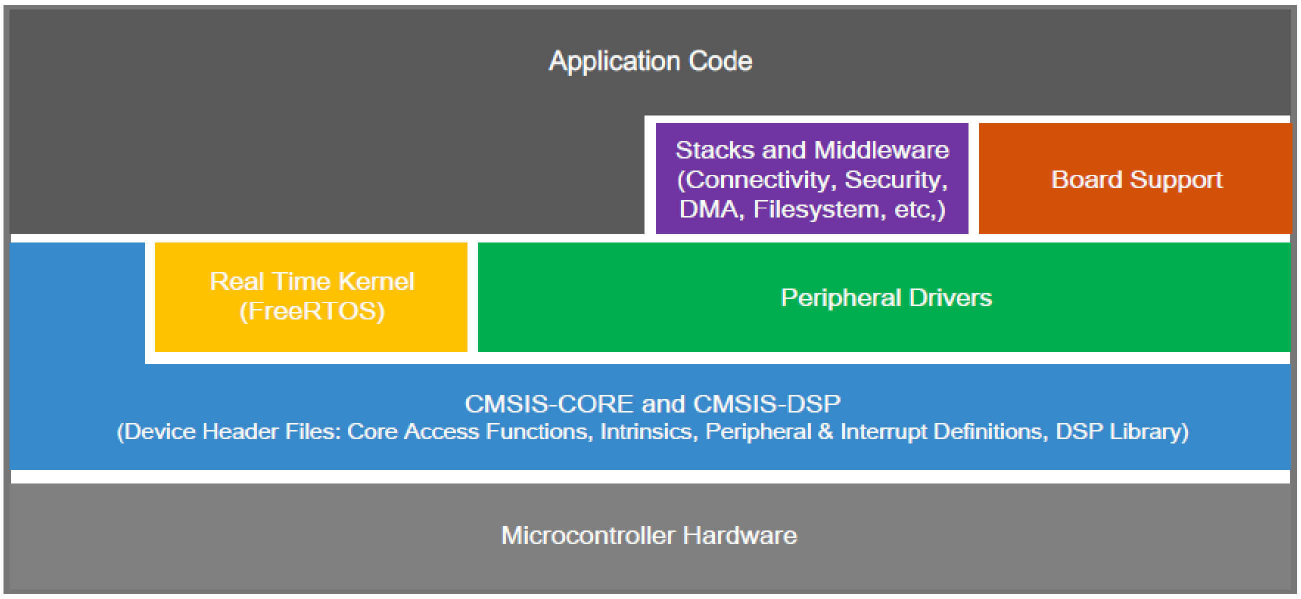
Getting Started with RV32M1 SDK (ARM)

1 Overview

RV32M1 has one ARM Cortex M0+ core, one ARM Cortex M4F core, one RISC-V RI5CY core and one RISC-V ZERORISCY core. By default, all the cores run in normal run mode (RUN mode), in this mode, the max core clock speed is 48MHz. Meanwhile, the device could be configured to high speed run mode (HSRUN mode) or very low power run mode (VLPR mode) for different use cases. The max core clock speed is 72MHz in HSRUN mode, and 8MHz in VLPR mode.

The RV32M1 Software Development Kit (SDK) provides comprehensive software support for RV32M1. The RV32M1 SDK includes a flexible set of peripheral drivers designed to speed up and simplify development of embedded applications. Along with the peripheral drivers, the RV32M1 SDK provides an extensive and rich set of example applications covering everything from basic peripheral use case examples to full demo applications. The RV32M1 SDK contains FreeRTOS and various other middleware to support rapid development.

NOTE: if you would like to use BLE function and need BLE middleware, please refer to Appendix A for more details.


Application Code
Stacks and Middleware (Connectivity, Security, DMA, Filesystem, etc,)
Board Support
Real Time Kernel (FreeRTOS)
Peripheral Drivers
CMSIS-CORE and CMSIS-DSP
(Device Header Files: Core Access Functions, Intrinsics, Peripheral & Interrupt Definitions, DSP Library)
Microcontroller Hardware

2 RV32M1 SDK Board Support Folders

RV32M1 SDK board support provides example applications for RV32M1-VEGA board. Board support packages are found inside of the top level boards folder, and each supported board has its own folder (an RV32M1 SDK package can support multiple boards). Within each <board_name> folder, there are various sub-folders to classify the type of examples they contain. These include (but are not limited to):

- demo_apps: Full-featured applications intended to highlight key functionality and use cases of the target MCU. These applications typically use multiple MCU peripherals and may leverage stacks and middleware.

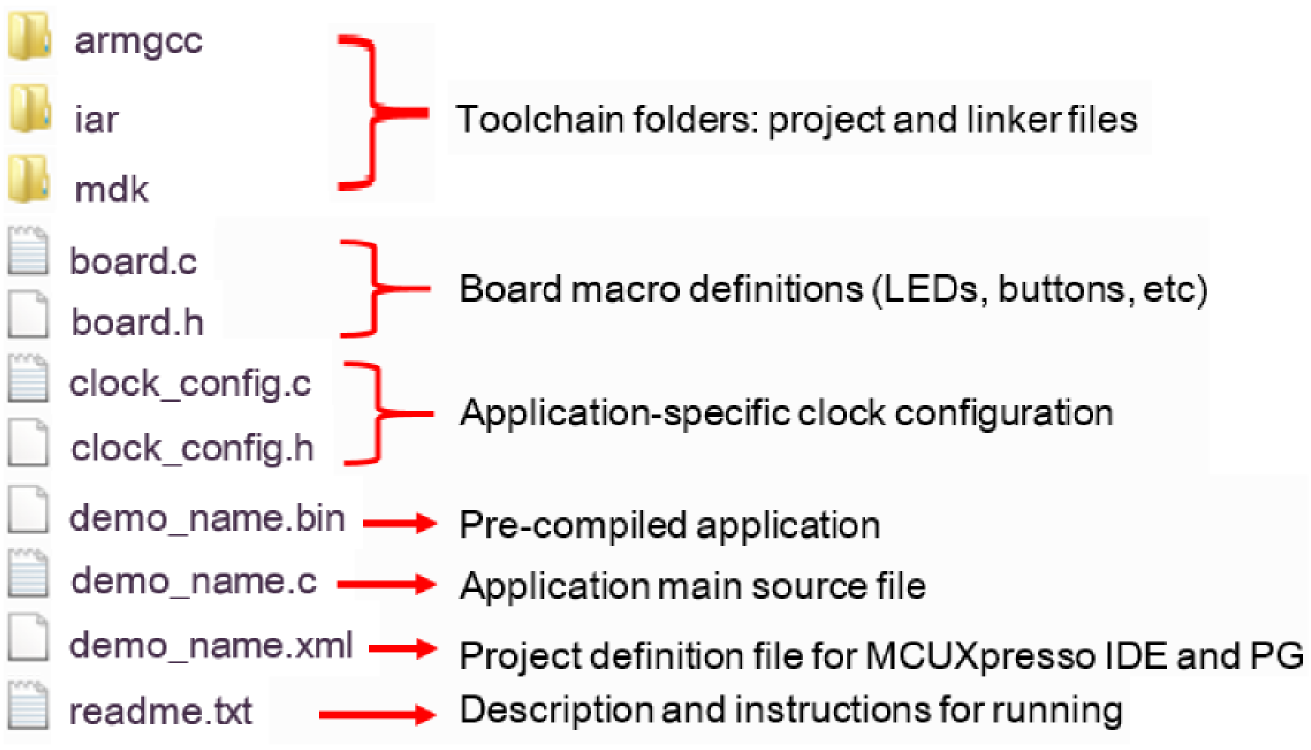
- driver_examples: Simple applications intended to concisely illustrate how to use the RV32M1 SDK's peripheral drivers for a single use case. These applications typically only use a single peripheral, but there are cases where multiple are used (for example, SPI conversion using DMA).
- rtos_examples: Basic FreeRTOS^{TM} OS examples showcasing the use of various RTOS objects (semaphores, queues, and so on) and interfacing with the RV32M1 SDK's RTOS drivers

2.1 Example Application Structure

This section describes how the various types of example applications interact with the other components in the RV32M1 SDK.

Each <board_name> folder in the boards directory contains a comprehensive set of examples that are relevant to that specific piece of hardware. We'll discuss the hello_world example (part of the demo_apps folder), but the same general rules apply to any type of example in the <board_name> folder.

In the hello_world application folder you see this:


armgcc
iar
mdk
Toolchain folders: project and linker files
board.c
board.h
Board macro definitions (LEDs, buttons, etc)
clock_config.c
clock_config.h
Application-specific clock configuration
demo_name.bin
Pre-compiled application
demo_name.c
Application main source file
demo_name.xml
Project definition file for MCUXpresso IDE and PG
readme.txt
Description and instructions for running

All files in the application folder are specific to that example, so it is easy to copy and paste an existing example to start developing a custom application based on a project provided in the RV32M1 SDK.

2.2 Locating Example Application Source Files

When opening an example application in any of the supported IDE, there are a variety of source files referenced. The RV32M1 SDK devices folder is the central component to all example applications. It means the examples reference the same source files and, if one of these files is modified, it could potentially impact the behavior of other examples.

The main areas of the RV32M1 SDK tree used in all example applications are:

- devices/<device_name>: The device's CMSIS header file, RV32M1 SDK feature file and a few other things.
- devices/<device_name>/drivers: All of the peripheral drivers for your specific MCU.

