Marvell
Wireless Microcontroller
Getting Started Guide for MW320

V1.7

April 30, 2019

CONFIDENTIAL

Document Classification: Proprietary Information

Marvell.Moving Forward Faster

Document Conventions

	Note: Provides related information or information of special importance.
Caution	**Caution:** Indicates potential damage to hardware or software, or loss of data.
	Warning: Indicates a risk of personal injury.

For more information, visit our website at: www.marvell.com

Date	Author	Revision	Changelog
April 30, 2019	Mayank Sirotiya	V1.7	Implemented Review Comments
April 25, 2019	Abhishek Misra	V1.6	Implemented Review Comments
April 24, 2019	Abhishek Misra	V1.5	Added screenshots in Section 4
April 22, 2019	Abhishek Misra	V1.4	Changes Provisioning details in Section 4. Formatting changes for commands to be run on mw3xx in section 4. Added Trouble Shooting instructions in section 5
April 17, 2019	Abhinav Kulkarni	V1.3	Minor changes, addition of debug method.
Feb 26 2019	Sudeep Mohanty	V1.2	Added cmake commands for building firmware
Feb 21 2019	Abhishek Misra	V1.1	Added section on provisioning Restructured section on flashing firmware
Feb 7 2019	Abhishek Misra	V1.0	Release V1.2.r4.p1

Table of Contents

1 Getting Started Guide for MW320

The AWS IoT Starter Kit is a development kit based on the MW302, the latest integrated Cortex M4 Microcontroller from Marvell. This integrates 802.11b/g/n Wi-Fi on a single microcontroller chip. The development kit is FCC certified and available for sale. The MW302 module is also FCC certified and available for customization and volume sale.

The first step for development of your application for the SDK board is to cross compile the application along with SDK on a host computer. Once compiled the generated binary file is loaded onto the board using tools provided along with SDK. Once the application starts running on the board you can debug or interact with it from the Serial console on your host computer.

The following host platforms are supported for development:

- Ubuntu 16.04

You may be able to use other platforms, but those platforms are not supported officially. This system will act as the host platform for development and debugging. You need to have permissions to install software on this host system. Following are the external tools required for building SDK successfully.

- Any of the above platforms.
- The ARM toolchain is required to cross compile your application and the SDK. The SDK takes advantage of the latest versions of the toolchain to optimize the image footprint and fit more functionality into less space. Using older toolchains is not recommended. The supported version of the tool chain at the time of writing this document is ***4_9_2015q3***.

Eclipse 4.9.0 IDE is supported

The development kit is pre-flashed with Wireless Microcontroller Demo Project Firmware.

1.1 Development Toolchain Requirements

For development purposes, at a minimum you will need the ARM toolchain (in addition to the tools bundled with the SDK).

1.1.1 GNU Toolchain

The SDK officially supports the GCC Compiler toolchain. The cross-compiler toolchain for GNU ARM is available from the following URL: https://launchpad.net/gcc-arm-embedded/4.9/4.9-2015-q3-update

The build system is configured to use the GNU toolchain by default. The Makefiles assume that the GNU compiler toolchain binaries are available on the user's PATH and can be invoked from the Makefiles. The Makefiles also assume that the GNU toolchain binaries are prefixed with ***`arm-none-eabi-`***.

The GCC toolchain can be used with GDB for debugging with OpenOCD (bundled with the SDK) providing the software interfacing to JTAG.

The current version of the gcc-arm-embedded toolchain at the time of writing this document is 4_9_2015q3

1.1.2 Linux Toolchain Setup Procedure

Setting up the GCC toolchain in Linux requires the steps outlined below.

- Download the toolchain tarball
  - Linux: gcc-arm-none-eabi-4_9-2015q3-20150921-linux.tar.bz2
- Copy the file to a directory of your choosing. Please ensure there are no spaces in the directory name.
- Untar the file using the “tar –vxf <filename>”
- Path of the installed toolchain should be added into system PATH.
  - For example, .**profile** file found under ***/home/<user>*** directory. Append the following line to the end of the file:
    - PATH=$PATH:<path to>gcc-arm-none-eabit-4_9_2015_q3/bin

Note

Newer distributions of Ubuntu might come with a Debian version of the GCC Cross Compiler. It is imperative that the native Cross Compiler is removed. Setup procedure outlined above should be followed.

