byteWIKI

Release 1.0

Johannes Böhm and others

Apr 11, 2024

ABOUT

Welcome to the

»byteWIKI«

CHAPTER

ONE

ABOUT THE COMPANY

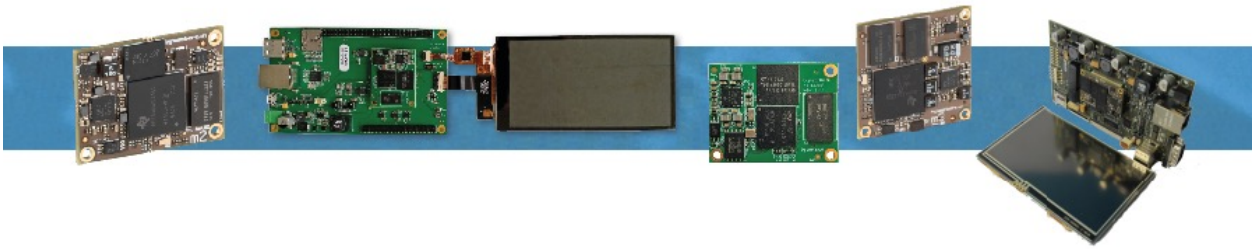
bytes at work is a modern Swiss Technology company specialized in industrial computing. Our focus lies on the development of hardware and embedded software, as well as customizing Linux systems. The entire development life cycle takes place in-house with transparent project management and customer involvement. This significantly reduces both development time and development costs.

We have years of experience in developing coordinated hardware and software solutions – from the prototype to the final product. We make your system usable end-to-end for your needs.

1.1 Our philosophy

Hardware and software for industrial computers have to fulfill an immense range of demanding challenges. They are used in completely different areas of industries and they have to be able to adapt unique and specific tasks. Our employees pay particular attention to each and every customer. That is why our products and services meet and even exceed our customers expectations.

We from bytes at work are aware that the current persistent industrial development also has its darker side. This is our motivation to be exemplary in terms of use of resources. No wonder that unconditional reliability, long service life and low power consumption are main features of all our products.



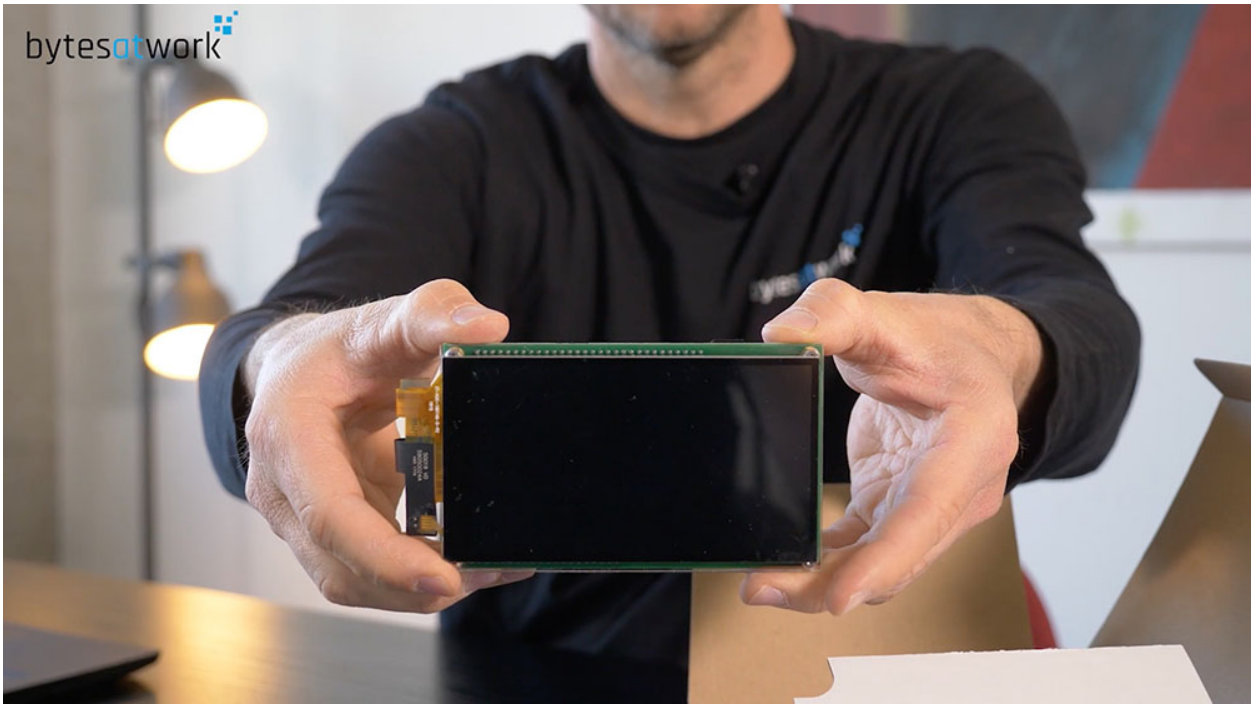
CHAPTER

TWO

UNBOXING BYTEDEVKIT STM32MP1

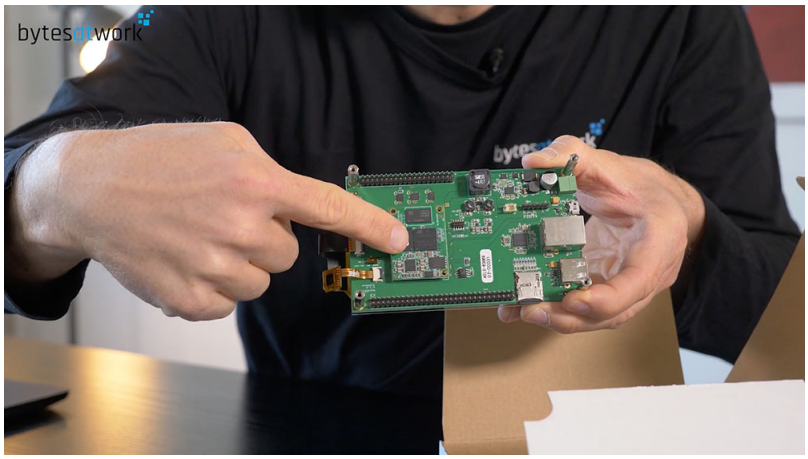
This guide delivers new users a brief overview of the package content and the functions of our byteDEVKIT STM32MP1. When unboxing you should find the following components:

- The byteDEVKIT STM32MP1 with a 5-inch touchscreen display

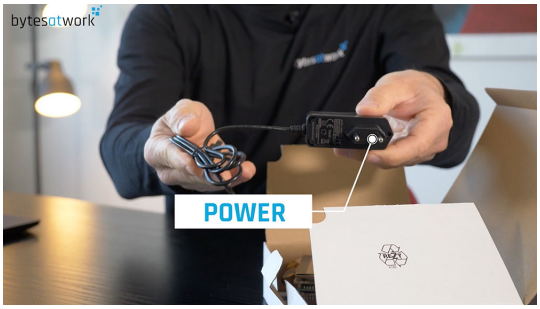


- The SOM STM32MP1x

Note: The SOM STM32MP1x is already connected with the byteDEVKIT STM32MP1.

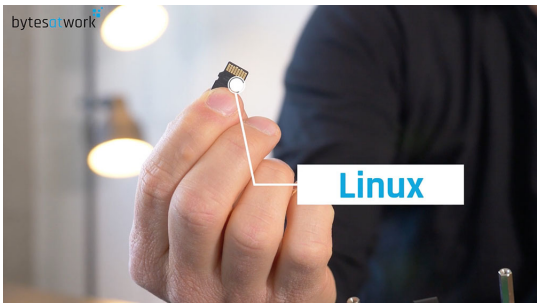


- The power supply for the byteDEVKIT STM32MP1



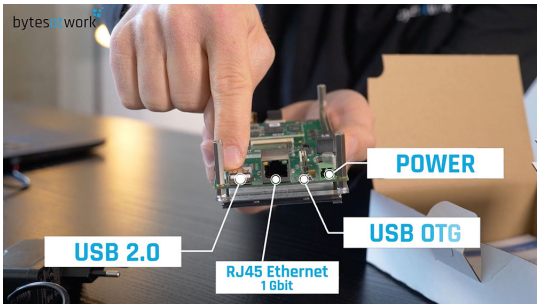
- The USB serial cable for the byteDEVKIT STM32MP1

- micro-SD card with preinstalled Linux

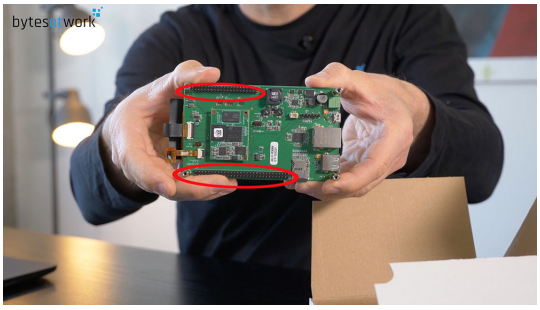


2.1 Technical overview byteDEVKIT STM32MP1

- The byteDEVKIT STM32MP1 offers the following connectors on the front side:
  - USB 2.0
  - RJ45 Ethernet 1 Gbit
  - USB OTG
  - Power connector



- You find the extension on the backside. The byteDEVKIT STM32MP1 offers:
  - 40 pin header compatible for the **rasperry pi**
  - 60 pin header with all the needed signals: **I2C, SPI, CAN, UART, I2S, LDC, GPIO and PWM**



- The micro-SD card slot contains a micro-SD card with preinstalled Linux OS:

Note: The micro-SD card is already slotted to the byteDEVKIT STM32MP1.

2.2 Unboxing Video Tutorial

CHAPTER

THREE

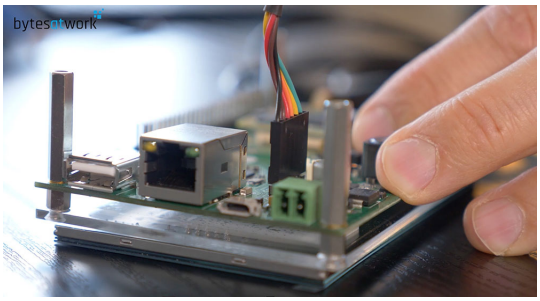
FIRST START BYTEDEVKIT STM32MP1

This guide helps with the first start of the byteDEVKIT STM32MP1:

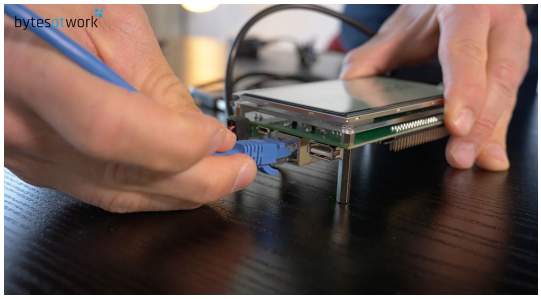
3.1 Connecting the Hardware and first Booting

- Prepare the USB serial cable for connection
- Locate the black cable of the serial connector.

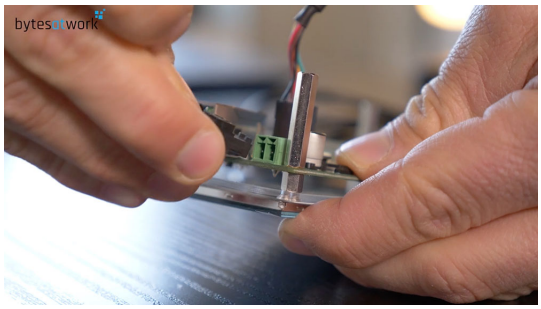
Caution: Connect the serial cable to the byteDEVKIT STM32MP1 as shown. The **black cable** must point towards the USB OTG connector.



- Connect the USB connector with USB port of your computer or laptop.
- Connect the ethernet RJ45 with the byteDEVKIT STM32MP1.

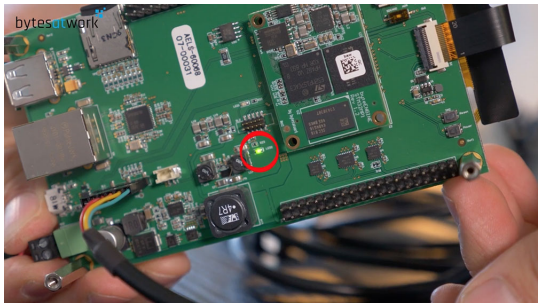


- Plug in the power socket.
- Connect the power supply cable to the power slot of the byteDEVKIT STM32MP1.



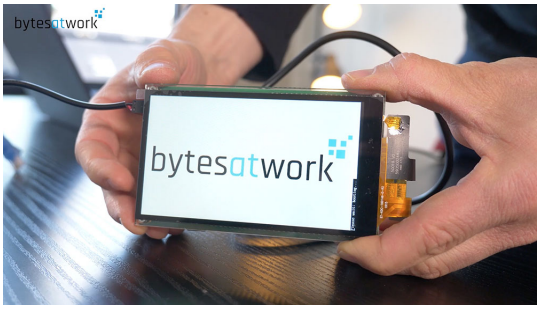
- A green LED on the backside of the byteDEVKIT STM32MP1 indicates the status of the power supply.

Attention: Your byteDEVKIT STM32MP1 is powered up, when the green LED lights up. If the LED doesn´t light up, check the connection of the power socket.



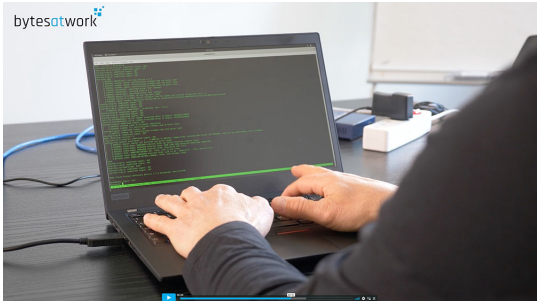
- The 5-inch touchscreen display shows the bytes at work-logo when booting.

Hint: The booting procedure will take a few seconds.



- Now you can access the byteDEVKIT STM32MP1 with your laptop.

Hint: For further information refer to: "Bring-up_byteDEVKIT_STM32MP1".



CHAPTER

FOUR

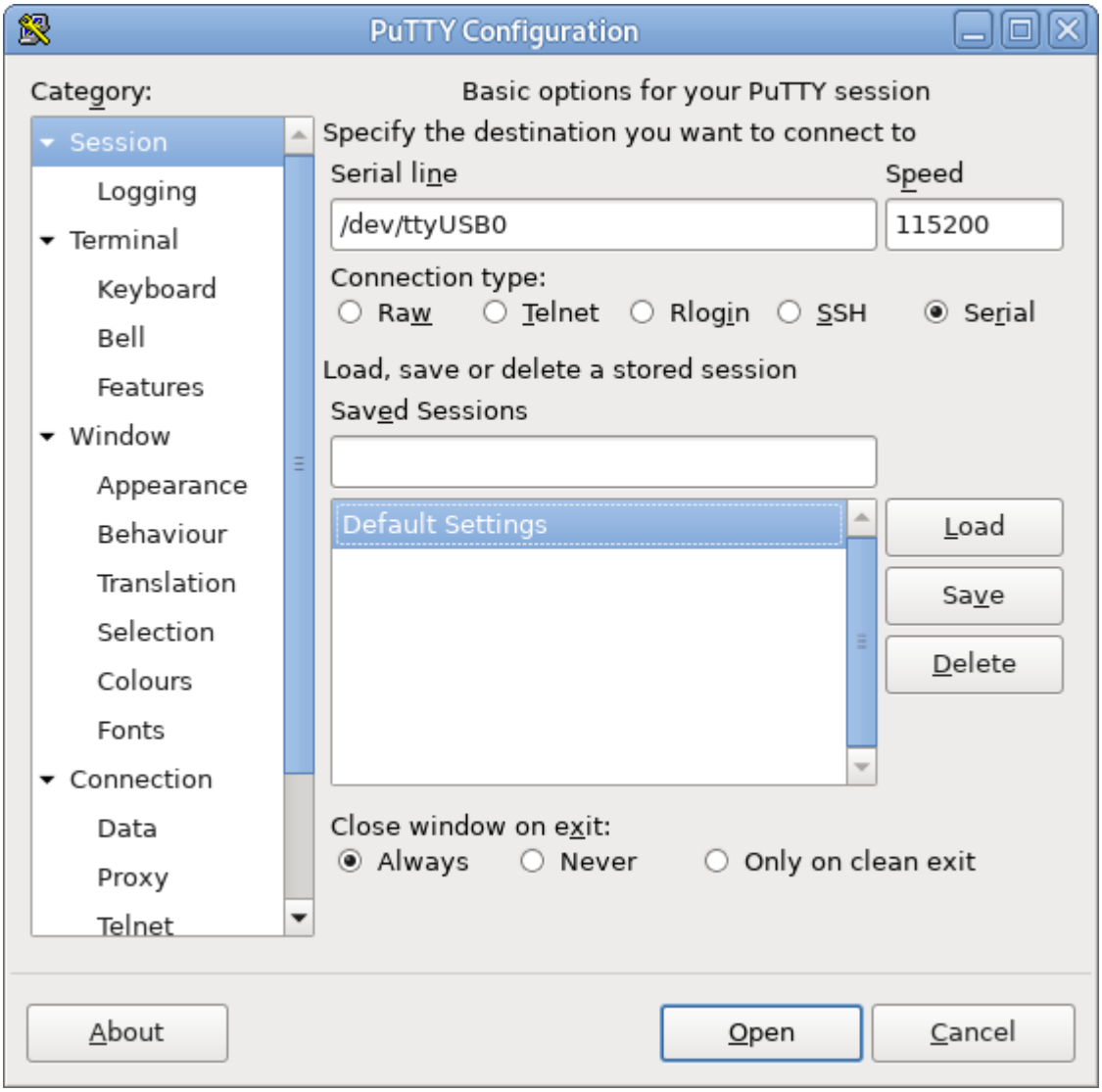
BRING-UP BYTEDEVKIT STM32MP1

4.1 How do I connect to byteDEVKIT using the serial console?

- **Use the serial port to connect the byteDEVKIT STM32MP1:**
  - Connect the debug cable with the byteDEVKIT STM32MP1 and your computer/laptop
  - Start a serial communication program on your computer/laptop (‹putty›, ‹minicom› or something else)
  - Set to 115200, 8N1, no flow control
  - login with: **user: “root”** and **password: “rootme”**

4.1.1 LINUX

- Start PuTTY

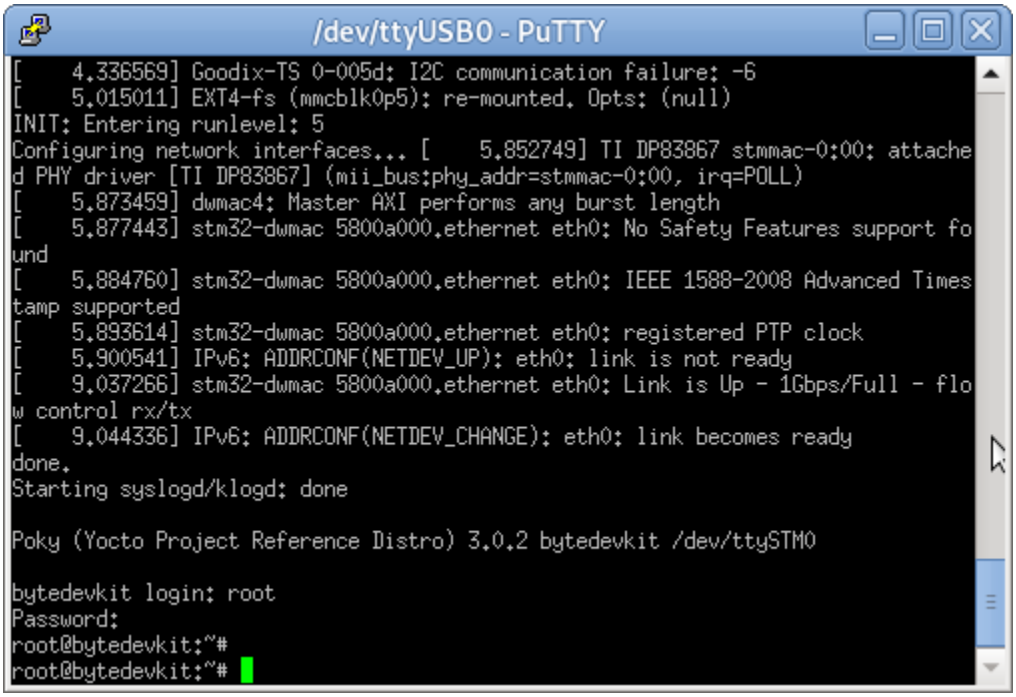


- Click “Serial”
- Change “Serial line” to “/dev/ttyUSB0”
- Change “Speed” to 115200
- Navigate to “Serial” in the menu “Connection”

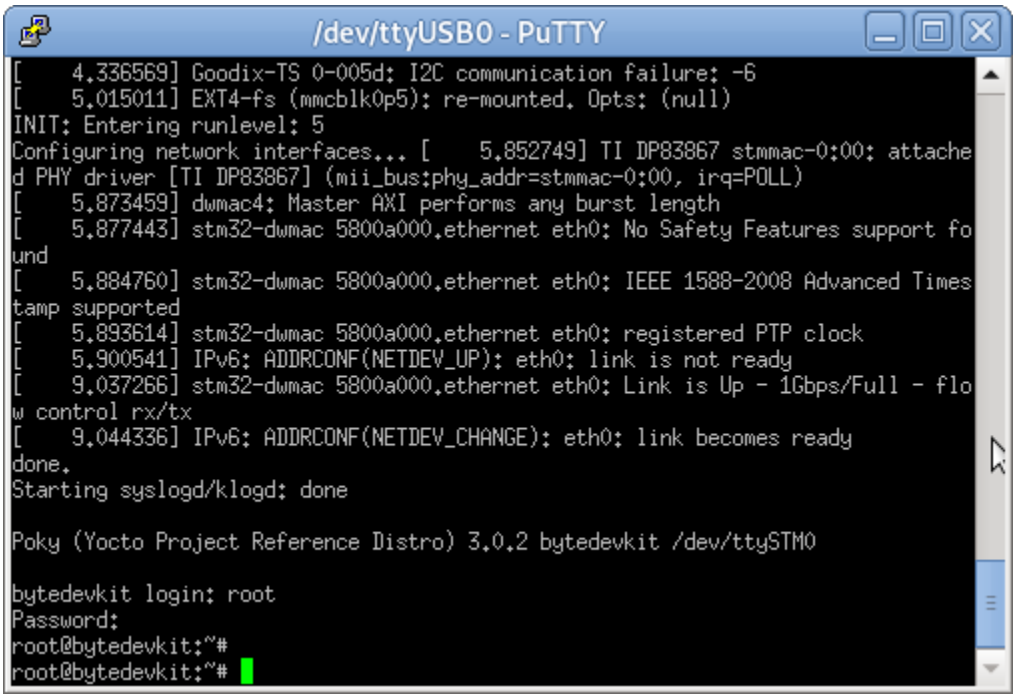
Hint: make sure you have Data bits set to 8, Stop bits set to 1, Parity to None, Flow control to None

- Click “Open”

- Power up the byteDEVKIT STM32MP1



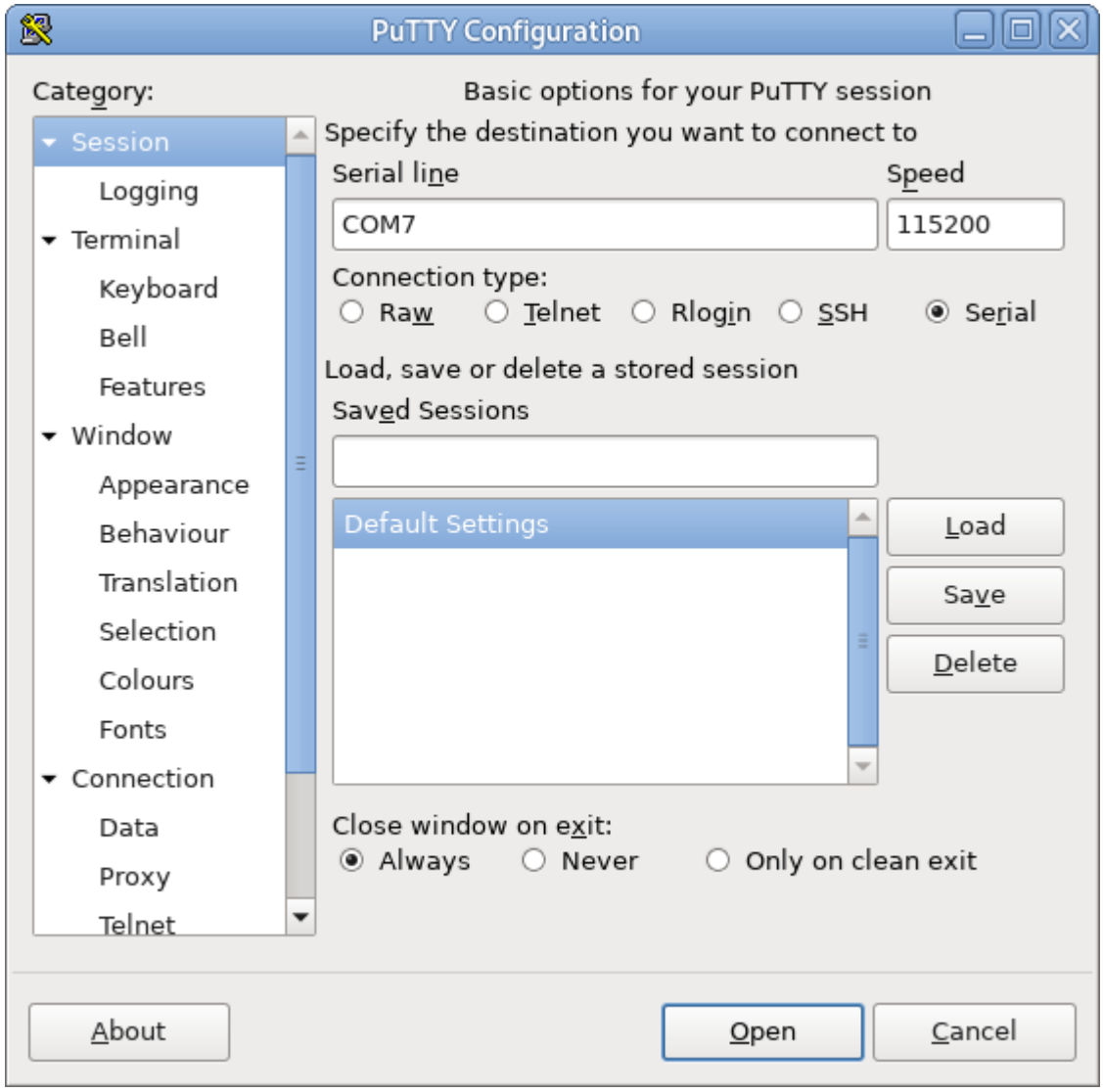
- Once the login prompt appears, login with user “root” and password “rootme”



