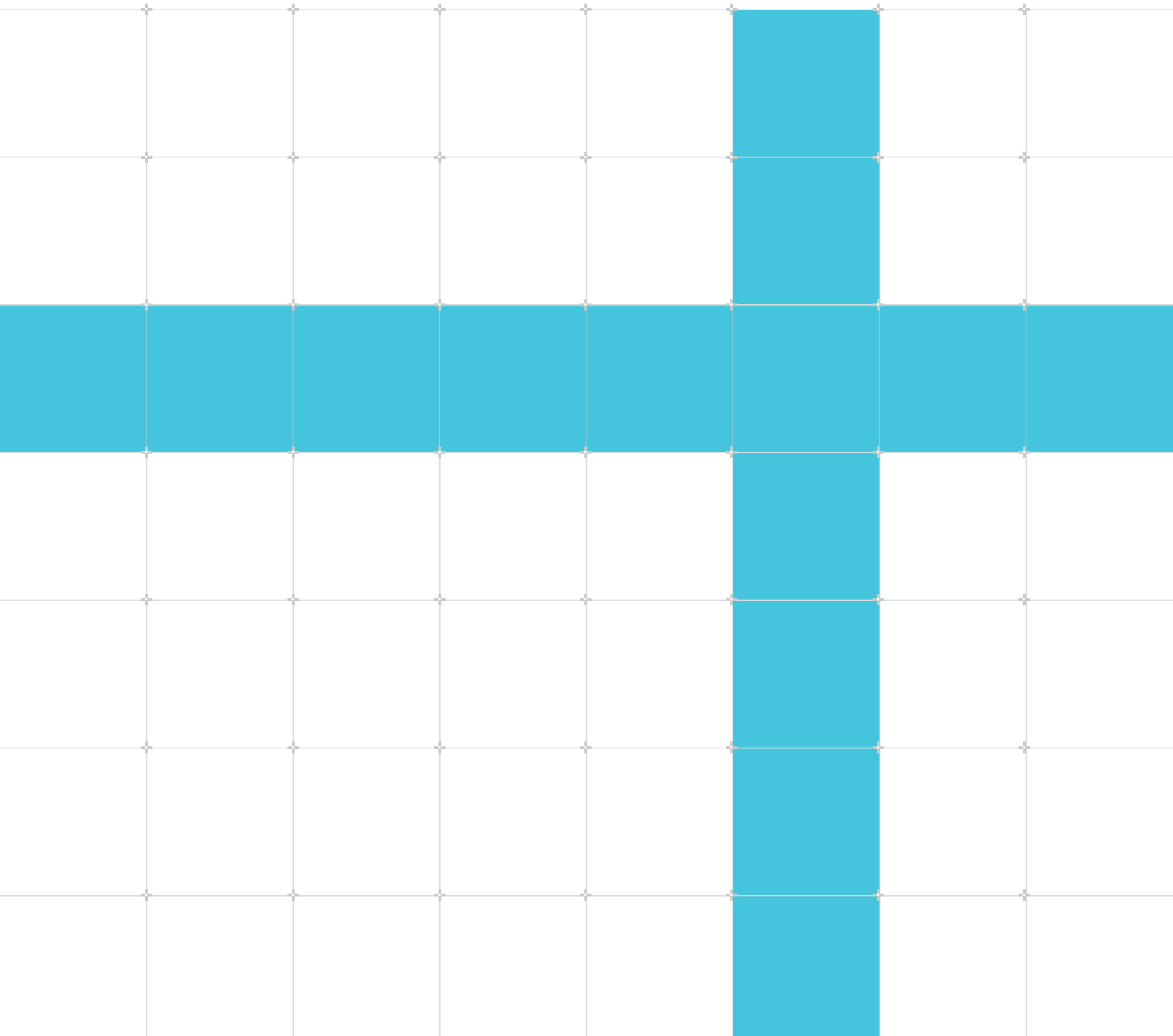
Morello Development Platform and Software

Version 1.0

Getting Started Guide

Issue 04
den0132_0100_04_en



Morello Development Platform and Software
Getting Started Guide

Release information

Document history

Issue	Date	Confidentiality	Change
0100-01	17 January 2022	Non-Confidential	Initial release
0100-02	7 March 2022	Non-Confidential	Minor modifications to the instructions for flashing the onboard SD card.
0100-03	14 April 2022	Non-Confidential	Minor modifications to the instructions for setting up the Morello Hardware Development Platform and installing the repo tool.
0100-04	17 June 2022	Non-Confidential	Minor modifications to the instructions for flashing the onboard SD card.