2 Working with a Linux Development Host

Linux development hosts can be used in lieu of Windows development hosts. Any modern Linux Desktop distribution such as Ubuntu or Fedora can be used, however it is recommended to upgrade to the most recent release. The below steps are explained and tested to work on Ubuntu 16.04.

2.1 Installing Packages

To enable quick setup of development environment on a newly setup Linux machine, a script is provided along with the SDK. The script will try to autodetect the machine type and install the appropriate software viz. C libraries, USB library, FTDI library, ncurses, python and latex.

Make sure you have root privileges, then go to amzsdk_bundle-x.y.z/ directory and run the following command:

```
#./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/installpkgs.sh
```

2.1.1 Avoiding 'sudo'

Your Linux development host can also be configured to perform 'flashprog' and 'ramload' operations without requiring the 'sudo' command to be executed each time. This can be done by executing the following command:

```
#./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/bin/perm_fix.sh
```

Note that fixing these permissions is ***mandatory*** for ensuring a smooth Eclipse IDE based experience.

2.2 Setup Serial Console

1. Insert the USB cable into the Linux host's USB slot as mentioned above. This will trigger the detection of the device and you should see messages like the following in the */var/log/messages* file (or after executing the *dmesg* command).

```
Jan  6 20:00:51 localhost kernel: usb 4-2: new full speed USB device using uhci_hcd and address 127
Jan  6 20:00:51 localhost kernel: usb 4-2: configuration #1 chosen from 1 choice
Jan  6 20:00:51 localhost kernel: ftdi_sio 4-2:1.0: FTDI USB Serial Device converter detected
Jan  6 20:00:51 localhost kernel: ftdi_sio: Detected FT2232C
Jan  6 20:00:51 localhost kernel: usb 4-2: FTDI USB Serial Device converter now attached to ttyUSB0
Jan  6 20:00:51 localhost kernel: ftdi_sio 4-2:1.1: FTDI USB Serial Device converter detected
Jan  6 20:00:51 localhost kernel: ftdi_sio: Detected FT2232C
Jan  6 20:00:51 localhost kernel: usb 4-2: FTDI USB Serial Device converter now attached to ttyUSB1
```

1. As can be seen two ttyUSB devices have been created. The second of this device is the serial console, in our case ttyUSB1
2. Execute minicom in setup mode (*minicom –s*). Alternatively, you can use other serial programs such as putty.

3. Go to Serial Port Setup
4. Perform the following settings:

```
| A -    Serial Device      : /dev/ttyUSB1
| B - Lockfile Location     : /var/lock
| C -   Callin Program      :
| D -  Callout Program      :
| E -    Bps/Par/Bits       : 115200 8N1
| F - Hardware Flow Control : No
| G - Software Flow Control : No
```

You can save these settings in minicom for future use. The minicom window will now show messages from the serial console.

5. Hit enter on the serial console window. This should show you a hash (#) on the screen.

Note

The development boards from Marvell have an FTDI chip. The FTDI chip exposes two USB interfaces for the host. The first interface is associated to JTAG functionality of the MCU and the second interface is associated with physical UARTx port of the MCU.

2.3 OpenOCD

OpenOCD version 0.9 will be required. It is also required for Eclipse functionality. If an earlier version (0.7) was installed on your Linux Host, please remove that repository with the appropriate command for the Linux distribution you are currently using.

OpenOCD can be installed with standard linux command 'apt-get install openocd'

If above mentioned command does not install v0.9 or higher, use following procedure to download source for openocd

- Install libusb-1.0 (sudo apt-get install libusb-1.0)
- Download openocd 0.9.0 (We get it in the form of source code) from http://openocd.org/
- Extract openocd and go to it's folder
- Configure openocd (./configure --enable-ftdi --enable-jlink)
- Make openocd (make install)

3 Eclipse Setup

Note

Please make sure that you have performed steps mentioned in "Avoiding sudo" section.

