7: Download Application Without Debugging

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 using the onboard J-Link OB on the EVB. 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.

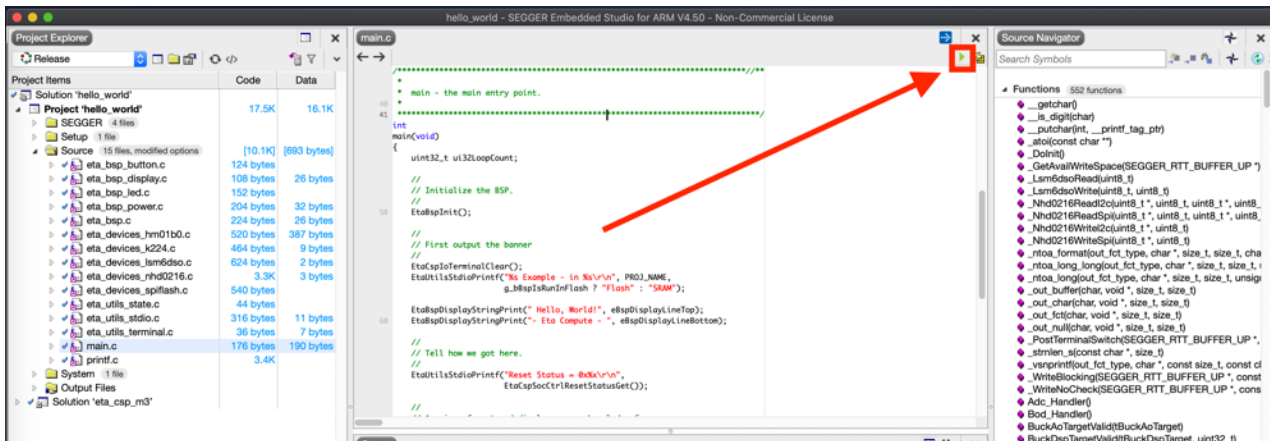


Figure 8.8: Start Execution