Note: You are now succesfully connected to the byteDEVKIT STM32MP1

4.1.2 WINDOWS

- Connect the USB serial adapter to the computer
- Windows installs the driver automatically (if the windows doesn´t install the driver reconnect the serial adapter cable)
- Open device manager and navigate to “Ports (COM & LPT)”
- The serial adapter shows up in the device tree: “Prolific USB-to-Serial Comm Port (COM7)”
- “COM7” is your serial port
- Install a serial terminal application, e.g. PuTTY (version 0.59 and newer) *https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html*
- Start PuTTY



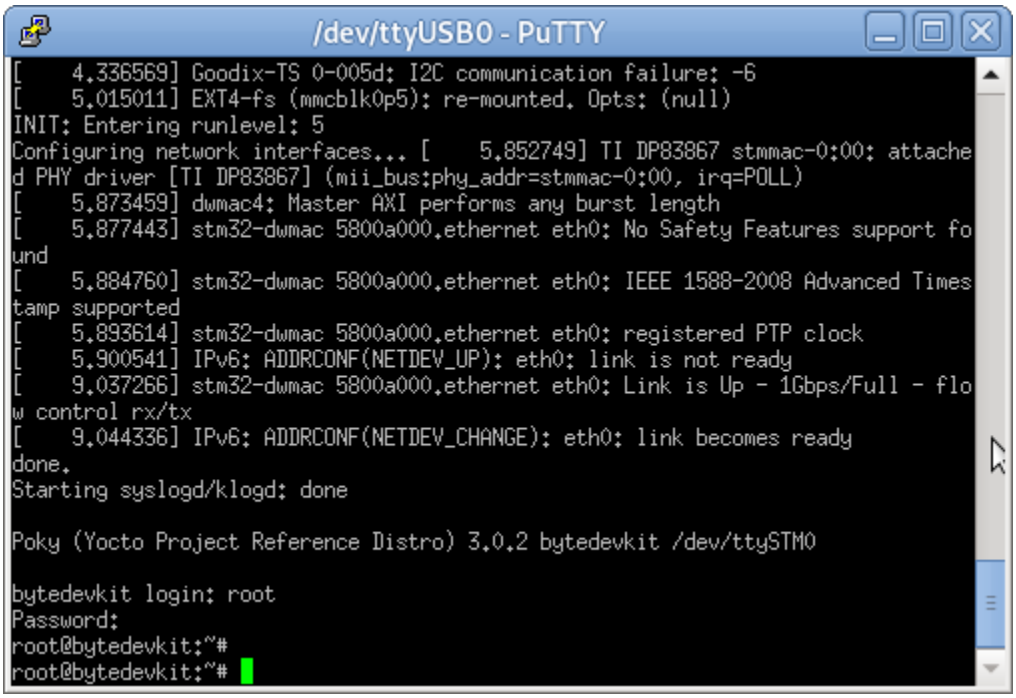
- Click “Serial”
- Change “Serial line” to serial port you found in device manager

- Change “Speed” to 115200
- Navigate to “Serial” in the menu “Connection”

Hint: make sure you have Data bits set to 8, Stop bits set to 1, Parity to None, Flow control to None

- Click “Open”

Power up the byteDEVKIT STM32MP1



Once the login prompt appears, login with user “root” and password “rootme”

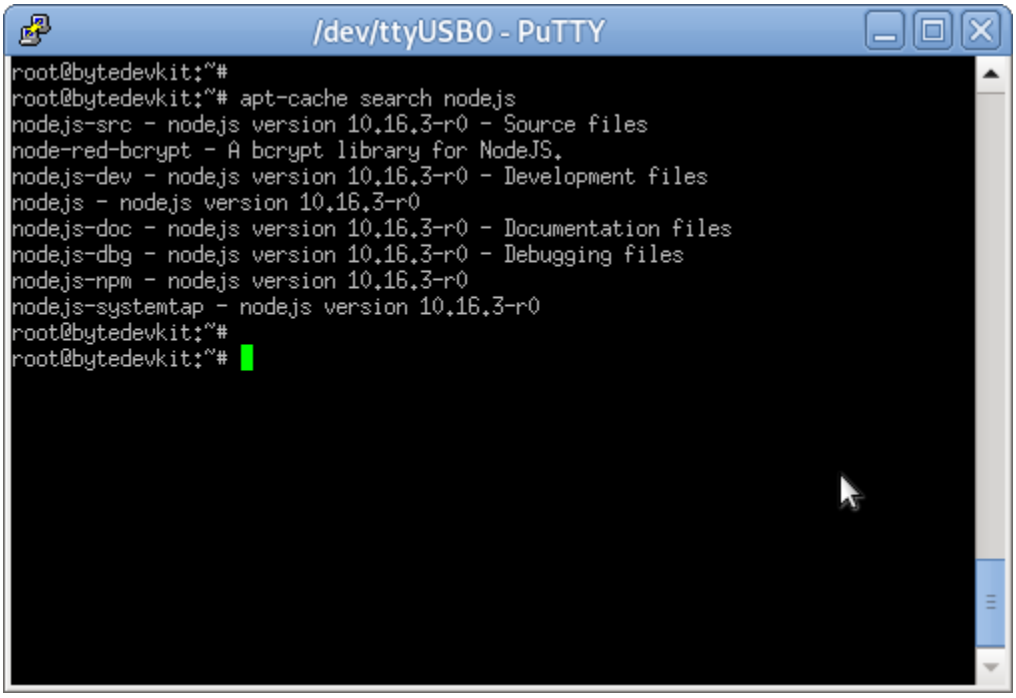
Note: You are now successfully connected to the byteDEVKIT STM32MP1

4.2 How to install additional software using apt

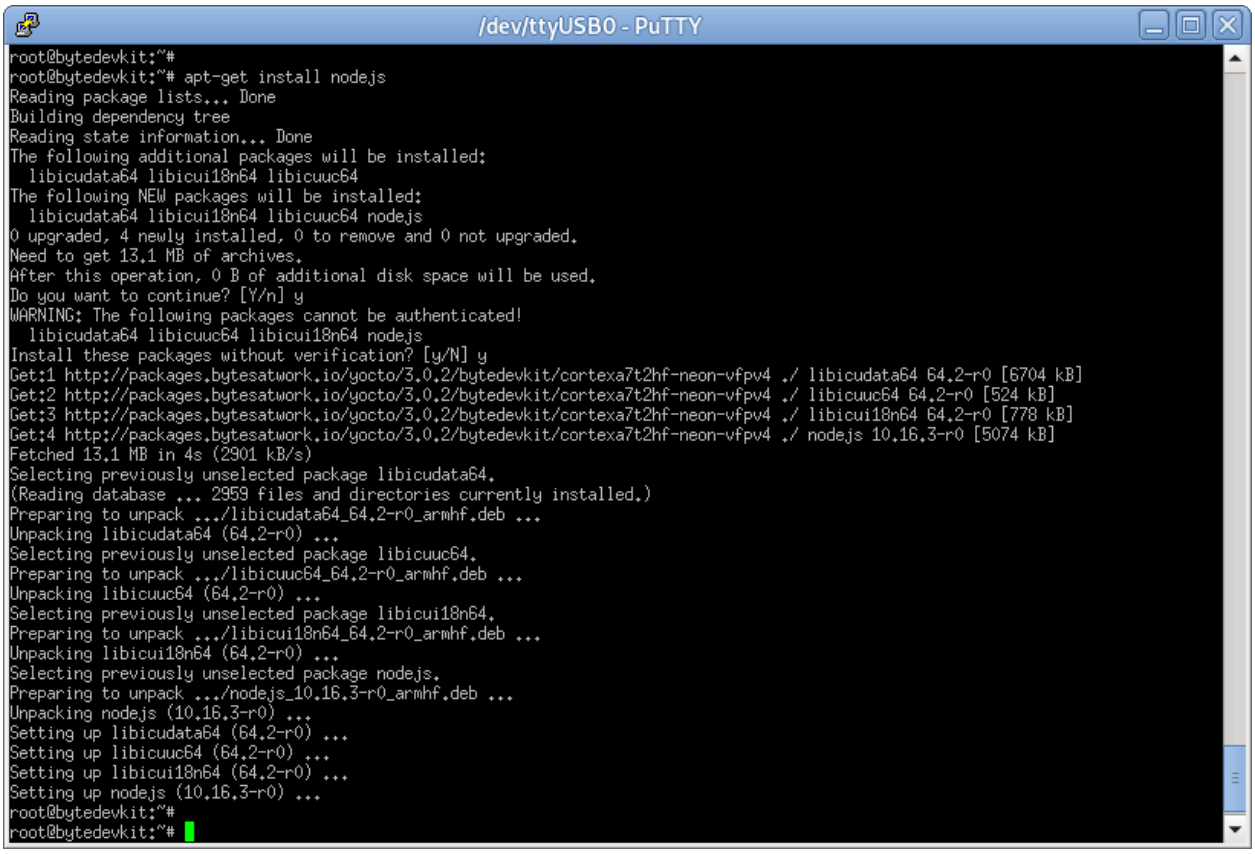
Hint: Follow the link for additional information about "apt": https://help.ubuntu.com/community/AptGet/Howto

Note: byteDEVKIT < V1.2: If you are using a LAN switch (hub) with no 1 GbE support see *STM32MP1 Ethernet*.

1. Connect the embedded device's ethernet to your LAN
2. Run: *apt-get update*
3. Run: *apt-cache search <software component>* to search for available packages e.g.: *apt-cache search nodejs*



5. Run: *apt-get install <software component>* to install additional software e.g.: *apt-get install nodejs*




bytesatwork

CHAPTER

FIVE

SOFTWARE DEVELOPMENT

The entire development life cycle is done in-house with transparent project management and customer involvement. We have proven experience in a wide range of industries, including industrial automation and custom solutions for consumer electronics. This section helps you step by step initiating the software development process.

Current software platforms and images are available directly under this navigation item. Older versions are found in the Archive.

Hint: `bytePANEL` has become outdated. Current software platforms will only support byteDEVKIT with am335x or stm32mp1 modules.

5.1 byteDEVKIT-am62x (Yocto 4.0)

5.1.1 Downloads

SD card image

Download	Checksum (SHA256)
bytesatwork-minimal-image-bytedevkit-am62x.wic.gz	0747dfb463edad01cd3bf7985bed602e717b1dfa2f09258ed6860c37b57c67cb
bytesatwork-minimal-image-bytedevkit-am62x.wic.bmap	3577b6bc71600903fcba120629a50f5595e25f9ceb63d6301efb3f46d3848115

Hint: Updating from an older image? You can update your older image by using: `apt-get update` and `apt-get upgrade`.

1. check for new version in the table above
2. edit `/etc/apt/sources.list` and point to the new package feed
3. run `apt-get update; apt-get upgrade`

As the yocto framework is based on several packages from various projects or suppliers, it is not guaranteed that an incremental upgrade by `apt-get upgrade` works automatically. Some manual adjustments might be needed.

Toolchain

Download	Checksum (SHA256)
poky-bytesatwork-glibc-x86_64-bytesatwork-minimal-image-aarch64-bytedevkit-am62x-toolchain-4.0.9.sh	a5e9e6706cbff94fb3e31b41e948cbe1665cabca457e1bf337c59d45

U-Boot

Description	Download	Checksum (SHA256)
SPL R5F	tiboot3.bin	53481b110634d711c43c47db40b2cfbce8b993cc6b63892d204d6563f35ea690
SPL A53	tispl.bin	ee581879fba5a58dc872395eda734e5fe4d5bfdc4a4eb48b7e09b21991827908
U-Boot A53	u-boot.img	7c14d88c61772c3bb36d4d1441eee46f3d64f4d5d5abbb1b0ba2a264247a20aa

5.1.2 Image

How do you flash the image?

Attention:

- You need a microSD card with **at least 8GB** capacity.
- **All existing data** on the microSD card will be lost.
- **Do not format** the microSD card before flashing.

Windows

1. Unzip the file `bytesatwork-minimal-image-bytedevkit-am62x.wic.gz` (e.g. with 7-zip)
2. Write the resulting file to the microSD card with a tool like Roadkils Disk Image

Linux

```
gunzip -c bytesatwork-minimal-image-bytedevkit-am62x.wic.gz | dd of=/dev/mmcblk<X> bs=8M␣
↪conv=fsync status=progress
```

Hint: To improve write performance, you could use bmap-tools under Linux:

```
bmaptool copy bytesatwork-minimal-image-bytedevkit-am62x.wic.gz /dev/mmcblk<X>
```

How do you build an image?

Use `repo` to download all necessary repositories:

```
$ mkdir -p ~/workdir/bytedevkit-am62x/4.0; cd ~/workdir/bytedevkit-am62x/4.0
$ repo init -b kirkstone -u https://github.com/bytesatwork/bsp-platform-ti.git
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for byteDEVKIT-am62x:

```
$ cd ~/workdir/bytedevkit-am62x/4.0
$ MACHINE=bytedevkit-am62x DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds the development image:

```
$ cd $BUILDDIR
$ bitbake bytesatwork-minimal-image
```

The output is found in:

```
~/workdir/bytedevkit-am62x/4.0/build/tmp/deploy/images/bytedevkit-am62x
```

Hint: For additional information about yocto images and how to build them, please visit: https://docs.yoctoproject.org/4.0.9/brief-yoctoprojectqs/index.html#building-your-image.

How to modify the image

The image recipes can be found in `~/workdir/bytedevkit-am62x/4.0/sources/meta-bytesatwork/recipes-core/images`

This is relative to where you started the `repo` command to fetch all the sources.

Edit the minimal-image recipe `bytesatwork-minimal-image.bb`

Add the desired software-package to `IMAGE_INSTALL` variable, for example add `net-tools` to `bytesatwork-minimal-image.bb`

Rebuild the image by:

```
$ cd ~/workdir/bytedevkit-am62x/4.0
$ MACHINE=bytedevkit-am62x DISTRO=poky-bytesatwork EULA=1 . setup-environment␣
↪build
$ bitbake bytesatwork-minimal-image
```

How to rename the image

If you want to rename or copy an image, simply rename or copy the image recipe by:

```
$ cd ~/workdir/bytedevkit-am62x/4.0/sources/meta-bytesatwork/recipes-core/
↪images
$ cp bytesatwork-minimal-image.bb customer-example-image.bb
```

Troubleshooting

- **Image size is too small**

  If you encounter that your image size is too small to install additional software, please have a look at the `IMAGE_ROOTFS_SIZE` variable under `~/workdir/bytedevkit-am62x/4.0/sources/meta-bytesatwork/recipes-core/images/bytesatwork-minimal-image.bb`. Increase the size if necessary.

5.1.3 Toolchain

How do you install the toolchain?

Simply download the toolchain and execute the downloaded file, which is a self-extracting shell script.

Hint: If you encounter problems when trying to install the toolchain, make sure the downloaded toolchain is executable. Run `chmod +x /<path>/<toolchain-file>.sh` to make it executable.

Important:

The following tools need to be installed on your development system:

- `xz` (Debian package: `xz-utils`)
- `python` (any version)
- `gcc`

How do you use the toolchain?

Source the installed toolchain:

```
source /opt/poky-bytesatwork/4.0.9/environment-setup-aarch64-poky-linux
```

Check if Cross-compiler is available in environment:

```
echo $CC
```

You should see the following output:

```
aarch64-poky-linux-gcc -fstack-protector-strong -O2 -D_FORTIFY_SOURCE=2 -Wformat -
↪Wformat-security -Werror=format-security --sysroot=/opt/poky-bytesatwork/4.0.9_
↪bytedevkit-am62x/sysroots/aarch64-poky-linux
```

Crosscompile the source code, e.g. by:

```
$CC helloworld.c -o helloworld
```

Check generated binary:

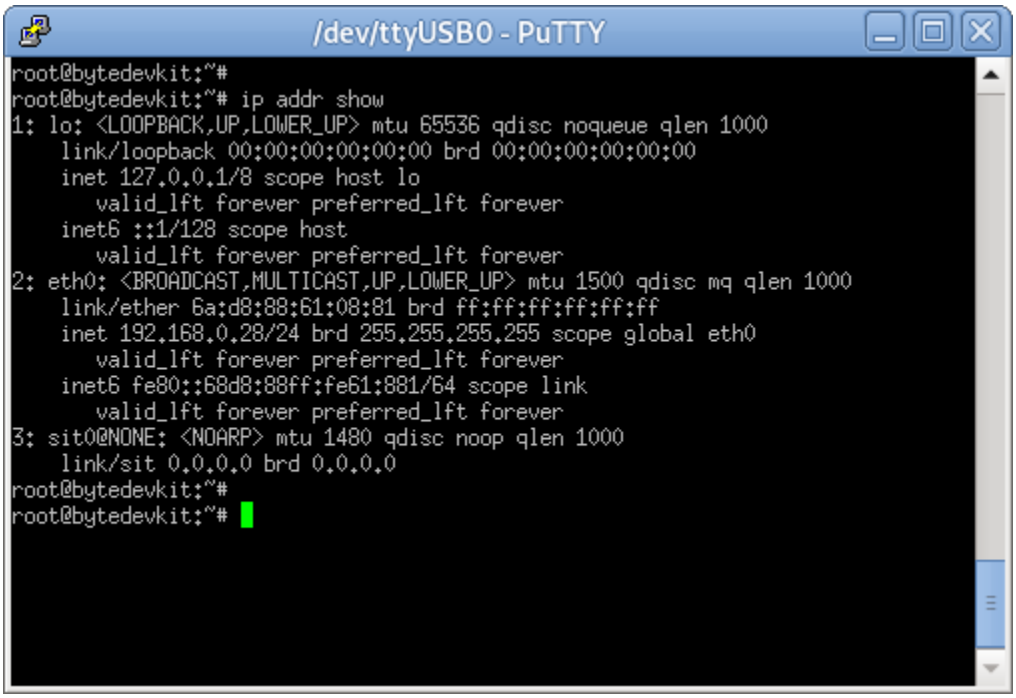
```
file helloworld
```

The output that is shown in prompt afterwards:

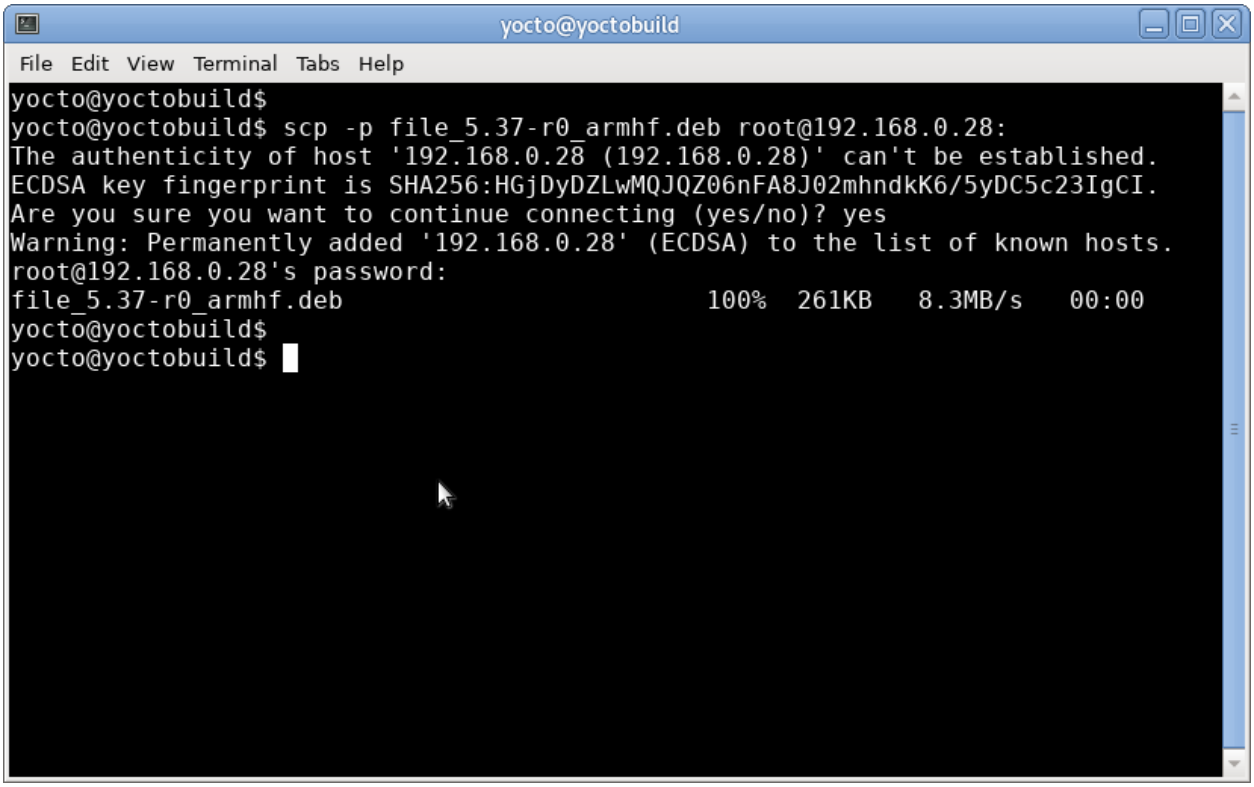
```
helloworld: ELF 64-bit LSB pie executable, ARM aarch64, version 1 (SYSV), dynamically␣
↪linked, interpreter /lib/ld-linux-aarch64.so.1,␣
↪BuildID[sha1]=257792938c3ed4fbf6b15d071c60973ab51b2f37, for GNU/Linux 3.14.0, with␣
↪debug_info, not stripped
```

How to bring your binary to the target?

1. Connect the embedded device's ethernet to your LAN
2. Determine the embedded target IP address by `ip addr show`



3. Copy your binary, e.g. `helloworld` to the target by `scp helloworld root@<ip address of target>:/tmp`



4. Run `chmod +x` on the target to make your binary executable: `chmod +x /<path>/<binary name>`
5. Run your binary on the target: `/<path>/<binary name>`

How do you build a toolchain?

```
$ cd ~/workdir/bytedevkit-am62x/4.0
$ repo init -b kirkstone -u https://github.com/bytesatwork/bsp-platform-ti.git
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for byteDEVKIT-am62x:

```
$ cd ~/workdir/bytedevkit-am62x/4.0
$ MACHINE=bytedevkit-am62x DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds an installable toolchain:

```
$ cd $BUILDDIR
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The toolchain is located under:

```
~/workdir/bytedevkit-am62x/4.0/build/tmp/deploy/sdk
```

How to modify your toolchain

Currently the bytesatwork toolchain is generated out of the bytesatwork-minimal-image recipe. If you want to add additional libraries and development headers to customize the toolchain, you need to modify the bytesatwork-minimal-image recipe. It can be found under `~/workdir/bytedevkit-am62x/4.0/sources/meta-bytesatwork/recipes-core/images`

For example if you want to develop your own ftp client and you need libftp and the corresponding header files, edit the recipe `bytesatwork-minimal-image.bb` and add `ftplib` to the `IMAGE_INSTALL` variable.

This will provide the ftplib libraries and development headers in the toolchain. After adding additional software components, the toolchain needs to be rebuilt by:

```
$ cd ~/workdir/bytedevkit-am62x/4.0
$ MACHINE=bytedevkit-am62x DISTRO=poky-bytesatwork EULA=1 . setup-environment build
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The newly generated toolchain will be available under:

```
~/workdir/bytedevkit-am62x/4.0/build/tmp/deploy/sdk
```

For additional information, please visit: https://docs.yoctoproject.org/4.0.9/overview-manual/concepts.html#cross-development-toolchain-generation.

5.1.4 Kernel

Download the Linux Kernel

Device	Branch	git URL
bytedevkit-am62x	baw-ti-linux-6.1.y	https://github.com/bytesatwork/ti-linux-kernel

Build the Linux Kernel

For both targets, an ARM toolchain is necessary. You can use the provided toolchain from *Toolchain* or any compatible toolchain (e.g. from your distribution)

Important:

The following tools need to be installed on your development system:

- `git`
- `make`
- `bc`

Note: The following instructions assume, you installed the provided toolchain for the respective target.

Important:

The following tools need to be installed on your development system:

- OpenSSL headers (Debian package: `libssl-dev`)
- `depmod` (Debian package: `kmod`)

1. Download kernel sources

   Download the appropriate kernel from *Download the Linux Kernel*.

2. Source toolchain

```
source /opt/poky-bytesatwork/4.0.9/environment-setup-aarch64-poky-linux
```

3. Create defconfig

```
make bytedevkit_am62x_defconfig
```

4. Build Linux kernel

```
make -j `nproc` Image dtbs modules
```

5. Install kernel and device tree

   To use the newly created kernel, device tree and/or module, the necessary files need to be installed on the target. This can be done either via Ethernet (e.g. `scp`) or by copying the files to the SD card.

   Note: For scp installation: Don't forget to mount /boot on the target.

