Microchip U-Boot:

Howto Guide

for release 2023.06
14. December 2023

1 U-Boot

This document conserns U-Boot and has been written with release 2023.06 in mind.

Chapter 2 is about SparX5 / arm64 (64bit), and chapter 3 is about lan966x / arm (32bit). The chapter are as identical as they can be. The main difference is, that lan966x has secure boot which means it must be wrapped into ARM Trusted Firmware[1] (TF-A).

It is assumed, that Docker is installed. This is described in the BSP documentation[2]. You can also run the script `mchp-get-docker.sh`[3] which has automated the process. The documentation is in the file.

The command

```
$ dr make ...
```

means that `make` is run in the docker environment and all the necessary tools is available. If you have installed the tools on your build machine and do not want to run docker then you just have to leave out the `dr` part.

[1] `https://www.trustedfirmware.org/projects/tf-a`

[2] `http://mscc-ent-open-source.s3-eu-west-1.amazonaws.com/public_root/bsp/mscc-brsdk-doc-latest.html`

[3] `https://github.com/microchip-ung/misc-scripts`. The instructions are in the script, but you only need to run step 1 and 2 as described in the script, since that will install docker.

Docker has been tested on ubuntu 20.04 and ubuntu 22.04.2 LTS. If it does not work on your build machine, then goto the BSP documentation for details.

2 U-Boot for SparX5 / arm64

- Section 2.1 is about howto find the U-Boot image in the BSP binary. If you just need to program/update the default U-Boot image on an evaluation board, then this is where to look.
- Section 2.2 describe howto build U-Boot from the BSP source with buildroot.
- Section 2.3 describe howto make your own U-Boot configuration (defconfig) and build it within buildroot
- Section 2.4 describe howto build U-Boot outside of buildroot. If you want to do U-Boot development, then this is a more feasible methode than doing it inside buildroot as describe in section 2.3.
- Section 2.5-2.6 describe howto build a DDR parameter file.
- Section 2.7 describe howto program/update U-Boot when a working U-Boot is in the NOR flash already.
- Section 2.8 short on U-Boot environment variables.

It is assumed that the binary BSP and toolchain is installed[1].

2.1 U-Boot in the BSP binary

The BSP provide the standard U-Boot binary. For SparX5 the `mscc-brsdk-arm64-2023.06.tar.gz`[2] need to be installed and installed with

```
$ sudo mkdir -p /opt/mscc
$ sudo tar xzf mscc-brsdk-arm64-2023.06.tar.gz -C /opt/mscc
```

The easiest way to find the U-Boot images is to run

```
$ find /opt/mscc/mscc-brsdk-arm64-2023.06 -name "u-boot*"
```

In this specific case the following U-Boot images are listed[3]

```
u-boot-sparx5_pcb13x_emmc.bin
u-boot-sparx5_pcb13x_nand.bin
```

Depending on whether the board in question has NAND or eMMC flash, one or the other need to be used.

[1] If it is not then run the commands

```
$ sudo mkdir -p /opt/mscc
$ wget http://mscc-ent-open-source.s3-eu-west-1.amazonaws.com/public_root/bsp/\
  mscc-brsdk-arm64-2023.06.tar.gz
$ sudo tar xzf mscc-brsdk-arm64-2023.06.tar.gz -C /opt/mscc
$ wget http://mscc-ent-open-source.s3-eu-west-1.amazonaws.com/public_root/toolchain/\
  mscc-toolchain-bin-2023.02.101.tar.gz
$ sudo tar xzf mscc-toolchain-bin-2023.02.101.tar.gz -C /opt/mscc
```

This is described in the BSP documentation, but for quick reference we state it here also.

[2] Download this file from `http://mscc-ent-open-source.s3-website-eu-west-1.amazonaws.com/?prefix=public_root/bsp/`

[3] Others are also listed, but they are not for the evaluations boards.