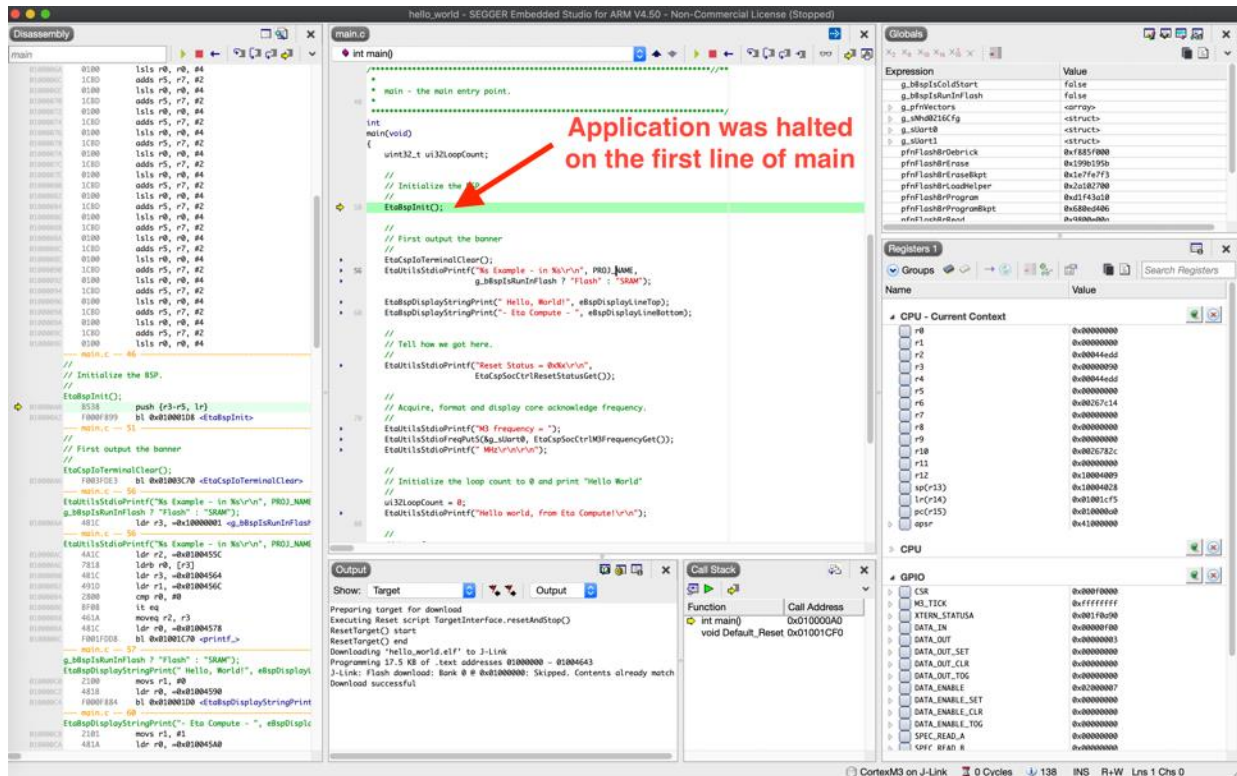


Figure 8.9: Debug View

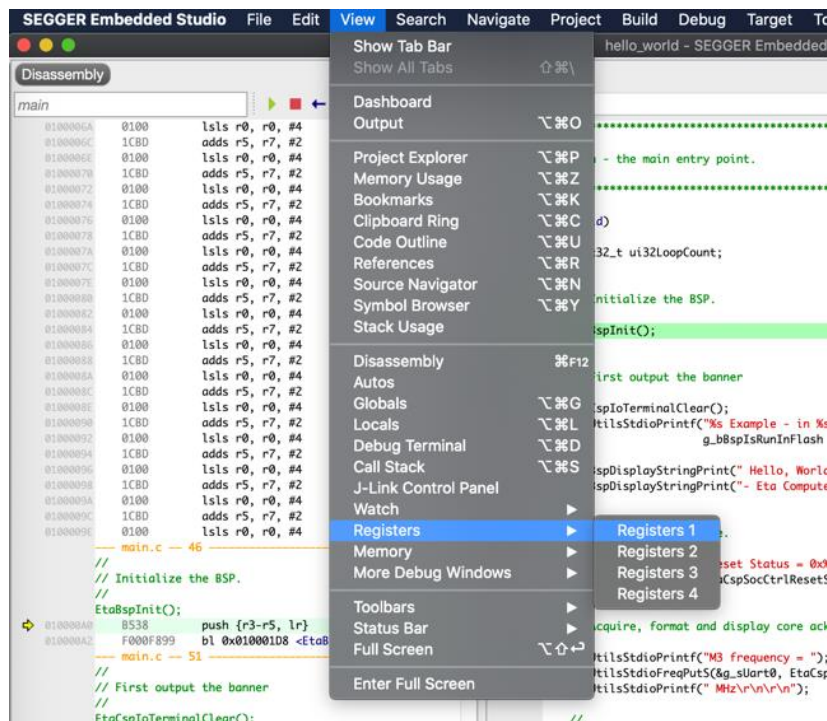


Figure 8.10: Add Different View Windows

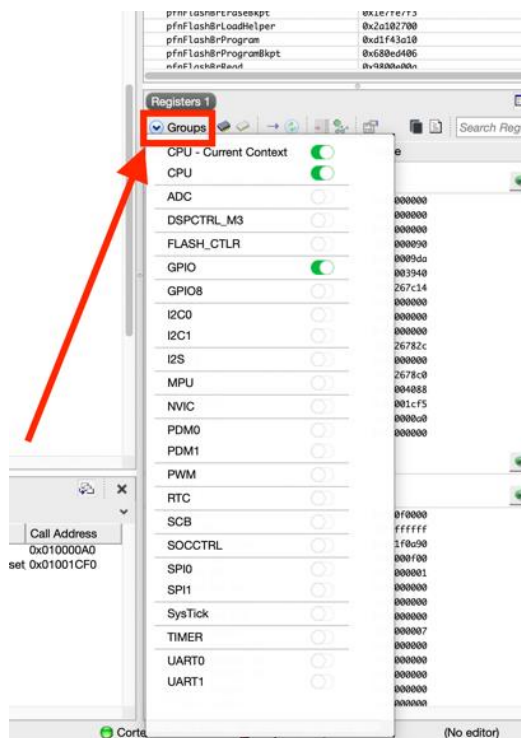


Figure 8.11: Add Different Peripheral Registers