File	Target path	Target partition
`arch/arm64/boot/Image`	`/boot/Image`	`/dev/mmcblk1p2`
`arch/arm64/boot/dts/ti/k3-am625-bytedevkit.dtb`	`/boot/k3-am62x-bytedevkit.dtb`	`/dev/mmcblk1p2`

   Note:

   After installing a new kernel, it often fails to load modules, as the _signature_ of the kernel changed and it fails to find its corresponding modules folder. This issue can often be resolved with a symlink:

```
ln -s /lib/modules/<EXISTING FOLDER> /lib/modules/`uname -r`
```

   Otherwise, please follow the instructions to copy the kernel modules

6. Install kernel modules

   To copy all available modules to the target, it's best to deploy them locally first and then copy all modules to the target.

```
mkdir /tmp/bytedevkit-am62x
make INSTALL_MOD_PATH=/tmp/bytedevkit-am62x modules_install
```

Now you can copy the content of the folder `/tmp/bytedevkit-am62x` into the target's root folder (`/`) which is partition `/dev/mmcblk1p2`.

5.1.5 U-Boot

Download U-Boot Source Code

Device	Branch	git URL
bytedevkit-am62x	baw-ti-u-boot-2023.04	https://github.com/bytesatwork/u-boot-ti

---

Build U-Boot

1. Install and get Dependencies
   - Cross toolchain
   - TI-linux-firmware
   - TF-A
   - OP-TEE

   Hint: Probably some tools are missing on your host:

   - A list can be found here https://docs.u-boot.org/en/latest/build/gcc.html#building-with-gcc
   - A non-exhaustive list of (additional) necessary tools

     ```
     sudo apt install bison flex swig libssl-dev python3-setuptools \
     python-dev python3-dev python3-yaml python3-jsonschema
     ```

   ---

2. Build TF-A

   TI TF-A build instructions

3. Build OP-TEE

   TI OP-TEE build instructions

4. Build u-boot

   You should have downloaded TI-linux-firmware and built TF-A, OP-TEE OS already.

   TI u-boot build instructions

Important: Use `am62x_bytedevkit_r5_defconfig` and `am62x_bytedevkit_a53_defconfig` instead of the TI defconfigs.

Note: Clean command: `make ARCH=arm CROSS_COMPILE=aarch64-linux-gnu- O=<your_dir> distclean`

Install SPL and U-Boot

SD Card

To use the newly created U-Boot, the necessary files need to be installed on the SD card. This can be done either on the host or on the target.

File	Target partition	Target partition label	File system
`tiboot3.bin` `tispl.bin` `u-boot.img`	`/dev/mmcblk1p1` (or `/dev/sdX`)	`boot`	FAT32

You need to copy the files to the boot partition. The example assumes that the boot partition is mounted on `/media/${USER}/boot`:

```
cp tiboot3.bin tispl.bin u-boot.img /media/${USER}/boot/
```

The next time the target is reset, it will start with the new U-Boot.

Hint: Copy the related files to SD card, see end of section TI u-boot build instructions

eMMC via SD Card

1. Copy the `tiboot3.bin`, `tispl.bin` and `u-boot.img` to the SD Card rootfs partition.
2. Program the `tiboot3.bin`, `tispl.bin` and `u-boot.img` from the SD card to the eMMC.

   In the u-boot shell `run update_emmc`

   Or manually by following commands

```
mmc dev 0 1
load mmc 1:2 ${loadaddr} tiboot3.bin
mmc write ${loadaddr} 0x0 0x400
load mmc 1:2 ${loadaddr} tispl.bin
mmc write ${loadaddr} 0x400 0xC00
load mmc 1:2 ${loadaddr} u-boot.img
mmc write ${loadaddr} 0x1000 0x1000
mmc dev 0 0
```

Note: The bootloader needs to be stored in the boot0 hardware partition of the eMMC. The layout of boot0 is defined so that it fits within 4 MiB, defined in blocks of 512 Bytes:

File	start	end	size
`tiboot3.bin`	0x0000	0x0400	0x0400 512 KiB
`tispl.bin`	0x0400	0x1000	0x0C00 1536 KiB
`u-boot.img`	0x1000	0x2000	0x1000 2048 KiB

5.2 byteDEVKIT-imx8mm (Yocto 4.0)

5.2.1 Downloads

SD card image

Download	Checksum (SHA256)
bytesatwork-minimal-image-bytedevkit-imx8mm.wic.gz	99ce54bf379fc97c11157bc48fa0a4fb91ac5f1776968e3bfe2a45471b878427
bytesatwork-minimal-image-bytedevkit-imx8mm.wic.bmap	c94c9177bf80a56fb493acd79df8d677cc7b11d70ea6b7b97256647c161872b4

Hint: Updating from an older image? You can update your older image by using: `apt-get update` and `apt-get upgrade`.

1. check for new version in the table above
2. edit `/etc/apt/sources.list` and point to the new package feed
3. run `apt-get update; apt-get upgrade`

As the yocto framework is based on several packages from various projects or suppliers, it is not guaranteed that an incremental upgrade by `apt-get upgrade` works automatically. Some manual adjustments might be needed.

Toolchain

Download	Checksum (SHA256)
poky-bytesatwork-glibc-x86_64-bytesatwork-minimal-image-cortexa53-crypto-bytedevkit-imx8mm-toolchain-4.0.9.sh	b558c84d3030628daa4d227ba122a3a4f5deccf476d291bd35842

U-Boot

Description	Download	Checksum (SHA256)
U-Boot (SD-card)	imx-boot-bytedevkit-imx8mm-sd.bin-flash_evk	ee2bddafa023d6c84b59474cd783b46fa3bfac7301ba8765d37486dd833b3d0a

5.2.2 Image

How do you flash the image?

Attention:

- You need a microSD card with **at least 8GB** capacity.
- **All existing data** on the microSD card will be lost.
- **Do not format** the microSD card before flashing.

Windows

1. Unzip the file `bytesatwork-minimal-image-bytedevkit-imx8mm.wic.gz` (e.g. with 7-zip)
2. Write the resulting file to the microSD card with a tool like Roadkils Disk Image

Linux

```
gunzip -c bytesatwork-minimal-image-bytedevkit-imx8mm.wic.gz | dd of=/dev/mmcblk<X>␣
↪bs=8M conv=fsync status=progress
```

Hint: To improve write performance, you could use bmap-tools under Linux:

```
bmaptool copy bytesatwork-minimal-image-bytedevkit-imx8mm.wic.gz /dev/mmcblk<X>
```

How do you build an image?

Use `repo` to download all necessary repositories:

```
$ mkdir -p ~/workdir/bytedevkit-imx8mm/4.0; cd ~/workdir/bytedevkit-imx8mm/4.0
$ repo init -b kirkstone -u https://github.com/bytesatwork/bsp-platform-nxp.git
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for byteDEVKIT-imx8mm:

```
$ cd ~/workdir/bytedevkit-imx8mm/4.0
$ MACHINE=bytedevkit-imx8mm DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds the development image:

```
$ cd $BUILDDIR
$ bitbake bytesatwork-minimal-image
```

The output is found in:

```
~/workdir/bytedevkit-imx8mm/4.0/build/tmp/deploy/images/bytedevkit-imx8mm
```

Hint: For additional information about yocto images and how to build them, please visit: https://docs.yoctoproject.org/4.0.9/brief-yoctoprojectqs/index.html#building-your-image.

How to modify the image

The image recipes can be found in `~/workdir/bytedevkit-imx8mm/4.0/sources/meta-bytesatwork/recipes-core/images`

This is relative to where you started the `repo` command to fetch all the sources.

Edit the minimal-image recipe `bytesatwork-minimal-image.bb`

Add the desired software-package to `IMAGE_INSTALL` variable, for example add `net-tools` to `bytesatwork-minimal-image.bb`

Rebuild the image by:

```
$ cd ~/workdir/bytedevkit-imx8mm/4.0
$ MACHINE=bytedevkit-imx8mm DISTRO=poky-bytesatwork EULA=1 . setup-environment␣
↪build
$ bitbake bytesatwork-minimal-image
```

How to rename the image

If you want to rename or copy an image, simply rename or copy the image recipe by:

```
$ cd ~/workdir/bytedevkit-imx8mm/4.0/sources/meta-bytesatwork/recipes-core/
↪images
$ cp bytesatwork-minimal-image.bb customer-example-image.bb
```

Troubleshooting

- **Image size is too small**

  If you encounter that your image size is too small to install additional software, please have a look at the `IMAGE_ROOTFS_SIZE` variable under `~/workdir/bytedevkit-imx8mm/4.0/sources/meta-bytesatwork/recipes-core/images/bytesatwork-minimal-image.bb`. Increase the size if necessary.

5.2.3 Toolchain

How do you install the toolchain?

Simply download the toolchain and execute the downloaded file, which is a self-extracting shell script.

Hint: If you encounter problems when trying to install the toolchain, make sure the downloaded toolchain is executable. Run `chmod +x /<path>/<toolchain-file>.sh` to make it executable.

Important:

The following tools need to be installed on your development system:

- `xz` (Debian package: `xz-utils`)
- `python` (any version)
- `gcc`

How do you use the toolchain?

Source the installed toolchain:

```
source /opt/poky-bytesatwork/4.0.9/environment-setup-cortexa53-crypto-poky-linux
```

Check if Cross-compiler is available in environment:

```
echo $CC
```

You should see the following output:

```
aarch64-poky-linux-gcc -mcpu=cortex-a53 -march=armv8-a+crc+crypto -fstack-protector-
↪strong -O2 -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror=format-security --
↪sysroot=/opt/poky-bytesatwork/4.0.9_bytedevkit-imx8mm/sysroots/cortexa53-crypto-poky-
↪linux
```

Crosscompile the source code, e.g. by:

```
$CC helloworld.c -o helloworld
```

Check generated binary:

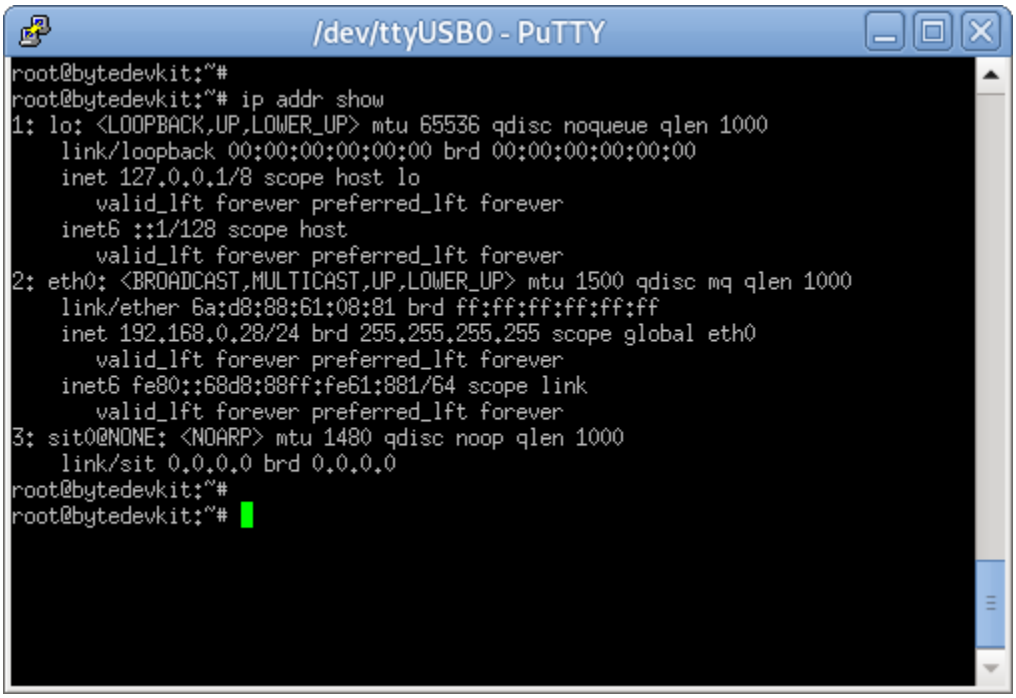
```
file helloworld
```

The output that is shown in prompt afterwards:

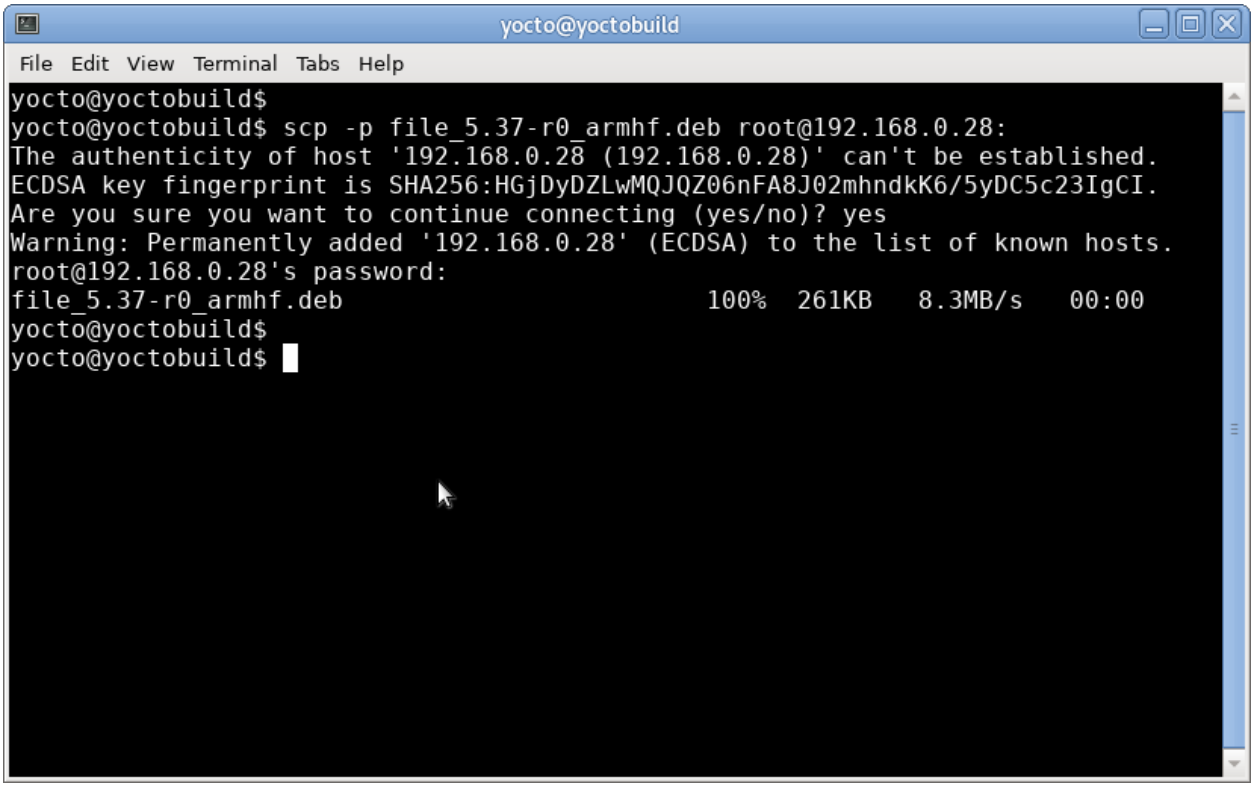
```
helloworld: ELF 64-bit LSB pie executable, ARM aarch64, version 1 (SYSV), dynamically␣
↪linked, interpreter /lib/ld-linux-aarch64.so.1,␣
↪BuildID[sha1]=c4a368203085c7897b632728f24bfa60eec34771, for GNU/Linux 3.14.0, with␣
↪debug_info, not stripped
```

How to bring your binary to the target?

1. Connect the embedded device's ethernet to your LAN
2. Determine the embedded target IP address by `ip addr show`



3. Copy your binary, e.g. `helloworld` to the target by `scp helloworld root@<ip address of target>:/tmp`



4. Run `chmod +x` on the target to make your binary executable: `chmod +x /<path>/<binary name>`
5. Run your binary on the target: `/<path>/<binary name>`

How do you build a toolchain?

```
$ cd ~/workdir/bytedevkit-imx8mm/4.0
$ repo init -b kirkstone -u https://github.com/bytesatwork/bsp-platform-nxp.git
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for byteDEVKIT-imx8mm:

```
$ cd ~/workdir/bytedevkit-imx8mm/4.0
$ MACHINE=bytedevkit-imx8mm DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds an installable toolchain:

```
$ cd $BUILDDIR
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The toolchain is located under:

```
~/workdir/bytedevkit-imx8mm/4.0/build/tmp/deploy/sdk
```

How to modify your toolchain

Currently the bytesatwork toolchain is generated out of the bytesatwork-minimal-image recipe. If you want to add additional libraries and development headers to customize the toolchain, you need to modify the bytesatwork-minimal-image recipe. It can be found under `~/workdir/bytedevkit-imx8mm/4.0/sources/meta-bytesatwork/recipes-core/images`

For example if you want to develop your own ftp client and you need libftp and the corresponding header files, edit the recipe `bytesatwork-minimal-image.bb` and add `ftplib` to the `IMAGE_INSTALL` variable.

This will provide the ftplib libraries and development headers in the toolchain. After adding additional software components, the toolchain needs to be rebuilt by:

```
$ cd ~/workdir/bytedevkit-imx8mm/4.0
$ MACHINE=bytedevkit-imx8mm DISTRO=poky-bytesatwork EULA=1 . setup-environment build
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The newly generated toolchain will be available under:

```
~/workdir/bytedevkit-imx8mm/4.0/build/tmp/deploy/sdk
```

For additional information, please visit: https://docs.yoctoproject.org/4.0.9/overview-manual/concepts.html#cross-development-toolchain-generation.

5.2.4 Kernel

Download the Linux Kernel

Device	Branch	git URL
bytedevkit-imx8mm	baw-lf-5.15.y	https://github.com/bytesatwork/linux-imx.git

Build the Linux Kernel

For both targets, an ARM toolchain is necessary. You can use the provided toolchain from *Toolchain* or any compatible toolchain (e.g. from your distribution)

Important:

The following tools need to be installed on your development system:

- `git`
- `make`
- `bc`

Note: The following instructions assume, you installed the provided toolchain for the respective target.

Important:

The following tools need to be installed on your development system:

- OpenSSL headers (Debian package: `libssl-dev`)
- `depmod` (Debian package: `kmod`)

1. Download kernel sources

   Download the appropriate kernel from *Download the Linux Kernel*.

2. Source toolchain

```
source /opt/poky-bytesatwork/4.0.9/environment-setup-cortexa53-crypto-poky-linux
```

3. Create defconfig

```
make bytedevkit_imx8mm_defconfig
```

4. Build Linux kernel

```
make -j `nproc` Image dtbs modules
```

5. Install kernel and device tree

   To use the newly created kernel, device tree and/or module, the necessary files need to be installed on the target. This can be done either via Ethernet (e.g. `scp`) or by copying the files to the SD card.

   Note: For scp installation: Don't forget to mount /boot on the target.

File	Target path	Target partition
`arch/arm64/boot/Image`	`/boot/Image`	`/dev/mmcblk1p1`
`arch/arm64/boot/dts/freescale/imx8mm-bytedevkit.dtb`	`/boot/imx8mm-bytedevkit.dtb`	`/dev/mmcblk1p1`

   Note:

   After installing a new kernel, it often fails to load modules, as the _signature_ of the kernel changed and it fails to find its corresponding modules folder. This issue can often be resolved with a symlink:

```
ln -s /lib/modules/<EXISTING FOLDER> /lib/modules/`uname -r`
```

   Otherwise, please follow the instructions to copy the kernel modules

6. Install kernel modules

   To copy all available modules to the target, it's best to deploy them locally first and then copy all modules to the target.

```
mkdir /tmp/bytedevkit-imx8mm
make INSTALL_MOD_PATH=/tmp/bytedevkit-imx8mm modules_install
```

Now you can copy the content of the folder `/tmp/bytedevkit-imx8mm` into the target's root folder (`/`) which is partition `/dev/mmcblk1p1`.

5.2.5 U-Boot

Additional information can be found under https://www.nxp.com/docs/en/user-guide/IMX_LINUX_USERS_GUIDE.pdf and https://docs.u-boot.org/en/latest/board/nxp/index.html.

Note: On i.MX 8M Mini, SPL and U-Boot are combined in a container file called `flash.bin` (Yocto: `imx-boot-bytedevkit-imx8mm-sd.bin-flash_evk`).

Download U-Boot Source Code

Device	Branch	git URL
bytedevkit-imx8mm	baw-imx_v2020.04_5.4.24_2.1.0	https://github.com/bytesatwork/u-boot-imx

Build U-Boot

To compile U-Boot, an ARM toolchain is necessary. You can use the provided toolchain from *Toolchain* or any compatible toolchain (e.g. from your distribution)

Important: A list of needed host tools can be found here https://docs.u-boot.org/en/latest/build/gcc.html#building-with-gcc, e.g.

```
sudo apt install bc bison build-essential coccinelle \
device-tree-compiler dfu-util efitools flex gdisk graphviz imagemagick \
liblz4-tool libgnutls28-dev libguestfs-tools libncurses-dev \
libpython3-dev libsdl2-dev libssl-dev lz4 lzma lzma-alone openssl \
pkg-config python3 python3-asteval python3-coverage python3-filelock \
python3-pkg-resources python3-pycryptodome python3-pyelftools \
python3-pytest python3-pytest-xdist python3-sphinxcontrib.apidoc \
python3-sphinx-rtd-theme python3-subunit python3-testtools \
python3-virtualenv swig uuid-dev
```

`fspi_packer.sh` additionally needs the package `xxd` to be installed on your host:

```
sudo apt install xxd
```

Note: The following instructions assume, you installed the provided toolchain for the respective target.

1. Download ARM-Trusted-Firmware sources

Device	Branch	git URL
bytedevkit-imx8mm	imx_5.4.24_2.1.0	https://github.com/nxp-imx/imx-atf

2. Build ARM-Trusted-Firmware

```
cd imx-atf
export CROSS_COMPILE=/opt/poky-bytesatwork/4.0.9/sysroots/x86_64-pokysdk-linux/usr/
↪bin/aarch64-poky-linux/aarch64-poky-linux-
make PLAT=imx8mm bl31
cd ..
```

3. Download IMX Firmware

```
wget https://www.nxp.com/lgfiles/NMG/MAD/YOCTO/firmware-imx-8.15.bin
chmod +x firmware-imx-8.15.bin
./firmware-imx-8.15.bin
```

4. Download U-Boot sources

   Download the appropriate U-Boot from *Download U-Boot Source Code*.

5. Source toolchain

```
source /opt/poky-bytesatwork/4.0.9/environment-setup-cortexa53-crypto-poky-linux
```

6. Copy necessary files into U-Boot folder

```
cp -pv ./firmware-imx-8.15/firmware/ddr/synopsys/lpddr4_pmu_train_* ./u-boot-imx/
cp -pv ./imx-atf/build/imx8mm/release/bl31.bin ./u-boot-imx/
```

7. Build `flash.bin`

   - SD Card

     ```
     cd u-boot-imx
     make distclean
     make bytedevkit_defconfig
     export ATF_LOAD_ADDR=0x920000
     make -j `nproc`
     make -j `nproc` flash.bin
     cd ..
     ```

   - SPI

     Building for SPI requires IMX mkimage tool

     ```
     git clone -b lf-5.15.5_1.0.0 https://github.com/nxp-imx/imx-mkimage.git
     ```

```
cd u-boot-imx
make distclean
make bytedevkit_fspi_defconfig
export ATF_LOAD_ADDR=0x920000
make -j `nproc`
make -j `nproc` flash.bin
../imx-mkimage/scripts/fspi_packer.sh ../imx-mkimage/scripts/fspi_header␣
↪0
cd ..
```

Important: The build command will overwrite the generated `flash.bin`, so you can not build a binary for the SD Card and the SPI at the same time.

Install SPL and U-Boot

To use the newly created U-Boot, the necessary file needs to be installed on the SD card. This can be done either on the host or on the target.

File	Target partition	Offset
`flash.bin` Yocto: `imx-boot-bytedevkit-imx8mm-sd.bin-flash_evk`	`/dev/mmcblk1` (or `/dev/sdX`)	33 KiB

You need to write the files to the respective "raw" partition, either on the host system or the target system:

```
dd if=./u-boot-imx/flash.bin of=/dev/mmcblk1 bs=1K seek=33
```

The next time the target is reset, it will start with the new U-Boot.

Note: Flash to SPI

1. Copy flash.bin to first SD card partition (root partition)
2. You need to boot into u-boot.
3. In the u-boot shell: `run update-spi`
4. Or do it manually by

```
sf probe; sf erase 0 0x200000; load mmc 1:1 ${loadaddr} flash.bin; sf␣
↪write $loadaddr 0 $filesize
```

5.3 byteDEVKIT-stm32mp1 (Yocto 4.0)

5.3.1 Downloads

SD card image

Download	Checksum (SHA256)
bytesatwork-minimal-image-bytedevkit-stm32mp1.wic.gz	72e629a3361f2f5529e6124a30ecf7637d0dc0e3045b310d7af8ddbcf3f7ca2b
bytesatwork-minimal-image-bytedevkit-stm32mp1.wic.bmap	9548f8d625f40a8e43009da3635cee5223235e4839043e28bb38c6873abc7747

Hint: Updating from an older image? You can update your older image by using: `apt-get update` and `apt-get upgrade`.

1. check for new version in the table above
2. edit `/etc/apt/sources.list` and point to the new package feed
3. run `apt-get update; apt-get upgrade`

As the yocto framework is based on several packages from various projects or suppliers, it is not guaranteed that an incremental upgrade by `apt-get upgrade` works automatically. Some manual adjustments might be needed.

Toolchain

Download	Checksum (SHA256)
poky-bytesatwork-glibc-x86_64-bytesatwork-minimal-image-cortexa7t2hf-neon-vfpv4-bytedevkit-stm32mp1-toolchain-4.0.9.sh	847997ab62d47598aa743b6192b36ba6425feef3e9d77961384

U-Boot

Note: The images come with a preinstalled U-Boot that supports 512 MB of RAM. If you have a module with 1 GB of RAM, you will have to *Install SPL and U-Boot* to unlock the full 1 GB of RAM.