2.2 U-Boot in BSP source

U-Boot can be build from the BSP source. In case changes has to be made, then this is the way to go. Download the `mscc-brsdk-source-2023.06.tar.gz`[4] file and run the commands

```
$ tar xzf mscc-brsdk-source-2023.06.tar.gz
$ cd mscc-brsdk-source-2023.06
```

Now that you are in the root of the source tree, you can build

```
$ dr make BR2_EXTERNAL=./external O=output/my-uboot arm64_bootloaders_defconfig
$ dr make BR2_EXTERNAL=./external O=output/my-uboot
```

The first command will configure what to build, and the second will build it. When done, the result is in `output/my-uboot/images`. As you can see both U-Boot images, i.e. one for NAND and another for eMMC, have been build. The configuration of each of these builds are in `output/my-uboot/build/mscc-muboot-xxx/configs`[5] in `mscc_sparx5_pcb13x_nand_defconfig` and `mscc_sparx5_pcb13x_emmc_defconfig`. The reason both are build is found in `external/boot/mscc-muboot/mscc-muboot.mk` where is can be seen, that `MSCC_MUBOOT_BUILD_CMDS` will build all the U-Boots listed in `BR2_MSCC_MUBOOT_TARGETS`. If you run

```
$ dr make BR2_EXTERNAL=./external O=output/my-uboot menuconfig
```

and do a search for `MUBOOT`, you can see, that in this particular case we have

```
BR2_MSCC_MUBOOT_TARGETS = mscc_sparx5_pcb13x_emmc   mscc_sparx5_pcb13x_nand
```

You can edit this entry if you want to build a different set of U-Boots. Probably you only want one, so remove the one you do not want.

2.3 Changing U-Boot configuration under buildroot

If you e.g. want to use `mscc_sparx5_pcb13x_emmc` but want to make some changes, one way is to make a copy like

```
$ cd output/my-uboot/build/mscc-muboot-xxx/configs
$ cp mscc_sparx5_pcb13x_emmc_defconfig my-uboot_defconfig
```

and then set

```
BR2_MSCC_MUBOOT_TARGETS = my-uboot
```

Then build U-Boot with the commands discussed earlier. If you now want to change the configuration then run

```
$ cd output/my-uboot/build/mscc-muboot-xxx
$ make menuconfig
$ cp .config configs/my-uboot_defconfig
```

[4] Download this file from `http://mscc-ent-open-source.s3-website-eu-west-1.amazonaws.com/?prefix=public_root/bsp/`

[5] For release 2023.06, the `xxx` is `45c55b...` and you can find it in `external/configs/arm64_bootloaders_defconfig` variable `BR2_MSCC_MUBOOT_VERSION`. If you look in `dl/mscc-muboot/` you can see, that there are more than one file that only differ in this `xxx` part. That is because the source code cover all targets, and here we happen only to be intereseted in SparX5.

and then run the build commands:

```
$ make BR2_EXTERNAL=./external O=output/my-uboot mscc-muboot-rebuild
$ make BR2_EXTERNAL=./external O=output/my-uboot
```

The first forces U-Boot to be rebuild[6] the U-Boots listed in `BR2_MSCC_MUBOOT_TARGETS`. So if the related xxx_defconfig(s) has changed, then that will take effect. And also if the source code in `output/my-uboot/build/mscc-muboot-xxx` has changed.

The second command finish up the entire build as before.

You need to make the copy since `MSCC_MUBOOT_BUILD_CMDS` will clear the build folder for mscc-muboot-xxx and start all over from the defconfig.

2.4 Build U-Boot without buildroot

If you have the `mscc-brsdk-source-2023.06` code as before and do[7]

```
$ cd mscc-brsdk-source-2023.06/dl/mscc-muboot
$ tar xzf mscc-brsdk-source-2023.06/dl/mscc-muboot/mscc-muboot-xxx.tar.gz
$ cd mscc-muboot-xxx
$ ./env.sh make mscc_sparx5_pcb13x_emmc_defconfig
$ ./env.sh make
```

Then you have `u-boot.bin`.