- devices/<device_name>/<tool_name>: Toolchain-specific startup code. Vector table definitions are here.
- devices/<device_name>/utilities: Items such as the debug console that are used by many of the example applications.

For examples containing an RTOS, there are references to the appropriate source code. RTOSes are in the *rtos* folder. Again, the core files of each of these are shared, so modifying them could have potential impacts on other projects that depend on them.

3 Run a demo using IAR

This section describes the steps required to build, run, and debug example applications provided in the RV32M1 SDK. The hello_world demo application targeted for the RV32M1-VEGA hardware platform is used as an example, although these steps can be applied to any example application in the RV32M1 SDK.

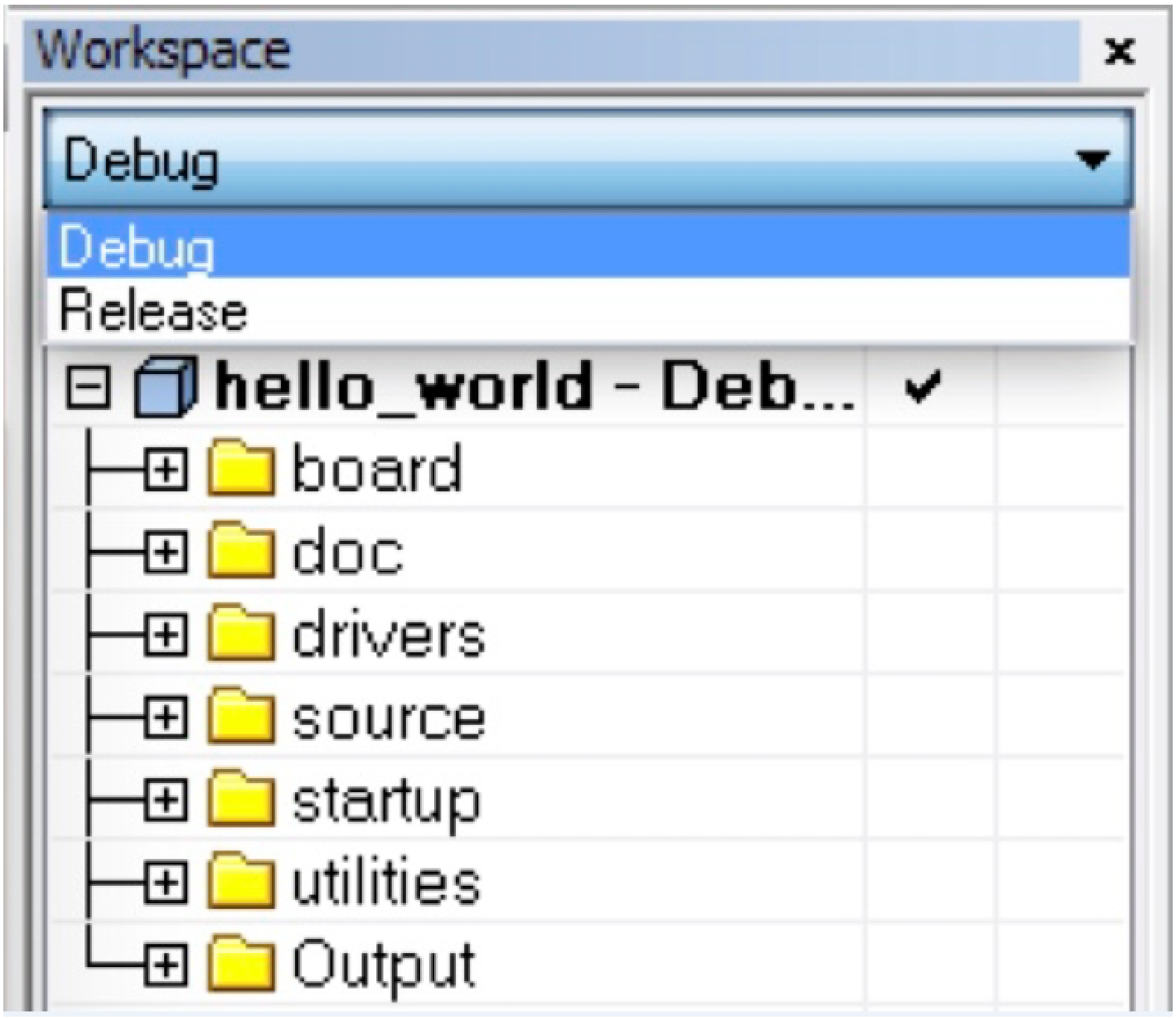
First download the IAR patch for RV32M1 from open-isa.org, and copy the file to *<iar_install_dir>*.

If downloading M0+ core project, let the MCU boot from the M0+ core. For detailed steps, see the hello_world demo readme.txt which is in folder *<install_dir>/boards/rv32m1_vega/demo_apps/hello_world/cm0plus*.

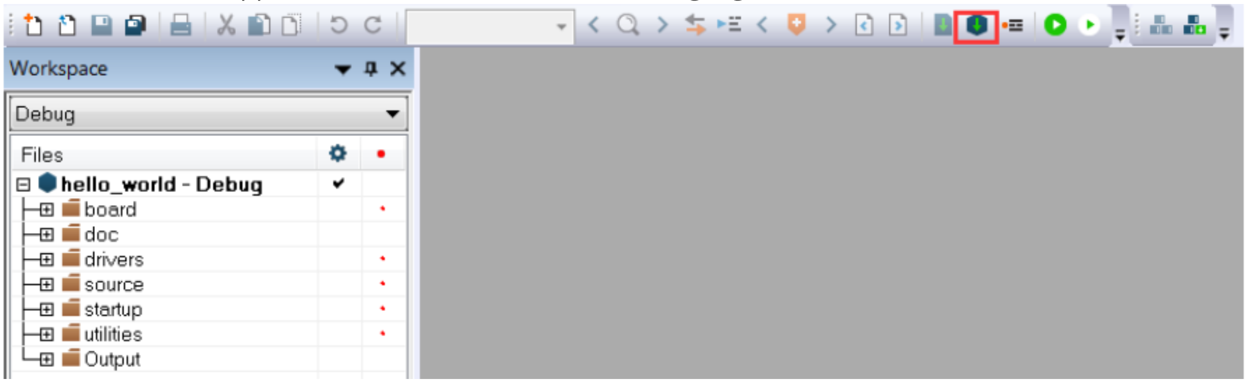
3.1 Build an example application

The following steps guide you through opening the hello_world example application. These steps may change slightly for other example applications as some of these applications may have additional layers of folders in their path.

1. If not already done, open the desired demo application workspace. Most example application workspace files can be located using the following path: *<install_dir>/boards/<board_name>/<example_type>/<application_name>/<core_type>/iar* Using the RV32M1-VEGA hardware platform as an example, the hello_world for CM4 core workspace is located in *<install_dir>/boards/rv32m1_vega/demo_apps/hello_world/cm4/iar/hello_world.eww*
2. Select the desired build target from the drop-down. For this example, select the "hello_world - Debug" target.



3. To build the demo application, click the "Make" button, highlighted in red below.



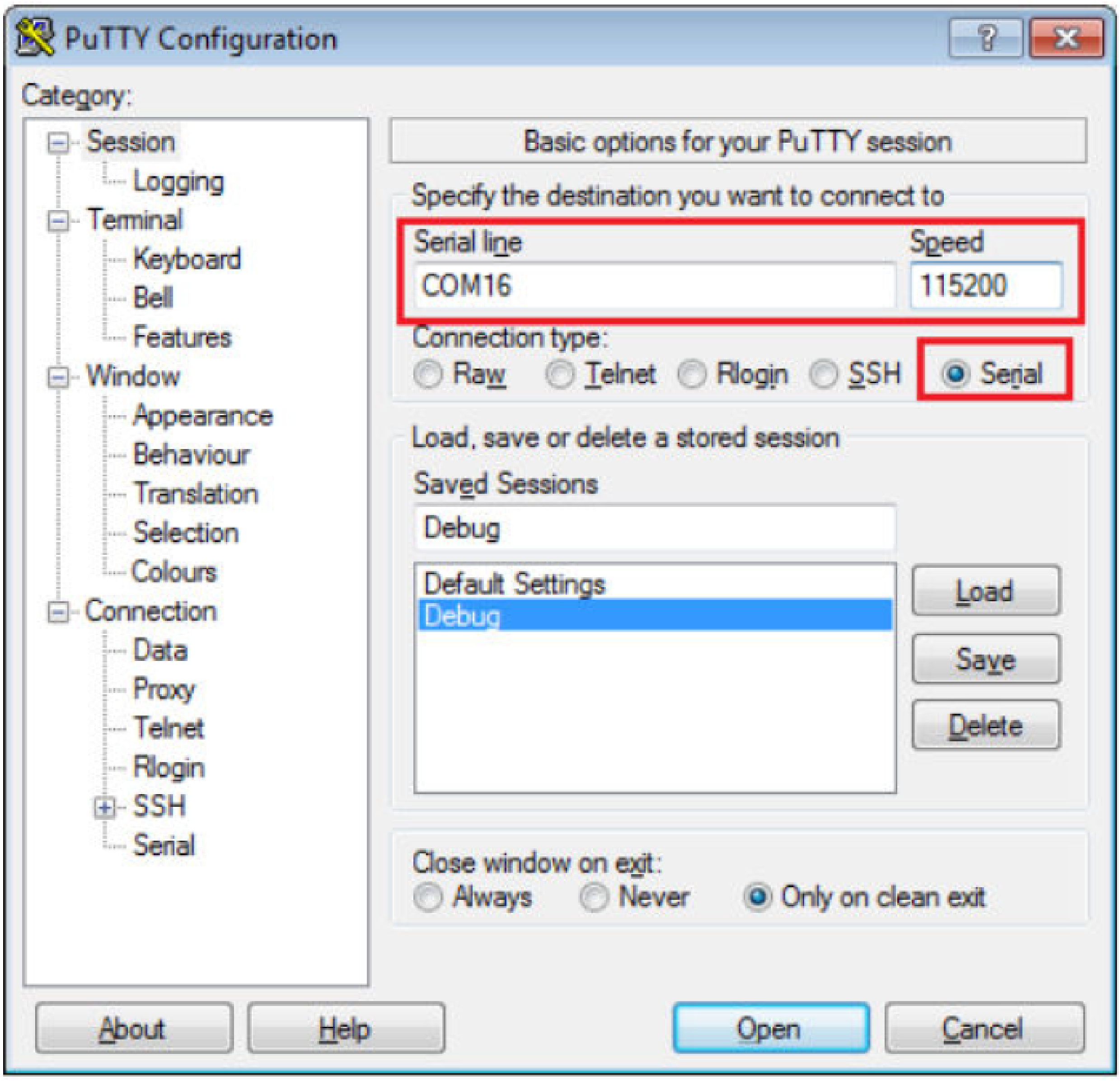
4. The build completes without errors.