Description	Download	Checksum (SHA256)
MLO (512 MB)	u-boot-spl.stm32-stm32mp157c-bytedevkit-v1-3-basic	0556b53f8f9ecff54af89f7fa1f32aec97549aef1a54a1723d3561677804317b
U-Boot (512 MB)	u-boot-stm32mp157c-bytedevkit-v1-3-basic.img	24fbb4bf87bc4a459d7dd9aeb5c906bceb47a3df8a9954e0f3e860e0a085abc
MLO (1 GB)	u-boot-spl.stm32-stm32mp157c-bytedevkit-v1-3-1g_ram	1cc7589cd4f39a6782d0276c890521c53a4ef6099fde35c4edbad5370f090d2
U-Boot (1 GB)	u-boot-stm32mp157c-bytedevkit-v1-3-1g_ram.img	aebe97b9be2c0862d4a9c9b156278325d70fe33fded7eb0b4bd51377835a3b

5.3.2 Image

How do you flash the image?

Attention:

- You need a microSD card with **at least 8GB** capacity.
- **All existing data** on the microSD card will be lost.
- **Do not format** the microSD card before flashing.

Windows

1. Unzip the file `bytesatwork-minimal-image-bytedevkit-stm32mp1.wic.gz` (e.g. with 7-zip)
2. Write the resulting file to the microSD card with a tool like Roadkils Disk Image

Linux

```
gunzip -c bytesatwork-minimal-image-bytedevkit-stm32mp1.wic.gz | dd of=/dev/mmcblk<X>␣
↪bs=8M conv=fsync status=progress
```

Hint: To improve write performance, you could use bmap-tools under Linux:

```
bmaptool copy bytesatwork-minimal-image-bytedevkit-stm32mp1.wic.gz /dev/mmcblk<X>
```

How do you build an image?

Use `repo` to download all necessary repositories:

```
$ mkdir -p ~/workdir/bytedevkit-stm32mp1/4.0; cd ~/workdir/bytedevkit-stm32mp1/4.0
$ repo init -b kirkstone -u https://github.com/bytesatwork/bsp-platform-st.git
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for byteDEVKIT-stm32mp1:

```
$ cd ~/workdir/bytedevkit-stm32mp1/4.0
$ MACHINE=bytedevkit-stm32mp1 DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds the development image:

```
$ cd $BUILDDIR
$ bitbake bytesatwork-minimal-image
```

The output is found in:

```
~/workdir/bytedevkit-stm32mp1/4.0/build/tmp/deploy/images/bytedevkit-stm32mp1
```

Hint: For additional information about yocto images and how to build them, please visit: https://docs.yoctoproject.org/4.0.9/brief-yoctoprojectqs/index.html#building-your-image.

How to modify the image

The image recipes can be found in `~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images`

This is relative to where you started the `repo` command to fetch all the sources.

Edit the minimal-image recipe `bytesatwork-minimal-image.bb`

Add the desired software-package to `IMAGE_INSTALL` variable, for example add `net-tools` to `bytesatwork-minimal-image.bb`

Rebuild the image by:

```
$ cd ~/workdir/<machine name>/<yocto version>
$ MACHINE=<machine name> DISTRO=poky-bytesatwork EULA=1 . setup-environment␣
↪build
$ bitbake bytesatwork-minimal-image
```

How to rename the image

If you want to rename or copy an image, simply rename or copy the image recipe by:

```
$ cd ~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-
↪core/images
$ cp bytesatwork-minimal-image.bb customer-example-image.bb
```

Troubleshooting

- **Image size is too small**

  If you encounter that your image size is too small to install additional software, please have a look at the `IMAGE_ROOTFS_SIZE` variable under `~/workdir/<machine-name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images/bytesatwork-minimal-image.bb`. Increase the size if necessary.

5.3.3 Toolchain

How do you install the toolchain?

Simply download the toolchain and execute the downloaded file, which is a self-extracting shell script.

Hint: If you encounter problems when trying to install the toolchain, make sure the downloaded toolchain is executable. Run `chmod +x /<path>/<toolchain-file>.sh` to make it executable.

Important:

The following tools need to be installed on your development system:

- `xz` (Debian package: `xz-utils`)
- `python` (any version)
- `gcc`

How do you use the toolchain?

Source the installed toolchain:

```
source /opt/poky-bytesatwork/4.0.9/environment-setup-cortexa7t2hf-neon-vfpv4-poky-linux-
↪gnueabi
```

Check if Cross-compiler is available in environment:

```
echo $CC
```

You should see the following output:

```
arm-poky-linux-gnueabi-gcc -mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7 -
↪fstack-protector-strong -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror=format-
↪security --sysroot=/opt/poky-bytesatwork/4.0.9/sysroots/cortexa7t2hf-neon-vfpv4-poky-
↪linux-gnueabi
```

Crosscompile the source code, e.g. by:

```
$CC helloworld.c -o helloworld
```

Check generated binary:

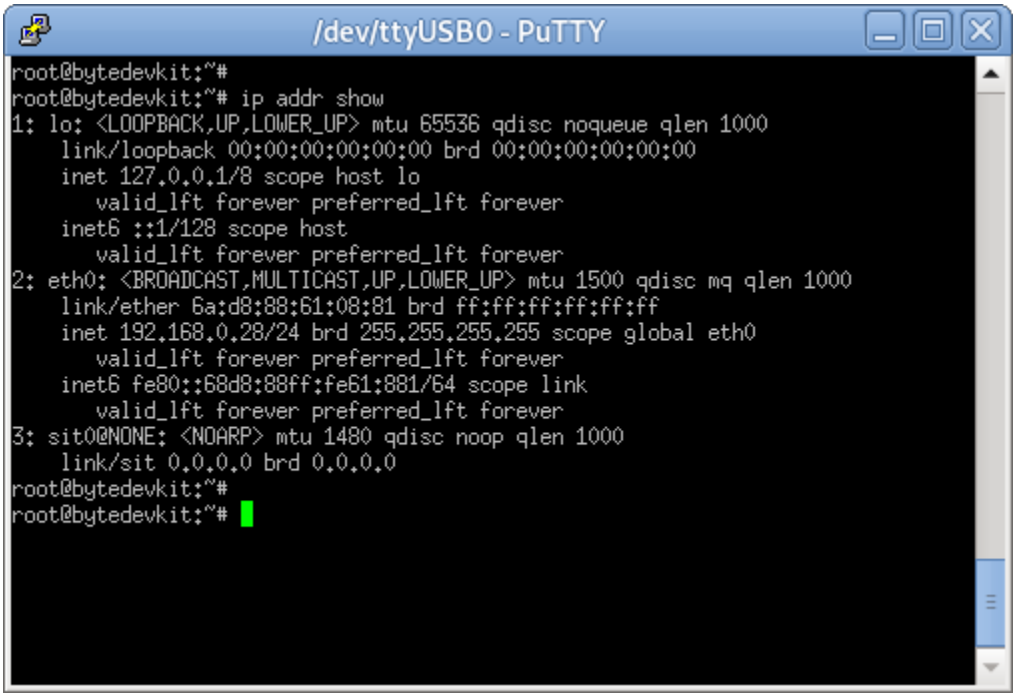
```
file helloworld
```

The output that is shown in prompt afterwards:

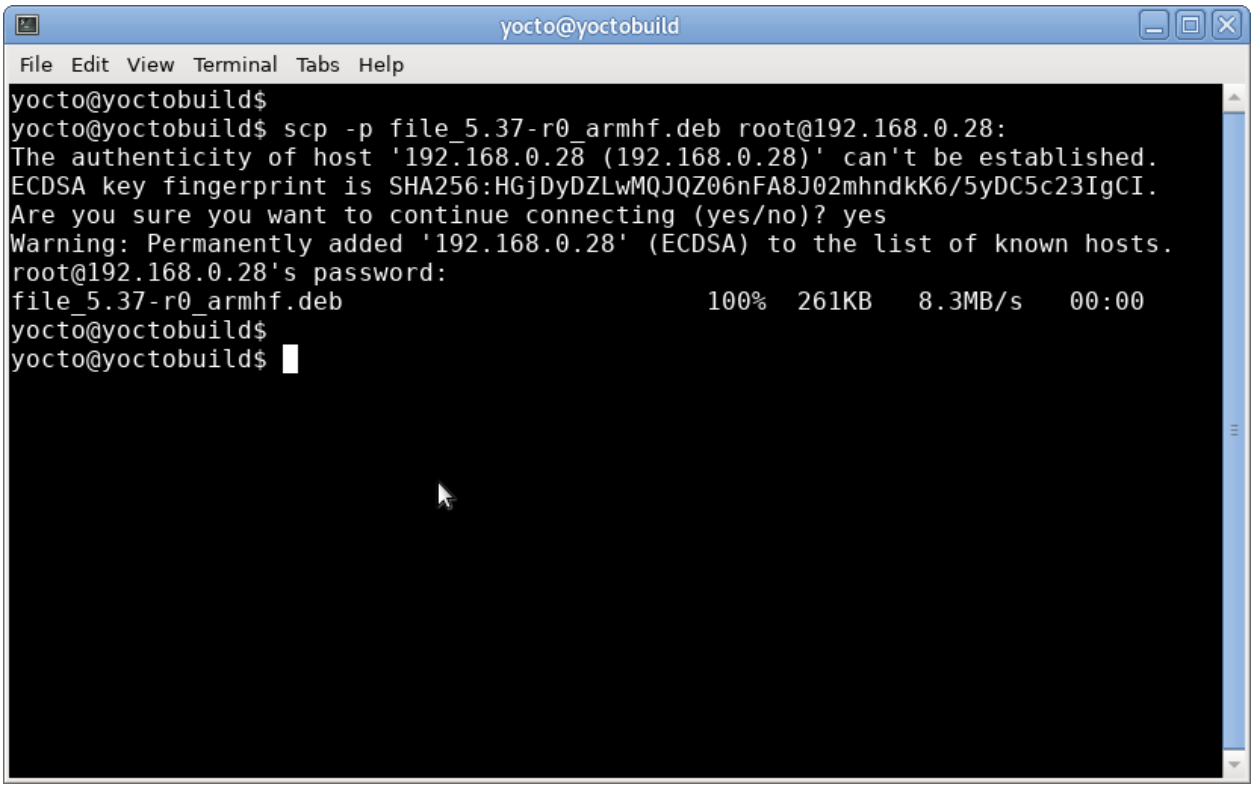
```
helloworld: ELF 32-bit LSB pie executable, ARM, EABI5 version 1
```

How to bring your binary to the target?

1. Connect the embedded device's ethernet to your LAN
2. Determine the embedded target IP address by `ip addr show`



3. Copy your binary, e.g. `helloworld` to the target by `scp helloworld root@<ip address of target>:/tmp`



4. Run `chmod +x` on the target to make your binary executable: `chmod +x /<path>/<binary name>`
5. Run your binary on the target: `/<path>/<binary name>`

How do you build a toolchain?

```
$ cd ~/workdir/bytedevkit-stm32mp1/4.0
$ repo init -b kirkstone -u https://github.com/bytesatwork/bsp-platform-st.git
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for byteDEVKIT-stm32mp1:

```
$ cd ~/workdir/bytedevkit-stm32mp1/4.0
$ MACHINE=bytedevkit-stm32mp1 DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds an installable toolchain:

```
$ cd $BUILDDIR
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The toolchain is located under:

```
~/workdir/bytedevkit-stm32mp1/4.0/build/tmp/deploy/sdk
```

How to modify your toolchain

Currently the bytesatwork toolchain is generated out of the bytesatwork-minimal-image recipe. If you want to add additional libraries and development headers to customize the toolchain, you need to modify the bytesatwork-minimal-image recipe. It can be found under `~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images`

For example if you want to develop your own ftp client and you need libftp and the corresponding header files, edit the recipe `bytesatwork-minimal-image.bb` and add `ftplib` to the `IMAGE_INSTALL` variable.

This will provide the ftplib libraries and development headers in the toolchain. After adding additional software components, the toolchain needs to be rebuilt by:

```
$ cd ~/workdir/<machine name>/<yocto version>
$ MACHINE=<machine> DISTRO=poky-bytesatwork EULA=1 . setup-environment build
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The newly generated toolchain will be available under:

```
~/workdir/<machine name>/<yocto version>/build/tmp/deploy/sdk
```

For additional information, please visit: https://docs.yoctoproject.org/4.0.9/overview-manual/concepts.html#cross-development-toolchain-generation.

5.3.4 Kernel

Download the Linux Kernel

Device	Branch	git URL
bytedevkit-stm32mp1	baw-v5.10-stm32mp-r2	https://github.com/bytesatwork/linux-stm32mp.git

Build the Linux Kernel

For both targets, an ARM toolchain is necessary. You can use the provided toolchain from *Toolchain* or any compatible toolchain (e.g. from your distribution)

Important:

The following tools need to be installed on your development system:

- `git`
- `make`
- `bc`

Note: The following instructions assume, you installed the provided toolchain for the respective target.

Important:

The following tools need to be installed on your development system:

- OpenSSL headers (Debian package: `libssl-dev`)
- `depmod` (Debian package: `kmod`)

1. Download kernel sources

   Download the appropriate kernel from *Download the Linux Kernel*.

2. Source toolchain

```
source /opt/poky-bytesatwork/4.0.9/environment-setup-cortexa7t2hf-neon-vfpv4-poky-
↪linux-gnueabi
```

3. Create defconfig

```
make multi_v7_defconfig
scripts/kconfig/merge_config.sh -m -r .config arch/arm/configs/fragment-*
make olddefconfig
```

4. Build Linux kernel

```
make LOADADDR=0xC2000040 -j `nproc` uImage stm32mp157c-bytedevkit-v1-3.dtb modules
```

5. Install kernel and device tree

   To use the newly created kernel, device tree and/or module, the necessary files need to be installed on the target. This can be done either via Ethernet (e.g. `scp`) or by copying the files to the SD card.

   Note: For scp installation: Don't forget to mount /boot on the target.

File	Target path	Target partition
arch/arm/boot/uImage	/boot/uImage	/dev/mmcblk0p4
arch/arm/boot/dts/stm32mp157c-bytedevkit-v1-3.dtb	/boot/stm32mp157c-bytedevkit-v1-3.dtb	/dev/mmcblk0p4

   Note:

   After installing a new kernel, it often fails to load modules, as the _signature_ of the kernel changed and it fails to find its corresponding modules folder. This issue can often be resolved with a symlink:

   ```
   ln -s /lib/modules/<EXISTING FOLDER> /lib/modules/`uname -r`
   ```

   Otherwise, please follow the instructions to copy the kernel modules

   Hint: If you have a byteDEVKIT V1.1, replace `v1-3` with `v1-1` in the file names above.

6. Install kernel modules

   To copy all available modules to the target, it's best to deploy them locally first and then copy all modules to the target.

   ```
   mkdir /tmp/bytedevkit-stm32mp1
   make INSTALL_MOD_PATH=/tmp/bytedevkit-stm32mp1 modules_install
   ```

   Now you can copy the content of the folder `/tmp/bytedevkit-stm32mp1` into the target's root folder (`/`) which is partition `/dev/mmcblk0p5`.

5.3.5 U-Boot

Download U-Boot Source Code

Device	Branch	git URL
bytedevkit-stm32mp1	baw-v2020.01-stm32mp-r1	https://github.com/bytesatwork/u-boot-stm32mp

Build U-Boot

To compile U-Boot, an ARM toolchain is necessary. You can use the provided toolchain from *Toolchain* or any compatible toolchain (e.g. from your distribution)

Important:

The following tools need to be installed on your development system:

- `git`
- `make`
- `bc`

Note: The following instructions assume, you installed the provided toolchain for the respective target.

1. Download U-Boot sources

   Download the appropriate U-Boot from *Download U-Boot Source Code*.

2. Source toolchain

```
source /opt/poky-bytesatwork/4.0.9/environment-setup-cortexa7t2hf-neon-vfpv4-poky-
↪linux-gnueabi
```

3. Create defconfig

```
make stm32mp157_bytedevkit_defconfig
```

Note: For the 1 GB RAM variant, use `make stm32mp157_bytedevkit_1g_defconfig` instead.

4. Build U-Boot and SPL

```
make -j `nproc`
```

Install SPL and U-Boot

To use the newly created U-Boot, the necessary files need to be installed on the SD card. This can be done either on the host or on the target.

File	Target partition
u-boot-spl.stm32	/dev/mmcblk0p1
u-boot-spl.stm32	/dev/mmcblk0p2
u-boot.img	/dev/mmcblk0p3

You need to write the files to the respective "raw" partition, either on the host system or the target system:

```
dd if=u-boot-spl.stm32 of=/dev/mmcblk0p1
dd if=u-boot-spl.stm32 of=/dev/mmcblk0p2
dd if=u-boot.img of=/dev/mmcblk0p3
```

The next time the target is reset, it will start with the new U-Boot.

5.4 Archive

Here you'll find informations on older images and platforms.

Note: Information in this section is EOL and not supported anymore.

5.4.1 byteDEVKIT-am335x (Yocto 3.1)

Image

Where do you get the SD card image?

Device	Yocto Version	Download	Checksum (SHA256)
bytedevkit-am335x	Yocto 3.1.3	bytesatwork-minimal-image-bytedevkit-am335x.wic.gz (wic.bmap)	d1429b5f68808450538d6354d7f40898828c73ef1079092d23663

Hint: Updating from an older image? You can update your older image by using: `apt-get update` and `apt-get upgrade`.

1. check for new version in the table above
2. edit `/etc/apt/sources.list` and point to the new package feed

3. run `apt-get update; apt-get upgrade`

As the yocto framework is based on several packages from various projects or suppliers, it is not guaranteed that an incremental upgrade by `apt-get upgrade` works automatically. Some manual adjustments might be needed.

How do you flash the image?

Attention:

- You need a microSD card with **at least 8GB** capacity.
- **All existing data** on the microSD card will be lost.
- **Do not format** the microSD card before flashing.

Windows

1. Unzip the file `bytesatwork-minimal-image-bytedevkit-am335x.wic.gz` (e.g. with 7-zip)
2. Write the resulting file to the microSD card with a tool like Roadkils Disk Image

Linux

```
gunzip -c bytesatwork-minimal-image-bytedevkit-am335x.wic.gz | dd of=/dev/mmcblk<X>␣
↪bs=8M conv=fdatasync status=progress
```

Hint: To improve write performance, you could use bmap-tools under Linux:

```
bmaptool copy bytesatwork-minimal-image-bytedevkit-am335x.wic.gz /dev/mmcblk<X>
```

How do you build an image?

Use `repo` to download all necessary repositories:

```
$ mkdir -p ~/workdir/bytedevkit-am335x/3.1; cd ~/workdir/bytedevkit-am335x/3.1
$ repo init -u https://github.com/bytesatwork/bsp-platform-ti.git -b dunfell
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for byteDEVKIT-am335x:

```
$ cd ~/workdir/bytedevkit-am335x/3.1
$ MACHINE=bytedevkit-am335x DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds the development image:

```
$ cd $BUILDDIR
$ bitbake bytesatwork-minimal-image
```

The output is found in:

```
~/workdir/bytedevkit-am335x/3.1/build/tmp/deploy/images/bytedevkit-am335x
```

Hint: For additional information about yocto images and how to build them, please visit: https://www.yoctoproject.org/docs/3.1/mega-manual/mega-manual.html#brief-building-your-image

How to modify the image

The image recipes can be found in `~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images`

This is relative to where you started the `repo` command to fetch all the sources.

Edit the minimal-image recipe `bytesatwork-minimal-image.bb`

Add the desired software-package to `IMAGE_INSTALL` variable, for example add `net-tools` to `bytesatwork-minimal-image.bb`

Rebuild the image by:

```
$ cd ~/workdir/<machine name>/<yocto version>
$ MACHINE=<machine name> DISTRO=poky-bytesatwork EULA=1 . setup-environment␣
↪build
$ bitbake bytesatwork-minimal-image
```

How to rename the image

If you want to rename or copy an image, simply rename or copy the image recipe by:

```
$ cd ~/workdir/<machine name>/<yocto version>/build/tmp/deploy/images/<machine␣
↪name>
$ cp bytesatwork-minimal-image.bb customer-example-image.bb
```

Troubleshooting

- **Image size is too small**

  If you encounter that your image size is too small to install additional software, please have a look at the `IMAGE_ROOTFS_SIZE` variable under `~/workdir/<machine-name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images/bytesatwork-minimal-image.bb`. Increase the size if necessary.

Toolchain

Where do you get the toolchain?

Device	Yocto Version	Download	Checksum (SHA256)
bytedevkit am335x	Yocto 3.1.3	poky-bytesatwork-glibc-x86_64-bytesatwork-minimal-image-armv7at2hf-neon-bytedevkit-am335x-toolchain-3.1.3.sh	8f36974f1635022a1744f0dfde9c3810fcd1a44422afdac

How do you install the toolchain?

Simply download the toolchain and execute the downloaded file, which is a self-extracting shell script.

Hint: If you encounter problems when trying to install the toolchain, make sure the downloaded toolchain is executable. Run `chmod +x /<path>/<toolchain-file>.sh` to make it executable.

Important:

The following tools need to be installed on your development system:

- `xz` (Debian package: `xz-utils`)
- `python` (any version)
- `gcc`

How do you use the toolchain?

Source the installed toolchain:

```
source /opt/poky-bytesatwork/3.1.3/environment-setup-armv7at2hf-neon-poky-linux-gnueabi
```

Check if Cross-compiler is available in environment:

```
echo $CC
```

You should see the following output:

```
arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb -mfpu=neon -mfloat-abi=hard -fstack-
↪protector-strong -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror=format-
↪security --sysroot=/opt/poky-bytesatwork/3.1.3/sysroots/armv7at2hf-neon-poky-linux-
↪gnueabi
```

Crosscompile the source code, e.g. by:

```
$CC helloworld.c -o helloworld
```

Check generated binary:

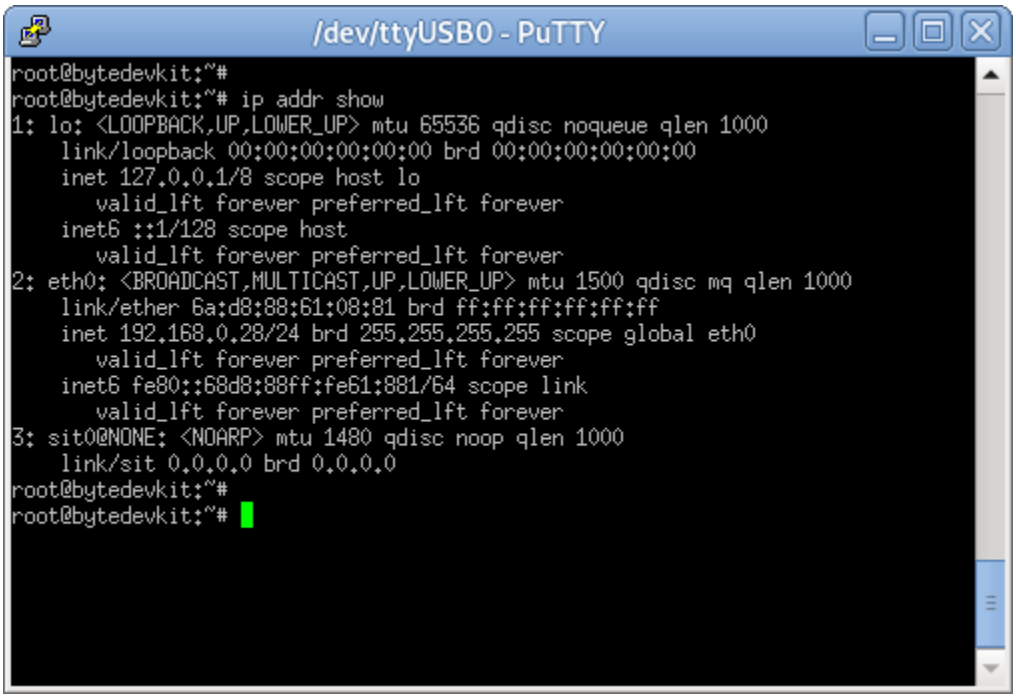
```
file helloworld
```

The output that is shown in prompt afterwards:

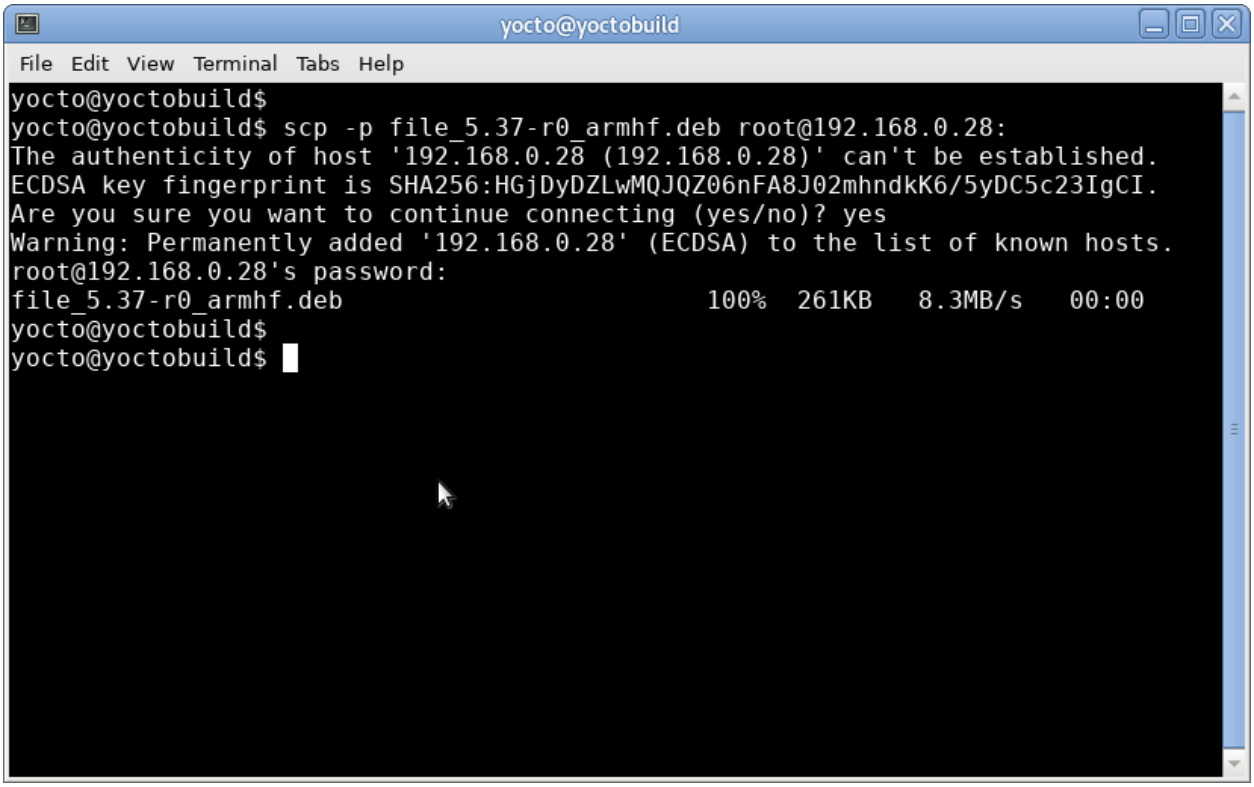
```
helloworld: ELF 32-bit LSB pie executable, ARM, EABI5 version 1
```

How to bring your binary to the target?