Proprietary Notice

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1. Overview

Morello is an experimental architecture protection model that extends the Armv8.2-A profile. This profile implements the 129-bit Capability Hardware Enhanced RISC Instructions (CHERI) capability protection model. Morello is backward compatible with Armv8-A architecture systems in AArch64.

CHERI is an Instruction Set Architecture (ISA) that improves system memory security. CHERI compartmentalizes memory and assigns permissions to each range of memory using structures called capabilities.

Compartmentalizing splits memory into sections that contain a set of permissions, bounds, and other metadata. The metadata associated with these sections are stored in capabilities. Capabilities are stored in memory in 16-byte (128 bit) memory locations and the associated validity tag is stored in a separate location.

Figure 1-1: Morello Board - Upright and Side Views

1.1 Capabilities for memory security

Capabilities are used in place of all language visible pointers in C/C++. Language-visible pointers can be exploited by attackers, usually by overwriting pointers or reinterpreting data as pointers. Morello capabilities ensure that only operations with the associated permissions can overwrite pointers.

Morello combats many different types of memory security issues, such as:

- Remote code injection through buffer overflows
- Data pointer corruption

Capabilities have the following protection properties to ensure the memory security of the system:

- Provenance: capabilities can only be changed or created through valid manipulations of other capabilities.
- Integrity: capabilities that become corrupted cannot be de-referenced.
- Monotonicity: the rights and permissions of a capability cannot increase.

When a processing element attempts to access a value from virtual memory, the capability associated with the memory location must be checked first. This check determines which permissions can be performed to or by the data in that memory location. The capability is then treated as an integer address processed by the Memory Management Unit (MMU) and memory subsystem.

To protect other capabilities and areas of memory, capability permissions change when writing to a capability register or capability-tagged memory as follows:

- If a non-capability tagged operation attempts to write to a capability register, the permissions granted to the capabilities do not increase. In addition, this writing attempt does not increase the set of reachable capabilities.
- If an unknown value is written to a capability register or capability-tagged memory, the write operation will not result in an increase the capability defined rights available to the software.

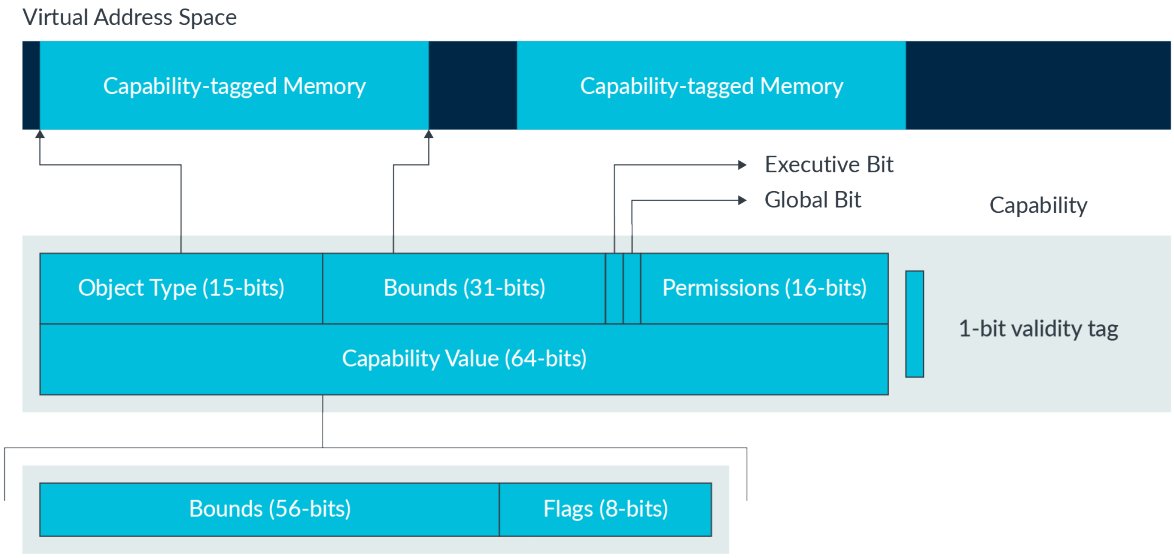
The Capability Program Counter and the Default Data Capability Registers are granted the maximum capability bounds and maximum permissions after every system reset. New maximum permission levels are created when new virtual address spaces are created.