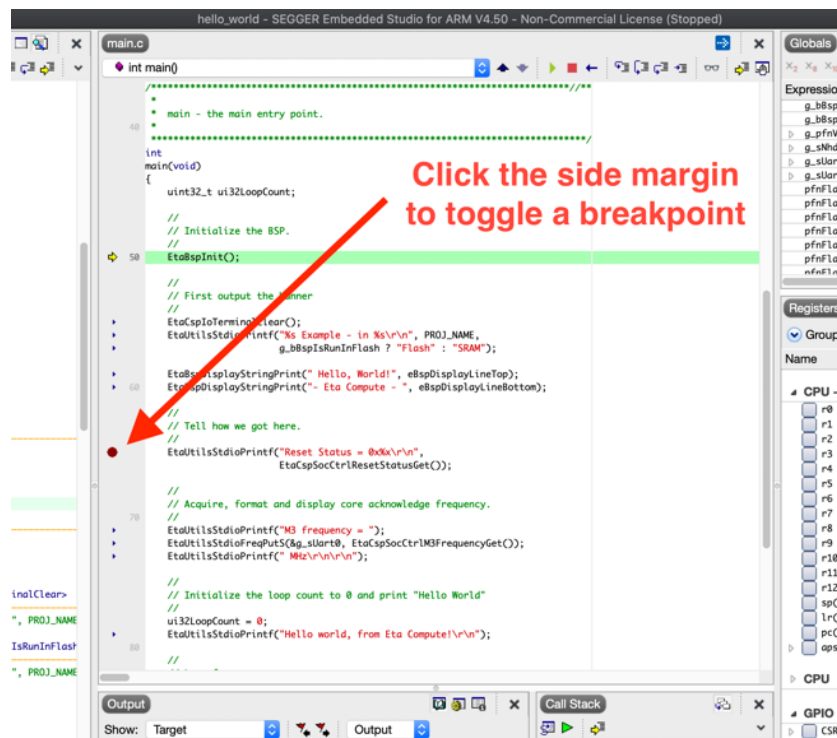


Figure 8.12: Toggle a Breakpoint



Figure 8.13: Free Run

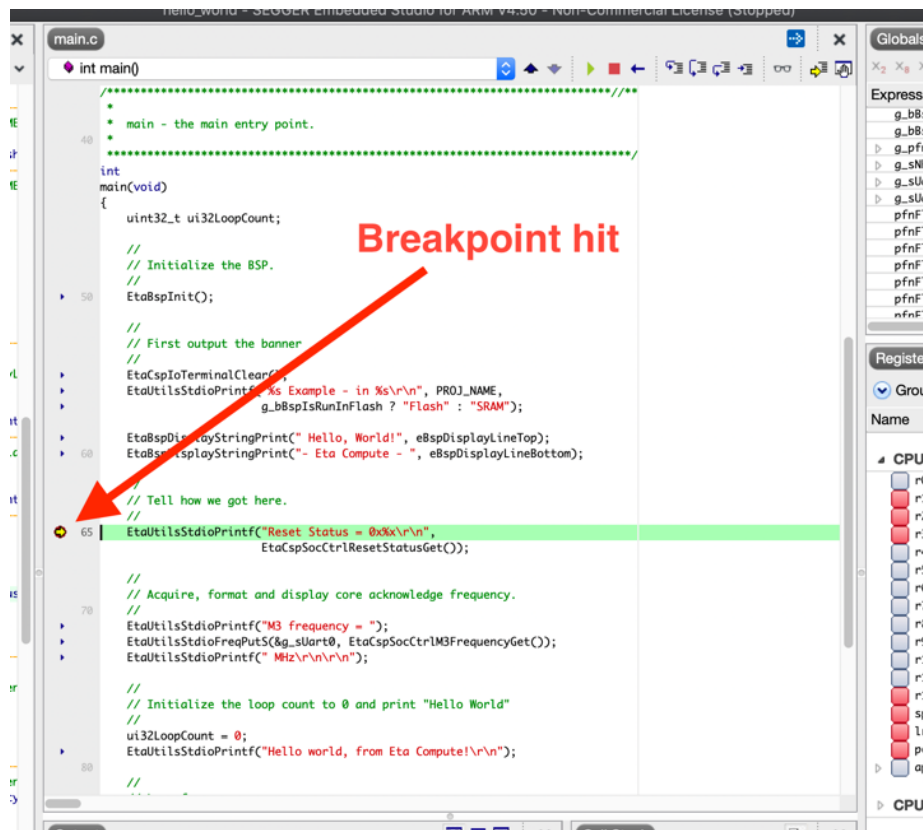


Figure 8.14: Breakpoint Hit

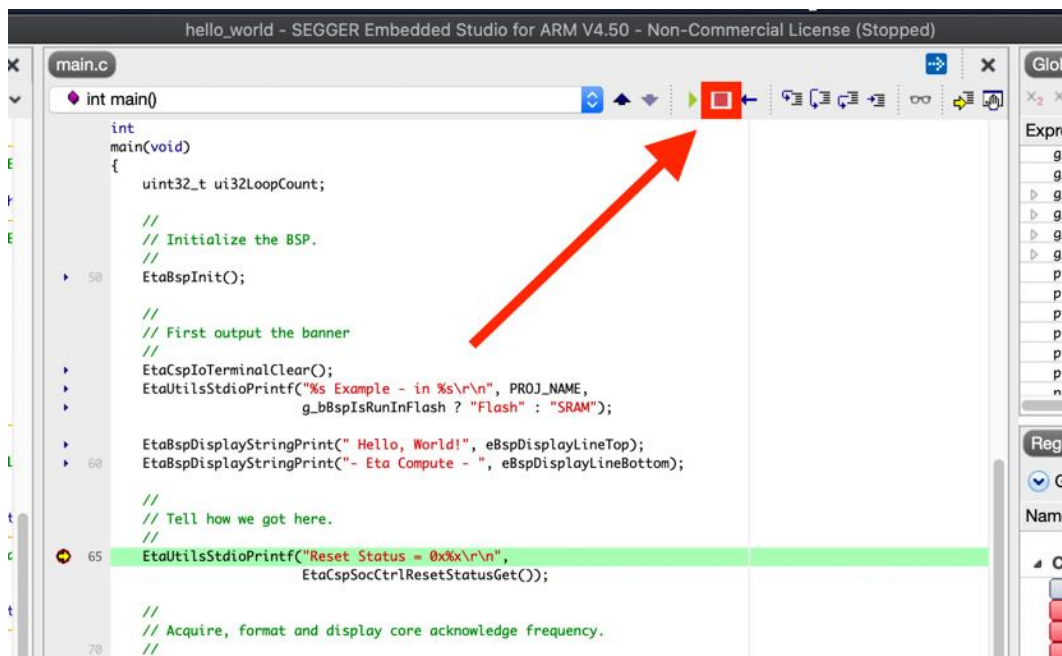


Figure 8.15: Stop Debugging

8.2 GCC

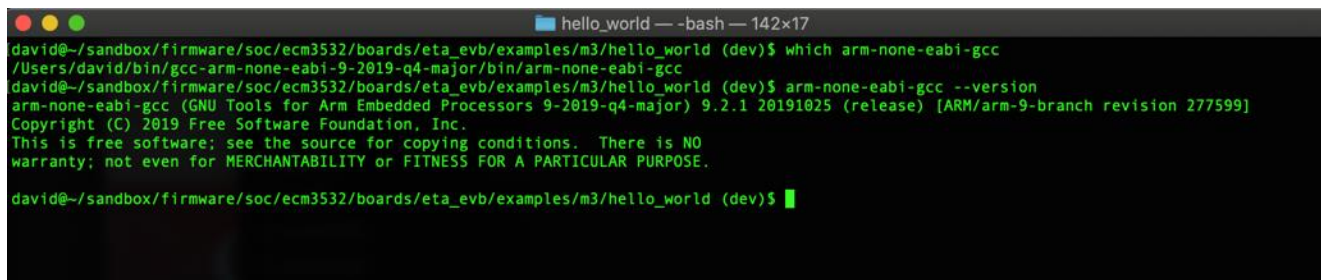
GCC is a cross-compilation compiler for devices based on the Arm Cortex-M and Cortex-R processors. GNU Arm Embedded toolchain contains the Arm Embedded GCC compiler, libraries and other GNU tools necessary for bare-metal software development.