1. Connect the embedded device's ethernet to your LAN
2. Determine the embedded target IP address by `ip addr show`



3. Copy your binary, e.g. `helloworld` to the target by `scp helloworld root@<ip address of target>:/tmp`



4. Run `chmod +x` on the target to make your binary executable: `chmod +x /<path>/<binary name>`
5. Run your binary on the target: `/<path>/<binary name>`

How do you build a toolchain?

```
$ cd ~/workdir/bytedevkit-am335x/3.1
$ repo init -u https://github.com/bytesatwork/bsp-platform-ti.git -b dunfell
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for byteDEVKIT-am335x:

```
$ cd ~/workdir/bytedevkit-am335x/3.1
$ MACHINE=bytedevkit-am335x DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds an installable toolchain:

```
$ cd $BUILDDIR
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The toolchain is located under:

```
~/workdir/bytedevkit-am335x/3.1/build/tmp/deploy/sdk
```

How to modify your toolchain

Currently the bytesatwork toolchain is generated out of the bytesatwork-minimal-image recipe. If you want to add additional libraries and development headers to customize the toolchain, you need to modify the bytesatwork-minimal-image recipe. It can be found under `~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images`

For example if you want to develop your own ftp client and you need libftp and the corresponding header files, edit the recipe `bytesatwork-minimal-image.bb` and add `ftplib` to the `IMAGE_INSTALL` variable.

This will provide the ftplib libraries and development headers in the toolchain. After adding additional software components, the toolchain needs to be rebuilt by:

```
$ cd ~/workdir/<machine name>/<yocto version>
$ MACHINE=<machine> DISTRO=poky-bytesatwork EULA=1 . setup-environment build
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The newly generated toolchain will be available under:

```
~/workdir/<machine name>/<yocto version>/build/tmp/deploy/sdk
```

For additional information, please visit: https://www.yoctoproject.org/docs/3.1/overview-manual/overview-manual.html#cross-development-toolchain-generation

Kernel

Download the Linux Kernel

Device	Branch	git URL
bytedevkit-am335x	baw-ti-linux-5.4.y	https://github.com/bytesatwork/ti-linux-kernel

---

Build the Linux Kernel

For both targets, an ARM toolchain is necessary. You can use the provided toolchain from *Where do you get the toolchain?* or any compatible toolchain (e.g. from your distribution)

Important:

The following tools need to be installed on your development system:

- `git`
- `make`
- `bc`

Note: The following instructions assume, you installed the provided toolchain for the respective target.

Important:

The following tools need to be installed on your development system:

- OpenSSL headers (Debian package: `libssl-dev`)
- `depmod` (Debian package: `kmod`)
- `mkimage` (Debian package: `u-boot-tools`)

1. Download kernel sources

   Download the appropriate kernel from *Download the Linux Kernel*.

2. Source toolchain

```
source /opt/poky-bytesatwork/3.1.3/environment-setup-armv7at2hf-neon-poky-linux-
↪gnueabi
```

3. Create defconfig

```
make multi_v7_defconfig
```

4. Build Linux kernel

```
make LOADADDR=0x80008000 -j `nproc` uImage am335x-bytedevkit.dtb modules
```

5. Install kernel and device tree

   To use the newly created kernel, device tree and/or module, the necessary files need to be installed on the target. This can be done either via Ethernet (e.g. `scp`) or by copying the files to the SD card.

   Note: For scp installation: Don't forget to mount /boot on the target.

File	Target path	Target partition
`arch/arm/boot/uImage`	`/boot/uImage`	`/dev/mmcblk0p4`
`arch/arm/boot/dts/am335x-bytedevkit.dtb`	`/boot/am335x-bytedevkit.dtb`	`/dev/mmcblk0p4`

   Note:

   After installing a new kernel, it often fails to load modules, as the _signature_ of the kernel changed and it fails to find its corresponding modules folder. This issue can often be resolved with a symlink:

```
ln -s /lib/modules/<EXISTING FOLDER> /lib/modules/`uname -r`
```

   Otherwise, please follow the instructions to copy the kernel modules

6. Install kernel modules

   To copy all available modules to the target, it's best to deploy them locally first and then copy all modules to the target.

```
mkdir /tmp/bytedevkit-am335x
make INSTALL_MOD_PATH=/tmp/bytedevkit-am335x modules_install
```

Now you can copy the content of the folder `/tmp/bytedevkit-am335x` into the target's root folder (`/`) which is partition `/dev/mmcblk0p5`.

5.4.2 byteDEVKIT-stm32mp1 (Yocto 3.1)

Downloads

SD card image

Download	Checksum (SHA256)
bytesatwork-minimal-image-bytedevkit-stm32mp1.wic.gz (wic.bmap)	6fa368ff5df6967480f3704c1a9e987f284fa0f8b78ec679c57be9f74e4520f7

Hint: Updating from an older image? You can update your older image by using: `apt-get update` and `apt-get upgrade`.

1. check for new version in the table above
2. edit `/etc/apt/sources.list` and point to the new package feed
3. run `apt-get update; apt-get upgrade`

As the yocto framework is based on several packages from various projects or suppliers, it is not guaranteed that an incremental upgrade by `apt-get upgrade` works automatically. Some manual adjustments might be needed.

Toolchain

Download	Checksum (SHA256)
poky-bytesatwork-glibc-x86_64-bytesatwork-minimal-image-cortexa7t2hf-neon-vfpv4-bytedevkit-stm32mp1-toolchain-3.1.11.sh	41e304ec75a26d3bcac7d1f9f2cb72fc07e6002d97f7de45f65

U-Boot

Note: The images come with a preinstalled U-Boot that supports 512 MB of RAM. If you have a module with 1 GB of RAM, you will have to *Install SPL and U-Boot* to unlock the full 1 GB of RAM.

Description	Download	Checksum (SHA256)
MLO (512 MB)	u-boot-spl.stm32-stm32mp157c-bytedevkit-basic	ffc3c38e453f7b8760b4edfabd0e6aa0c55fb3e386d8a5a80b90e3a12d0e900d
U-Boot (512 MB)	u-boot-stm32mp157c-bytedevkit-basic.img	c0fe5de015ceefa8b3e9a761007523b33fb0e0dddda9ee39d7c3d55382a13ccb
MLO (1 GB)	u-boot-spl.stm32-stm32mp157c-bytedevkit-1g_ram	99b88a246879e704f92a4f934a9641db8cf64262033e81dbc69b73b6bdba1d2(
U-Boot (1 GB)	u-boot-stm32mp157c-bytedevkit-1g_ram.img	8fa044532a61bfe82621bafad4b640710cb5406bc280f43e026a4709d269cb45

Image

How do you flash the image?

Attention:

- You need a microSD card with **at least 8GB** capacity.
- **All existing data** on the microSD card will be lost.
- **Do not format** the microSD card before flashing.

Windows

1. Unzip the file `bytesatwork-minimal-image-bytedevkit-stm32mp1.wic.gz` (e.g. with 7-zip)
2. Write the resulting file to the microSD card with a tool like Roadkils Disk Image

Linux

```
gunzip -c bytesatwork-minimal-image-bytedevkit-stm32mp1.wic.gz | dd of=/dev/mmcblk<X>␣
↪bs=8M conv=fdatasync status=progress
```

Hint: To improve write performance, you could use bmap-tools under Linux:

```
bmaptool copy bytesatwork-minimal-image-bytedevkit-stm32mp1.wic.gz /dev/mmcblk<X>
```

How do you build an image?

Use `repo` to download all necessary repositories:

```
$ mkdir -p ~/workdir/bytedevkit-stm32mp1/3.1; cd ~/workdir/bytedevkit-stm32mp1/3.1
$ repo init -u https://github.com/bytesatwork/bsp-platform-st.git -b dunfell
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for byteDEVKIT-stm32mp1:

```
$ cd ~/workdir/bytedevkit-stm32mp1/3.1
$ MACHINE=bytedevkit-stm32mp1 DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds the development image:

```
$ cd $BUILDDIR
$ bitbake bytesatwork-minimal-image
```

The output is found in:

```
~/workdir/bytedevkit-stm32mp1/3.1/build/tmp/deploy/images/bytedevkit-stm32mp1
```

Hint: For additional information about yocto images and how to build them, please visit: https://docs.yoctoproject.org/3.1.11/brief-yoctoprojectqs/brief-yoctoprojectqs.html#building-your-image.

How to modify the image

The image recipes can be found in `~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images`

This is relative to where you started the `repo` command to fetch all the sources.

Edit the minimal-image recipe `bytesatwork-minimal-image.bb`

Add the desired software-package to `IMAGE_INSTALL` variable, for example add `net-tools` to `bytesatwork-minimal-image.bb`

Rebuild the image by:

```
$ cd ~/workdir/<machine name>/<yocto version>
$ MACHINE=<machine name> DISTRO=poky-bytesatwork EULA=1 . setup-environment␣
↪build
$ bitbake bytesatwork-minimal-image
```

How to rename the image

If you want to rename or copy an image, simply rename or copy the image recipe by:

```
$ cd ~/workdir/<machine name>/<yocto version>/build/tmp/deploy/images/<machine␣
↪name>
$ cp bytesatwork-minimal-image.bb customer-example-image.bb
```

Troubleshooting

- **Image size is too small**

  If you encounter that your image size is too small to install additional software, please have a look at the `IMAGE_ROOTFS_SIZE` variable under `~/workdir/<machine-name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images/bytesatwork-minimal-image.bb`. Increase the size if necessary.

Toolchain

How do you install the toolchain?

Simply download the toolchain and execute the downloaded file, which is a self-extracting shell script.

Hint: If you encounter problems when trying to install the toolchain, make sure the downloaded toolchain is executable. Run `chmod +x /<path>/<toolchain-file>.sh` to make it executable.

Important:

The following tools need to be installed on your development system:

- `xz` (Debian package: `xz-utils`)
- `python` (any version)
- `gcc`

How do you use the toolchain?

Source the installed toolchain:

```
source /opt/poky-bytesatwork/3.1.11/environment-setup-cortexa7t2hf-neon-vfpv4-poky-linux-
↪gnueabi
```

Check if Cross-compiler is available in environment:

```
echo $CC
```

You should see the following output:

```
arm-poky-linux-gnueabi-gcc -mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7 -
↪fstack-protector-strong -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror=format-
↪security --sysroot=/opt/poky-bytesatwork/3.1.11/sysroots/cortexa7t2hf-neon-vfpv4-poky-
↪linux-gnueabi
```

Crosscompile the source code, e.g. by:

```
$CC helloworld.c -o helloworld
```

Check generated binary:

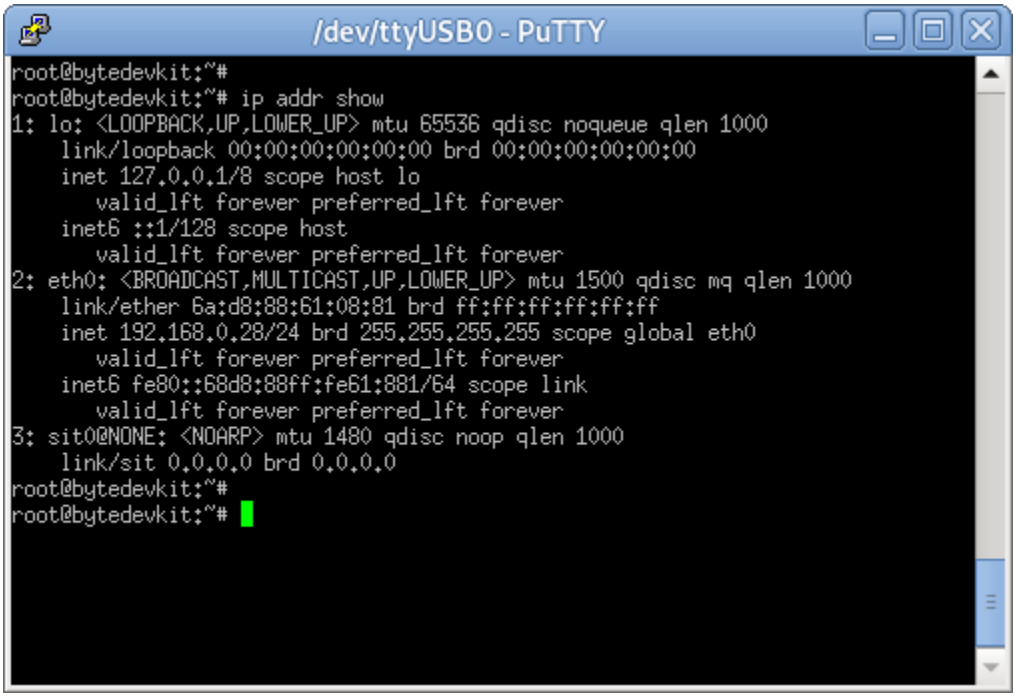
```
file helloworld
```

The output that is shown in prompt afterwards:

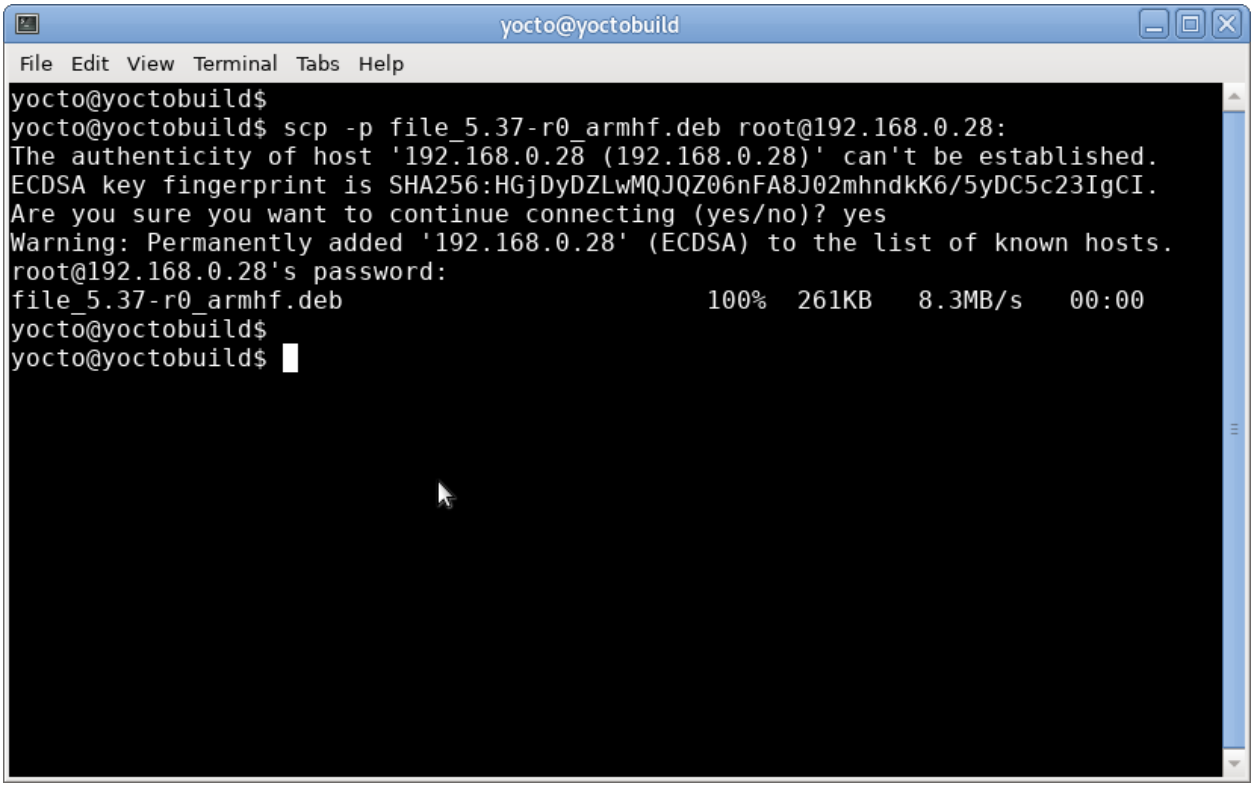
```
helloworld: ELF 32-bit LSB pie executable, ARM, EABI5 version 1
```

How to bring your binary to the target?

1. Connect the embedded device's ethernet to your LAN
2. Determine the embedded target IP address by `ip addr show`



3. Copy your binary, e.g. `helloworld` to the target by `scp helloworld root@<ip address of target>:/tmp`



4. Run `chmod +x` on the target to make your binary executable: `chmod +x /<path>/<binary name>`
5. Run your binary on the target: `/<path>/<binary name>`

How do you build a toolchain?

```
$ cd ~/workdir/bytedevkit-stm32mp1/3.1
$ repo init -u https://github.com/bytesatwork/bsp-platform-st.git -b dunfell
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for byteDEVKIT-stm32mp1:

```
$ cd ~/workdir/bytedevkit-stm32mp1/3.1
$ MACHINE=bytedevkit-stm32mp1 DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds an installable toolchain:

```
$ cd $BUILDDIR
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The toolchain is located under:

```
~/workdir/bytedevkit-stm32mp1/3.1/build/tmp/deploy/sdk
```

How to modify your toolchain

Currently the bytesatwork toolchain is generated out of the bytesatwork-minimal-image recipe. If you want to add additional libraries and development headers to customize the toolchain, you need to modify the bytesatwork-minimal-image recipe. It can be found under `~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images`

For example if you want to develop your own ftp client and you need libftp and the corresponding header files, edit the recipe `bytesatwork-minimal-image.bb` and add `ftplib` to the `IMAGE_INSTALL` variable.

This will provide the ftplib libraries and development headers in the toolchain. After adding additional software components, the toolchain needs to be rebuilt by:

```
$ cd ~/workdir/<machine name>/<yocto version>
$ MACHINE=<machine> DISTRO=poky-bytesatwork EULA=1 . setup-environment build
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The newly generated toolchain will be available under:

```
~/workdir/<machine name>/<yocto version>/build/tmp/deploy/sdk
```

For additional information, please visit: https://docs.yoctoproject.org/3.1.11/overview-manual/overview-manual-concepts.html#cross-development-toolchain-generation.

Kernel

Download the Linux Kernel

Device	Branch	git URL
bytedevkit-stm32mp1	baw-v5.10-stm32mp-r1	https://github.com/bytesatwork/linux-stm32mp.git

---

Build the Linux Kernel

For both targets, an ARM toolchain is necessary. You can use the provided toolchain from *Toolchain* or any compatible toolchain (e.g. from your distribution)

Important:

The following tools need to be installed on your development system:

- `git`
- `make`
- `bc`

Note: The following instructions assume, you installed the provided toolchain for the respective target.

Important:

The following tools need to be installed on your development system:

- OpenSSL headers (Debian package: `libssl-dev`)
- `depmod` (Debian package: `kmod`)

1. Download kernel sources

   Download the appropriate kernel from *Download the Linux Kernel*.

2. Source toolchain

```
source /opt/poky-bytesatwork/3.1.11/environment-setup-cortexa7t2hf-neon-vfpv4-poky-
↪linux-gnueabi
```

3. Create defconfig

```
make multi_v7_defconfig
scripts/kconfig/merge_config.sh -m -r .config arch/arm/configs/fragment-*
make olddefconfig
```

4. Build Linux kernel

```
make LOADADDR=0xC2000040 -j `nproc` uImage stm32mp157c-bytedevkit.dtb modules
```

5. Install kernel and device tree

   To use the newly created kernel, device tree and/or module, the necessary files need to be installed on the target. This can be done either via Ethernet (e.g. `scp`) or by copying the files to the SD card.

   Note: For scp installation: Don't forget to mount /boot on the target.

File	Target path	Target partition
`arch/arm/boot/uImage`	`/boot/uImage`	`/dev/mmcblk0p4`
`arch/arm/boot/dts/stm32mp157c-bytedevkit.dtb`	`/boot/stm32mp157c-bytedevkit.dtb`	`/dev/mmcblk0p4`

   Note:

   After installing a new kernel, it often fails to load modules, as the _signature_ of the kernel changed and it fails to find its corresponding modules folder. This issue can often be resolved with a symlink:

```
ln -s /lib/modules/<EXISTING FOLDER> /lib/modules/`uname -r`
```

   Otherwise, please follow the instructions to copy the kernel modules

6. Install kernel modules

To copy all available modules to the target, it's best to deploy them locally first and then copy all modules to the target.

```
mkdir /tmp/bytedevkit-stm32mp1
make INSTALL_MOD_PATH=/tmp/bytedevkit-stm32mp1 modules_install
```

Now you can copy the content of the folder `/tmp/bytedevkit-stm32mp1` into the target's root folder (`/`) which is partition `/dev/mmcblk0p5`.

U-Boot

Download U-Boot

Device	Branch	git URL
bytedevkit-stm32mp1	baw-v2020.01-stm32mp-r1	https://github.com/bytesatwork/u-boot-stm32mp

Build U-Boot

To compile U-Boot, an ARM toolchain is necessary. You can use the provided toolchain from *Toolchain* or any compatible toolchain (e.g. from your distribution)

Important:

The following tools need to be installed on your development system:

- `git`
- `make`
- `bc`

Note: The following instructions assume, you installed the provided toolchain for the respective target.

1. Download U-Boot sources

   Download the appropriate U-Boot from *Download U-Boot*.

2. Source toolchain

   ```
   source /opt/poky-bytesatwork/3.1.11/environment-setup-cortexa7t2hf-neon-vfpv4-poky-
   ↪linux-gnueabi
   ```

3. Create defconfig

```
make stm32mp157_bytedevkit_defconfig
```

Note: For the 1 GB RAM variant, use `make stm32mp157_bytedevkit_1g_defconfig` instead.

4. Build U-Boot and SPL

```
make -j `nproc`
```

Install SPL and U-Boot

To use the newly created U-Boot, the necessary files need to be installed on the SD card. This can be done either on the host or on the target.

File	Target partition
u-boot-spl.stm32	/dev/mmcblk0p1
u-boot-spl.stm32	/dev/mmcblk0p2
u-boot.img	/dev/mmcblk0p3

You need to write the to the respective "raw" partition, either on the host system or the target system:

```
dd if=u-boot-spl.stm32 of=/dev/mmcblk0p1
dd if=u-boot-spl.stm32 of=/dev/mmcblk0p2
dd if=u-boot.img of=/dev/mmcblk0p3
```

The next time the target is reset, it will start with the new U-Boot.

bytesatwork

5.4.3 byteDEVKIT-stm32mp1 (Yocto 3.2)

Image

Where do you get the SD card image?

Device	Yocto Version	Download	Checksum (SHA256)
bytedevkit-stm32mp1	Yocto 3.2.2	bytesatwork-minimal-image-bytedevkit-stm32mp1.wic.gz (wic.bmap)	efc3ed1e56d5c017c7e72549fab30d9909ce24e63c8b0192a8a53

Hint: Updating from an older image? You can update your older image by using: `apt-get update` and `apt-get upgrade`.

1. check for new version in the table above

2. edit `/etc/apt/sources.list` and point to the new package feed
3. run `apt-get update; apt-get upgrade`

As the yocto framework is based on several packages from various projects or suppliers, it is not guaranteed that an incremental upgrade by `apt-get upgrade` works automatically. Some manual adjustments might be needed.

How do you flash the image?

Attention:

- You need a microSD card with **at least 8GB** capacity.
- **All existing data** on the microSD card will be lost.
- **Do not format** the microSD card before flashing.

Windows

1. Unzip the file `bytesatwork-minimal-image-bytedevkit-stm32mp1.wic.gz` (e.g. with 7-zip)
2. Write the resulting file to the microSD card with a tool like Roadkils Disk Image

Linux

```
gunzip -c bytesatwork-minimal-image-bytedevkit-stm32mp1.wic.gz | dd of=/dev/mmcblk<X>␣
↪bs=8M conv=fdatasync status=progress
```

Hint: To improve write performance, you could use bmap-tools under Linux:

```
bmaptool copy bytesatwork-minimal-image-bytedevkit-stm32mp1.wic.gz /dev/mmcblk<X>
```

How do you build an image?

Use `repo` to download all necessary repositories:

```
$ mkdir -p ~/workdir/bytedevkit-stm32mp1/3.2; cd ~/workdir/bytedevkit-stm32mp1/3.2
$ repo init -u https://github.com/bytesatwork/bsp-platform-st.git -b gatesgarth
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for byteDEVKIT-stm32mp1:

```
$ cd ~/workdir/bytedevkit-stm32mp1/3.2
$ MACHINE=bytedevkit-stm32mp1 DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds the development image:

```
$ cd $BUILDDIR
$ bitbake bytesatwork-minimal-image
```

The output is found in:

```
~/workdir/bytedevkit-stm32mp1/3.2/build/tmp/deploy/images/bytedevkit-stm32mp1
```

Hint: For additional information about yocto images and how to build them, please visit: https://docs.yoctoproject.org/3.2.2/singleindex.html#building-your-image

How to modify the image

The image recipes can be found in `~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images`

This is relative to where you started the `repo` command to fetch all the sources.