3.2 Run an example application

To download and run the application, perform these steps:

1. Connect the development platform J12 to your PC via USB cable.
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port. Configure the terminal with these settings: a. 115200 baud rate b. No parity c. 8 data bits d. 1 stop

bit


PuTTY Configuration

Category:

- Session
  - Logging
- Terminal
  - Keyboard
  - Bell
  - Features
- Window
  - Appearance
  - Behaviour
  - Translation
  - Selection
  - Colours
- Connection
  - Data
  - Proxy
  - Telnet
  - Rlogin
  - SSH
  - Serial

Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM16

Speed: 115200

Connection type: Raw Telnet Rlogin SSH [✓ Serial]

Load, save or delete a stored session

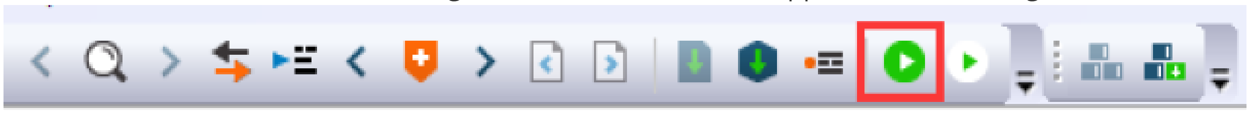
Saved Sessions: Debug

Default Settings

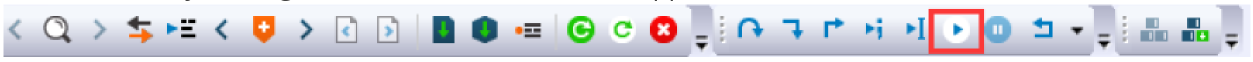
Debug

Close window on exit: Always Never [✓ Only on clean exit]

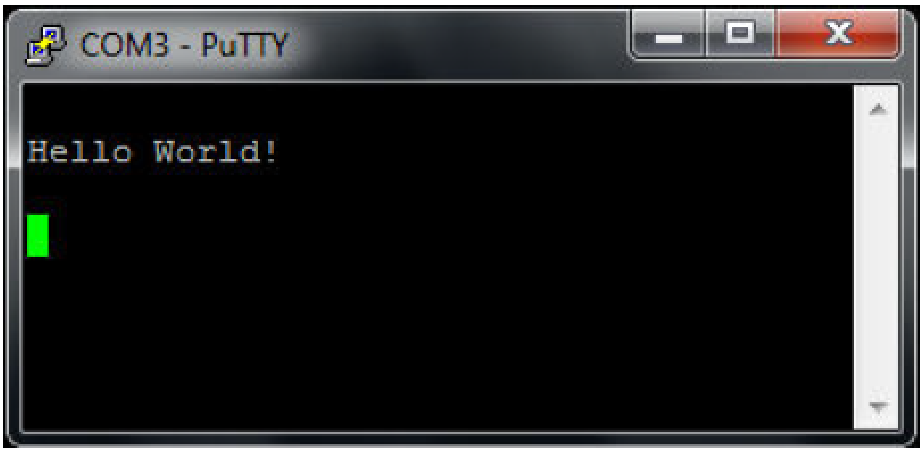
3. In IAR, click the "Download and Debug" button to download the application to the target.



4. The application is then downloaded to the target and automatically runs to the main() function.
5. Run the code by clicking the "Go" button to start the application.



6. The hello_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections



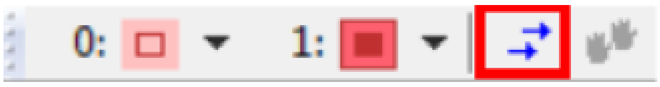
```
Hello World!
```

3.3 Build a multicore example application

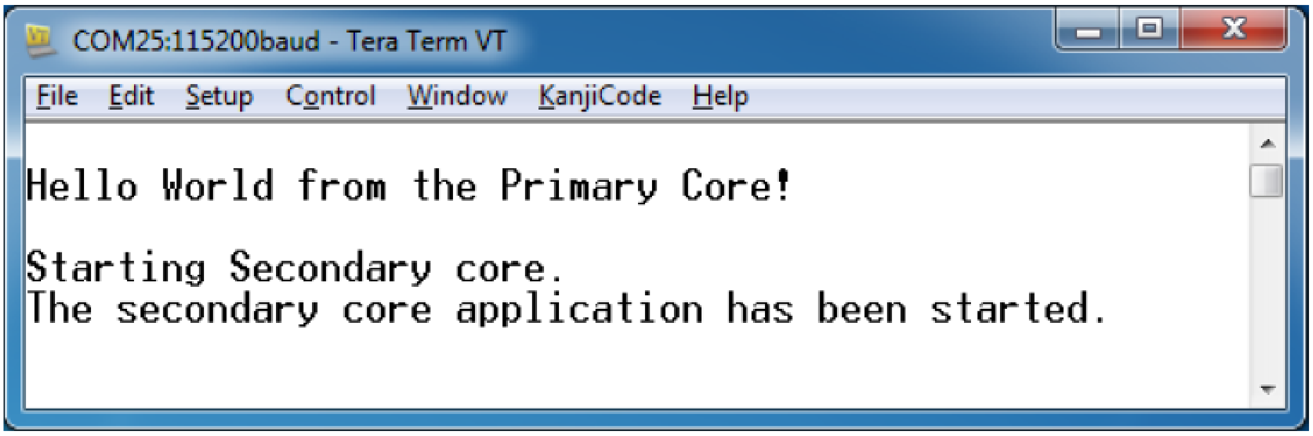
This section describes the particular steps that need to be done in order to build and run a dual-core application. The demo applications workspace files are located in this folder: *<install_dir>/boards/<board_name>/multicore_examples/<application_name>/<core_type>/iar* Begin with a simple dual-core version of the Hello World application. The multicore Hello World IAR workspaces are located in this folder: *<install_dir>/boards/rv32m1_vega/multicore_examples/hello_world/cm0plus/iar/hello_world_cm0plus.eww* *<install_dir>/boards/rv32m1_vega/multicore_examples/hello_world/cm4/iar/hello_world_cm4.eww* Build both applications separately by clicking the "Make" button. It is requested to build the application for the auxiliary core (cm0plus) first, because the primary core application project (cm4) needs to know the auxiliary core application binary when running the linker. It is not possible to finish the primary core linker when the auxiliary core application binary is not ready.

3.4 Run a multicore example application

The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 - 4 as described in Section 3.2, "Run an example application". These steps are common for both single core and dual-core applications in IAR. After clicking the "Download and Debug" button, the auxiliary core project is opened in the separate EWARM instance. Both the primary and auxiliary image are loaded into the device flash memory, and the primary core application is executed. It stops at the default C language entry point in the main() function. Run both cores by clicking the "Start all cores" button to start the multicore application.

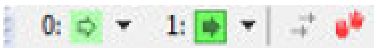


During the primary core code execution, the auxiliary core is released from the reset. The hello_world multicore application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.


COM25:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help

```
Hello World from the Primary Core!

Starting Secondary core.
The secondary core application has been started.
```

An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and is running correctly. When both cores are running, use the "Stop all cores" and "Start all cores" control buttons to stop or run both cores simultaneously.



4 Run a demo using Keil® MDK/μVision

This section describes the steps required to build, run, and debug example applications provided in the RV32M1 SDK. The hello_world demo application targeted for the RV32M1-VEGA hardware platform is used as an example, although these steps can be applied to any demo or example application in the RV32M1 SDK. If downloading the M0+ core project, let the MCU boot from the M0+ core. For detailed steps, see the hello_world demo readme.txt which is in folder <install_dir>/boards/rv32m1_vega/demo_apps/hello_world/cm0plus.