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)$

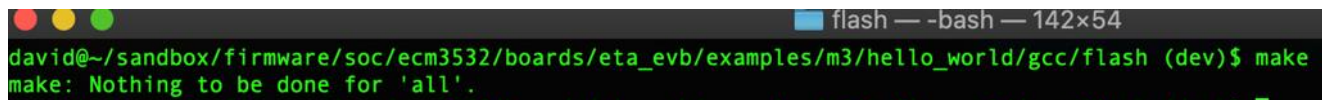
```

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.



```
flash — -bash — 142x54
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”.

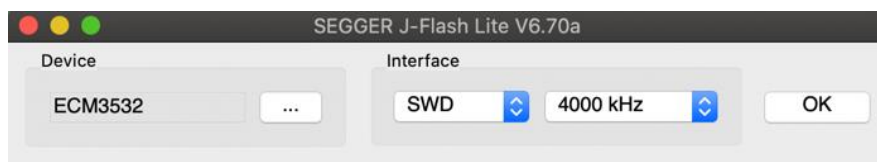


Fig 8.17: JFlashLite Device Selection.

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.

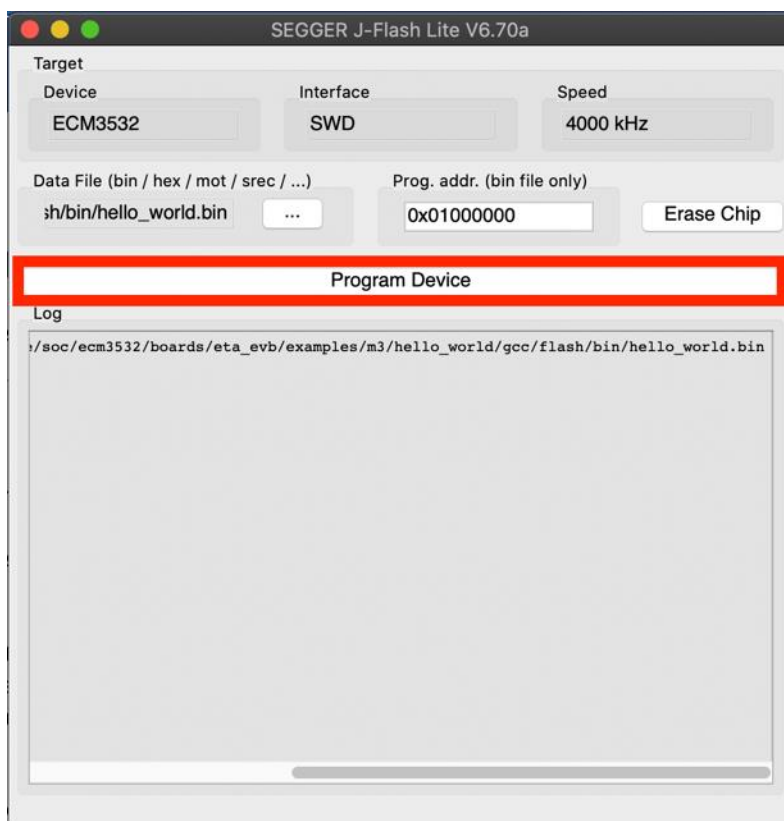
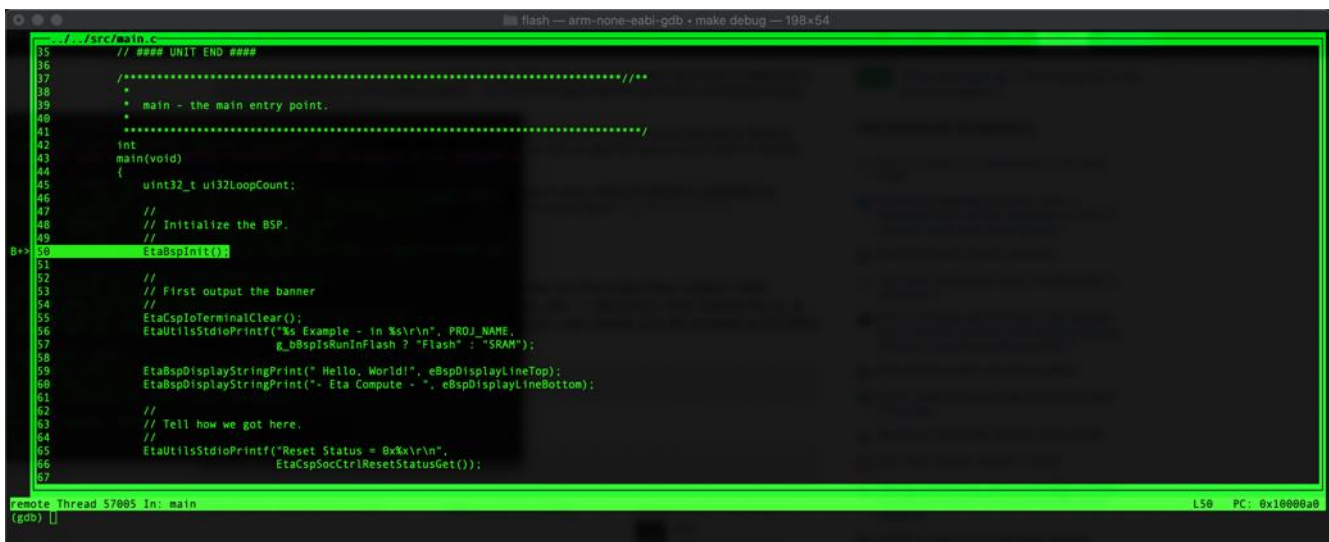