Eclipse is a preferred IDE for application development and debugging. It provides a rich, user-friendly IDE with integrated debugging support including thread aware debugging. This section describes common Eclipse setup for all the development hosts supported.

3.1 Download and Install

3.1.1 Java Run Time Environment

Eclipse requires Java Run Time Environment (JRE) to be installed. It is recommended to install this first, although it can be installed after Eclipse is installed. The JRE version (32/64 bit) must match the version of Eclipse (32/64bit) that will be installed. JRE can be downloaded from URL: http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html

3.1.2 Eclipse

Download and install "Eclipse IDE for C/C++ Developers" from http://www.eclipse.org. The supported Eclipse version is Eclipse 4.9.0 or above. Installation is just extracting the downloaded archive. Platform specific Eclipse executable can be executed to start the eclipse

4 Build and Run the Amazon FreeRTOS Demo Project

4.1 Provisioning

Depending upon if user wants to use test or demo application, user will need to set provisioning data in below files

./tests/common/include/aws_clientcredential.h

./demos/common/include/aws_clientcredential.h

example:

#define clientcredentialWIFI_SSID "Paste Wi-Fi SSID here"

#define clientcredentialWIFI_PASSWORD "Paste Wi-Fi password here"

#define clientcredentialWIFI_SECURITY Paste Wi-Fi Security

Note: Possible values are - eWiFiSecurityOpen, eWiFiSecurityWEP, eWiFiSecurityWPA and eWiFiSecurityWPA2

Note: SSID and Passphrase should be enclosed in “”

4.2 Working with Command line

4.2.1 Building

Building demo or test application is straight forward. Follow these commands for building a demo application:

$ cmake -DVENDOR=marvell -DBOARD=mw300_rd -DCOMPILER=arm-gcc -S . -Bbuild -DAFR_ENABLE_TESTS=0

Make sure you are getting o/p as in image below

```
marvell@pe-lt586:amzsdk$
marvell@pe-lt586:amzsdk$ cmake -DVENDOR=marvell -DBOARD=mw300_rd -DCOMPILER=arm-gcc -S . -
Bbuild -DAFR_ENABLE_TESTS=0

====================Configuration for Amazon FreeRTOS====================
  Version:                 v1.2.4
  Git version:             AMZSDK_V1.2.r6.p1-12-gdd17d10

Target microcontroller:
  vendor:                  Marvell
  board:                   mw300_rd
  description:             Marvell Board for AmazonFreeRTOS
  family:                  Wireless Microcontroller
  data ram size:           512KB
  program memory size:     2MB

Host platform:
  OS:                      Linux-4.15.0-47-generic
  Toolchain:               arm-gcc
  Toolchain path:          /home/marvell/Software/gcc-arm-none-eabi-4_9-2015q3
  CMake generator:         Unix Makefiles

Amazon FreeRTOS modules:
  Modules to build:        kernel, freertos_plus_tcp, bufferpool, crypto, greengrass, mqtt
, ota, pkcs11, secure_sockets, shadow, tls, wifi
  Disabled by user:
  Disabled by dependency:  posix

  Available demos:         demo_key_provisioning, demo_logging, demo_mqtt_hello_world, dem
o_mqtt_pubsub, demo_tcp, demo_shadow, demo_greengrass, demo_ota
  Available tests:         test_crypto, test_greengrass, test_mqtt, test_ota, test_pkcs11,
 test_secure_sockets, test_tls, test_shadow, test_wifi, test_memory
=========================================================================

-- Configuring done
-- Generating done
-- Build files have been written to: /home/marvell/gerrit/amzsdk/build
marvell@pe-lt586:amzsdk$
```