1.2 Capability structure

Capabilities are given specific access permissions and are linked to a virtual address space, not a physical space. This access and address space causes memory to be viewed as an unordered collection of objects, which is more difficult for an attacker to bypass. Capabilities point to the sections of memory they are associated with using bounds, which are stored as a base address value and either a length or size limit that defines the range of virtual memory covered by the capability. If access is gained to a memory location during an attack, memory locations that do not share the same capability are protected.

Figure 1.3 shows how capabilities are stored separately to regular memory areas:

Figure 1-2: Capability structure



Capabilities are structured as follows:

- Validity tag (1-bit)
- Executive bit (1-bit)
- Global bit (1-bit)
- Permissions (16-bits). To see how the permissions are encoded, refer to the Morello Architecture Reference Manual in the Arm Architecture Reference Manual Supplement - Morello for A-profile Architecture.
- ObjectType (15 bits)
- Bounds (87-bits):
  - 56 bits of the bounds are encoded in the 64-bit capability value
  - 31 bits of the bounds are stored in the capability separate from the capability value
- Flags (8 bits) encoded in the 64-bit capability value

The 64-bit capability value can be accessed as either an absolute value or as an address that is offset from the lower bound of the capability. The permissions of the capability are 16-bits long and encoded with the executive bit and the global bit.

1.3 ObjectType and Sealing capabilities

Sealing and unsealing are capability functions used to support higher level software encapsulation. Sealed capabilities can only perform unseal operations. Generally, capabilities can only be unsealed

during an unseal operation by citing a capability of the same type. Some capabilities with hardware types can be unsealed by the code pointed to by the unsealed version of the capability.

If a memory location is used to store critical data and only certain functions can access the location, the function can unseal the capability associated with the memory location. The function also uses the data stored at the location and seals the associated capability at the end of the process to prevent use from functions that must not have access.

ObjectType controls whether the capability is sealed or unsealed. If the ObjectType of a capability is 0, the capability is unsealed but if the value is non-zero, the capability is sealed. The different values of ObjectType correspond to the ways that the valid, sealed capability can be unsealed.

For more information on sealing and unsealing capabilities, refer to the Morello Architecture reference manual in the Arm Architecture Reference Manual Supplement - Morello for A-profile Architecture.

2. Setting up the Morello Hardware Development Platform

Before you set up your board, ensure you have the following setup requirements:

- USB drive with 4GB or more of space
- Monitor with HDMI Input Support for at least 1920x1080@60Hz
- Ubuntu Linux 18.04 LTS running on an x86_64 machine