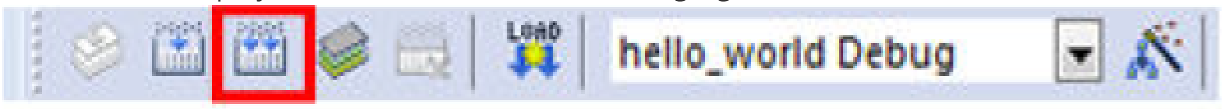
4.1 Install CMSIS device pack

After the MDK tools are installed, Cortex® Microcontroller Software Interfac Standard (CMSIS) device packs must be installed to fully support the device from a debug perspective. These packs include things such as memory map information, register definitions and flash programming algorithms. Follow these steps to install the RV32M1 CMSIS pack.

1. Download the pack from open-isa.org.
2. After downloading the DFP, double click to install it.

4.2 Build an example application

1. Open the desired example application workspace in: <install_dir>/boards/<board_name>/<example_type> /<application_name>/<core_type>/mdk The workspace file is named <demo_name>.uvmpw, so for this specific example, the actual path is: <install_dir>/boards/rv32m1_vega/demo_apps/hello_world/cm4/mdk/hello_world.uvmpw
2. To build the demo project, select the "Rebuild" button, highlighted in red.

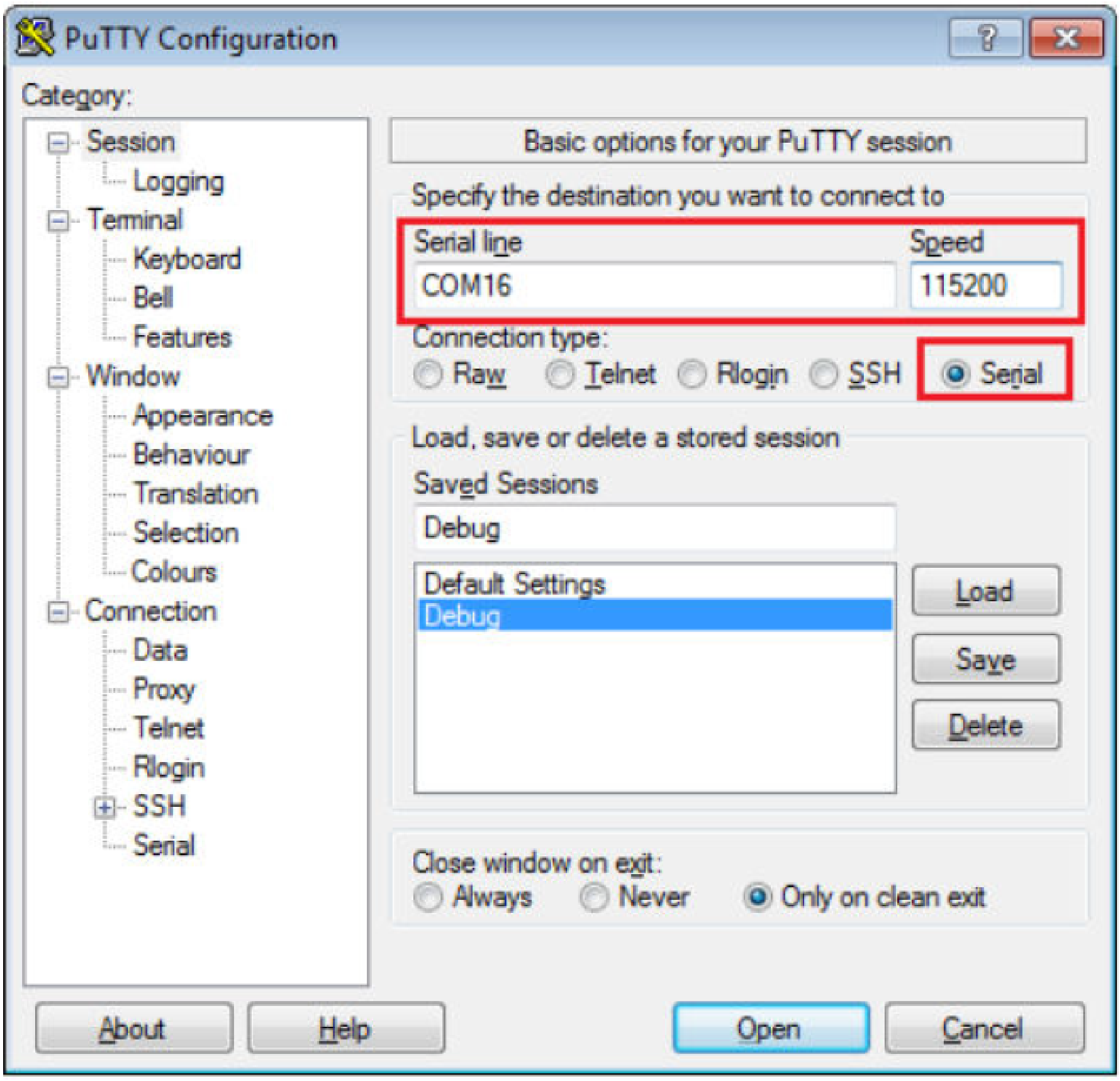


3. The build completes without errors.

4.2 Run an example application

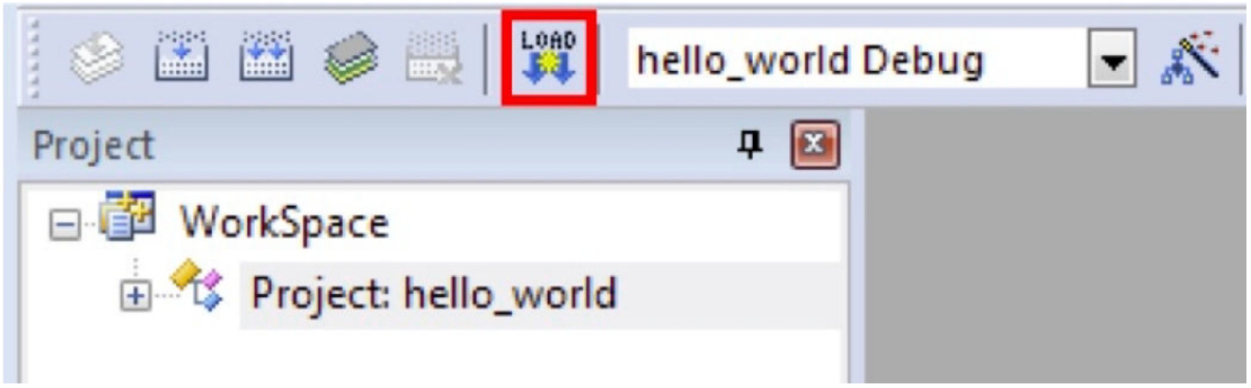
To download and run the application, perform these steps:

1. Connect the development platform J12 to your PC via USB cable.
2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port. Configure the terminal with these settings: a. 115200 baud rate b. No parity c. 8 data bits d. 1 stop bit

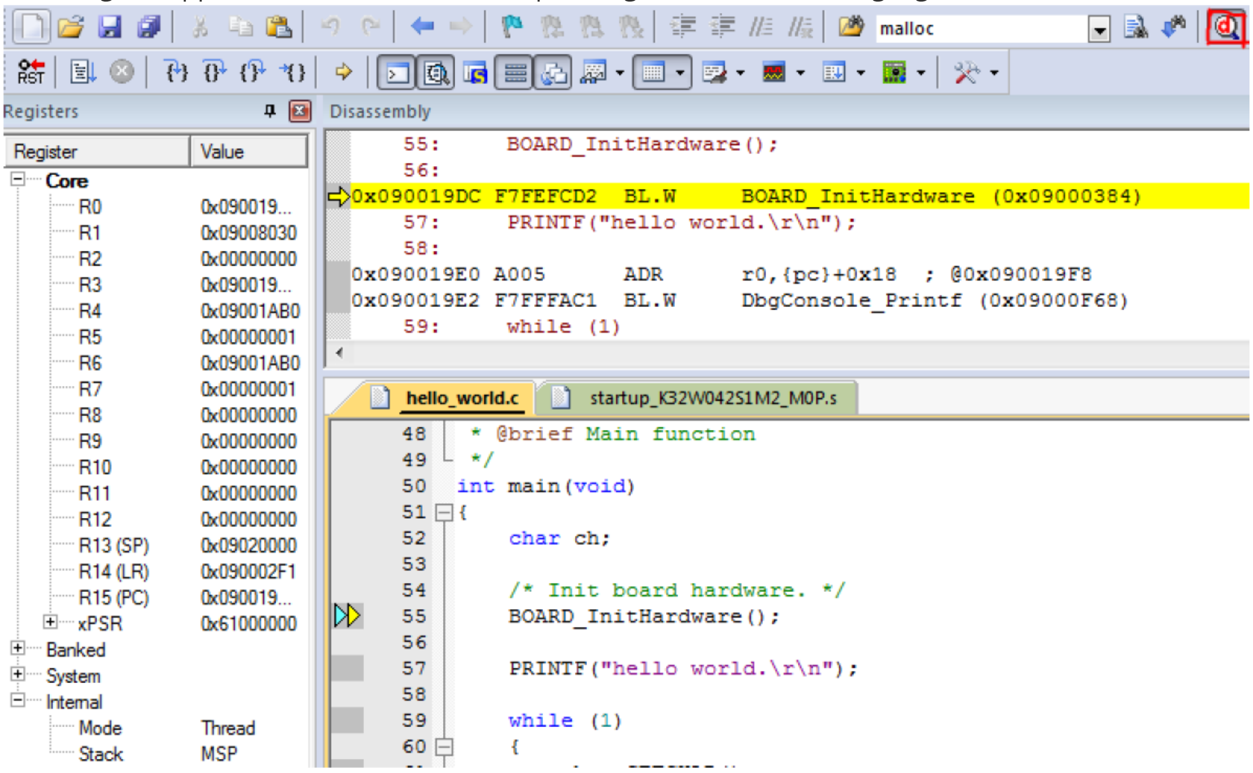


Serial line: COM16
Speed: 115200
Connection type: Raw Telnet Rlogin SSH [✓ Serial]
Saved Sessions: Debug
Close window on exit: Always Never [✓ Only on clean exit]

3. In µVision, after the application is properly built, click the "Download" button to download the application to the target.