cd build
make all -j4

Make sure you are getting o/p as in image below

```
[ 92%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborencoder.c.obj
[ 93%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborencoder_close
_container_checked.c.obj
[ 93%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborerrorstrings.
c.obj
[ 93%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborparser.c.obj
[ 94%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborparser_dup_st
ring.c.obj
[ 94%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/tinycbor/cborpretty.c.obj
[ 94%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/jsmn/jsmn.c.obj
[ 95%] Building C object CMakeFiles/afr_ota.dir/demos/common/logging/aws_logging_task_dyna
mic_buffers.c.obj
[ 95%] Building C object CMakeFiles/afr_ota.dir/demos/common/demo_runner/aws_demo_runner.c
.obj
[ 95%] Linking C static library afr_ota.a
[ 95%] Built target afr_ota
[ 96%] Linking C executable aws_demos.axf
[100%] Built target aws_demos
marvell@pe-lt586:build$
```

Follow these commands for building a test application:

$ cmake -DVENDOR=marvell -DBOARD=mw300_rd -DCOMPILER=arm-gcc -S . -Bbuild -DAFR_ENABLE_TESTS=1
$ cd build
$ make all -j4

Make sure to run the cmake command every time you switch between the aws_demos project and the aws_tests project.

4.2.2 Loading to Flash

This method writes the firmware image to the flash of the development board. The firmware then gets executed on a reset of the development board.

4.2.2.1 Loading Layout and Boot2

Before we flash the firmware image, let's prepare the development board's flash with some common components which are namely Layout and Boot2. This can be done as:

$ cd amzsdk_bundle-x.y.z

$ ./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py -l lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt --boot2 lib/third_party/mcu_vendor/marvell/WMSDK/mw320/boot2/bin/boot2.bin

Some comments about what is being done here:

• Layout: The flashprog utility is first instructed to write a layout to the flash. The layout is like a partition information of the flash. The default layout is available in the location /lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt.

• boot2: This is the boot-loader used by the WMSDK. The flashprog is also writing a bootloader to the flash. It is the boot-loader's job to load the microcontroller's firmware image once we flash it subsequently.

Make sure you are getting o/p as in image below

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245458s (115.728 KiB/s)
verified 29088 bytes in 0.350004s (81.160 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Erasing primary flash...done
Writing new flash layout...done
Writing "boot2" @0x0 (primary)...done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

4.2.2.2 Flashing the Wi-Fi Firmware

Notice that this firmware uses the Wi-Fi chipset for its functionality. The Wi-Fi chipset has its own firmware that must be present in flash for this to work. We go back to our flashprog.py utility. Remember we had used this utility to flash the boot2 boot-loader and the MCU firmware to flash. The Wi-Fi firmware can be flashed as:

$ cd amzsdk_bundle-x.y.z

$ ./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --wififw ./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/wifi-firmware/mw30x/mw30x_uapsta_W14.88.36.p135.bin

Make sure you are getting o/p as in image below

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245498s (115.709 KiB/s)
verified 29088 bytes in 0.350229s (81.108 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "wififw" @0x12a000 (primary).....done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
```

.py --wififw

4.2.2.3 Loading MCU Firmware

Once layout and boot2 are flashed, you can flash the microcontroller firmware as under

$ cd amzsdk_bundle-x.y.z

$ ./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --mcufw build/cmake/vendors/marvell/mw300_rd/aws_demos.bin -r

Once this is flashed, on resetting the board you should see the logs for demo app. To run the test app, please flash the aws_tests.bin binary present at the same location.

Make sure you are getting o/p as in image below

```
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
downloaded 29088 bytes in 0.245499s (115.708 KiB/s)
verified 29088 bytes in 0.350231s (81.107 KiB/s)
semihosting is enabled