Figure 8.18: Program Device

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.



```

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     //
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     //
65     EtaUtilsStdioPrintf("Reset Status = 0x%x\r\n",
66                         EtaCspSocCtrlResetStatusGet());
67

```

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     //
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     //
65     EtaUtilsStdioPrintf("Reset Status = 0x%x\r\n",
66                        EtaCspSocCtrlResetStatusGet());
67
remote Thread 57005 In: main
(gdb) b 65
Breakpoint 2 at 0x1000000: 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 serial output on my terminal**

Double check:

1. The board is powered over USB (J19).
2. Terminal settings match what is configured on the device.
 - a. The default is 115,200 baud, 8N1 with no flow control.
3. All jumpers match the default configuration.

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

Double check:

1. The board is powered over USB (J19).
2. The device is not in sleep or stall mode. To ensure this, press the POR button (S1) 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.
4. All jumpers match the default configuration.

APPENDIX B: Revision History

Date	Version	Changes
03-31-2020	0.1	Alpha release of ECM3532EVB User Guide
04-01-2020	0.2	Updated EVB photo and jumpers
04-02-2020	0.3	Updated GCC and Embedded Studio and add silkscreen fig

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

ETA COMPUTE INTENDS FOR THE CONTENT CONTAINED IN THE DOCUMENT TO BE ACCURATE AND RELIABLE. THIS CONTENT MAY, HOWEVER, CONTAIN TECHNICAL INACCURACIES, TYPOGRAPHICAL ERRORS OR OTHER MISTAKES. ETA COMPUTE MAY MAKE CORRECTIONS OR OTHER CHANGES TO THIS CONTENT AT ANY TIME. ETA COMPUTE AND ITS SUPPLIERS RESERVE THE RIGHT TO MAKE CORRECTIONS, MODIFICATIONS, ENHANCEMENTS, IMPROVEMENTS AND OTHER CHANGES TO ITS PRODUCTS, PROGRAMS AND SERVICES AT ANY TIME OR TO DISCONTINUE ANY PRODUCTS, PROGRAMS, OR SERVICES WITHOUT NOTICE.