The defconfig files, in this case `mscc_sparx5_pcb13x_emmc_defconfig`, are located under `configs`, and in this it can be seen what device-tree go into the build In this particular case we have

```
CONFIG_OF_LIST="sparx5_pcb134_emmc sparx5_pcb135_emmc"
CONFIG_DEFAULT_DEVICE_TREE="sparx5_pcb135_emmc"
```

The device-trees are in `arch/arm/dts`. Just append a `.dts` to the name in `CONFIG_OF_LIST` to find the relecant device-tree, like e.g. `sparx5_pcb135_emmc.dts`.

So this is where you change device-tree if necessary. This become relevant when changing the DDR configuration in the next sections.

Code that is specific for sparx5 can be found in

```
arch/arm/mach-sparx5/
board/mscc/sparx5/
```

For example the `ddr_init()` is defined in `ddr_umctl.c` first location and called from `dram.c` in the second - the board folder.

Since `CONFIG_OF_LIST` contains a number of device trees, U-Boot must decide which one to use. Since this is a board specific thing, a guess is that the code is in board folder. In

[6] In the buidroot manual `https://buildroot.org/downloads/manual/manual.html` describe how the rebuild target works. For our muboot (mulitple U-Boot) we have implemented this rebuild target.

[7] The `env.sh` is

```
#!/bin/sh
ARCH=arm64 CROSS_COMPILE=/opt/mscc/mscc-brsdk-arm64-2023.06/arm64-armv8_a-linux-gnu/\
xstax/release/x86_64-linux/usr/bin/aarch64-linux- $@
```

so it just setup the `ARCH` and `CROSS_COMPILE` environment variables for `make` so that is can use the correct cross compiler.

sparx5.c you will find the funciton embedded_dtb_select(), and it is enabled if CONFIG_DTB_RESELECT is defined. If you check the .config file, you will see that it is. In this function you can see, that it detect of U-Boot is running on pcb134 or pcb135.

2.5 DDR configuration

U-Boot has the resonsibility of configuring the DDR memory. This is done via device-trees. In the root mscc-muboot-xxx of the U-Boot source tree, the .dts files are in arch/arm/dts. List the relevant files with

```
$ ls *sparx5*
```

For example the sparx5_pcb135_emmc.dts is used for the SparX5 evaluation board PCB135 w. eMMC. This file will in turn include mscc,sparx5_ddr3.dtsi that with &ddr provide the DDR configuration.

2.6 DDR tool

The DDR tool[8] is used to generate the dtsi file with the &ddr configuration. As an example, we can generate the a dtsi with the same &ddr configuration as in the mscc,sparx5_ddr3.dtsi

```
$ ./scripts/gen_cfg.rb -f devicetree configs/profiles/pcb135_ddr3.yaml \
  > my-pcb135_ddr3_config.dtsi
```

You can diff the file you generated with the one in the release

```
diff my-pcb135_ddr3_config.dtsi ~/p2/bsp/mscc-brsdk-source-2023.06/dl/\
     mscc-muboot/mscc-muboot-45xx/arch/arm/dts/mscc,sparx5_ddr3.dtsi
8c8
<       microchip,mem-name = "pcb135_ddr3 2023-09-21-13:23:10 236061e";
---
>       microchip,mem-name = "pcb135_ddr3 2023-05-11-12:52:54 00e206508b93";
53,55c53,55
<               0x00181818      /* addrmap1 */
<               0x00000000      /* addrmap2 */
<               0x00000000      /* addrmap3 */
---
>               0x00040401      /* addrmap1 */
>               0x01010100      /* addrmap2 */
>               0x13131303      /* addrmap3 */
```

From this you can see, that in the ddr-umctl source tree, I happen to be on git tag 236061e which can be verify with

```
$ git log
```

[8] https://github.com/microchip-ung/ddr-umctl.