Edit the minimal-image recipe `bytesatwork-minimal-image.bb`

Add the desired software-package to `IMAGE_INSTALL` variable, for example add `net-tools` to `bytesatwork-minimal-image.bb`

Rebuild the image by:

```
$ cd ~/workdir/<machine name>/<yocto version>
$ MACHINE=<machine name> DISTRO=poky-bytesatwork EULA=1 . setup-environment␣
↪build
$ bitbake bytesatwork-minimal-image
```

How to rename the image

If you want to rename or copy an image, simply rename or copy the image recipe by:

```
$ cd ~/workdir/<machine name>/<yocto version>/build/tmp/deploy/images/<machine␣
↪name>
$ cp bytesatwork-minimal-image.bb customer-example-image.bb
```

Troubleshooting

- **Image size is too small**

  If you encounter that your image size is too small to install additional software, please have a look at the `IMAGE_ROOTFS_SIZE` variable under `~/workdir/<machine-name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images/bytesatwork-minimal-image.bb`. Increase the size if necessary.

Toolchain

Where do you get the toolchain?

Device	Yocto Version	Download	Checksum (SHA256)
bytedevkit-stm32mp1	Yocto 3.2.2	poky-bytesatwork-glibc-x86_64-bytesatwork-minimal-image-cortexa7t2hf-neon-vfpv4-bytedevkit-stm32mp1-toolchain-3.2.2.sh	8f8fc481de6d891392a3b3e5edbfcee58788a47366f45

How do you install the toolchain?

Simply download the toolchain and execute the downloaded file, which is a self-extracting shell script.

Hint: If you encounter problems when trying to install the toolchain, make sure the downloaded toolchain is executable. Run `chmod +x /<path>/<toolchain-file>.sh` to make it executable.

Important:

The following tools need to be installed on your development system:

- `xz` (Debian package: `xz-utils`)
- `python` (any version)
- `gcc`

How do you use the toolchain?

Source the installed toolchain:

```
source /opt/poky-bytesatwork/3.2.2/environment-setup-cortexa7t2hf-neon-vfpv4-poky-linux-
↪gnueabi
```

Check if Cross-compiler is available in environment:

```
echo $CC
```

You should see the following output:

```
arm-poky-linux-gnueabi-gcc -mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7 -
↪fstack-protector-strong -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror=format-
↪security --sysroot=/opt/poky-bytesatwork/3.2.2/sysroots/cortexa7t2hf-neon-vfpv4-poky-
↪linux-gnueabi
```

Crosscompile the source code, e.g. by:

```
$CC helloworld.c -o helloworld
```

Check generated binary:

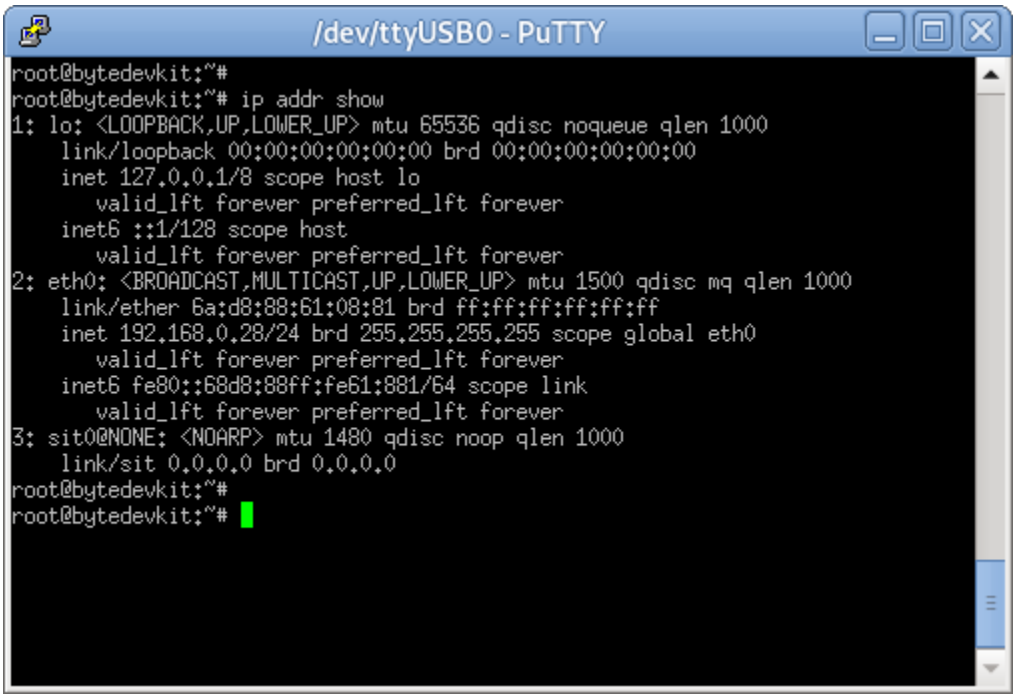
```
file helloworld
```

The output that is shown in prompt afterwards:

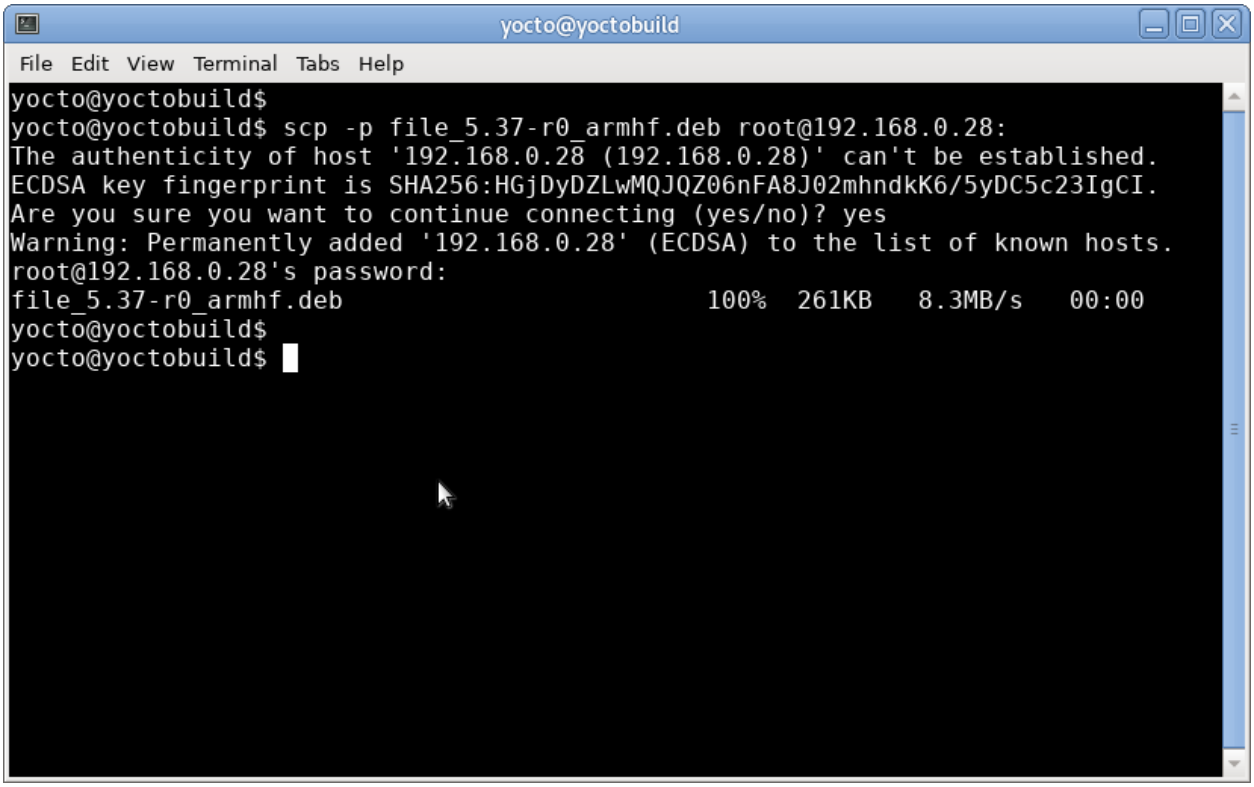
```
helloworld: ELF 32-bit LSB pie executable, ARM, EABI5 version 1
```

How to bring your binary to the target?

1. Connect the embedded device's ethernet to your LAN
2. Determine the embedded target IP address by `ip addr show`



3. Copy your binary, e.g. `helloworld` to the target by `scp helloworld root@<ip address of target>:/tmp`



4. Run `chmod +x` on the target to make your binary executable: `chmod +x /<path>/<binary name>`
5. Run your binary on the target: `/<path>/<binary name>`

How do you build a toolchain?

```
$ cd ~/workdir/bytedevkit-stm32mp1/3.2
$ repo init -u https://github.com/bytesatwork/bsp-platform-st.git -b gatesgarth
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for byteDEVKIT-stm32mp1:

```
$ cd ~/workdir/bytedevkit-stm32mp1/3.2
$ MACHINE=bytedevkit-stm32mp1 DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds an installable toolchain:

```
$ cd $BUILDDIR
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The toolchain is located under:

```
~/workdir/bytedevkit-stm32mp1/3.2/build/tmp/deploy/sdk
```

How to modify your toolchain

Currently the bytesatwork toolchain is generated out of the bytesatwork-minimal-image recipe. If you want to add additional libraries and development headers to customize the toolchain, you need to modify the bytesatwork-minimal-image recipe. It can be found under `~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images`

For example if you want to develop your own ftp client and you need libftp and the corresponding header files, edit the recipe `bytesatwork-minimal-image.bb` and add `ftplib` to the `IMAGE_INSTALL` variable.

This will provide the ftplib libraries and development headers in the toolchain. After adding additional software components, the toolchain needs to be rebuilt by:

```
$ cd ~/workdir/<machine name>/<yocto version>
$ MACHINE=<machine> DISTRO=poky-bytesatwork EULA=1 . setup-environment build
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The newly generated toolchain will be available under:

```
~/workdir/<machine name>/<yocto version>/build/tmp/deploy/sdk
```

For additional information, please visit: https://docs.yoctoproject.org/3.2.2/overview-manual/overview-manual-concepts.html#cross-development-toolchain-generation

Kernel

Download the Linux Kernel

Device	Branch	git URL
bytedevkit-stm32mp1	baw-v5.4-stm32mp-r2	https://github.com/bytesatwork/linux-stm32mp.git

---

Build the Linux Kernel

For both targets, an ARM toolchain is necessary. You can use the provided toolchain from *Where do you get the toolchain?* or any compatible toolchain (e.g. from your distribution)

Important:

The following tools need to be installed on your development system:

- `git`
- `make`
- `bc`

Note: The following instructions assume, you installed the provided toolchain for the respective target.

Important:

The following tools need to be installed on your development system:

- OpenSSL headers (Debian package: `libssl-dev`)
- `depmod` (Debian package: `kmod`)

1. Download kernel sources

   Download the appropriate kernel from *Download the Linux Kernel*.

2. Source toolchain

```
source /opt/poky-bytesatwork/3.2.2/environment-setup-cortexa7t2hf-neon-vfpv4-poky-
↪linux-gnueabi
```

3. Create defconfig

```
make multi_v7_defconfig
scripts/kconfig/merge_config.sh -m -r .config arch/arm/configs/fragment-*
make olddefconfig
```

4. Build Linux kernel

```
make LOADADDR=0xC2000040 -j `nproc` uImage stm32mp157c-bytedevkit.dtb modules
```

5. Install kernel and device tree

   To use the newly created kernel, device tree and/or module, the necessary files need to be installed on the target. This can be done either via Ethernet (e.g. `scp`) or by copying the files to the SD card.

   Note: For scp installation: Don't forget to mount /boot on the target.

File	Target path	Target partition
`arch/arm/boot/uImage`	`/boot/uImage`	`/dev/mmcblk0p4`
`arch/arm/boot/dts/stm32mp157c-bytedevkit.dtb`	`/boot/stm32mp157c-bytedevkit.dtb`	`/dev/mmcblk0p4`

   Note:

   After installing a new kernel, it often fails to load modules, as the _signature_ of the kernel changed and it fails to find its corresponding modules folder. This issue can often be resolved with a symlink:

```
ln -s /lib/modules/<EXISTING FOLDER> /lib/modules/`uname -r`
```

   Otherwise, please follow the instructions to copy the kernel modules

6. Install kernel modules

To copy all available modules to the target, it's best to deploy them locally first and then copy all modules to the target.

```
mkdir /tmp/bytedevkit-stm32mp1
make INSTALL_MOD_PATH=/tmp/bytedevkit-stm32mp1 modules_install
```

Now you can copy the content of the folder `/tmp/bytedevkit-stm32mp1` into the target's root folder (`/`) which is partition `/dev/mmcblk0p5`.

U-Boot

Download U-Boot

Device	Branch	git URL
bytedevkit-stm32mp1	baw-v2020.01-stm32mp-r2	https://github.com/bytesatwork/u-boot-stm32mp

Build U-Boot

To compile U-Boot, an ARM toolchain is necessary. You can use the provided toolchain from *Where do you get the toolchain?* or any compatible toolchain (e.g. from your distribution)

Important:

The following tools need to be installed on your development system:

- `git`
- `make`
- `bc`

Note: The following instructions assume, you installed the provided toolchain for the respective target.

1. Download U-Boot sources

   Download the appropriate U-Boot from *Download U-Boot*.

2. Source toolchain

   ```
   source /opt/poky-bytesatwork/3.2.2/environment-setup-cortexa7t2hf-neon-vfpv4-poky-
   ↪linux-gnueabi
   ```

3. Create defconfig

```
make stm32mp157_bytedevkit_defconfig
```

Note: For the 1 GB RAM variant, use `make stm32mp157_bytedevkit_1g_defconfig` instead.

4. Build U-Boot and SPL

```
make -j `nproc`
```

5. Install SPL and U-Boot

   To use the newly created U-Boot, the necessary files need to be installed on the SD card. This can be done either on the host or on the target.

File	Target partition
u-boot-spl.stm32	/dev/mmcblk0p1
u-boot-spl.stm32	/dev/mmcblk0p2
u-boot.img	/dev/mmcblk0p3

   You need to write the to the respective "raw" partition, either on the host system or the target system:

```
dd if=u-boot-spl.stm32 of=/dev/mmcblk0p1
dd if=u-boot-spl.stm32 of=/dev/mmcblk0p2
dd if=u-boot.img of=/dev/mmcblk0p3
```

   The next time the target is reset, it will start with the new U-Boot.

5.4.4 byteDEVKIT (Yocto 3.0)

Image

Where do you get the SD card image?

Device	Yocto Version	Download	Checksum (SHA256)
byteDEVKIT	Yocto 3.0.3	bytesatwork-minimal-image-bytedevkit.wic.gz (wic.bmap)	1c1d442ef80de24f3bb02704880cf8c2124c88008aefca0264bf5850bd

Hint: Updating from an older image? You can update your older image by using: `apt-get update` and `apt-get upgrade`.

1. check for new version in the table above
2. edit `/etc/apt/sources.list` and point to the new package feed

3. run `apt-get update; apt-get upgrade`

As the yocto framework is based on several packages from various projects or suppliers, it is not guaranteed that an incremental upgrade by `apt-get upgrade` works automatically. Some manual adjustments might be needed.

How do you flash the image?

Attention:

- You need a microSD card with **at least 8GB** capacity.
- **All existing data** on the microSD card will be lost.
- **Do not format** the microSD card before flashing.

Windows

1. Unzip the file `bytesatwork-minimal-image-bytedevkit.wic.gz` (e.g. with 7-zip)
2. Write the resulting file to the microSD card with a tool like Roadkils Disk Image

Linux

```
gunzip -c bytesatwork-minimal-image-bytedevkit.wic.gz | dd of=/dev/mmcblk<X> bs=8M␣
↪conv=fdatasync status=progress
```

Hint: To improve write performance, you could use bmap-tools under Linux:

```
bmaptool copy bytesatwork-minimal-image-bytedevkit.wic.gz /dev/mmcblk<X>
```

How do you build an image?

Use `repo` to download all necessary repositories:

```
$ mkdir -p ~/workdir/bytedevkit/3.0; cd ~/workdir/bytedevkit/3.0
$ repo init -u https://github.com/bytesatwork/bsp-platform-st.git -b zeus
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for byteDEVKIT:

```
$ cd ~/workdir/bytedevkit/3.0
$ MACHINE=bytedevkit DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds the development image:

```
$ cd $BUILDDIR
$ bitbake bytesatwork-minimal-image
```

The output is found in:

```
~/workdir/bytedevkit/3.0/build/tmp/deploy/images/bytedevkit
```

Hint: For additional information about yocto images and how to build them, please visit: https://www.yoctoproject.org/docs/3.0/mega-manual/mega-manual.html#brief-building-your-image

How to modify the image

The image recipes can be found in `~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images`

This is relative to where you started the `repo` command to fetch all the sources.

Edit the minimal-image recipe `bytesatwork-minimal-image.bb`

Add the desired software-package to `IMAGE_INSTALL` variable, for example add `net-tools` to `bytesatwork-minimal-image.bb`

Rebuild the image by:

```
$ cd ~/workdir/<machine name>/<yocto version>
$ MACHINE=<machine name> DISTRO=poky-bytesatwork EULA=1 . setup-environment␣
↪build
$ bitbake bytesatwork-minimal-image
```

How to rename the image

If you want to rename or copy an image, simply rename or copy the image recipe by:

```
$ cd ~/workdir/<machine name>/<yocto version>/build/tmp/deploy/images/<machine␣
↪name>
$ cp bytesatwork-minimal-image.bb customer-example-image.bb
```

Troubleshooting

- **Image size is too small**

  If you encounter that your image size is too small to install additional software, please have a look at the `IMAGE_ROOTFS_SIZE` variable under `~/workdir/<machine-name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images/bytesatwork-minimal-image.bb`. Increase the size if necessary.

Toolchain

Where do you get the toolchain?

Device	Yocto Version	Download	Checksum (SHA256)
byteDE-VKIT	Yocto 3.0.3	poky-bytesatwork-glibc-x86_64-bytesatwork-minimal-image-cortexa7t2hf-neon-vfpv4-bytedevkit-toolchain-3.0.3.sh	fe182429d8bf6d91ca2a556452894612b273141fd168af5

How do you install the toolchain?

Simply download the toolchain and execute the downloaded file, which is a self-extracting shell script.

Hint: If you encounter problems when trying to install the toolchain, make sure the downloaded toolchain is executable. Run `chmod +x /<path>/<toolchain-file>.sh` to make it executable.

Important:

The following tools need to be installed on your development system:

- `xz` (Debian package: `xz-utils`)
- `python` (any version)
- `gcc`

How do you use the toolchain?

Source the installed toolchain:

```
source /opt/poky-bytesatwork/3.0.3/environment-setup-cortexa7t2hf-neon-vfpv4-poky-linux-
↪gnueabi
```

Check if Cross-compiler is available in environment:

```
echo $CC
```

You should see the following output:

```
arm-poky-linux-gnueabi-gcc -mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7 -
↪fstack-protector-strong -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror=format-
↪security --sysroot=/opt/poky-bytesatwork/3.0.3/sysroots/cortexa7t2hf-neon-vfpv4-poky-
↪linux-gnueabi
```

Crosscompile the source code, e.g. by:

```
$CC helloworld.c -o helloworld
```

Check generated binary:

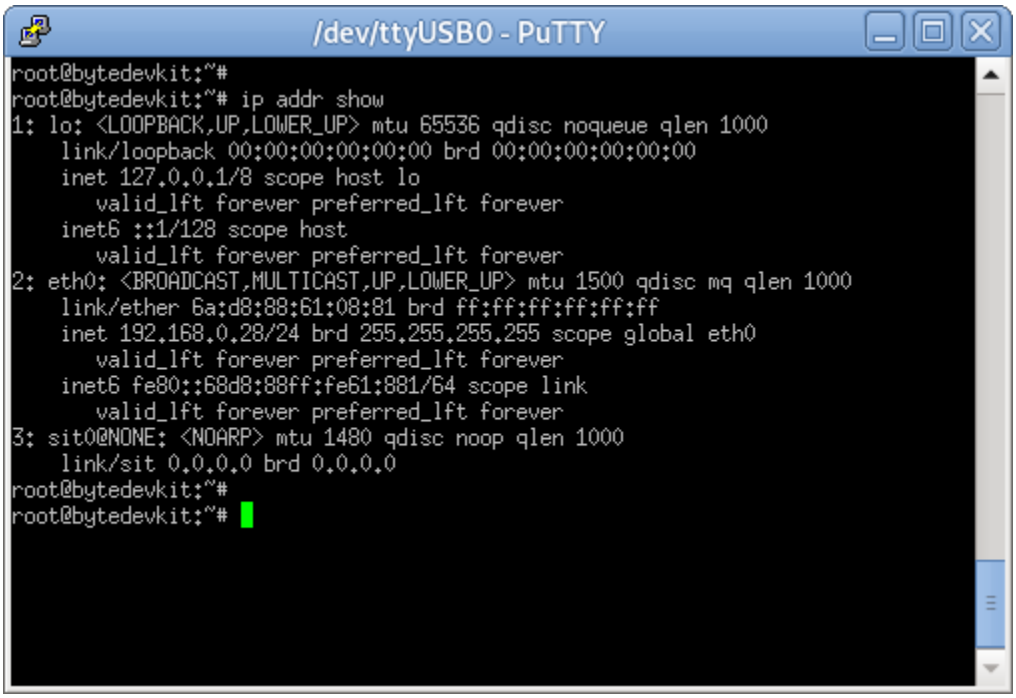
```
file helloworld
```

The output that is shown in prompt afterwards:

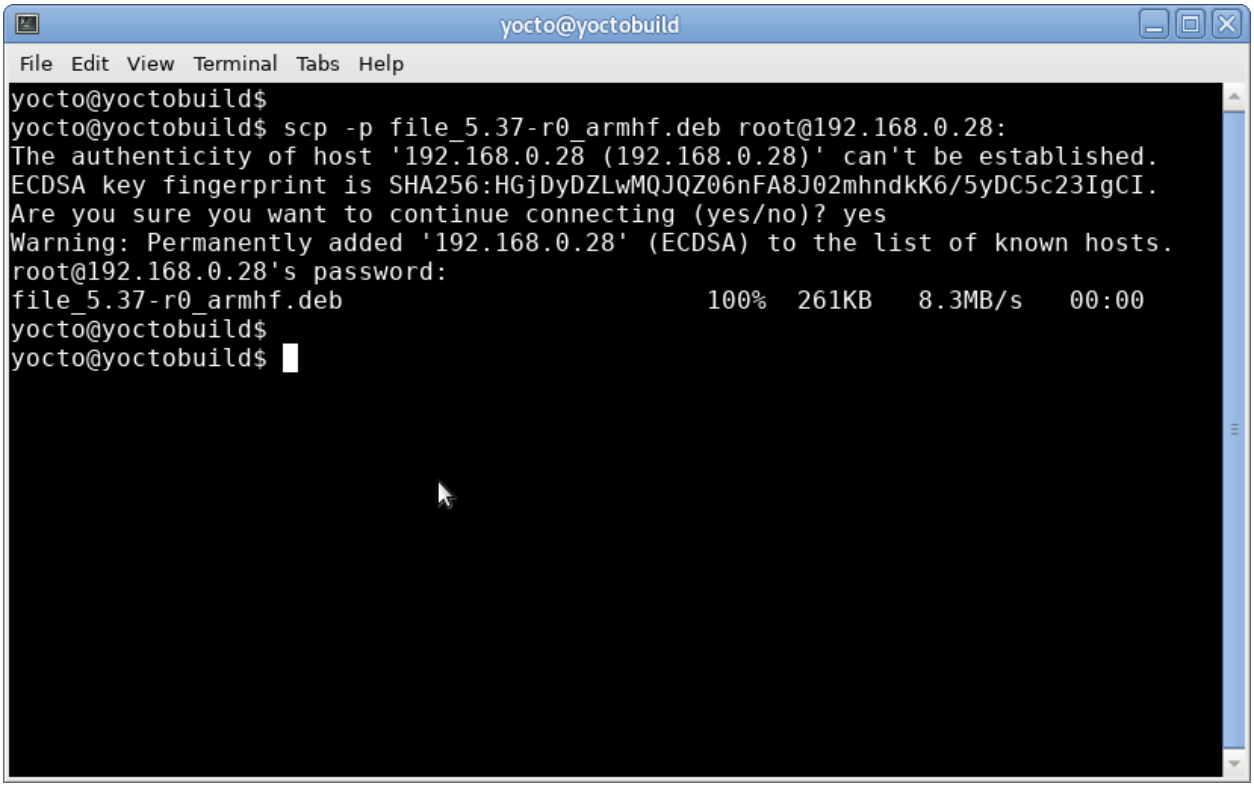
```
helloworld: ELF 32-bit LSB pie executable, ARM, EABI5 version 1
```

How to bring your binary to the target?