Flashprog version: 2.1.0
Writing "mcufw" @0x6a000 (primary)...done
semihosting: *** application exited ***
Flashprog Complete
shutdown command invoked


target state: halted
target halted due to breakpoint, current mode: Thread
xPSR: 0x21000000 pc: 0x00100658 msp: 0x0015ffe4, semihosting
Resetting board...
Using OpenOCD interface file ftdi.cfg
Open On-Chip Debugger 0.9.0 (2015-07-15-15:28)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 3000 kHz
adapter_nsrst_delay: 100
Info : auto-selecting first available session transport "jtag". To override use 'transport
 select <transport>'.
jtag_ntrst_delay: 100
cortex_m reset_config sysresetreq
sh_load
Info : clock speed 3000 kHz
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
Info : wmcore.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
shutdown command invoked
Resetting board done...
```

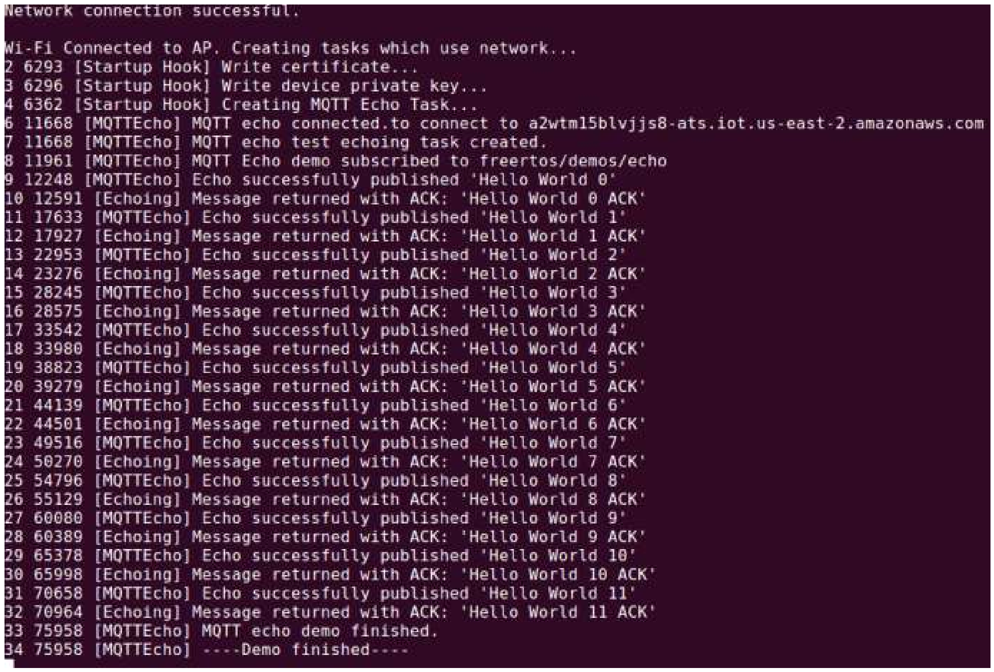
Note:

If you are changing only MCU FW then you need not load boot2, layout and WiFi FW again. If there is any change in layout then it is better to reflash all components again.

Using -r option with flashprog.py causes device reset

4.2.3 Output of Demo App

Make sure you get below o/p once Demo App is flashed and board is reset.



```
Network connection successful.