It is checked out with

```
$ git clone https://github.com/microchip-ung/ddr-umctl
$ cd ddr-umctl
```

The `mscc,sparx5_ddr3.dtsi` in the BSP source tree has been generated with `00e206508b93`. If I want to generate the same thing, then I can say

```
$ git checkout 00e206508b93
$ ./scripts/gen_cfg.rb -f devicetree configs/profiles/pcb135_ddr3.yaml \
  > my-pcb135_ddr3_config.dtsi
```

We then get the above diff

```
8c8
<       microchip,mem-name = "pcb135_ddr3 2023-09-20-17:16:30 00e2065";
---
>       microchip,mem-name = "pcb135_ddr3 2023-05-11-12:52:54 00e206508b93";
```

since the build time stamp is different, but everything else is the same, as it should be.

In order to build U-Boot with a new DDR configuration, the approach in section 2.4 is suggested.

2.7 Programming U-Boot

If you have a working system, then the new U-Boot can be programmed from the existing by stoppeing the boot process and running the commands

```
m => dhcp <TFTP-IP>:u-boot.bin
m => sf probe
m => sf update ${loadaddr} 0 ${filesize}
```

Now you can start the new U-Boot by power cycling the board or by saying

```
m => reset
```

When the new U-Boot start it is suggested to stop it by pressing a key, and then reset the environment variables

```
m => env default -a
m => save
```

You may want to take a copy of the old enviroment before doing this, if you have configured proprotaiy things.

If the NOR is empty, or you have programmed a broken image, then a flash programmer must be used.

2.8 Running U-Boot

When U-Boot start up, you can stop the boot process by pressing a key. It will at some point print

```
Hit any key to stop autoboot: <count-down>
```

If you do that you get an `m =>` prompt. You can give the command `printenv` in order to print the U-Boot environment variables. The most interesting variable is `bootcmd` which is the one that will be run, if you do not stop the boot process.

3 U-Boot for LAN966x / arm

- Section 3.1 is about howto find the U-Boot image in TF-A. If you just need to program/update the default U-Boot image on an evaluation board, then this is where to look.
- Section 3.2 is about howto find the U-Boot image in the BSP binary and wrap it into TF-A, i.e. generating a Firmware Image Package[1] (`.fip`).
- Section 3.3 describe howto build U-Boot from the BSP source with buildroot.
- Section 3.4 describe howto make your own U-Boot configuration (defconfig) and build it withing buildroot
- Section 3.5 describe howto build U-Boot outside of buildroot. If you want to do U-Boot development, then this is a more feasible methode than doing it inside buildroot as describe in section 2.3.
- Section 3.6 describe howto build a DDR parameter file.
- Section 3.7 describe howto program/update U-Boot when work U-Boot is in the NOR flash already.

It is assumed that the binary BSP and toolchain is installed[2].

3.1 TF-A (TrustedFirmware-A)

A programmable boot image is not provided directly in the BSP, but can be found on `https://github.com/microchip-ung/arm-trusted-firmware`. This procedure is described in `AN4905`[3].

You need `lan966x_b0-release-bl2normal-auth.fip`[4]. The reason U-Boot is not directly in the BSP is, that LAN966x revision B support TF-A[5] so U-Boot need to be wrapped into this secure software. The `.fip` file above is the U-Boot binary wrapped into TF-A as describe in the next section.

[1] `https://trustedfirmware-a.readthedocs.io/en/latest/getting_started/image-terminology.html#firmware-image-package-fip`

[2] If it is not then run the commands

```
$ sudo mkdir -p /opt/mscc
$ wget http://mscc-ent-open-source.s3-eu-west-1.amazonaws.com/public_root/bsp/\
  mscc-brsdk-arm-2023.06.tar.gz
$ sudo tar xzf mscc-brsdk-arm-2023.06.tar.gz -C /opt/mscc
$ wget http://mscc-ent-open-source.s3-eu-west-1.amazonaws.com/public_root/toolchain/\
  mscc-toolchain-bin-2023.02.101.tar.gz
$ sudo tar xzf mscc-toolchain-bin-2023.02.101.tar.gz -C /opt/mscc
```

This is described in the BSP documentation, but for quick reference we state it here also.