What is included with the board

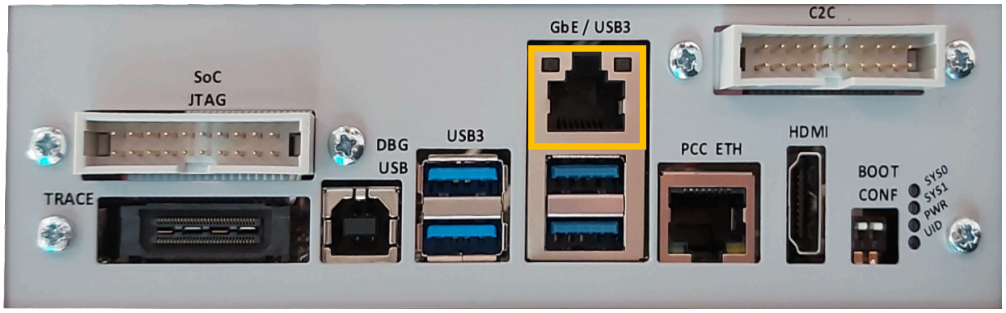
- ATX case or rack mount
- ATX Power supply with power supply cables
- 2GB SD card with firmware
- USB Type A-B cable for debug and configuration
- 250 GB SATA SSD
- HDMI cable
- 3m Cat5 Ethernet cable
- UK/USA/EU mains cables

The firmware on the SD card boots the board to UEFI to ensure that the board is working. This is not the firmware that is needed to use the board; this can be installed using the instructions found in Installing the repo tool.

To set up the Morello development board:

1. Plug the USB-B cable into the debug (DBG) USB port on the Morello board and into your host PC.
2. Plug one end of the HDMI cable into the HDMI port on the Morello board, and the other end into a monitor.
3. Plug one end of the ethernet cable into the gigabit ethernet port (GbE) on the device and plug the other end of the ethernet cable into your local network router.

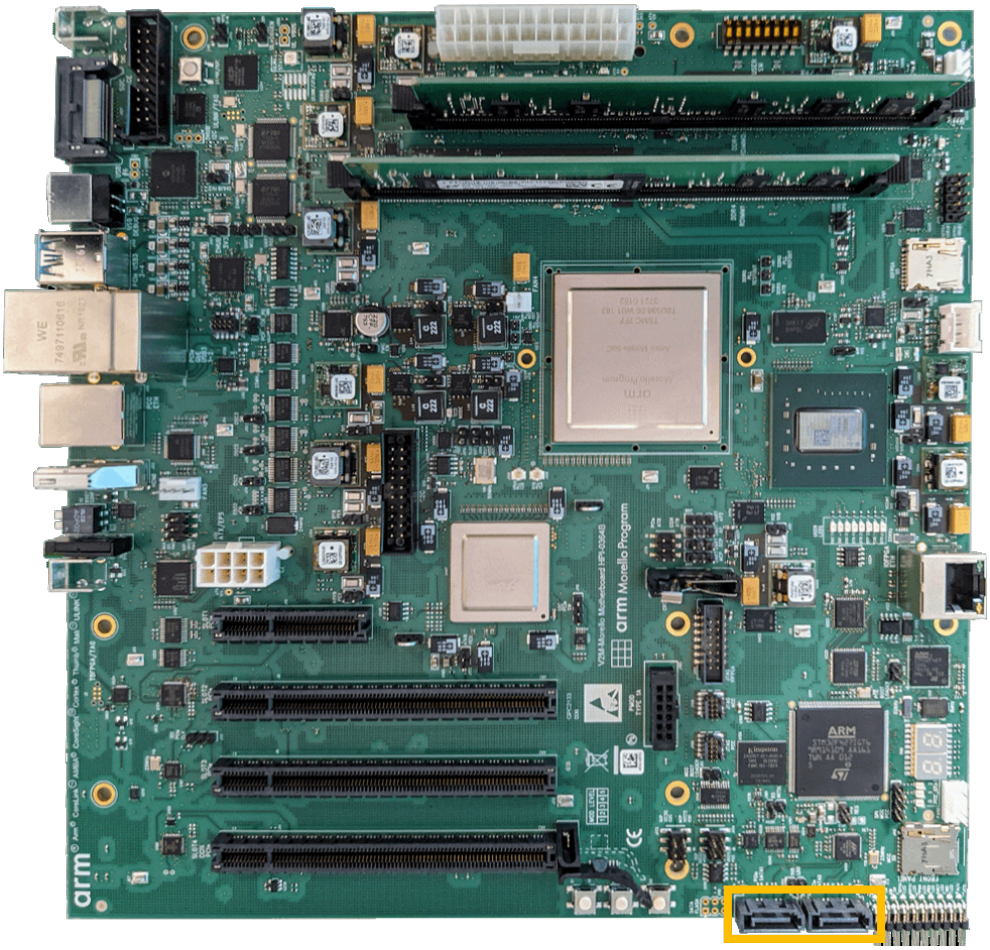
Figure 2-1: Gigabit Ethernet Port



The Morello development platform can accomodate two SATA storage devices. To install an additional SATA HDD or SATA SSD:

1. Locate the SATA power cables by removing the metal side panel.
2. Unfold the power cables and connect them to the hard drive.
3. Plug the data cable into the hard drive and the other end into the board as indicated in the following diagram.

Figure 2-2: SATA Slot on Board



To test that the development platform can boot, turn on the board using the power supply switch at the rear of the chassis.

Note

There will not be any HDMI output at this stage of set up.

To format the Morello board's pre-installed SATA SSD, turn off the Morello board and the host PC. Unplug the SATA SSD's data and power cables from the Morello board and plug it into your host PC using SATA power and data cables. Turn your host PC on and then, in your Linux terminal, type `lsblk`. The output of the `lsblk` command will look like this:

```
NAME   MAJ:MIN  RM    SIZE  RO  TYPE  MOUNTPOINT
sda      8:0     0     64G   0  disk
|--sda1  8:1     0     62G   0  part  /
|--sda2  8:2     0      2G   0  part
sdb      8:16    0    250G   0  disk
```

In this example, the SSD is `sdb`. First, a partition must be created using:

```
sudo fdisk /dev/sdb
```

There are a few commands that can be used now:

1. `m` is the help command, it will print a list of all of the supported parameters that can be used.
2. `n` will create a new partition
3. `d` will delete the partition
4. `p` allows the user to check the partition table

Enter `n` to create a new partition. An option to create a primary or extended partition will be created; enter `p` to create a primary partition and enter `1` to create a single partition. Once the partition is created, it can be formatted using the command:

```
sudo mkfs.ext4 /dev/sdb
```

This will look for any of the partitions that were just created. To confirm that you wish to format the drive, enter `y` and wait for the disk to format.

After this, turn off the Host PC then unplug and remove the Morello SSD. Make sure you remove the correct SSD, Then re-insert it into the Morello Board and plud in the SATA Power and data cables. The Morello Board can now be switched on.

3. Building the Morello software stack

The Morello development platform is supported by many open-source software stacks and supporting tools. These resources provide a foundation for ecosystem research.

For more details, see Morello Platform Open Source Software.

The Morello board comes with pre-installed software that allows the board to be booted. The pre-installed software is not the full software stack, the full package still needs to be installed into the board.

To build the software stack, follow the instructions in the Morello User Guide. This guide tells you how to build the software stack and run it on the Morello Fixed Virtual Platform (FVP) and the Morello Development Board.

To run the software stack on a physical or remote access Morello board, follow the user guide until the Running the software on FVP section, then continue with the steps given in this guide to complete the process.

We recommend that you use the Android build of the software stack, because this build has been more thoroughly tested.

4. Installing the repo tool

The supported configuration of the repo tool cannot be installed using `apt-get`. To install the repo tool, use the following code to ensure you have a binary directory (`/bin`) included in the home directory path:

```
$ mkdir -p ~/.bin
$ PATH="${HOME}/.bin:${PATH}"
```

Then, use the following code to download the repo launcher from Google APIs:

```
$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/.bin/repo
$ chmod a+rx ~/.bin/repo
```

5. Create a bootable USB drive

After the software stack is built, you must create a bootable USB drive.