1. Connect the embedded device's ethernet to your LAN
2. Determine the embedded target IP address by `ip addr show`



3. Copy your binary, e.g. `helloworld` to the target by `scp helloworld root@<ip address of target>:/tmp`



4. Run `chmod +x` on the target to make your binary executable: `chmod +x /<path>/<binary name>`
5. Run your binary on the target: `/<path>/<binary name>`

How do you build a toolchain?

```
$ cd ~/workdir/bytedevkit/3.0
$ repo init -u https://github.com/bytesatwork/bsp-platform-st.git -b zeus
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for byteDEVKIT:

```
$ cd ~/workdir/bytedevkit/3.0
$ MACHINE=bytedevkit DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds an installable toolchain:

```
$ cd $BUILDDIR
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The toolchain is located under:

```
~/workdir/bytedevkit/3.0/build/tmp/deploy/sdk
```

How to modify your toolchain

Currently the bytesatwork toolchain is generated out of the bytesatwork-minimal-image recipe. If you want to add additional libraries and development headers to customize the toolchain, you need to modify the bytesatwork-minimal-image recipe. It can be found under `~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images`

For example if you want to develop your own ftp client and you need libftp and the corresponding header files, edit the recipe `bytesatwork-minimal-image.bb` and add `ftplib` to the `IMAGE_INSTALL` variable.

This will provide the ftplib libraries and development headers in the toolchain. After adding additional software components, the toolchain needs to be rebuilt by:

```
$ cd ~/workdir/<machine name>/<yocto version>
$ MACHINE=<machine> DISTRO=poky-bytesatwork EULA=1 . setup-environment build
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The newly generated toolchain will be available under:

```
~/workdir/<machine name>/<yocto version>/build/tmp/deploy/sdk
```

For additional information, please visit: https://www.yoctoproject.org/docs/3.0.3/overview-manual/overview-manual.html#cross-development-toolchain-generation

Kernel

Download the Linux Kernel

Device	Branch	git URL
byteDEVKIT	baw-v4.19-stm32mp	https://github.com/bytesatwork/linux-stm32mp.git

---

Build the Linux Kernel

For both targets, an ARM toolchain is necessary. You can use the provided toolchain from *Where do you get the toolchain?* or any compatible toolchain (e.g. from your distribution)

Important:

The following tools need to be installed on your development system:

- `git`
- `make`
- `bc`

Note: The following instructions assume, you installed the provided toolchain for the respective target.

Important:

The following tools need to be installed on your development system:

- OpenSSL headers (Debian package: `libssl-dev`)
- `depmod` (Debian package: `kmod`)

1. Download kernel sources

   Download the appropriate kernel from *Download the Linux Kernel*.

2. Source toolchain

```
source /opt/poky-bytesatwork/3.0.3/environment-setup-cortexa7t2hf-neon-vfpv4-poky-
↪linux-gnueabi
```

3. Create defconfig

```
make multi_v7_defconfig
scripts/kconfig/merge_config.sh -m -r .config arch/arm/configs/fragment-*
make olddefconfig
```

4. Build Linux kernel

```
make LOADADDR=0xC2000040 -j `nproc` uImage stm32mp157c-bytedevkit-v1-1.dtb modules
```

5. Install kernel and device tree

   To use the newly created kernel, device tree and/or module, the necessary files need to be installed on the target. This can be done either via Ethernet (e.g. `scp`) or by copying the files to the SD card.

   Note: For scp installation: Don't forget to mount /boot on the target.

File	Target path	Target partition
`arch/arm/boot/uImage`	`/boot/uImage`	`/dev/mmcblk0p4`
`arch/arm/boot/dts/stm32mp157c-bytedevkit-v1-1.dtb`	`/boot/stm32mp157c-bytedevkit.dtb`	`/dev/mmcblk0p4`

   Note:

   After installing a new kernel, it often fails to load modules, as the _signature_ of the kernel changed and it fails to find its corresponding modules folder. This issue can often be resolved with a symlink:

```
ln -s /lib/modules/<EXISTING FOLDER> /lib/modules/`uname -r`
```

   Otherwise, please follow the instructions to copy the kernel modules

6. Install kernel modules

To copy all available modules to the target, it's best to deploy them locally first and then copy all modules to the target.

```
mkdir /tmp/bytedevkit
make INSTALL_MOD_PATH=/tmp/bytedevkit modules_install
```

Now you can copy the content of the folder `/tmp/bytedevkit` into the target's root folder (`/`) which is partition `/dev/mmcblk0p5`.

bytesatwork

5.4.5 bytePANEL (Yocto 3.0)

Image

Where do you get the SD card image?

Device	Yocto Version	Download	Checksum (SHA256)
bytePAN	Yocto 3.0	bytesatwork-minimal-image-bytepanel-emmc.wic.gz (wic.bmap)	e3e166f28fb815b09c6372bbcae4b4c8fcd00f93e57e96084bdee90c2

Hint: Updating from an older image? You can update your older image by using: `apt-get update` and `apt-get upgrade`.

1. check for new version in the table above
2. edit `/etc/apt/sources.list` and point to the new package feed
3. run `apt-get update; apt-get upgrade`

As the yocto framework is based on several packages from various projects or suppliers, it is not guaranteed that an incremental upgrade by `apt-get upgrade` works automatically. Some manual adjustments might be needed.

How do you flash the image?

Attention:

- You need a microSD card with **at least 8GB** capacity.
- **All existing data** on the microSD card will be lost.
- **Do not format** the microSD card before flashing.

Windows

1. Unzip the file `bytesatwork-minimal-image-bytepanel-emmc.wic.gz` (e.g. with 7-zip)
2. Write the resulting file to the microSD card with a tool like Roadkils Disk Image

Linux

```
gunzip -c bytesatwork-minimal-image-bytepanel-emmc.wic.gz | dd of=/dev/mmcblk<X> bs=8M␣
↪conv=fdatasync status=progress
```

Hint: To improve write performance, you could use bmap-tools under Linux:

bmaptool copy bytesatwork-minimal-image-bytepanel-emmc.wic.gz /dev/mmcblk<X>

How do you build an image?

Use `repo` to download all necessary repositories:

```
$ mkdir -p ~/workdir/bytepanel/3.0; cd ~/workdir/bytepanel/3.0
$ repo init -u https://github.com/bytesatwork/bsp-platform-ti.git -b zeus
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for bytePANEL:

```
$ cd ~/workdir/bytepanel/3.0
$ MACHINE=bytepanel DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds the development image:

```
$ cd $BUILDDIR
$ bitbake bytesatwork-minimal-image
```

The output is found in:

```
~/workdir/bytepanel/3.0/build/tmp/deploy/images/bytepanel
```

Hint: For additional information about yocto images and how to build them, please visit: https://www.yoctoproject.org/docs/3.0/mega-manual/mega-manual.html#brief-building-your-image

---

How to modify the image

The image recipes can be found in `~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images`

This is relative to where you started the `repo` command to fetch all the sources.

Edit the minimal-image recipe `bytesatwork-minimal-image.bb`

Add the desired software-package to `IMAGE_INSTALL` variable, for example add `net-tools` to `bytesatwork-minimal-image.bb`

Rebuild the image by:

```
$ cd ~/workdir/<machine name>/<yocto version>
$ MACHINE=<machine name> DISTRO=poky-bytesatwork EULA=1 . setup-environment␣
↪build
$ bitbake bytesatwork-minimal-image
```

How to rename the image

If you want to rename or copy an image, simply rename or copy the image recipe by:

```
$ cd ~/workdir/<machine name>/<yocto version>/build/tmp/deploy/images/<machine␣
↪name>
$ cp bytesatwork-minimal-image.bb customer-example-image.bb
```

Troubleshooting

- **Image size is too small**

  If you encounter that your image size is too small to install additional software, please have a look at the `IMAGE_ROOTFS_SIZE` variable under `~/workdir/<machine-name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images/bytesatwork-minimal-image.bb`. Increase the size if necessary.

Toolchain

Where do you get the toolchain?

Device	Yocto Version	Download	Checksum (SHA256)
bytePA	Yocto 3.0	poky-bytesatwork-glibc-x86_64-bytesatwork-minimal-image-armv7at2hf-neon-bytepanel-emmc-toolchain-3.0.2.sh	a90763d7ff408e9e5f0556b051eccd3ea85c43406099c9a6

How do you install the toolchain?

Simply download the toolchain and execute the downloaded file, which is a self-extracting shell script.

Hint: If you encounter problems when trying to install the toolchain, make sure the downloaded toolchain is executable. Run `chmod +x /<path>/<toolchain-file>.sh` to make it executable.

Important:

The following tools need to be installed on your development system:

- `xz` (Debian package: `xz-utils`)
- `python` (any version)
- `gcc`

How do you use the toolchain?

Source the installed toolchain:

```
source /opt/poky-bytesatwork/3.0.2/environment-setup-armv7at2hf-neon-poky-linux-gnueabi
```

Check if Cross-compiler is available in environment:

```
echo $CC
```

You should see the following output:

```
arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb -mfpu=neon -mfloat-abi=hard --sysroot=/
↪opt/poky-bytesatwork/3.0.2/sysroots/armv7at2hf-neon-poky-linux-gnueabi
```

Cross-compile the source code, e.g. by:

```
$CC helloworld.c -o helloworld
```

Check generated binary:

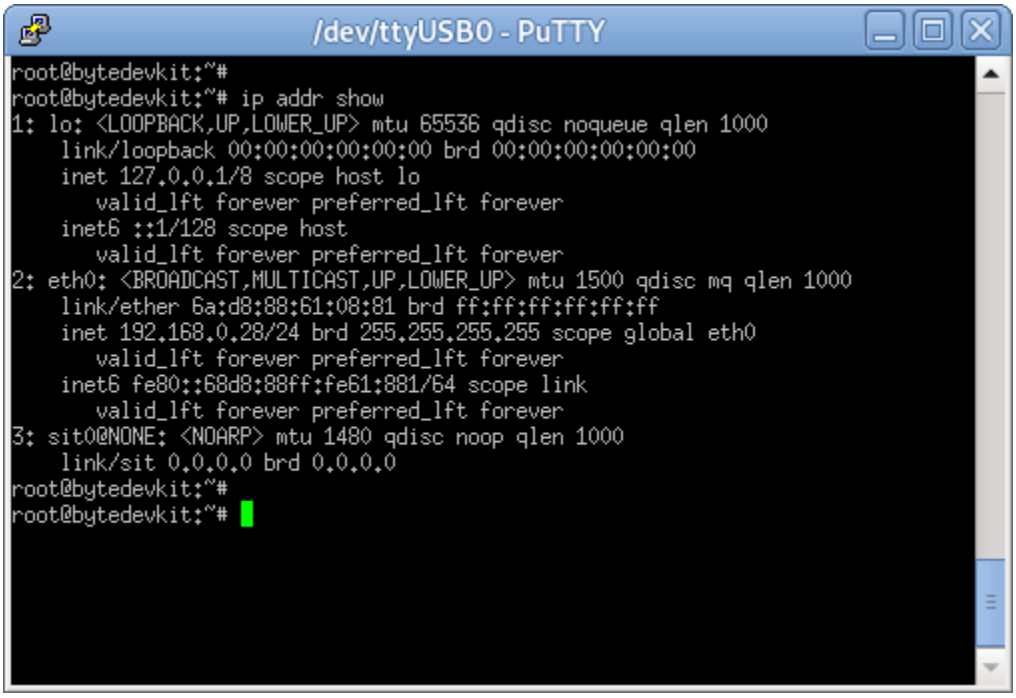
```
file helloworld
```

The output that is shown in prompt afterwards:

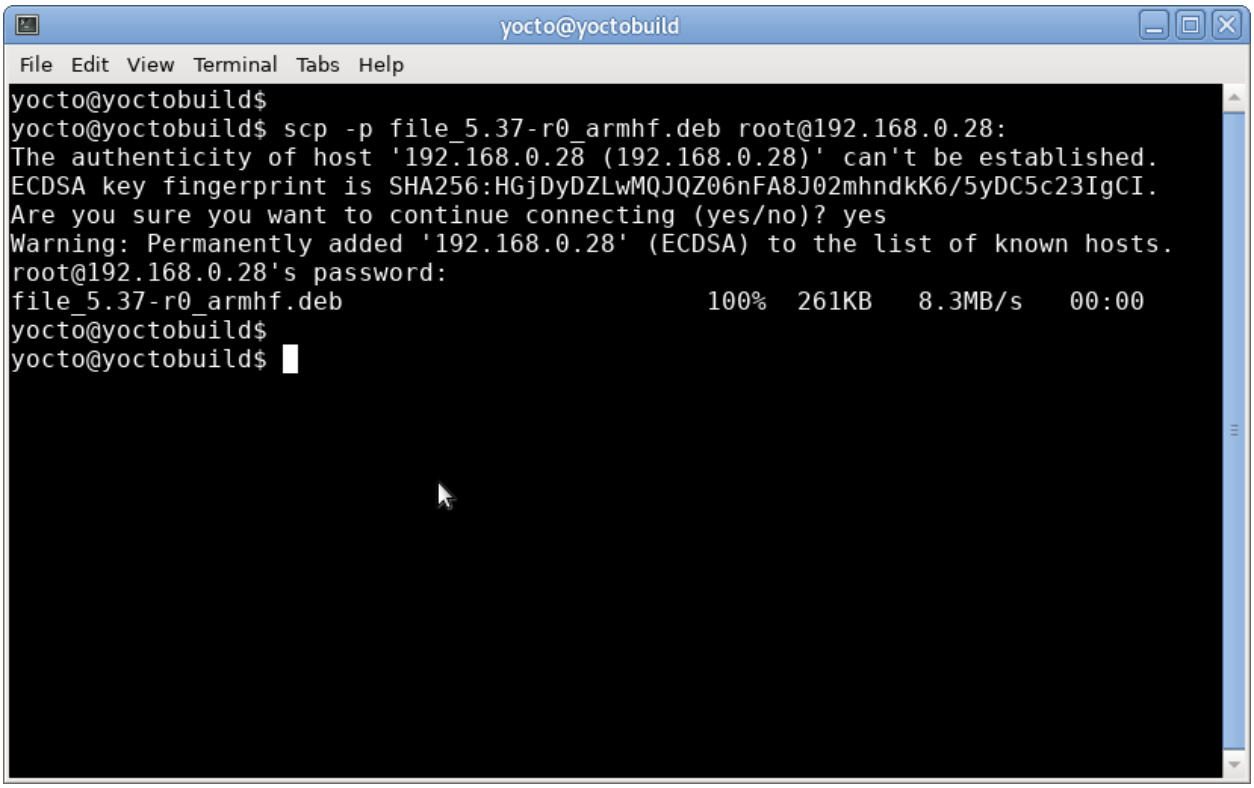
```
helloworld: ELF 32-bit LSB pie executable, ARM, EABI5 version 1
```

How to bring your binary to the target?

1. Connect the embedded device's ethernet to your LAN
2. Determine the embedded target IP address by `ip addr show`



3. Copy your binary, e.g. `helloworld` to the target by `scp helloworld root@<ip address of target>:/tmp`



4. Run `chmod +x` on the target to make your binary executable: `chmod +x /<path>/<binary name>`
5. Run your binary on the target: `/<path>/<binary name>`

How do you build a toolchain?

```
$ cd ~/workdir/bytepanel/3.0
$ repo init -u https://github.com/bytesatwork/bsp-platform-ti.git -b zeus
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for bytePANEL:

```
$ cd ~/workdir/bytepanel/3.0
$ MACHINE=bytepanel DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds an installable toolchain:

```
$ cd $BUILDDIR
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The toolchain is located under:

```
~/workdir/bytepanel/3.0/build/tmp/deploy/sdk
```

How to modify your toolchain

Currently the bytesatwork toolchain is generated out of the bytesatwork-minimal-image recipe. If you want to add additional libraries and development headers to customize the toolchain, you need to modify the bytesatwork-minimal-image recipe. It can be found under `~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images`

For example if you want to develop your own ftp client and you need libftp and the corresponding header files, edit the recipe `bytesatwork-minimal-image.bb` and add `ftplib` to the `IMAGE_INSTALL` variable.

This will provide the ftplib libraries and development headers in the toolchain. After adding additional software components, the toolchain needs to be rebuilt by:

```
$ cd ~/workdir/<machine name>/<yocto version>
$ MACHINE=<machine> DISTRO=poky-bytesatwork EULA=1 . setup-environment build
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The newly generated toolchain will be available under:

```
~/workdir/<machine name>/<yocto version>/build/tmp/deploy/sdk
```

For additional information, please visit: https://www.yoctoproject.org/docs/3.0.3/overview-manual/overview-manual.html#cross-development-toolchain-generation

Kernel

Download the Linux Kernel

Device	Branch	git URL
bytePANEL	baw-ti-linux-4.19.y	https://github.com/bytesatwork/ti-linux-kernel.git

---

Build the Linux Kernel

For both targets, an ARM toolchain is necessary. You can use the provided toolchain from *Where do you get the toolchain?* or any compatible toolchain (e.g. from your distribution)

Important:

The following tools need to be installed on your development system:

- `git`
- `make`
- `bc`

Note: The following instructions assume, you installed the provided toolchain for the respective target.

Important:

The following tools need to be installed on your development system:

- `u-boot-tools`

---

1. Download kernel sources

   Download the appropriate kernel from *Download the Linux Kernel*.

2. Source toolchain

```
source /opt/poky-bytesatwork/3.0.2/environment-setup-armv7at2hf-neon-poky-linux-
↪gnueabi
```

3. Create defconfig

```
make bytepanel_defconfig
```

4. Build Linux kernel

```
make LOADADDR=0x80008000 -j `nproc` uImage bytepanel.dtb
```

5. Install kernel and device tree

   To use the newly created kernel and device tree, the necessary files need to be installed on the target. This can be done either via Ethernet (e.g. `scp`) or by copying the files to the SD card.

   Note: For scp installation: Don't forget to mount /boot on the target.

File	Target path	Target partition
`arch/arm/boot/uImage`	`/boot/uImage`	`/dev/mmcblk0p1`
`arch/arm/boot/dts/bytepanel.dtb`	`/boot/devtree.dtb`	`/dev/mmcblk0p1`

5.4.6 byteDEVKIT (Yocto 2.7)

Image

Where do you get the SD card image?

Device	Yocto Version	Download	Checksum (SHA256)
byteDEVKIT	Yocto 2.7	flashlayout_bytesatwork-minimal-image_FlashLayout_sdcard_stm32mp157c-bytedevkit.raw.gz	7e62644473c21d200603b52d0080894a0ccfd950dd4a2f3c7c

Hint: Updating from an older image? You can update your older image by using: `apt-get update` and `apt-get upgrade`.

1. check for new version in the table above
2. edit `/etc/apt/sources.list` and point to the new package feed
3. run `apt-get update; apt-get upgrade`

As the yocto framework is based on several packages from various projects or suppliers, it is not guaranteed that an incremental upgrade by `apt-get upgrade` works automatically. Some manual adjustments might be needed.

How do you flash the image?

Attention:

- You need a microSD card with **at least 8GB** capacity.
- **All existing data** on the microSD card will be lost.
- **Do not format** the microSD card before flashing.

Windows

1. Unzip the file `flashlayout_bytesatwork-minimal-image_FlashLayout_sdcard_stm32mp157c-bytedevkit.raw.gz` (e.g. with 7-zip)
2. Write the resulting file to the microSD card with a tool like Roadkils Disk Image

Linux

```
gunzip -c flashlayout_bytesatwork-minimal-image_FlashLayout_sdcard_stm32mp157c-
↪bytedevkit.raw.gz | dd of=/dev/mmcblk<X> bs=8M conv=fdatasync status=progress
```

How do you build an image?

Use `repo` to download all necessary repositories:

```
$ mkdir -p ~/workdir/bytedevkit/2.7; cd ~/workdir/bytedevkit/2.7
$ repo init -u https://github.com/bytesatwork/bsp-platform-st.git -b warrior
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for byteDEVKIT:

```
$ cd ~/workdir/bytedevkit/2.7
$ MACHINE=bytedevkit DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds the development image:

```
$ cd $BUILDDIR
$ bitbake devbase-image-bytesatwork
```

The output is found in:

```
~/workdir/bytedevkit/2.7/build/tmp/deploy/images/bytedevkit
```

How to modify the image

The image recipes can be found in `~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images`

This is relative to where you started the `repo` command to fetch all the sources.

Edit the minimal-image recipe `bytesatwork-minimal-image.bb`

Add the desired software-package to `IMAGE_INSTALL` variable, for example add `net-tools` to `bytesatwork-minimal-image.bb`

Rebuild the image by:

```
$ cd ~/workdir/<machine name>/<yocto version>
$ MACHINE=<machine name> DISTRO=poky-bytesatwork EULA=1 . setup-environment␣
↪build
$ bitbake bytesatwork-minimal-image
```

How to rename the image

If you want to rename or copy an image, simply rename or copy the image recipe by:

```
$ cd ~/workdir/<machine name>/<yocto version>/build/tmp/deploy/images/<machine␣
↪name>
$ cp bytesatwork-minimal-image.bb customer-example-image.bb
```

Troubleshooting

- **Image size is too small**

  If you encounter that your image size is too small to install additional software, please have a look at the `IMAGE_ROOTFS_SIZE` variable under `~/workdir/<machine-name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images/bytesatwork-minimal-image.bb`. Increase the size if necessary.

Toolchain

Where do you get the toolchain?

Device	Yocto Version	Download	Checksum (SHA256)
byteDE-VKIT	Yocto 2.7	poky-bytesatwork-glibc-x86_64-devbase-image-bytesatwork-cortexa7t2hf-neon-vfpv4-bytedevkit-toolchain-2.7.1.sh	61896873ac7c75ac711a0b8e439ded6721d1a794deec26b

---

How do you install the toolchain?

Simply download the toolchain and execute the downloaded file, which is a self-extracting shell script.

Hint: If you encounter problems when trying to install the toolchain, make sure the downloaded toolchain is executable. Run `chmod +x /<path>/<toolchain-file>.sh` to make it executable.

Important:

The following tools need to be installed on your development system:

- `xz` (Debian package: `xz-utils`)
- `python` (any version)
- `gcc`

How do you use the toolchain?

Source the installed toolchain:

```
source /opt/poky-bytesatwork/3.0.3/environment-setup-cortexa7t2hf-neon-vfpv4-poky-linux-
↪gnueabi
```

Check if Cross-compiler is available in environment:

```
echo $CC
```

You should see the following output:

```
arm-poky-linux-gnueabi-gcc -mthumb -mfpu=neon-vfpv4 -mfloat-abi=hard -mcpu=cortex-a7 -
↪fstack-protector-strong -D_FORTIFY_SOURCE=2 -Wformat -Wformat-security -Werror=format-
↪security --sysroot=/opt/poky-bytesatwork/3.0.3/sysroots/cortexa7t2hf-neon-vfpv4-poky-
↪linux-gnueabi
```

Crosscompile the source code, e.g. by:

```
$CC helloworld.c -o helloworld
```

Check generated binary:

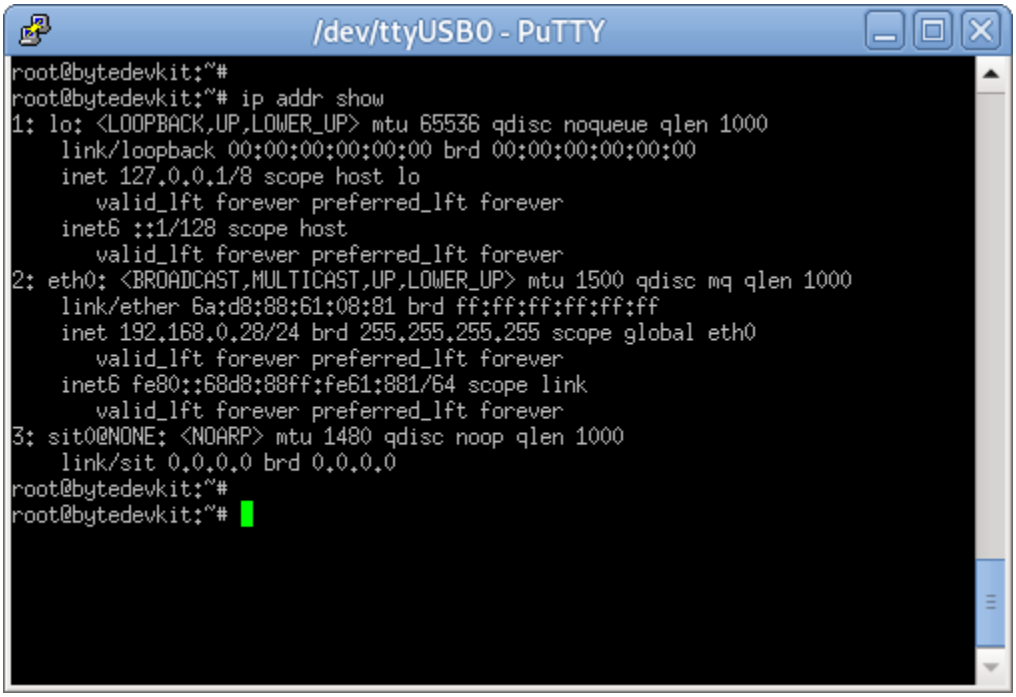
```
file helloworld
```

The output that is shown in prompt afterwards:

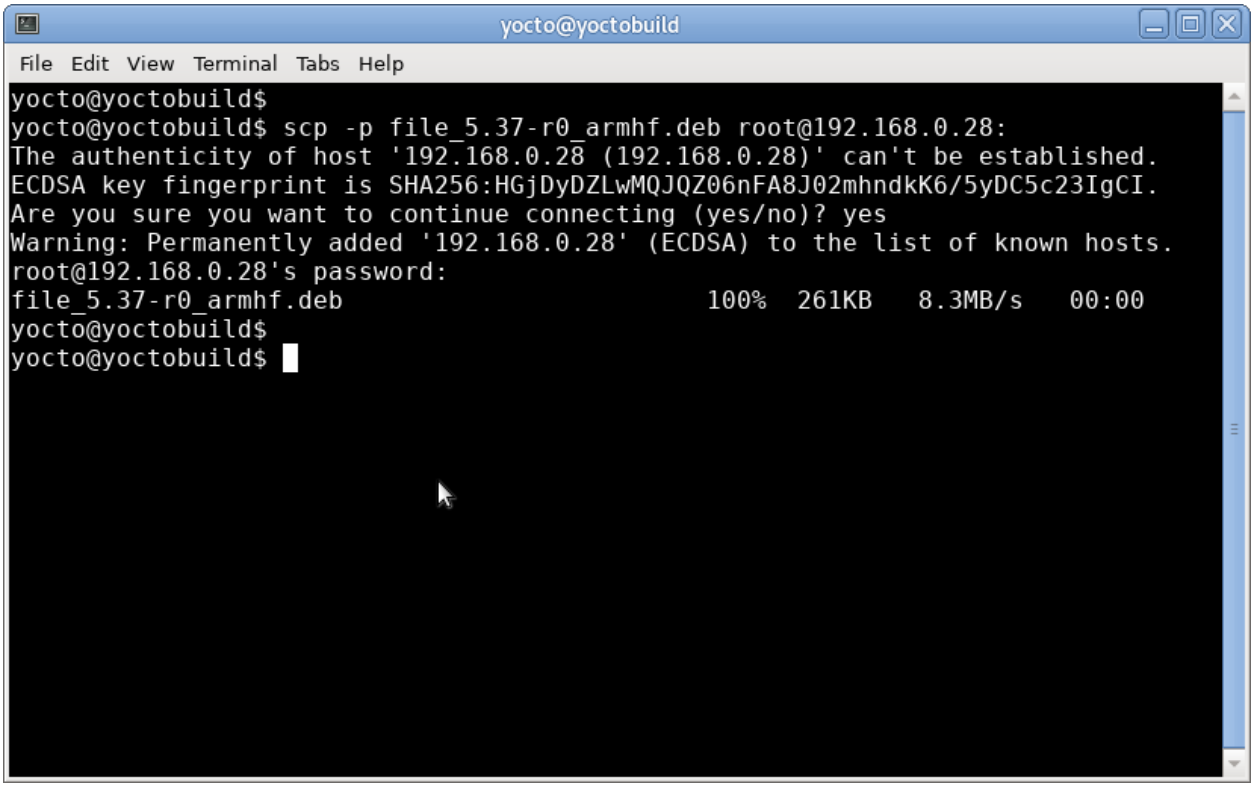
```
helloworld: ELF 32-bit LSB pie executable, ARM, EABI5 version 1
```

How to bring your binary to the target?