[3] This can be found on `https:/www.microchip.com`. Search for LAN9662 and click on Document. The page jumps a little around and then click on Documantation. The first in this list is called "Collateral documents for device". Click on the `Link` on the right side of this item. You need to have access to these documents.

[4] `https://github.com/microchip-ung/arm-trusted-firmware/releases/tag/mchp_v1.0.5`

[5] See `https://www.trustedfirmware.org`

3.2 Building a TF-A/fip image from U-Boot

The BSP provide the U-Boot binary, but without the TF-A wrapping. For LAN966x the `mscc-brsdk-arm-2023.06.tar.gz` need to be installed[6]. The easiest way to find the U-Boot images is to run

```
$ find /opt/mscc/mscc-brsdk-arm-2023.06 -name "u-boot*"
```

In this specific case the following U-Boot image is listed[7]

```
u-boot-mchp_lan966x_evb.bin
```

In order to build a `.fip` image from this, it is suggested to run the `mchp-get-tfa.sh`[8].

If you want to wrap U-Boot into TF-A by yourself, then it is probably because of one or both of the following reasons

- You want to configure U-Boot differently or modify the code. In this case you'll need to build U-Boot. See section 3.3 and 3.4 for two approaches.
- You are using a different DDR RAM setup, and need TF-A to configure it differently. See section 3.6.

If this is not the case, then you may want to take the approach in section 3.1.

3.3 U-Boot in BSP source

What is described here is automated in the script `build-uboot-2023.06-lan966x.sh`[9]. You may find it more instructive to download this script an run it, and then just read through the script to see the steps taken.

U-Boot can be build from the BSP source. Download the `mscc-brsdk-source-2023.06.tar.gz`[10] file and run the commands

```
$ tar xzf mscc-brsdk-source-2023.06.tar.gz
$ cd mscc-brsdk-source-2023.06
```

Now that you are in the root of the source tree, you can build U-Boot in the buildroot framework with

```
$ dr make BR2_EXTERNAL=./external O=output/my-uboot \
                                  arm_bootloaders_lan966x_defconfig
$ dr make BR2_EXTERNAL=./external O=output/my-uboot
```

---

[6] Download this file from `http://mscc-ent-open-source.s3-website-eu-west-1.amazonaws.com/?prefix=public_root/bsp/` and installed with

```
$ sudo mkdir -p /opt/mscc
$ sudo tar xzf mscc-brsdk-arm-2023.06.tar.gz -C /opt/mscc
```

[7] Others are also listed, but they are not for the evaluations board.

[8] The `mchp-get-tfa.sh` can be downloaded from `https://github.com/microchip-ung/misc-scripts`. The instructions are in the file.

[9] Go to `https://github.com/microchip-ung/misc-scripts` and get `build-uboot-2023.06-lan966x.sh`. The documentation is in this script.

[10] Download this file from `http://mscc-ent-open-source.s3-website-eu-west-1.amazonaws.com/?prefix=public_root/bsp/`

The first command will configure what to build, and the second will build it. When done, the results are in `output/my-uboot/images`. In general more than one U-Boot image is build as listed in the `BR2_MSCC_MUBOOT_TARGETS` variable. In this specific case we have

```
BR2_MSCC_MUBOOT_TARGETS  =  mchp_lan966x_evb  mchp_lan966x_svb
```

The configuration of each of these builds are in `output/my-uboot/build/mscc-muboot-xxx/configs`[11] in `mchp_lan966x_evb_defconfig` and `mchp_lan966x_svb_defconfig`. You may want to remove the latter from the list since that is for an internal test target. This is done by running

```
$ make BR2_EXTERNAL=./external O=output/my-uboot menuconfig
```

and search for `MUBOOT`. Then you can see where this variable is and change it.