To create a bootable USB drive:

1. Plug the USB drive into one of the host PC's USB ports to flash the firmware.
2. To identify the USB drive device, use the `lsblk` command as shown in the following example:

```
$ lsblk
NAME                MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
sda                   8:0    0 931.5G  0 disk
|-sda1                8:1    0 931.5G  0 part
|-g_sda-data_sda 253:2    0 931.5G  0 lvm  /data_sda
sdb                   8:16   1  14.6G  0 disk
```

   The USB drive can be identified by the size of the storage options. In this example, the USB device is `sdb`. It is important to get the right storage service and not confuse the host PC's internal drives with the USB drive to ensure that the data on the host PC does not get deleted.
3. The desired firmware image can then be flashed onto the USB drive and synced:

```
$ sudo dd if=<IMAGE> of=/dev/sdb conv=fsync bs=1M
$ sync
```

   Replace <IMAGE> with the necessary image:

   - <morello_workspace>/output/soc/busybox.img
   - <morello_workspace>/output/soc/android-swr.img
   - <morello_workspace>/output/soc/android-nano.img

6. Flash the onboard SD card

The internal microSD card in the Morello board must be mounted over the DBG USB connection to your host PC to copy the required files. The microSD card should be FAT16 formatted.

To mount the microSD card:

1. Use the `lsblk` command before and after the debug USB is plugged in to identify the Morello microSD card.

   In this example, the SD card partition is listed as `sdx1`. The `x` may be a different letter in your system
2. Mount the SD card using the following code:

```
$ sudo mount /dev/sdx1 /mnt
```

On the host PC, navigate to your `/dev` directory. List the contents of the directory using `ls` to show the following TTY USB devices:

- `MCC PORT - ttyUSB(x)`
- `PCC PORT  - ttyUSB(x+1)`
- `AP0 PORT - ttyUSB(x+2)`
- `SCP PORT - ttyUSB(x+3)`
- `MCP PORT - ttyUSB(x+4)`
- `IOFPGA1 PORT  - ttyUSB(x+5)`
- `IOFPGA2 PORT - ttyUSB(x+6)`
- `AP2 (secure) PORT - ttyUSB(x+7)`

For example, these ports will be labelled like `ttyUSB0`, `ttyUSB1` and so on.

The port settings are:

- 115200 Baud
- 8N1
- No parity
- 1 stop bit
- No hardware or software flow control

To flash the onboard SD card:

1. Before running the deliverables on the Morello board, ensure both BOOT CONF switches are in the OFF position.

2. Connect to the MCC device using your preferred terminal emulator on your host PC as follows:

```
sudo screen /dev/ttyUSB(x) 115200
```


Note

`screen` is used here for portability, but any terminal emulator is acceptable.

3. Turn on the Morello board power supply. The MCC window is shown. Type `?` to see the MCC firmware version and a list of commands, as shown in the following example:

```
Cmd> ?
Arm Morello MCC Firmware v2.1.8
Build Date: Jan 12 2022
Build Time: 14:18:16
+ command ------------------+ function ---------------------------------+
| CAP "fname" [/A]          | captures serial data to a file            |
|                           |  [/A option appends data to a file]       |
| FILL "fname" [nnnn]       | create a file filled with text            |
|                           |  [nnnn - number of lines, default=1000]   |
| TYPE "fname"              | displays the content of a text file       |
| REN "fname1" "fname2"     | renames a file 'fname1' to 'fname2'       |
| COPY "fin" ["fin2"] "fout"| copies a file 'fin' to 'fout' file        |
|                           |  ['fin2' option merges 'fin' and 'fin2']  |
| DEL "fname"               | deletes a file                            |
| DIR "[mask]"              | displays a list of files in the directory |
| FORMAT [label]            | formats Flash Memory Card                 |
| USB_ON                    | Enable usb                                |
| USB_OFF                   | Disable usb                               |
| SHUTDOWN                  | Shutdown PSU (leave micro running)        |
| REBOOT                    | Power cycle system and reboot             |
| RESET                     | Reset Board using CB_nRST                 |
| DEBUG                     | Enters debug menu                         |
| EEPROM                    | Enters eeprom menu                        |
| HELP  or  ?               | displays this help                        |
|                                                                       |
| THE FOLLOWING COMMANDS ARE ONLY AVAILABLE IN RUN MODE                 |
|                                                                       |
| CASE_FAN_SPEED "SPEED"    | Choose from SLOW, MEDIUM, FAST            |
| READ_AXI "fname"          | Read system memory to file 'fname'        |
|          "address"        | from address to end address               |
|          "end_address"    |                                           |
| WRITE_AXI "fname"         | Write file 'fname' to system memory       |
|           "address"       | at address                                |
+---------------------------+-------------------------------------------+
Cmd>
```