4. To debug the application, click the "Start/Stop Debug Session" button, highlighted in red.

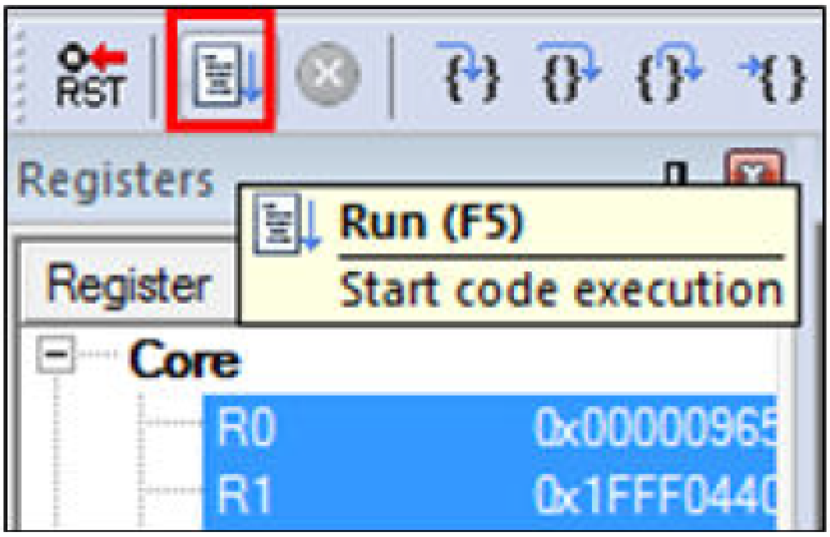


Register	Value
Core	
R0	0x090019...
R1	0x09008030
R2	0x00000000
R3	0x090019...
R4	0x09001AB0
R5	0x00000001
R6	0x09001AB0
R7	0x00000001
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x09020000
R14 (LR)	0x090002F1
R15 (PC)	0x090019...
xPSR	0x61000000
Banked	
System	
Internal	
Mode	Thread
Stack	MSP

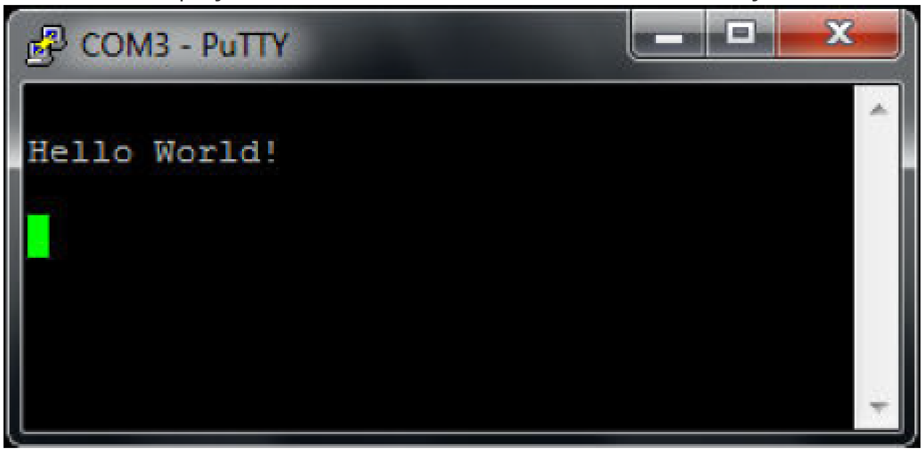
```
    55:     BOARD_InitHardware();
    56:
0x090019DC F7FEFCD2  BL.W     BOARD_InitHardware (0x09000384)
    57:     PRINTF("hello world.\r\n");
    58:
0x090019E0 A005      ADR      r0,{pc}+0x18  ; @0x090019F8
0x090019E2 F7FFFAC1  BL.W     DbgConsole_Printf (0x09000F68)
    59:     while (1)
```

```
 * @brief Main function
 */
int main(void)
{
    char ch;

    /* Init board hardware. */
    BOARD_InitHardware();

    PRINTF("hello world.\r\n");

    while (1)
    {
```

5. Run the code by clicking the "Run" button to start the application.



The hello_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.



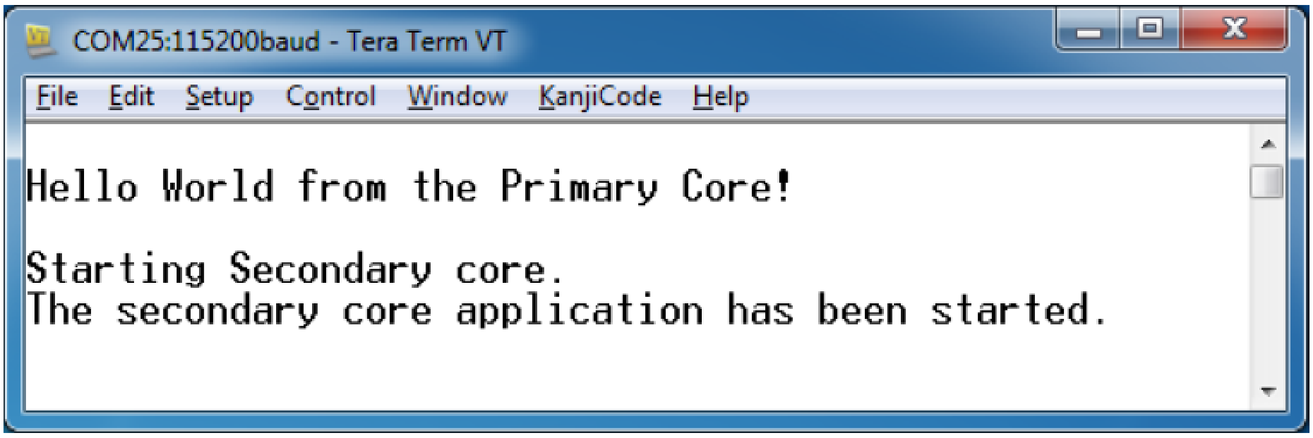
```
Hello World!
```

4.4 Build a multicore example application

This section describes the particular steps that need to be done in order to build and run a dual-core application. The demo applications workspace files are located in this folder: *<install_dir>/boards/<board_name>/multicore_examples/<application_name>/<core_type>/mdk* Begin with a simple dual-core version of the Hello World application. The multicore Hello World Keil MSDK/µVision® workspaces are located in this folder: *<install_dir>/boards/rv32m1_vega/multicore_examples/hello_world/cm0plus/mdk/hello_world_cm0plus.uvmpw* *<install_dir>/boards/rv32m1_vega/multicore_examples/hello_world/cm4/mdk/hello_world_cm4.uvmpw* Build both applications separately by clicking the "Rebuild" button. Build the application for the auxiliary core (cm0plus) first, because the primary core application project (cm4) needs to know the auxiliary core application binary when running the linker. It is not possible to finish the primary core linker when the auxiliary core application binary is not ready.