The reason both are build can be found in `external/boot/mscc-muboot/mscc-muboot.mk` where is can be seen, that `MSCC_MUBOOT_BUILD_CMDS` will build all the U-Boots listed in `BR2_MSCC_MUBOOT_TARGETS`.

3.4 Changing U-Boot configuration under buildroot

If you want to use `mchp_lan966x_evb` but want to make some changes, one way is to make a copy like

```
cd output/my-uboot/build/mscc-muboot-xxx/configs
cp mchp_lan966x_evb_defconfig my-uboot_defconfig
```

and then set

```
BR2_MSCC_MUBOOT_TARGETS = my-uboot
```

Then build uboot with the commands discussed earlier.

If you now want to change the configuration then run

```
$ cd output/my-uboot/build/mscc-muboot-xxx
$ make menuconfig
$ cp .config configs/my-uboot_defconfig
```

and then run the build commands:

```
$ make BR2_EXTERNAL=./external O=output/my-uboot mscc-muboot-rebuild
$ make BR2_EXTERNAL=./external O=output/my-uboot
```

The first forces U-Boot to be rebuild[12] the U-Boots listed in `BR2_MSCC_MUBOOT_TARGETS`. So if the related xxx_defconfig(s) has changed, then that will take effect. And also if the source code in `output/my-uboot/build/mscc-muboot-xxx` has changed.

The second command finish up the entire build as before.

When done, the image is in `output/my-uboot/images`. Then you have wrap it into TF-A as described in 3.2.

[11] For release 2023.06, the `xxx` is `45c55b...` and you can find it in `external/configs/arm_bootloaders_lan966x_defconfig` variable `BR2_MSCC_MUBOOT_VERSION`. If you look in `dl/mscc-muboot/` you can see, that there are more than one file that only differ in this `xxx` part. That is because the source code cover all targets, and here we happen only to be intereseted in SparX5.

[12] In the buidroot manual `https://buildroot.org/downloads/manual/manual.html` describe how the rebuild target works. For our muboot (multiple U-Boot) we have implemented this rebuild target.

3.5 Build U-Boot without buildroot

If you have the `mscc-brsdk-source-2023.06` code as before and do[13]

```
$ cd mscc-brsdk-source-2023.06/dl/mscc-muboot
$ tar xzf mscc-brsdk-source-2023.06/dl/mscc-muboot/mscc-muboot-xxx.tar.gz
$ cd mscc-muboot-xxx
$ ./env.sh make mchp_lan966x_evb_defconfig
$ ./env.sh make
```

Then you have `u-boot.bin`, which you need to wrap into TF-A as described in 3.2.

3.6 DDR configuration

If you have done what is explained in section 3.2, i.e. have run the `mchp-get-tfa.sh` script, then you can go into the TF-A code.

In the file `plat/microchip/lan966x/common/common.mk` the two files related to DDR configuration are

```
ddr_umctl.c
lan966x_ddr_config.c
```

In the first file, the `lan966x_ddr_init()` function can be found, and that is the entry point for setting up the DDR. It uses the variable `lan966x_ddr_config` which come from the second file. It contains the parameters.

This parameter file `lan966x_ddr_config.c` is generated with the DDR tool.

The DDR tool[14] is used to generate this parameter file. and run it with

```
$ ./scripts/gen_cfg.rb -f source configs/profiles/lan966x.yaml \
  > my-lan966x_ddr_config.c
```

You can compare the configuration that you have generated with the one in the TFA source tree:

```
$ diff my-lan966x_ddr_config.c ../sw-arm-trusted-firmware/plat/microchip/\
  lan966x/common/lan966x_ddr_config.c
```

In my case I get:

```
11c11
<                .name = "lan966x 2023-09-20-16:28:01 236061e",
---
>                .name = "lan966x 2023-05-11-12:32:23 00e206508b93",
```

[13] The `env.sh` is

```
#!/bin/sh
ARCH=arm CROSS_COMPILE=/opt/mscc/mscc-brsdk-arm-2023.06/arm-cortex_a8-linux-gnu/xstax/\
release/x86_64-linux/usr/bin/arm-linux- $@
```

so it just setup the `ARCH` and `CROSS_COMPILE` environment variables for `make` so that is can use the correct cross compiler.