4. Set the time and date using the debug command as shown:

```
Cmd> debug
Debug> time
Debug> date
Debug> exit
```

5. Enable the use of the MCC USB with the following code:

```
Cmd> USB_ON
```

6. Exit the MCC Terminal session. When using the screen command, to exit, press Ctrl+A then K to exit the MCC menu and kill the screen session.
7. Copy board firmware binaries to the mounted microSD card using the following code:

```
$ rm -rf /mnt/*
```

This will delete any modified platform configuration files that have been previously been modified.

To update the board and prebuilt SoC firmware use:

```
$ sudo cp -r <morello_workspace>/bsp/board-firmware/* /mnt/
```

To update the source build SoC firmware:

```
$ sudo cp -r <morello_workspace>/output/soc/firmware/* /mnt/SOFTWARE/
$ sudo sync
$ sudo umount /mnt
```

Failure to `umount` the SD card storage while issuing an MCC reboot command may corrupt the SD card and will mark the file system as 'dirty'.

8. You can now plug the bootable USB drive (prepared in Create a bootable USB drive) into one of the four USB ports on the back of the board.
9. Connect to the MCC console:

```
sudo screen /dev/ttyUSB(x) 115200
```

`screen` is used here for portability, but any terminal emulator is acceptable.

10. Reboot the board using the `REBOOT` command. Ensure that the SD-Card storage is NOT mounted when the reboot command is issued:

```
Cmd> REBOOT
```

11. Press Ctrl+A then K to exit the MCC menu and kill the screen session.

12. Connect to the AP UART console using the following command

```
sudo screen /dev/ttyUSB(x+2) 115200
```

13. Make sure to press the Esc key within 10 seconds to enter the UEFI Boot Manager menu and select the disk.
14. The system will boot into the software image environment previously selected.

7. Morello tools operation

The Morello toolchain is derived from Clang, an open-source compiler for C/C++. The toolchain also provides an lld linker, the lldb debugger, and other utilities such as an assembler and disassembler. The toolchain can generate code for Plain AArch64, capability enabled AArch64 with an ABI backward compatible with AArch64 ABI, and a pure capability ABI for which the code generation is restricted to the C64 Instruction set.

Only capability enabled AArch64 and pure capability versions of the code can be used with the Morello board. For details about how to use the LLVM compiler with Morello support, refer to the LLVM complier with Morello support user guide.

When compiling, select the appropriate root directory for the target using the following `sysroot` option:

```
--sysroot=<host_path_to_morello_workspace>
```

7.1 Compile, link, and debug with capability enabled AArch 64

Select the A64 ISA using the following code:

```
> clang -target <aarch64 target> -march=morello -mabi=aapcs -02 -wall -o
helloworld helloworld.c
```

The `-mabi=aapcs` option is present in the line to demonstrate how to add the `-mabi` flag. You could omit `-mabi=aapcs` from this command because this is the default setting.

After compiling, the compiled code is linked using the `lld` linker as shown:

```
> ld.lld -o output -entry _start --gc-sections in0.o in1.o --morello-c64-plt
```

7.2 Compile and link pure capability ABI

Select the pure capability ISA using the following command:

```
> clang -target <aarch64 target> -march=morello+c64 -mabi=purecap -02 -Wall -o
helloworld helloworld.c
```

The `-mabi=purecap` argument is added to the command because pure capability is not the default ISA.

