

# Marvell Wireless Microcontroller

Getting Started Guide for MW320/MW322

V1.9

June 24, 2019 CONFIDENTIAL Document Classification: Proprietary Information

Marvell.Moving Forward Faster



### **Document Conventions**

Caution
4

Note: Provides related information or information of special importance.

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

#### Disclaimer

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose, without the express written permission of Marvell. Marvell retains the right to make changes to this document at any time, without notice. Marvell makes no warranty of any kind, expressed or implied, with regard to any information contained in this document, including, but not limited to, the implied warranties of merchantability or fitness for any particular purpose. Further, Marvell does not warrant the accuracy or completeness of the information, text, graphics, or other items contained within this document. Marvell products are not designed for use in life-support equipment or applications that would cause a life-threatening situation if any such products failed. Do not use Marvell products in these types of equipment or applications.

With respect to the products described herein, the user or recipient, in the absence of appropriate U.S. government authorization, agrees:

1) Not to re-export or release any such information consisting of technology, software or source code controlled for national security reasons by the U.S. Export Control Regulations ("EAR"), to a national of EAR Country Groups D:1 or E:2;

2) Not to export the direct product of such technology or such software, to EAR Country Groups D:1 or E:2, if such technology or software and direct products thereof are controlled for national security reasons by the EAR; and,

3) In the case of technology controlled for national security reasons under the EAR where the direct product of the technology is a complete plant or component of a plant, not to export to EAR Country Groups D:1 or E:2 the direct product of the plant or major component thereof, if such direct product is controlled for national security reasons by the EAR, or is subject to controls under the U.S. Munitions List ("USML").

At all times hereunder, the recipient of any such information agrees that they shall be deemed to have manually signed this document in connection with their receipt of any such information.

Copyright © 2019. Marvell International Ltd. All rights reserved. Marvell, the Marvell logo, Moving Forward Faster, Alaska, Fastwriter, Datacom Systems on Silicon, Libertas, Link Street, NetGX, PHYAdvantage, Prestera, Raising The Technology Bar, The Technology Within, Virtual Cable Tester, and Yukon are registered trademarks of Marvell. Ants, AnyVoltage, Discovery, DSP Switcher, Feroceon, GalNet, GalTis, Horizon, Marvell Makes It All Possible, UniMAC, and VCT are trademarks of Marvell. Intel Xscale is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States and other countries. All other trademarks are the property of their respective owners.

Date	Author	Revision	Changelog
June 24, 2019	Abhinav Kulkarni	V1.9	Changes with respect to major release from amazon.
May 06, 2019	Abhishek Misra	V1.8	Fixed command for flashing wifi fw
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	5
1.1	Development Toolchain Requirements	5
	1.1.2 Linux Toolchain Setup Procedure	6
2	Working with a Linux Development Host	7
2.1	Installing Packages 2.1.1 Avoiding 'sudo'	7 7
2.2	Setup Serial Console	7
2.3	OpenOCD	8
3	Build and Run the Amazon FreeRTOS Demo Project	9
3.1	Provisioning	9
3.2	Working with Command line	10
	3.2.1 Building	10
	3.2.2 Loading to Flash	
	3.2.4 Loading to SRAM	
Trou	bleshooting	17
3.3	Enable additional logs	17
3.4	Using GDB	17

# **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: <u>https://launchpad.net/gcc-arm-embedded/4.9/4.9-2015-q3-update</u>

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

- o 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



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:

#./vendors/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:

#./vendors/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

 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: ftdi_sio: Detected FT2232C
Jan 6 20:00:51 localhost kernel: ftdi_sio: Detected FT2232C
```

- 1. As can be seen two ttyUSB devices have been created. The second of this device is the serial console, in our case ttyUSB1
- 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 -	Seria	L Dev:	Lce	:	/dev/ttyUSB1
B -	Lockfile	Locat	tion	:	/var/lock
C -	Callin	Prog	ram	:	
D -	Callout	Prog	ram	:	
E -	Bps/Pa	ar/Bit	s	:	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.



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.

#### OpenOCD 2.3

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/ ٠
- Extract openocd and go to it's folder ٠
- Configure openocd (./configure --enable-ftdi --enable-jlink) •
- Make openocd (make install)

June 24, 2019

# Build and Run the Amazon FreeRTOS Demo Project

# 3.1 **Provisioning**

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

./tests/include/aws\_clientcredential.h

./demos/include/aws\_clientcredential.h

#### example:

#define clientcredentialWIFI_SSID	
#define clientcredentialWIFI_PASSWORD	
#define clientcredentialWIFI_SECURITY	

"Paste Wi-Fi SSID here" "Paste Wi-Fi password here" Paste Wi-Fi Security

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

Note: SSID and Passphrase should be enclosed in ""



# 3.2 Working with Command line

## 3.2.1 Building

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

\$ cmake -DVENDOR=marvell -DBOARD=mw320 -DCOMPILER=arm-gcc -S . -Bbuild -DAFR\_ENABLE\_TESTS=0

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

=====Config	uration 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, crvpto, greengrass, mgt
, ota, pkcsll, secure_sock	ets, shadow, tls, wifi
Disabled by dependency:	nogiv
bisabled by dependency.	posix
Available demos: o_mqtt_pubsub, demo_tcp, d	<pre>demo_key_provisioning, demo_logging, demo_mqtt_hello_world, de emo_shadow, demo_greengrass, demo_ota</pre>
Available tests:	<pre>test_crypto, test_greengrass, test_mqtt, test_ota, test_pkcsll</pre>
<pre>test_secure_sockets, test</pre>	_tls, test_shadow, test_wifi, test_memory
Configuring done	
Generating done	
Build files have been w	ritten to: /home/marvell/gerrit/amzsdk/build

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/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
[ 94%] Building C object CMakeFiles/afr_ota.dir/lib/third_party/jsmn/jsmn.c.obj
[ 95%] 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=mw320 -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.



## 3.2.2 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.

#### 3.2.2.1 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

\$ ./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py -l vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/mw300/layout.txt --boot2 vendors/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

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

### 3.2.2.2 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

\$ ./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py --wififw ./vendors/marvell/WMSDK/mw320/wifi-firmware/mw30x/mw30x\_uapsta\_W14.88.36.p135.bin

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





### 3.2.2.3 Loading MCU Firmware

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

\$ cd amzsdk\_bundle-x.y.z

\$./vendors/marvell/WMSDK/mw320/sdk/tools/OpenOCD/flashprog.py build/cmake/vendors/marvell/mw300\_rd/aws\_demos.bin -r -mcufw

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

```
hal
target halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
29088 bytes written at address 0x00100000
lownloaded 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
emihosting: *** 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
pen On-Chip Debugger 0.9.0 (2015-07-15-15:28)
icensed 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>'.
tag_ntrst_delay: 100
ortex_m reset_config sysresetreq
h load
Info : clock speed 3000 kHz
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver: |
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
ĸ4)
hutdown 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

# 3.2.3 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 a2wtm15blvjjs8-ats.iot.us-east-2.amazonaws.u	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 [MQTIECNO] ECHO SUCCESSTULLY PUBLISHED 'HELLO WORLD 9'	
28 60389 [Echoing] Message returned with ACK: 'Hello World 9 ACK'	
29 653/8 [MQTTECNO] ECHO SUCCESSTULLY PUBLISHED 'HELLO WORLD 10'	
30 55998 [Echoing] Message returned with ACK: 'Hello World 10 ACK'	
31 70538 [MUTTECHO] ECHO SUCCESSIULTY Published 'Hello World 11'	
32 76964 [Echoing] Message returned with ACK: 'Hello World II ACK'	
33 75958 [MUTEcho] MUTEcho demo finished.	
34 /S958 [MQTTECHO]Demo TINISNEd	



## 3.2.4 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

\$ ./vendors/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>'.
tag_ntrst_delay: 100
ortex_m reset_config sysresetreq
h_load
Info : clock speed 3000 kHz
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver:
ĸ4)
Info : wmcore.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : JTAG tap: wmcore.cpu tap/device found: 0x4ba00477 (mfg: 0x23b, part: 0xba00, ver:
x4)
arget state: halted
arget halted due to debug-request, current mode: Thread
xPSR: 0x01000000 pc: 0x00007f14 msp: 0x20001000
5072 bytes written at address 0x00100000
 bytes written at address 0x00112540
468 bytes written at address 0x20000040
lownloaded 75548 bytes in 0.636127s (115.979 KiB/s)
erified 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.

# Troubleshooting

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

# 3.3 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:

#### **3.4** 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

![](_page_17_Picture_0.jpeg)

VEL

-

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