[14] `https://github.com/microchip-ung/ddr-umctl`.

It is checked out with

```
$ git clone https://github.com/microchip-ung/ddr-umctl
$ cd ddr-umctl
```

```
13c13
<               .size = 0x38000000,
---
>               .size = 0x40000000,
22c22
<               .ecccfg0 = 0x003f7f44,
---
>               .ecccfg0 = 0x003f7f40,
```

From this you can see, that in the `ddr-umctl` source tree, I happen to be on git tag `236061e` which can be verify with

```
 $ git log
```

The `lan966x_ddr_config.c` in the TFA source tree has been generated with `00e206508b93`. If I want to generate the same thing, then I can say

```
 $ git checkout 00e206508b93
 $ ./scripts/gen_cfg.rb -f source configs/profiles/lan966x.yaml \
   > my-lan966x_ddr_config.c
```

We then get the above diff

```
11c11
<               .name = "lan966x 2023-09-20-16:37:06 00e2065",
---
>               .name = "lan966x 2023-05-11-12:32:23 00e206508b93",
```

since the build time stamp is different, but everything else is the same, as it should be.

3.7 Programming U-Boot

If you have a working system, then the new U-Boot can be programmed from the existing by stoppeing the boot process and running the commands

```
 m => dhcp <TFTP-IP>:u-boot.bin
 m => sf probe
 m => sf update ${loadaddr} 0 ${filesize}
```

Now you can start the new U-Boot by power cycling the board or by saying

```
 m => reset
```

When the new U-Boot start it is suggested to stop it by pressing a key, and then reset the environment variables

```
 m => env default -a
 m => save
```

You may want to take a copy of the old enviroment before doing this, if you have configured proprotaiy things.

If the NOR is empty, or you have programmed a broken image, then a flash programmer can be used.

If you have access to FlexCom3 (FC3) serial port, then the `fwu.html`[15] tool can be used. This will e.g. work on EVB-LAN9662 / UNG8291 board. In the LAN9662 datasheet look

[15] `https://github.com/microchip-ung/arm-trusted-firmware/releases/download/mchp_v1.0.5/fwu.html`

at table 2-8 or search for `BOOTLOADER STRAP DESCRIPTION`. Set the vcore to "Monitor at FC3" i.e. `vcore[3:0]=1000`.

If you do that, disconnect any terminalt to the board and reboot it, then you can connect `fwu.html` to the board.

Run `fwu.html` with Chrome and java enabled.

- When it starts press `Connect device` and then select `MCP2221` from the menu and press `Connect`.
- Now you should be presented a new page. Press `Download BL2U`. This will upload a small program to the LAN9662.
- When it is done and started, you will presented a new page. Press `Choose File` and e.g. select a `.fip` image, e.g. `lan966x_b0-release-bl2normal-auth.fip` from section 3.1.
- Press `Upload file` This will upload the `lan966x_b0-release-bl2normal-auth.fip` to the LAN9662.
- To the right of the `Write FIP Image` button change the dropdown box to "NOR flash". and then press the `Write FIP Image` button.
- Change the `vcore[3:0]=0001`, i.e. boot from NOR and boot the device.

As mentioned earlier, you should default the environemt variables when a new U-Boot is programmed, i.e. run

```
m => env default -a
m => save
```

3.8 Running U-Boot

When U-Boot start up, you can stop the boot process by pressing a key. It will at some point print

```
Hit any key to stop autoboot: <count-down>
```

If you do that you get an `m =>` prompt. You can give the command `printenv` in order to print the U-Boot environment variables. The most interesting variable is `bootcmd` which is the one that will be run, if you do not stop the boot process.