Wi-Fi Connected to AP. Creating tasks which use network...
2 6293 [Startup Hook] Write certificate...
3 6296 [Startup Hook] Write device private key...
4 6362 [Startup Hook] Creating MQTT Echo Task...
6 11668 [MQTTEcho] MQTT echo connected.to connect to a2wtml5blvjjs8-ats.iot.us-east-2.amazonaws.com
7 11668 [MQTTEcho] MQTT echo test echoing task created.
8 11961 [MQTTEcho] MQTT Echo demo subscribed to freertos/demos/echo
9 12248 [MQTTEcho] Echo successfully published 'Hello World 0'
10 12591 [Echoing] Message returned with ACK: 'Hello World 0 ACK'
11 17633 [MQTTEcho] Echo successfully published 'Hello World 1'
12 17927 [Echoing] Message returned with ACK: 'Hello World 1 ACK'
13 22953 [MQTTEcho] Echo successfully published 'Hello World 2'
14 23276 [Echoing] Message returned with ACK: 'Hello World 2 ACK'
15 28245 [MQTTEcho] Echo successfully published 'Hello World 3'
16 28575 [Echoing] Message returned with ACK: 'Hello World 3 ACK'
17 33542 [MQTTEcho] Echo successfully published 'Hello World 4'
18 33980 [Echoing] Message returned with ACK: 'Hello World 4 ACK'
19 38823 [MQTTEcho] Echo successfully published 'Hello World 5'
20 39279 [Echoing] Message returned with ACK: 'Hello World 5 ACK'
21 44139 [MQTTEcho] Echo successfully published 'Hello World 6'
22 44501 [Echoing] Message returned with ACK: 'Hello World 6 ACK'
23 49516 [MQTTEcho] Echo successfully published 'Hello World 7'
24 50270 [Echoing] Message returned with ACK: 'Hello World 7 ACK'
25 54796 [MQTTEcho] Echo successfully published 'Hello World 8'
26 55129 [Echoing] Message returned with ACK: 'Hello World 8 ACK'
27 60080 [MQTTEcho] Echo successfully published 'Hello World 9'
28 60389 [Echoing] Message returned with ACK: 'Hello World 9 ACK'
29 65378 [MQTTEcho] Echo successfully published 'Hello World 10'
30 65998 [Echoing] Message returned with ACK: 'Hello World 10 ACK'
31 70658 [MQTTEcho] Echo successfully published 'Hello World 11'
32 70964 [Echoing] Message returned with ACK: 'Hello World 11 ACK'
33 75958 [MQTTEcho] MQTT echo demo finished.
34 75958 [MQTTEcho] ----Demo finished----
```

4.2.4 Loading to SRAM

This method loads the firmware image in SRAM and directly the execution is started. Since loading the firmware in SRAM is a faster operation, this is what you will most commonly use during iterative development.

The firmware can be loaded into SRAM as follows:

$ cd amzsdk_bundle-x.y.z

$ ./lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/tools/OpenOCD/ramload.py build/cmake/vendors/marvell/mw300_rd/aws_demos.axf