THE CONTENT IN THIS DOCUMENT IS PROVIDED "AS IS". ETA COMPUTE AND ITS RESPECTIVE SUPPLIERS MAKE NO REPRESENTATIONS ABOUT THE SUITABILITY OF THIS CONTENT FOR ANY PURPOSE AND DISCLAIM ALL WARRANTIES AND CONDITIONS WITH REGARD TO THIS CONTENT, INCLUDING BUT NOT LIMITED TO, ALL IMPLIED WARRANTIES AND CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHT.

ETA COMPUTE DOES NOT WARRANT OR REPRESENT THAT ANY LICENSE, EITHER EXPRESS OR IMPLIED, IS GRANTED UNDER ANY PATENT RIGHT, COPYRIGHT, MASK WORK RIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT OF ETA COMPUTE COVERING OR RELATING TO THIS CONTENT OR ANY COMBINATION, MACHINE, OR PROCESS TO WHICH THIS CONTENT RELATE OR WITH WHICH THIS CONTENT MAY BE USED.

USE OF THE INFORMATION IN THIS DOCUMENT MAY REQUIRE A LICENSE FROM A THIRD PARTY UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF THAT THIRD PARTY, OR A LICENSE FROM ETA COMPUTE UNDER THE PATENTS OR OTHER INTELLECTUAL PROPERTY OF ETA COMPUTE.

INFORMATION IN THIS DOCUMENT IS PROVIDED SOLELY TO ENABLE SYSTEM AND SOFTWARE IMPLEMENTERS TO USE ETA COMPUTE PRODUCTS. THERE ARE NO EXPRESS OR IMPLIED COPYRIGHT LICENSES GRANTED HEREUNDER TO DESIGN OR FABRICATE ANY INTEGRATED CIRCUITS OR INTEGRATED CIRCUITS BASED ON THE INFORMATION IN THIS DOCUMENT. ETA COMPUTE RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN. ETA COMPUTE MAKES NO WARRANTY, REPRESENTATION OR GUARANTEE REGARDING THE SUITABILITY OF ITS PRODUCTS FOR ANY PARTICULAR PURPOSE, NOR DOES ETA COMPUTE ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT, AND SPECIFICALLY DISCLAIMS ANY AND ALL LIABILITY, INCLUDING WITHOUT LIMITATION CONSEQUENTIAL OR INCIDENTAL DAMAGES. "TYPICAL" PARAMETERS WHICH MAY BE PROVIDED IN ETA COMPUTE DATA SHEETS AND/OR SPECIFICATIONS CAN AND DO VARY IN DIFFERENT APPLICATIONS AND ACTUAL PERFORMANCE MAY VARY OVER TIME. ALL OPERATING PARAMETERS, INCLUDING "TYPICALS" MUST BE VALIDATED FOR EACH CUSTOMER APPLICATION BY CUSTOMER'S TECHNICAL EXPERTS. ETA COMPUTE DOES NOT CONVEY ANY LICENSE UNDER NEITHER ITS PATENT RIGHTS NOR THE RIGHTS OF OTHERS. ETA COMPUTE PRODUCTS ARE NOT DESIGNED, INTENDED, OR AUTHORIZED FOR USE AS COMPONENTS IN SYSTEMS INTENDED FOR SURGICAL IMPLANT INTO THE BODY, OR OTHER APPLICATIONS INTENDED TO SUPPORT OR SUSTAIN LIFE, OR FOR ANY OTHER APPLICATION IN WHICH THE FAILURE OF THE ETA COMPUTE PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR. SHOULD BUYER PURCHASE OR USE ETA COMPUTE PRODUCTS FOR ANY SUCH UNINTENDED OR UNAUTHORIZED APPLICATION, BUYER SHALL INDEMNIFY AND HOLD ETA COMPUTE AND ITS OFFICERS, EMPLOYEES, SUBSIDIARIES, AFFILIATES, AND DISTRIBUTORS HARMLESS AGAINST ALL CLAIMS, COSTS, DAMAGES, AND EXPENSES, AND REASONABLE ATTORNEY FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PERSONAL INJURY OR DEATH ASSOCIATED WITH SUCH UNINTENDED OR UNAUTHORIZED USE, EVEN IF SUCH CLAIM ALLEGES THAT ETA COMPUTE WAS NEGLIGENT REGARDING THE DESIGN OR MANUFACTURE OF THE PART.