To link the compiled code and produce a pure capability ABI binary using ld.lld, use the following command:

```
> ld.lld -o output -entry _start --gc-sections in0.o in1.o
```

7.3 Debug both ABI options

LLDB for Morello has only been tested on Android. Copy the `lldb-server` to your Android device and launch using the following commands:

```
>adb push <llvm_build>/runtimes_ndk_cxx/aarch64/lldb-server /data/local/tmp
>port=5039
>adb forward tcp:$port tcp:$port
>adb shell /data/local/tmp/lldb-server platform --listen "*:$port"
```

If a physical board is being used to debug, using LLDB shell, run the following commands:

```
(LLDB) setting set platform.use-module-cache false
(LLDB) settings set target.auto-install-main-executeable false
(LLDB) platform select remote-android
(LLDB) platform connect connect://localhost:$PORT
(LLDB) target create -r $EXE_PATH_ON_DEVICE $LOCAL_EXE_PATH
```

If the board is accessed using remote access, run the following commands on the LLDB shell:

```
(LLDB) setting set platform.use-module-cache false
(LLDB) settings set target.auto-install-main-executeable false
(LLDB) platform select remote-android
(LLDB) platform connect connect://remotehost:$PORT
(LLDB) target create -r $EXE_PATH_ON_DEVICE $LOCAL_EXE_PATH
```

For more details on how to debug, refer to Remote Debugging.

7.4 Load a program into LLDB

Select the program to be debugged. `Test.app` is used in the following example:

```
(LLDB) file /projects/sketch/Debug/Test.app
Current executable set to '/projects/sketch/Debug/Test.app' (x86_64)
```

7.5 Set breakpoints

Breakpoints can be set using the `breakpoint` command.

To set a breakpoint in the `Test.c` file at line 13, use the following command:

```
(LLDB) breakpoint set --file Test.c --line 13
```

To break at a function named `foo`, use the following command:

```
(LLDB) breakpoint set --name foo
```

To set breakpoints at multiple functions named `foo` and `bar`, use the following command:

```
(LLDB) breakpoint set --name foo --name bar
```

When setting many breakpoints, it can be useful to have shortened versions of the available commands. The command `breakpoint` can be shortened to `br` using the `alias` command:

```
(LLDB) command alias br breakpoint
```

This alias allows the user to type `br` instead of `breakpoint`.

The LLDB command interpreter automatically matches a unique string to a command name, like autofill. Using this command interpreter, the following two lines of code execute the same command:

```
(LLDB) breakpoint set --name foo
(LLDB) br s -n foo
```

The length of the command lines can be shortened as follows:

- To set a breakpoint in the Test.c file at line 13, use the following code:

  ```
  (LLDB) br s -f Test.c -l 13
  ```

- To break at a function named `foo`, use the following code:

  ```
  (LLDB) br s -n foo
  ```

- To set breakpoints at multiple functions named `foo` and `bar`, use the following code:

  ```
  (LLDB) br s -n foo -n bar
  ```

8. Example code usage

When moving, compiling, linking, and debugging .c files, the process is the same for a physical board and a board that is remotely accessed.

After the board or FVP setup has been completed, the `helloworld` binary file can be executed directly from the AP UART terminal using the following command:

```
./morello-helloworld
```

Check for the following console message in `uart0.log` of the AP UART (ttyUSBx+7):

```
/# ./morello-helloworld
Hello from Morello!!
```

The following two commands can also be used to test the boards capabilities:

```
./morello-stack
./morello-heap
```

`./morello-stack` demonstrates that when an array with pre-determined bounds is attempted to be writren to outside of the boundaries, the board will return a segmentation fault (or SEGFAULT). `./morello-heap` allows the user to write a string to an array that is 11 characters long. If the input string is longer than 11 characters, it will return a segmentation fault.

9. Related information

The following resources are related to material in this guide:

- Arm Architecture Reference Manual Supplement - Morello for A-profile Architecture
- Morello Platform Open Source Software
- Morello User Guide

10. Next steps

In this guide, you learned about the Morello Development Platform, how to build a Morello software stack, and set up a Morello development board. As a next step, you can learn more about the Morello Project on the Morello Program site. This site contains reference manuals, tools, videos, and a user forum for the Morello platform.