```
Using OpenOCD interface file ftdi.cfg
Open On-Chip Debugger 0.9.0 (2015-07-15-15:28)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
adapter speed: 3000 kHz
adapter_nsrst_delay: 100
Info : auto-selecting first available session transport "jtag". To override use 'transport
 select <transport>'.
jtag_ntrst_delay: 100
cortex_m reset_config sysresetreq
sh_load
Info : clock speed 3000 kHz
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
Info : wmcore.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: 0
x4)
target state: halted
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
75072 bytes written at address 0x00100000
8 bytes written at address 0x00112540
468 bytes written at address 0x20000040
downloaded 75548 bytes in 0.636127s (115.979 KiB/s)
verified 75548 bytes in 0.959023s (76.930 KiB/s)
shutdown command invoked
```

Once you execute the above command, you should the logs for demo app.

4.3 Working with Eclipse

Before setting up an Eclipse work space, please make sure that you run the cmake command as mentioned in section 4.2.1.

To work with the aws_demos Eclipse project, please run the below command first on the command line:

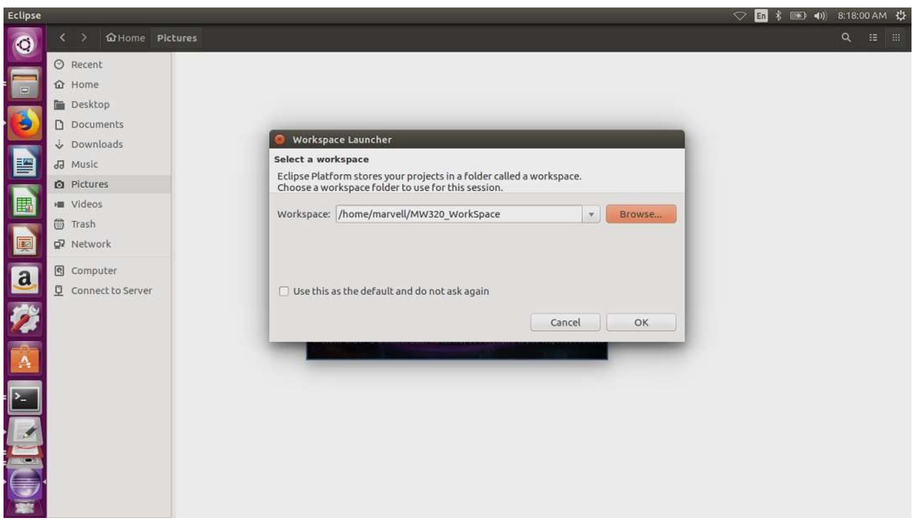
$ cmake -DVENDOR=marvell -DBOARD=mw300_rd -DCOMPILER=arm-gcc -S . -Bbuild -DAFR_ENABLE_TESTS=0

To work with the aws_tests Eclipse project, please run the below command first on the command line:

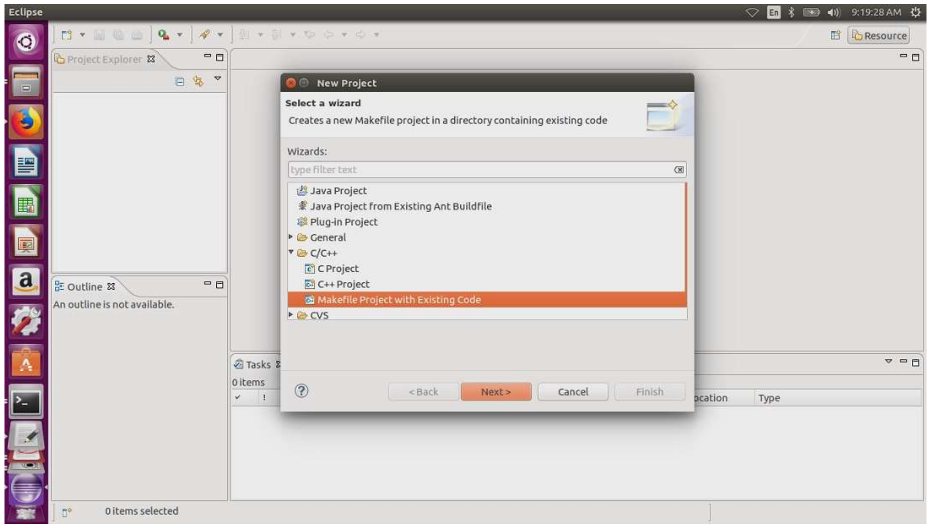
$ cmake -DVENDOR=marvell -DBOARD=mw300_rd -DCOMPILER=arm-gcc -S . -Bbuild -DAFR_ENABLE_TESTS=1

Make sure to run the cmake command every time you switch between the aws_demos project and the aws_tests project.

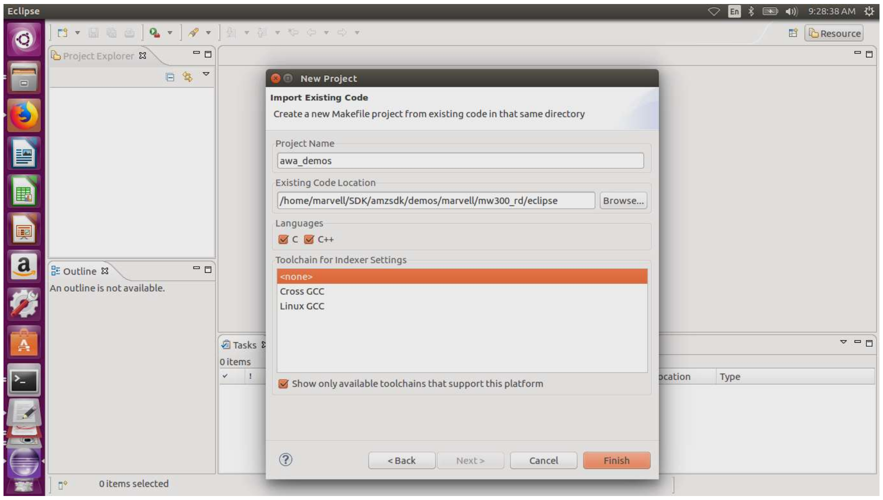
Select your Eclipse Work Space



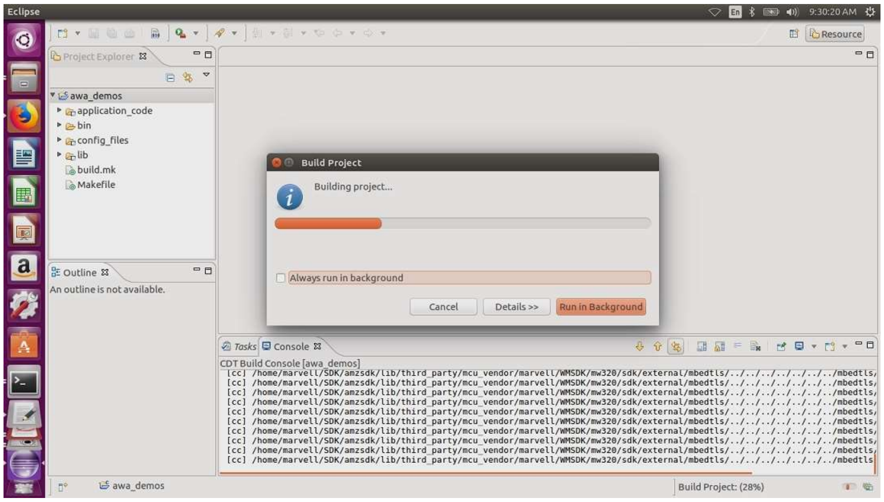
Create Makefile Project with existing code Load Demo Project


Eclipse
9:19:28 AM
Resource
Project Explorer
Outline
An outline is not available.
Tasks