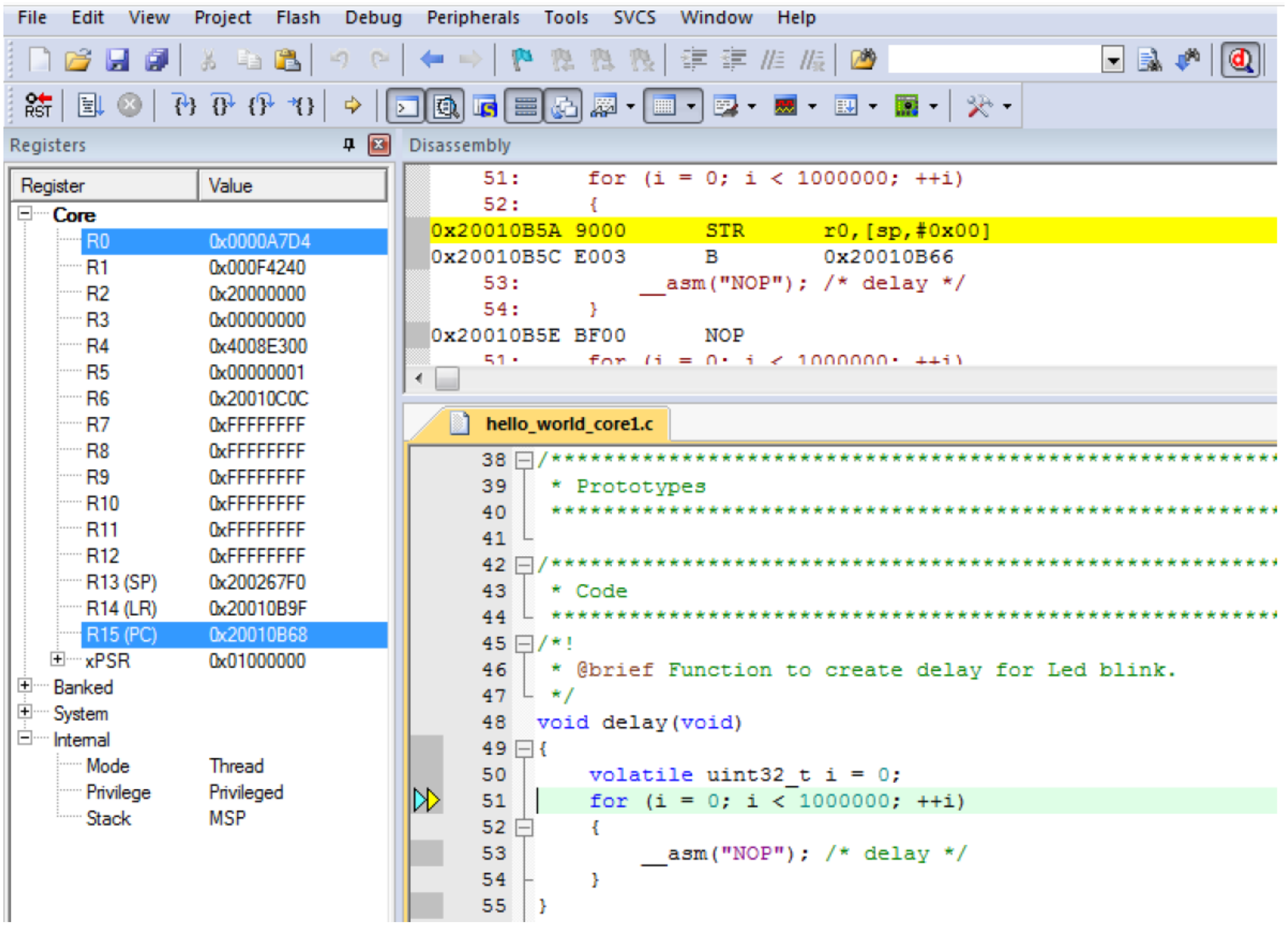
4.5 Run a multicore example application

The primary core debugger handles flashing of both the primary and the auxiliary core applications into the SoC flash memory. To download and run the multicore application, switch to the primary core application project and perform steps 1 – 5 as described in Section 4.3, "Run an example application". These steps are common for both single core and dual-core applications in µVision. Both the primary and the auxiliary image is loaded into the device flash memory. After clicking the "Run" button, the primary core application is executed. During the primary core code execution, the auxiliary core is released from the reset. The hello_world multicore application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.


COM25:115200baud - Tera Term VT
File Edit Setup Control Window KanjiCode Help

```
Hello World from the Primary Core!

Starting Secondary core.
The secondary core application has been started.
```

An LED controlled by the auxiliary core starts flashing, indicating that the auxiliary core has been released from the reset and is running correctly. Attach the running application of the auxiliary core by opening the auxiliary core project in the second µVision instance, and clicking the "Start/Stop Debug Session" button. After doing this, the second debug session is opened and the auxiliary core application can be debugged.



Register	Value
Core	
R0	0x0000A7D4
R1	0x000F4240
R2	0x20000000
R3	0x00000000
R4	0x4008E300
R5	0x00000001
R6	0x20010C0C
R7	0xFFFFFFFF
R8	0xFFFFFFFF
R9	0xFFFFFFFF
R10	0xFFFFFFFF
R11	0xFFFFFFFF
R12	0xFFFFFFFF
R13 (SP)	0x200267F0
R14 (LR)	0x20010B9F
R15 (PC)	0x20010B68
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP

```
    51:     for (i = 0; i < 1000000; ++i)
    52:     {
0x20010B5A 9000      STR      r0,[sp,#0x00]
0x20010B5C E003      B        0x20010B66
    53:         __asm("NOP"); /* delay */
    54:     }
0x20010B5E BF00      NOP
```

```
/*******************************************************************
 * Prototypes
 ******************************************************************/

/*******************************************************************
 * Code
 ******************************************************************/
/*!
 * @brief Function to create delay for Led blink.
 */
void delay(void)
{
    volatile uint32_t i = 0;
    for (i = 0; i < 1000000; ++i)
    {
        __asm("NOP"); /* delay */
    }
}
```

5 Run a demo using Arm® GCC

This section describes the steps to configure the command line Arm® GCC tools to build, run, and debug demo applications and necessary driver libraries provided in the RV32M1 SDK. The hello_world demo application targeted for the RV32M1-VEGA hardware platform is used as an example, though these steps can be applied to any board, demo or example application in the RV32M1 SDK. If downloading M0+ core project, let the MCU boot from the M0+ core. For detailed steps, see the hello_world demo readme.txt which is in folder <install_dir>/boards/rv32m1_vega/demo_apps/hello_world/cm0plus.

5.1 Set up toolchain

This section contains the steps to install the necessary components required to build and run a MCUXpresso SDK demo application with the Arm GCC toolchain, as supported by the MCUXpresso SDK. There are many ways to use Arm GCC tools, but this example focuses on a Windows operating system environment.

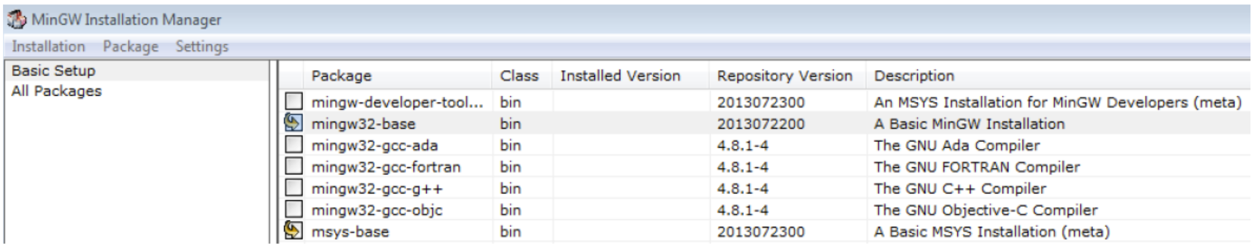
5.1.1 Install GCC ARM Embedded tool chain

Download and run the installer from launchpad.net/gcc-arm-embedded. This is the actual toolset (in other words, compiler, linker, etc.). The GCC toolchain should correspond to the latest supported version.

5.1.2 Install MinGW (only required on Windows OS)

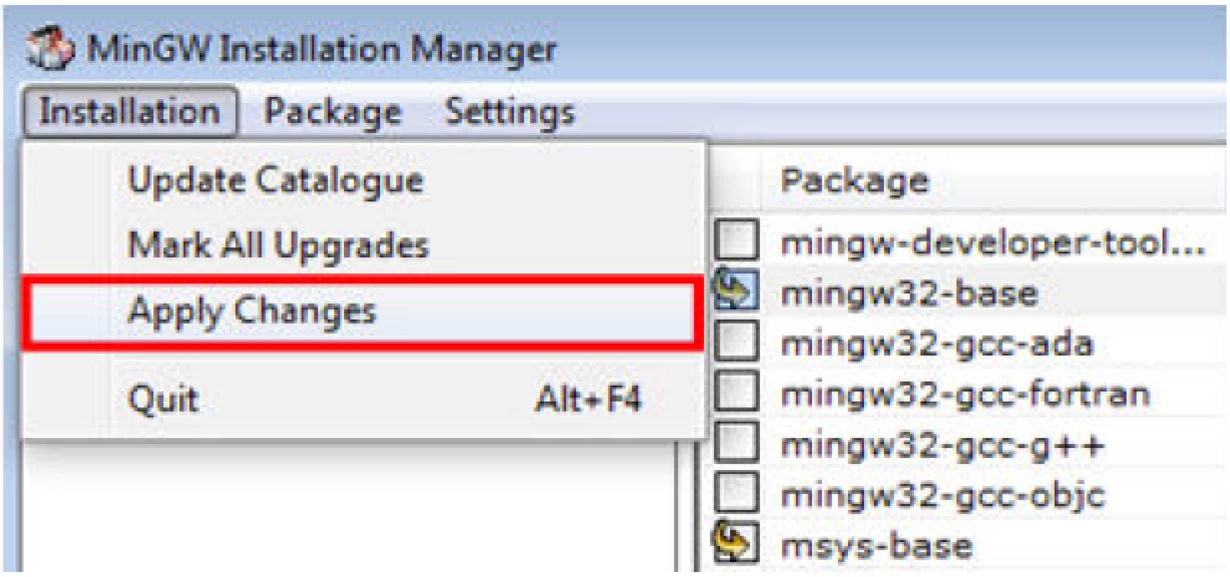
The Minimalist GNU for Windows (MinGW) development tools provide a set of tools that are not dependent on third party C-Runtime DLLs (such as Cygwin). The build environment used by the SDK does not utilize the MinGW build tools, but does leverage the base install of both MinGW and MSYS. MSYS provides a basic shell with a Unix-like interface and tools.

1. Download the latest MinGW mingw-get-setup installer from sourceforge.net/projects/mingw/files/Installer/.
2. Run the installer. The recommended installation path is C:\MinGW, however, you may install to any location. **NOTE** The installation path cannot contain any spaces.
3. Ensure that the "mingw32-base" and "msys-base" are selected under Basic Setup.