1. Connect the embedded device's ethernet to your LAN
2. Determine the embedded target IP address by `ip addr show`



3. Copy your binary, e.g. `helloworld` to the target by `scp helloworld root@<ip address of target>:/tmp`



4. Run `chmod +x` on the target to make your binary executable: `chmod +x /<path>/<binary name>`
5. Run your binary on the target: `/<path>/<binary name>`

How do you build a toolchain?

```
$ cd ~/workdir/bytedevkit/2.7
$ repo init -u https://github.com/bytesatwork/bsp-platform-st.git -b warrior
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for byteDEVKIT:

```
$ ~/workdir/bytedevkit/2.7
$ MACHINE=bytedevkit DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds an installable toolchain:

```
$ cd $BUILDDIR
$ bitbake devbase-image-bytesatwork -c populate_sdk
```

The toolchain is located under:

```
~/workdir/bytedevkit/2.7/build/tmp/deploy/sdk
```

How to modify your toolchain

Currently the bytesatwork toolchain is generated out of the bytesatwork-minimal-image recipe. If you want to add additional libraries and development headers to customize the toolchain, you need to modify the bytesatwork-minimal-image recipe. It can be found under `~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images`

For example if you want to develop your own ftp client and you need libftp and the corresponding header files, edit the recipe `bytesatwork-minimal-image.bb` and add `ftplib` to the `IMAGE_INSTALL` variable.

This will provide the ftplib libraries and development headers in the toolchain. After adding additional software components, the toolchain needs to be rebuilt by:

```
$ cd ~/workdir/<machine name>/<yocto version>
$ MACHINE=<machine> DISTRO=poky-bytesatwork EULA=1 . setup-environment build
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The newly generated toolchain will be available under:

```
~/workdir/<machine name>/<yocto version>/build/tmp/deploy/sdk
```

For additional information, please visit: https://www.yoctoproject.org/docs/2.7.2/overview-manual/overview-manual.html#cross-development-toolchain-generation

---

Kernel

Download the Linux Kernel

Device	Branch	git URL
byteDEVKIT	baw-v4.19-stm32mp	https://github.com/bytesatwork/linux-stm32mp.git

---

Build the Linux Kernel

For both targets, an ARM toolchain is necessary. You can use the provided toolchain from *Where do you get the toolchain?* or any compatible toolchain (e.g. from your distribution)

Important:

The following tools need to be installed on your development system:

- `git`
- `make`
- `bc`

Note: The following instructions assume, you installed the provided toolchain for the respective target.

Important:

The following tools need to be installed on your development system:

- OpenSSL headers (Debian package: `libssl-dev`)
- `depmod` (Debian package: `kmod`)

1. Download kernel sources

   Download the appropriate kernel from *Download the Linux Kernel*.

2. Source toolchain

```
source /opt/poky-bytesatwork/3.0.3/environment-setup-cortexa7t2hf-neon-vfpv4-poky-
↪linux-gnueabi
```

3. Create defconfig

```
make multi_v7_defconfig
scripts/kconfig/merge_config.sh -m -r .config arch/arm/configs/fragment-*
make olddefconfig
```

4. Build Linux kernel

```
make LOADADDR=0xC2000040 -j `nproc` uImage stm32mp157c-bytedevkit-v1-1.dtb modules
```

5. Install kernel and device tree

   To use the newly created kernel, device tree and/or module, the necessary files need to be installed on the target. This can be done either via Ethernet (e.g. `scp`) or by copying the files to the SD card.

   Note: For scp installation: Don't forget to mount /boot on the target.

File	Target path	Target partition
`arch/arm/boot/uImage`	`/boot/uImage`	`/dev/mmcblk0p4`
`arch/arm/boot/dts/stm32mp157c-bytedevkit-v1-1.dtb`	`/boot/stm32mp157c-bytedevkit.dtb`	`/dev/mmcblk0p4`

   Note:

   After installing a new kernel, it often fails to load modules, as the _signature_ of the kernel changed and it fails to find its corresponding modules folder. This issue can often be resolved with a symlink:

```
ln -s /lib/modules/<EXISTING FOLDER> /lib/modules/`uname -r`
```

Otherwise, please follow the instructions to copy the kernel modules

6. Install kernel modules

   To copy all available modules to the target, it's best to deploy them locally first and then copy all modules to the target.

```
mkdir /tmp/bytedevkit
make INSTALL_MOD_PATH=/tmp/bytedevkit modules_install
```

   Now you can copy the content of the folder `/tmp/bytedevkit` into the target's root folder (`/`) which is partition `/dev/mmcblk0p5`.

5.4.7 bytePANEL (Yocto 2.7)

Image

Where do you get the SD card image?

Device	Yocto Version	Download	Checksum (SHA256)
bytePAN	Yocto 2.7	devbase-image-bytesatwork-bytepanel-emmc-20190729194430.sdimg.gz	3b3e51d83c68f68d6ebbc2983d6b41b9e21d4878c1c9570804e694

Hint: Updating from an older image? You can update your older image by using: `apt-get update` and `apt-get upgrade`.

1. check for new version in the table above
2. edit `/etc/apt/sources.list` and point to the new package feed
3. run `apt-get update; apt-get upgrade`

As the yocto framework is based on several packages from various projects or suppliers, it is not guaranteed that an incremental upgrade by `apt-get upgrade` works automatically. Some manual adjustments might be needed.

How do you flash the image?

Attention:

- You need a microSD card with **at least 8GB** capacity.
- **All existing data** on the microSD card will be lost.
- **Do not format** the microSD card before flashing.

Windows

1. Unzip the file `devbase-image-bytesatwork-bytepanel-emmc-20190729194430.sdimg.gz` (e.g. with 7-zip)
2. Write the resulting file to the microSD card with a tool like Roadkils Disk Image

Linux

```
gunzip -c devbase-image-bytesatwork-bytepanel-emmc-20190729194430.sdimg.gz | dd of=/dev/
↪mmcblk<X> bs=8M conv=fdatasync status=progress
```

How do you build an image?

Use `repo` to download all necessary repositories:

```
$ mkdir -p ~/workdir/bytepanel/2.7; cd ~/workdir/bytepanel/2.7
$ repo init -u https://github.com/bytesatwork/bsp-platform.git -b warrior
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for bytePANEL:

```
$ cd ~/workdir/bytepanel/2.7
$ MACHINE=bytepanel DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds the development image:

```
$ cd $BUILDDIR
$ bitbake devbase-image-bytesatwork
```

The output is found in:

```
~/workdir/bytepanel/2.7/build/tmp/deploy/images/bytepanel
```

How to modify the image

The image recipes can be found in `~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images`

This is relative to where you started the `repo` command to fetch all the sources.

Edit the minimal-image recipe `bytesatwork-minimal-image.bb`

Add the desired software-package to `IMAGE_INSTALL` variable, for example add `net-tools` to `bytesatwork-minimal-image.bb`

Rebuild the image by:

```
$ cd ~/workdir/<machine name>/<yocto version>
$ MACHINE=<machine name> DISTRO=poky-bytesatwork EULA=1 . setup-environment␣
↪build
$ bitbake bytesatwork-minimal-image
```

How to rename the image

If you want to rename or copy an image, simply rename or copy the image recipe by:

```
$ cd ~/workdir/<machine name>/<yocto version>/build/tmp/deploy/images/<machine␣
↪name>
$ cp bytesatwork-minimal-image.bb customer-example-image.bb
```

Troubleshooting

- **Image size is too small**

  If you encounter that your image size is too small to install additional software, please have a look at the `IMAGE_ROOTFS_SIZE` variable under `~/workdir/<machine-name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images/bytesatwork-minimal-image.bb`. Increase the size if necessary.

Toolchain

Where do you get the toolchain?

Device	Yocto Version	Download	Checksum (SHA256)
bytePAl	Yocto 2.7	poky-bytesatwork-glibc-x86_64-devbase-image-bytesatwork-armv7at2hf-neon-bytepanel-toolchain-2.7.3.sh	b25e4a3f764eaf583ad0e6a3e0edcac9a1a9314ab6d1f4aad

---

How do you install the toolchain?

Simply download the toolchain and execute the downloaded file, which is a self-extracting shell script.

Hint: If you encounter problems when trying to install the toolchain, make sure the downloaded toolchain is executable. Run `chmod +x /<path>/<toolchain-file>.sh` to make it executable.

Important:

The following tools need to be installed on your development system:

- `xz` (Debian package: `xz-utils`)
- `python` (any version)
- `gcc`

How do you use the toolchain?

Source the installed toolchain:

```
source /opt/poky-bytesatwork/3.0.2/environment-setup-armv7at2hf-neon-poky-linux-gnueabi
```

Check if Cross-compiler is available in environment:

```
echo $CC
```

You should see the following output:

```
arm-poky-linux-gnueabi-gcc -march=armv7-a -mthumb -mfpu=neon -mfloat-abi=hard --sysroot=/
↪opt/poky-bytesatwork/3.0.2/sysroots/armv7at2hf-neon-poky-linux-gnueabi
```

Cross-compile the source code, e.g. by:

```
$CC helloworld.c -o helloworld
```

Check generated binary:

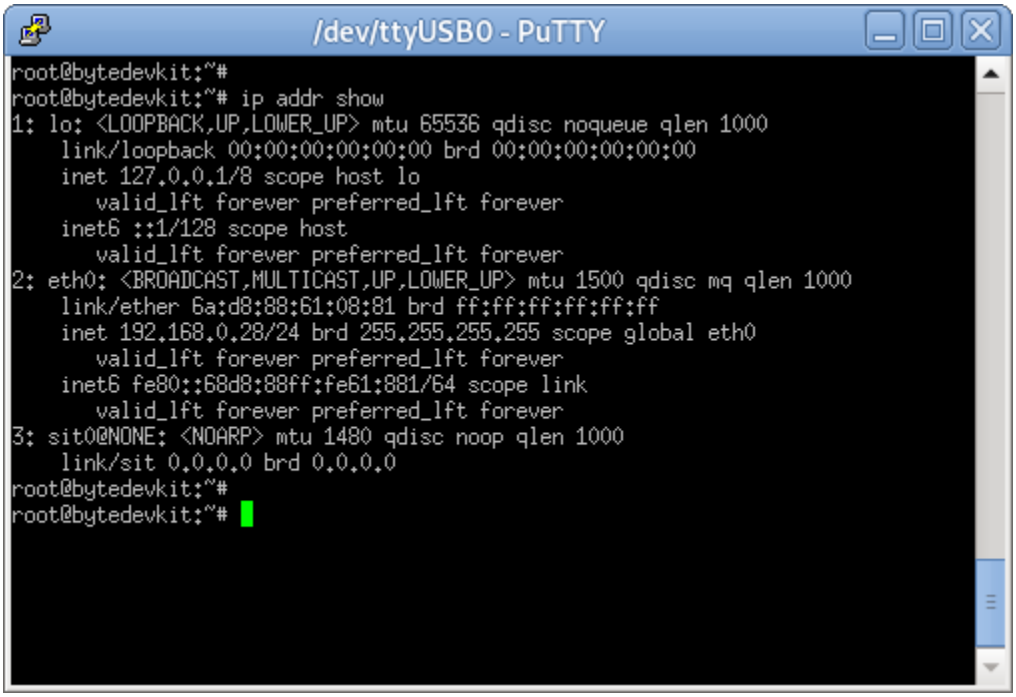
```
file helloworld
```

The output that is shown in prompt afterwards:

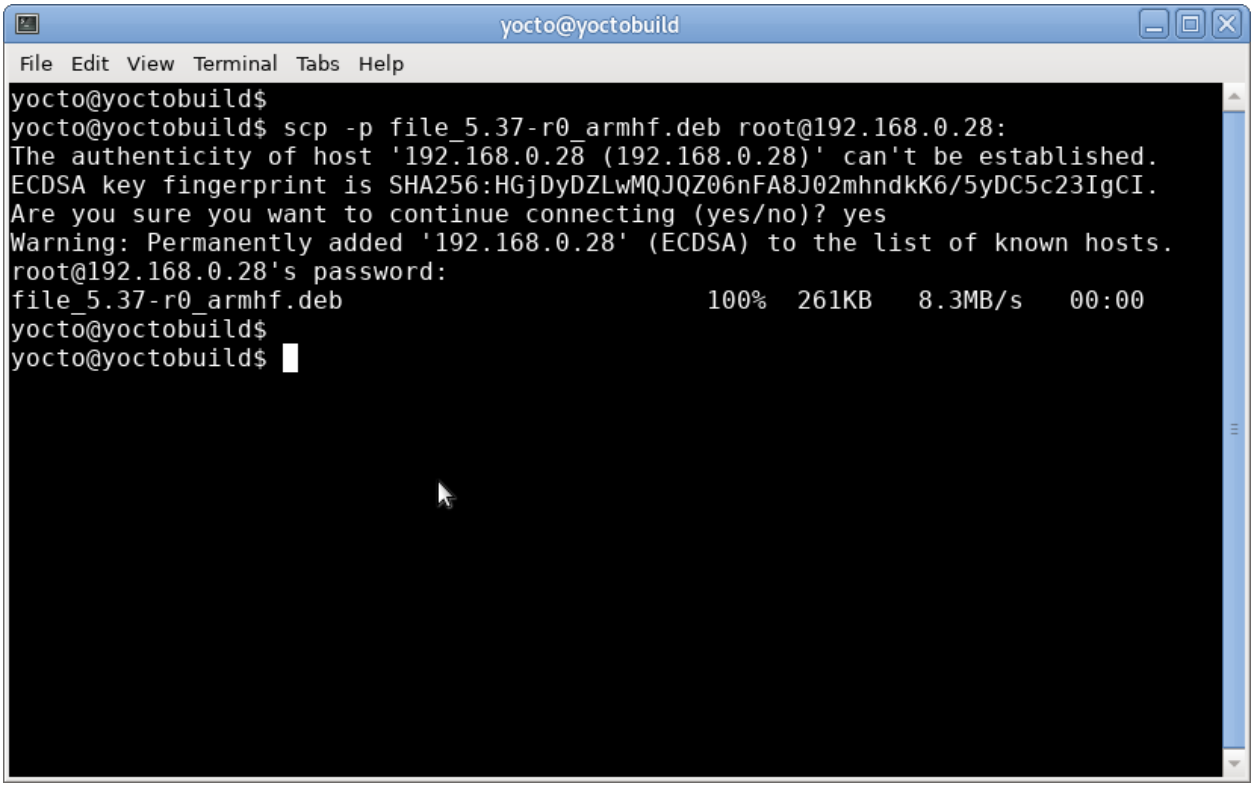
```
helloworld: ELF 32-bit LSB pie executable, ARM, EABI5 version 1
```

How to bring your binary to the target?

1. Connect the embedded device's ethernet to your LAN
2. Determine the embedded target IP address by `ip addr show`



3. Copy your binary, e.g. `helloworld` to the target by `scp helloworld root@<ip address of target>:/tmp`



4. Run `chmod +x` on the target to make your binary executable: `chmod +x /<path>/<binary name>`
5. Run your binary on the target: `/<path>/<binary name>`

How do you build a toolchain?

```
$ cd ~/workdir/bytepanel/2.7
$ repo init -u https://github.com/bytesatwork/bsp-platform.git -b warrior
$ repo sync
```

If those commands are completed successfully, the following command will set up a Yocto Project environment for bytePANEL:

```
$ cd ~/workdir/bytepanel/2.7
$ MACHINE=bytepanel DISTRO=poky-bytesatwork EULA=1 . setup-environment build
```

The final command builds an installable toolchain:

```
$ cd $BUILDDIR
$ bitbake devbase-image-bytesatwork -c populate_sdk
```

The toolchain is located under:

```
~/workdir/bytepanel/2.7/build/tmp/deploy/sdk
```

How to modify your toolchain

Currently the bytesatwork toolchain is generated out of the bytesatwork-minimal-image recipe. If you want to add additional libraries and development headers to customize the toolchain, you need to modify the bytesatwork-minimal-image recipe. It can be found under `~/workdir/<machine name>/<yocto version>/sources/meta-bytesatwork/recipes-core/images`

For example if you want to develop your own ftp client and you need libftp and the corresponding header files, edit the recipe `bytesatwork-minimal-image.bb` and add `ftplib` to the `IMAGE_INSTALL` variable.

This will provide the ftplib libraries and development headers in the toolchain. After adding additional software components, the toolchain needs to be rebuilt by:

```
$ cd ~/workdir/<machine name>/<yocto version>
$ MACHINE=<machine> DISTRO=poky-bytesatwork EULA=1 . setup-environment build
$ bitbake bytesatwork-minimal-image -c populate_sdk
```

The newly generated toolchain will be available under:

```
~/workdir/<machine name>/<yocto version>/build/tmp/deploy/sdk
```

For additional information, please visit: https://www.yoctoproject.org/docs/2.7.4/overview-manual/overview-manual.html#cross-development-toolchain-generation

Kernel

Download the Linux Kernel

Device	Branch	git URL
bytePANEL	baw-ti-linux-4.19.y	https://github.com/bytesatwork/ti-linux-kernel.git

---

Build the Linux Kernel

For both targets, an ARM toolchain is necessary. You can use the provided toolchain from *Where do you get the toolchain?* or any compatible toolchain (e.g. from your distribution)

Important:

The following tools need to be installed on your development system:

- `git`
- `make`
- `bc`

Note: The following instructions assume, you installed the provided toolchain for the respective target.

Important:

The following tools need to be installed on your development system:

- `u-boot-tools`

1. Download kernel sources

   Download the appropriate kernel from *Download the Linux Kernel*.

2. Source toolchain

```
source /opt/poky-bytesatwork/3.0.2/environment-setup-armv7at2hf-neon-poky-linux-
↪gnueabi
```

3. Create defconfig

```
make bytepanel_defconfig
```

4. Build Linux kernel

```
make LOADADDR=0x80008000 -j `nproc` uImage bytepanel.dtb
```

5. Install kernel and device tree

   To use the newly created kernel and device tree, the necessary files need to be installed on the target. This can be done either via Ethernet (e.g. `scp`) or by copying the files to the SD card.

   Note: For scp installation: Don't forget to mount /boot on the target.

File	Target path	Target partition
`arch/arm/boot/uImage`	`/boot/uImage`	`/dev/mmcblk0p1`
`arch/arm/boot/dts/bytepanel.dtb`	`/boot/devtree.dtb`	`/dev/mmcblk0p1`

bytesatwork

CHAPTER

SIX

HARDWARE DEVELOPMENT

We provide the development for a wide range of embedded systems, from small-scale embedded components to sophisticated embedded systems with increased security requirements. Our engineers are certified hardware experts and provide long experience in business.

6.1 byteENGINE AM335x

- **General Information**: The byteENGINE AM335x is a high performance industrial oriented computing module. It allows a short time-to-market, while reducing development costs and substantial design risks. The system on module (SOM) uses the Texas Instruments AM335x industrial applications processor family. The AM335x features a PowerVRTM SGX Graphics Accelerator Subsystem for 3D graphics acceleration. The Programmable Real-Time Unit and Industrial Communication Subsystem (PRU-ICSS) allows independent operation from the ARM processor. PRU-ICSS enables real-time protocols such as EtherCAT, PROFINET, EtherNet/IP, PROFIBUS, Ethernet Powerlink and Sercos.

  The byteENGINE AM335x is a high performance industrial oriented computing module. It allows a short time-to-market, while reducing development costs and substantial design risks.

  The system on module (SOM) uses the Texas Instruments AM335x industrial applications processor family. The AM335x features a PowerVRTM SGX Graphics Accelerator Subsystem for 3D graphics acceleration. The Programmable Real-Time Unit and Industrial Communication Subsystem (PRU-ICSS) allows independent operation from the ARM processor. PRU-ICSS enables real-time protocols such as EtherCAT, PROFINET, EtherNet/IP, PROFIBUS, Ethernet Powerlink and Sercos.

- **Datasheet AM335x**: https://www.bytesatwork.io/wp-content/uploads/2019/03/Datasheet_byteENGINE_AM335x-12.pdf
- **Prepared Pinmux file AM335x**: https://download.bytesatwork.io/documentation/byteENGINE/ressources/byteEngineM2-20160922.pinmux
- **Detailed pinout AM335x**: https://download.bytesatwork.io/documentation/byteENGINE/ressources/PinmuxConfigSummary_byteEngineM2-20160922.xlsx
- **Datasheet Connectors Neltron 2001S-100G-270-020**: https://download.bytesatwork.io/documentation/byteENGINE/ressources/Neltron_2000P.pdf
- **Schematic of the connectors X1 and X2**: https://download.bytesatwork.io/documentation/byteENGINE/ressources/m2-connector.pdf
- **Texas Instruments Sitara™ AM335x Processors**: http://www.ti.com/processors/sitara-arm/am335x-cortex-a8/overview.html
- **AM335x Technical Reference Manual**: https://www.ti.com/lit/ug/spruh73q/spruh73q.pdf
- **TPS65910x Integrated Power-Management Unit**: http://www.ti.com/lit/ds/symlink/tps65910.pdf

6.2 byteENGINE STM32MP1x

- **General Information**: The byteENGINE STM32MP1x is a high performance industrial oriented computing module. It allows you a short time-to-market, reducing development costs and substantial design risks.

  The system on module (SOM) uses the STM32MP15xxAC devices which are based on the high-performance dual-core ARM® Cortex®-A7 32-bit RISC core operating at up to 650MHz/800MHz. The STM32MP15xxAC devices also embed a Cortex®-M4 32-bit RISC core operating at up to 200 MHz frequency. The Cortex®-M4 core features a floating point unit (FPU) single precision which supports ARM® single-precision dataprocessing instructions and data types.

  Furthermore, the STM32MP15xxAC devices embed a 3D graphic processing unit (Vivante® - OpenGL® ES 2.0) running at up to 533 MHz, with performances up to 26 Mtriangle/s, 133 Mpixel/s.

- **Factsheet STM32MP1x**: https://www.bytesatwork.io/wp-content/uploads/2019/04/Fact-Sheet-byteENGINE_STM32MP1x.pdf
- **Datasheet STM32MP1x**: https://www.bytesatwork.io/wp-content/uploads/2019/12/Datasheet_byteENGINE_STM32MP1x-6.pdf
- **Detailed pinout STM32MP1x**: https://download.bytesatwork.io/documentation/byteENGINE/ressources/byteENGINE-M5-pinout.xlsx
- **Datasheet Connectors Neltron 2001S-100G-270-020**: https://download.bytesatwork.io/documentation/byteENGINE/ressources/Neltron_2000P.pdf
- **Schematic of the connectors X1 and X2**: https://download.bytesatwork.io/documentation/byteENGINE/ressources/m5-connector-pinout.pdf
- **STMicroelectronics STM32MP1**: https://www.st.com/en/microcontrollers-microprocessors/stm32mp1-series.html
- **STPMIC1 power management IC**: https://www.st.com/en/power-management/stpmic1.html
- **Datasheet STM32MP157C**: https://www.st.com/resource/en/datasheet/stm32mp157c.pdf
- **STM32CubeMX Software Download**: https://www.st.com/en/development-tools/stm32cubemx.html
- **STM32MP1x prepared CubeMX Project**: https://download.bytesatwork.io/documentation/byteENGINE/ressources/byteENGINE_STM32MP1.ioc
- **Prepared project: step model STM32MP1x**: https://download.bytesatwork.io/documentation/byteENGINE/ressources/byteengine-m5.step
- **Altium Library Neltron 2001S-100G-270-020**: https://download.bytesatwork.io/documentation/byteENGINE/ressources/2001s-100G-270-020.zip
- **Altium Library byteENGINE STM32MP1x (X1/X2 position mask on layer 21)**: https://download.bytesatwork.io/documentation/byteENGINE/ressources/Footprint-byteENGINE-M5.zip

CHAPTER

SEVEN

ERRATA

Known issues

7.1 byteDEVKIT < V1.2

7.1.1 STM32MP1 Ethernet

Due to a hardware issue at the ethernet PHY autonegotiation is disabled.
Using the ethernet setting from Device Tree of 1 GbE will not work on ethernet switches < 1 GbE.
As a workaround the `ethtool` could be used to set the speed manually.

Download it from here, copy it to the SD card and install it on the target with:

```
dpkg -i ethtool_5.4-r0_armhf.deb
```

Set the desired speed manually:

```
ethtool -s eth0 speed 100 duplex full
# or even
ethtool -s eth0 speed 10 duplex half
```

bytesatwork

bytesatwork