0 items
0 items selected
New Project
Select a wizard
Creates a new Makefile project in a directory containing existing code
Wizards:
type filter text
Java Project
Java Project from Existing Ant Buildfile
Plug-in Project
General
C/C++
C Project
C++ Project
Makefile Project with Existing Code
CVS
< Back
Next >
Cancel
Finish
Type

Browse to and load aws_demos project



Select aws_demos project and use right-click menu to build project



On Successful build *aws_demos.bin* will be generated at below path
build/cmake/vendors/marvell/mw300_rd/aws_demos.bin

After this command line tools can be used to flash Layout file, Boot2 binary, MCU Firmware Binary i.e. layout.txt, boot2.bin, aws_demos.bin and WiFi Firmware.

5 Troubleshooting

Check if network credentials are valid if network is not getting up.

5.1 Enable additional logs

To enable additional logs related to wifi and Marvell modules please follow below mentioned procedure

To enable board specific logs: Enable call to *wmstdio_init(UART0_ID, 0)* in function *prvMiscInitialization* file *main.c* file(tests or demos)

To enable WiFi logs: Enable macro CONFIG_WLCMGR_DEBUG in *AFR_HOME/lib/third_party/mcu_vendor/marvell/WMSDK/mw320/sdk/src/incl/autoconf.h* file.

cd AFR_HOME/lib/third_party/mcu_vendor/marvell/WMSDK/mw320

Run this command to connect to GDB:

5.2

arm-none-eabi-gdb -x ./sdk/tools/OpenOCD/gdbinit ../../../../../../build/cmake/vendors/marvell/mw300_rd/aws_demos.axf

cd AFR_HOME/lib/third_party/mcu_vendor/marvell/WMSDK/mw320

Run below command to connect to GDB:

arm-none-eabi-gdb -x sdk/tools/OpenOCD/gdbinit <path to axf file to be debugged>/aws_demos.axf

e.g.

arm-none-eabi-gdb -x ./sdk/tools/OpenOCD/gdbinit ../../../../../../build/cmake/vendors/marvell/mw300_rd/aws_demos.axf

Marvell Semiconductor, Inc.
5488 Marvell Lane
Santa Clara, CA 95054, USA

Tel: 1.408.222.2500
Fax: 1.408.752.9028

www.marvell.com

Marvell. Moving Forward Faster