	Package	Class	Installed Version	Repository Version	Description
☐	mingw-developer-tool...	bin		2013072300	An MSYS Installation for MinGW Developers (meta)
✓	mingw32-base	bin		2013072200	A Basic MinGW Installation
☐	mingw32-gcc-ada	bin		4.8.1-4	The GNU Ada Compiler
☐	mingw32-gcc-fortran	bin		4.8.1-4	The GNU FORTRAN Compiler
☐	mingw32-gcc-g++	bin		4.8.1-4	The GNU C++ Compiler
☐	mingw32-gcc-objc	bin		4.8.1-4	The GNU Objective-C Compiler
✓	msys-base	bin		2013072300	A Basic MSYS Installation (meta)

4. Click "Apply Changes" in the "Installation" menu and follow the remaining instructions to complete the installation.



5. Add the appropriate item to the Windows operating system path environment variable. It can be found under Control Panel -> System and Security -> System -> Advanced System Settings in the "Environment Variables..." section. The path is: <mingw_install_dir>\bin Assuming the default installation path, C:\MinGW, an example is shown below. If the path is not set correctly, the toolchain does not work. **NOTE** If you have "C:\MinGW\msys\x.x\bin" in your PATH variable, remove it to ensure that the new GCC build system works correctly.

5.1.3 Add a new system environment variable for ARMGCC_DIR

Create a new system environment variable and name it ARMGCC_DIR. The value of this variable should point to the Arm GCC Embedded tool chain installation path. For this example, the path is: C:\Program Files (x86)\GNU Tools ARM Embedded\5.2 2016q3 Reference the installation folder of the GNU ARM GCC Embedded tools for the exact path name of your installation.

Also, add the path C:\Program Files (x86)\GNU Tools ARM Embedded\5.2 2016q3\bin to system enviroment variable PATH.

5.1.4 Install CMake

1. Download CMake 3.0.x from here.
2. Install CMake, ensuring that the option "Add CMake to system PATH" is selected when installing. The user chooses to select whether it is installed into the PATH for all users or just the current user. In this example, it is installed for all users.
3. Follow the remaining instructions of the installer.
4. You may need to reboot your system for the PATH changes to take effect.
5. Make sure "sh.exe" is not in the Environment Variable PATH. This is a limitation of mingw32-make.

5.2 Build an example application

The projects are in the folder <install_dir>/boards/<board_name>/<example_type>/<core_type>/<application_name>/armgcc double click the "*.bat" file to build.

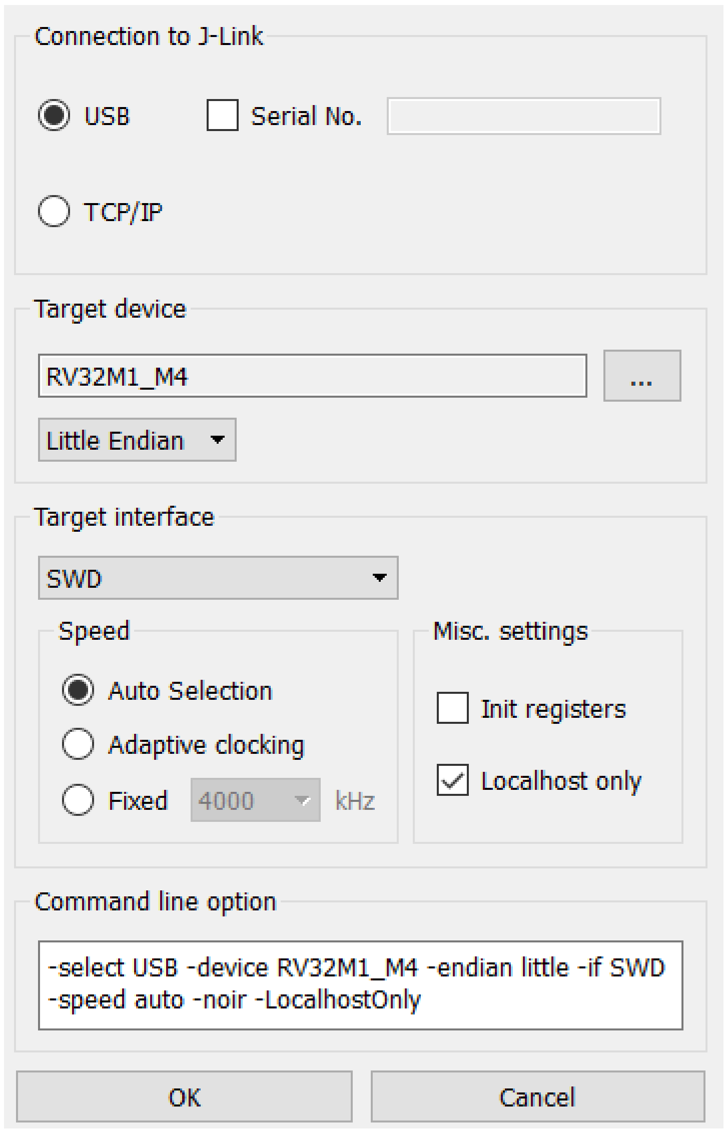
5.3 Run an example application

This section describes the steps to download and run the iamge using Jlink. Before started, please download the Jlink patch for RV32M1 from open-isa.org and copy to the Jlink install folder. For the file JLinkDevices.xml, copy the device section to your original JLinkDevices.xml in Jlink install folder.

1. Connect the development platform J12 to your PC via USB cable, connect Jlink to J7.

2. Open the terminal application on the PC, such as PuTTY or TeraTerm, and connect to the debug COM port. Configure the terminal with these settings: a. 115200 baud rate b. No parity c. 8 data bits d. 1 stop bit

3. 3. Open the J-Link GDB Server application. Assuming the J-Link software is installed, the application can be launched by going to the Windows operating system Start menu and selecting "Programs -> SEGGER -> J-Link <version> J-Link GDB Server".
4. Modify the settings as shown below.


Connection to J-Link

[● USB] Serial No. ☐

○ TCP/IP

Target device: RV32M1_M4

Little Endian

Target interface: SWD

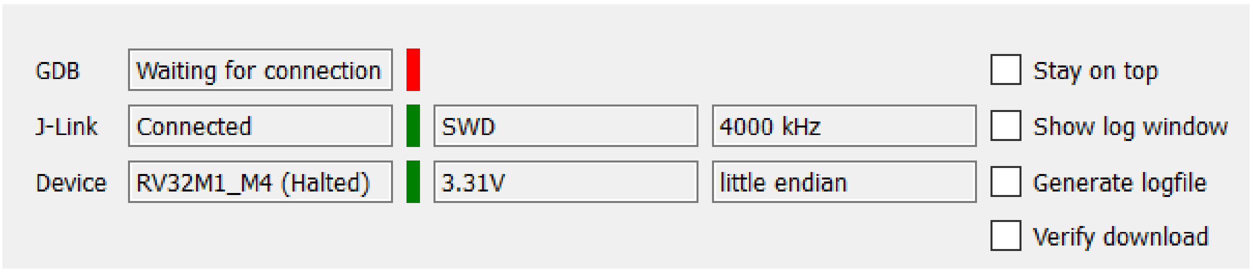
Speed: [● Auto Selection] ○ Adaptive clocking ○ Fixed 4000 kHz

Misc. settings: ☐ Init registers ☑ Localhost only

Command line option

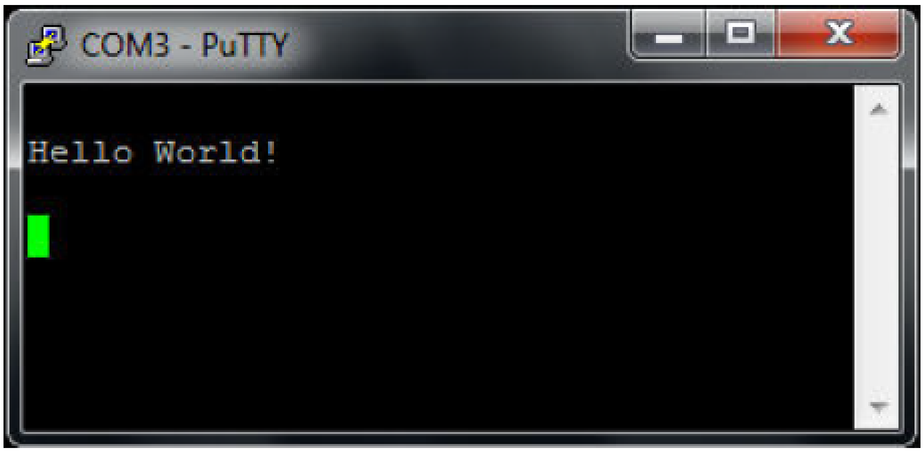
```
-select USB -device RV32M1_M4 -endian little -if SWD
-speed auto -noir -LocalhostOnly
```

5. After it is connected, the screen should resemble this figure:



GDB	Waiting for connection	🟥			☐ Stay on top
J-Link	Connected	🟩	SWD	4000 kHz	☐ Show log window
Device	RV32M1_M4 (Halted)	🟩	3.31V	little endian	☐ Generate logfile
					☐ Verify download

6. Start command prompt and change to the directory <install_dir>/boards/<board_name>/<example_type>/<core_type>/<application_name>/armgcc
7. Run the command `arm-none-eabi-gdb <application_name>.elf`.
8. Run these commands: a. "target remote localhost:2331" b. "monitor reset" c. "monitor halt" d. "load"
9. The application is now downloaded and halted at the reset vector. Execute the "monitor go" command to start the demo application. The hello_world application is now running and a banner is displayed on the terminal. If this is not true, check your terminal settings and connections.



```
Hello World!
```

5.4 Build a multicore example application

Build the cm4 core and cm0plus core binaries refering Section 5.2.

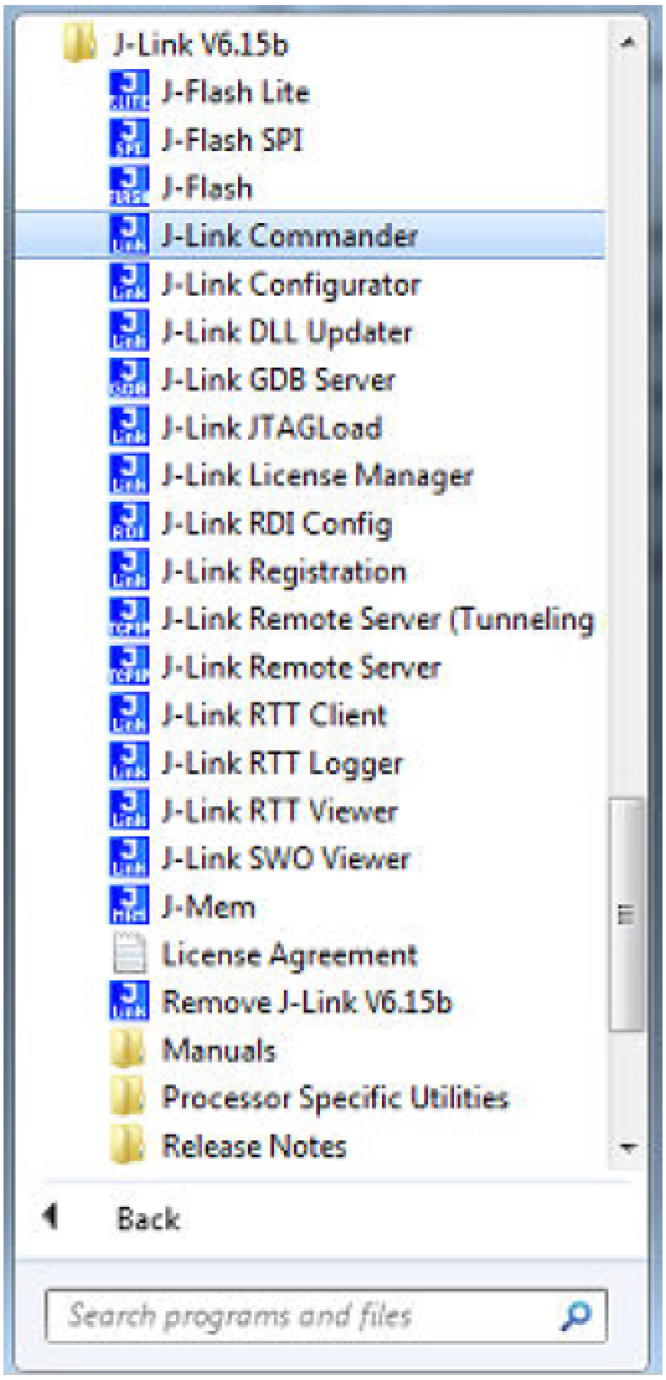
5.5 Run a multicore example application

Flashing of both the primary and the auxiliary core applications into the SoC flash memory needs to be done separately for each core. First, start with the auxiliary core image. Once the application is built, switch to the build output directory and convert the elf format file to the binary format by

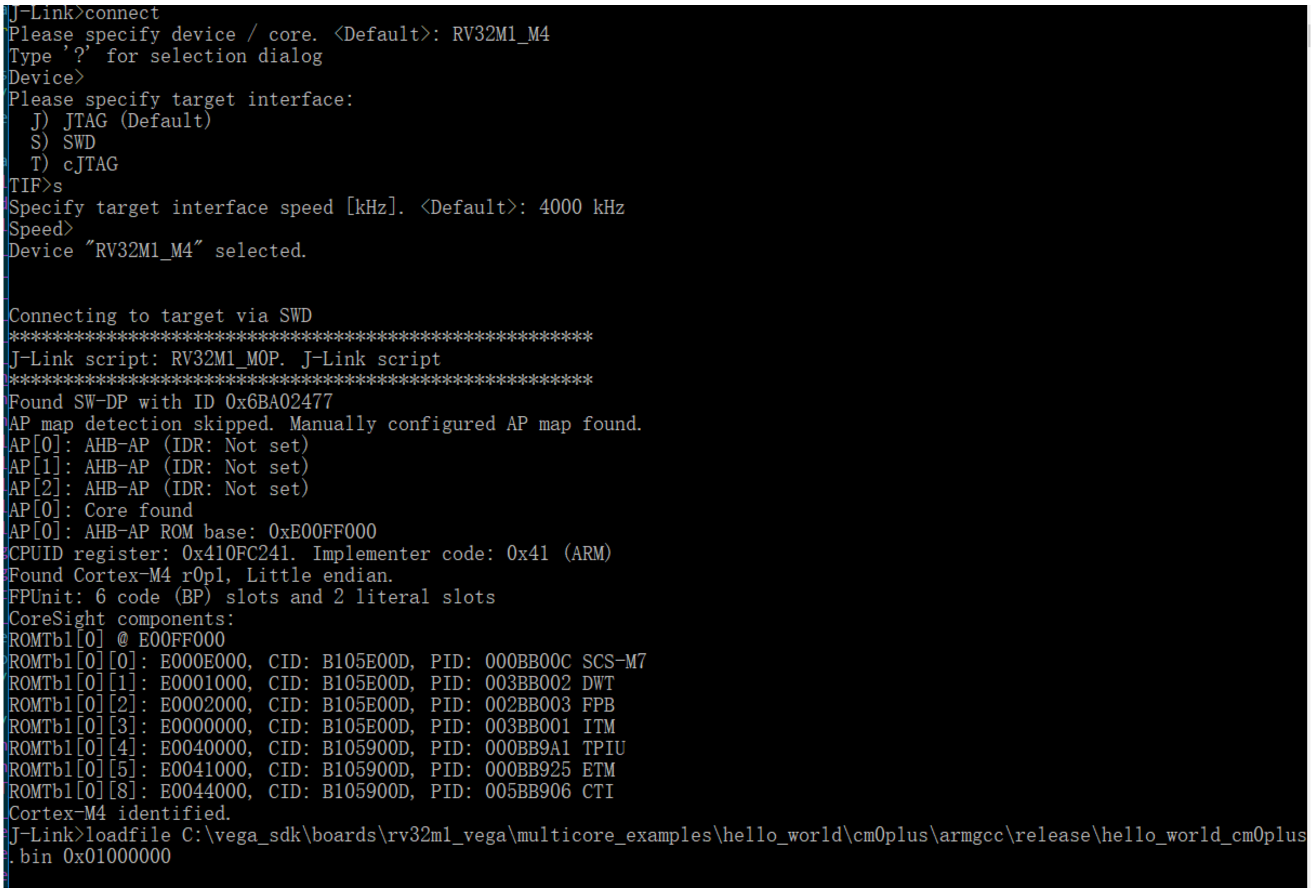
```
arm-none-eabi-objcopy -O binary hello_world_cm0plus.elf hello_world_cm0plus.bin
```

command on the GCC console:

Now, the binary file can be loaded into the CM0+ core flash memory via J-Link commander. Open the J-Link Commander from the Windows operating system 'Start' menu:



Once the J-Link Commander is opened, connect to the target device using the connect command and specifying the RV32M1_M4 as the device name, SWD as the target interface, and the default speed. Then, apply the following command to load the auxiliary core application image into the flash memory:



```
J-Link>connect
Please specify device / core. <Default>: RV32M1_M4
Type '?' for selection dialog
Device>
Please specify target interface:
  J) JTAG (Default)
  S) SWD
  T) cJTAG
TIF>s
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>
Device "RV32M1_M4" selected.


Connecting to target via SWD
******************************************************
J-Link script: RV32M1_M0P. J-Link script
******************************************************
Found SW-DP with ID 0x6BA02477
AP map detection skipped. Manually configured AP map found.
AP[0]: AHB-AP (IDR: Not set)
AP[1]: AHB-AP (IDR: Not set)
AP[2]: AHB-AP (IDR: Not set)
AP[0]: Core found
AP[0]: AHB-AP ROM base: 0xE00FF000
CPUID register: 0x410FC241. Implementer code: 0x41 (ARM)
Found Cortex-M4 r0p1, Little endian.
FPUnit: 6 code (BP) slots and 2 literal slots
CoreSight components:
ROMTbl[0] @ E00FF000
ROMTbl[0][0]: E000E000, CID: B105E00D, PID: 000BB00C SCS-M7
ROMTbl[0][1]: E0001000, CID: B105E00D, PID: 003BB002 DWT
ROMTbl[0][2]: E0002000, CID: B105E00D, PID: 002BB003 FPB
ROMTbl[0][3]: E0000000, CID: B105E00D, PID: 003BB001 ITM
ROMTbl[0][4]: E0040000, CID: B105900D, PID: 000BB9A1 TPIU
ROMTbl[0][5]: E0041000, CID: B105900D, PID: 000BB925 ETM
ROMTbl[0][8]: E0044000, CID: B105900D, PID: 005BB906 CTI
Cortex-M4 identified.
J-Link>loadfile C:\vega_sdk\boards\rv32m1_vega\multicore_examples\hello_world\cm0plus\armgcc\release\hello_world_cm0plus
.bin 0x01000000
```

Then download and run the M4 image refer to the Section 5.3.

6 Appendix A - BLE stack

The BLE controller library is not included in this SDK package. If you would like to use BLE function, please send an email to support@open-isa.org to ask for this BLE controller library. Otherwise all the BLE examples in this SDK package cannot work.