DOCUMENTATION

emtrion
embedded systems

emSBC-Argon Yocto Manual

Yocto Based BSP Manual

Rev. 5 / 17.08.2021
Author: Saptarshi Mitra
Last Change: August 17, 2021

Revision: Rev. 5 / 17.08.2021

Rev.	Date/Initial	Changes
1	11.09.2019/Mi	First Version of emSBC-Argon Yocto Manual
2	05.11.2019/Mi	Minor changes for easier comprehension
3	14.07.2021/Ml	Changing to Yocto 3.1 dunfell
4	26.07.2021/Ml	Support rotation of display(90°/180°/270°)
5	17.08.2021/Ml	update U-Boot and tf-a before Version dunfell

Contents

1 Introduction **5**

2 Terms and definitions **7**

3 Scope of the delivery **8**

4 Host Development Machine **10**
4.1 Linux distribution . . . 10
4.2 Further requirements . . . 10
4.2.1 Predefined directories . . . 10
4.2.2 Providing the delivery . . . 11
4.2.3 Installation emtrion's yocto layers . . . 11
4.2.4 Installation emtrion's yocto setup script . . . 11
4.2.5 Preparing NFS server . . . 11
4.2.6 Preparing the prebuilt target RFS for booting by NFS . . . 11
4.2.7 Preparing the SDK . . . 11
4.2.8 Preparing emtrion's update mechanism emPURS . . . 11
4.2.9 Serial port . . . 11
4.2.10 Serial terminal . . . 12
4.3 Device Startup . . . 12
4.4 Device Network Setup . . . 13
4.5 Prebuilt images installations . . . 14
4.5.1 Target Root Filesystem . . . 14

5 Setting up Yocto Buildsystem **15**
5.1 Setting up the machine specific build directory . . . 15
5.2 Yocto Setup Script . . . 16
5.2.1 Parameters . . . 16
5.2.2 Input File Guidelines . . . 16
5.2.3 Functions . . . 17
5.3 Emtrion's Yocto Layers . . . 18

6 Image Creation **21**
6.1 Output files . . . 21
6.2 Root File System . . . 22
6.3 Boot directory . . . 22

7 Booting, updating and restoring **23**
7.1 Updating from Yocto thud to dunfell . . . 23

7.2 Default boot and other . . . 23
7.2.1 fitimage linux . . . 24
7.2.1.1 Display rotation . . . 24
7.3 U-boot . . . 24
7.3.1 Basic Uboot Operation . . . 24
7.3.2 Using U-Boot to change boot device or update the system . . . 25
7.3.2.1 Updating of the root file system and kernel . . . 25
7.3.2.2 Updating of U-Boot and tf-a . . . 26
7.3.2.3 Booting . . . 27
7.4 NFS Setup . . . 28

8 Using ARM Cortex M4 processor **29**
8.1 emtm4demo . . . 29
8.2 OpenAMP_raw . . . 30

9 SDK **32**
9.1 Installing the SDK . . . 32
9.1.1 Setting up the SDK environment . . . 32
9.2 Qt5 with Yocto development . . . 32

10 Further information **34**

1 Introduction

Emtrion produces and offers various base boards and modules which are available in https://support.emtrion.de/en/home.html. One of the newest additions to the product line is a single board computer named emSBC-Argon. This manual provides instructions and pointers for efficient use of Yocto project development tool with the hardware.

The **emSBC-Argon** is based on the **STM32MP1** board from STMicroelectronics. It supports open source software development with a fully functional Linux kernel. The system can be tailored according to requirements using Yocto Project as well as Debian. The board is an MPU, providing the user benefits of a **Dual-Cortex A7** and **Cortex-M4** processor.

The **Yocto version** for the product is **Yocto dunfell (3.1**), launched in April 2020. The Yocto layers provided by STM and emtrion are based on this Yocto version. The layers used from STM repositories are **meta-st-openstlinux** and **meta-st-stm32mp**. Emtrion provides two Yocto layers to add the necessary functions: **"meta-emtrion"** and **"meta-emtrion-bsp"** layers. While the meta-emtrion layer provides a general layer for all the functions common to the entire product range, the emtrion-bsp layer, as the name suggests is board specific. In case of the board under consideration, the relevant layer is **meta-emtrion-emsbc-argon**. Some new recipes have been added to the emtrion layers and existing recipes have been modified to tailor for the device.

The BSP is a **Linux Kernel** based on version **5.4.56** and is forked from STMicroelectonics/linux, hosted at emtrion's github. The developer kit is managed by the specific kernel configuration file and the device tree file, along with the required changes for the correct operation of the associated peripherals.

The U-Boot system implemented the board is based on the version **v2020.01** and also forked from STMicroelectonics repos, hosted at emtrion's github. The U-Boot was adapted by some patches to give a working version for the emsbc-argon.

The important version numbers for the various tools and packages after the update are listed below.

Name	Version
Linux stm32mp1	v5.4.56
Uboot stm32mp1	v2020.01
tf-a stm32mp1	v2.2
Yocto	dunfell v3.1.7
Bitbake	v1.46.0
GCC	9.3
Openssh	v8.2
Busybox	v1.31.1
systemd	v1.244.5
Netbase	v6.1
Qt	v5.14.2
Gstreamer	v1.16.3
Mesa	v20.0.2
Graphics support	OpenGL_ES 2.0
Graphics driver	Gcnano v6.4.3
layer/ meta-emtrion	tag: emsbc-argon_dunfell_v1.0
layer/ meta-emtrion-bsp	tag: emsbc-argon_dunfell_v1.2

Table 1.1: *Packages and versions*

This manual describes the scope of the developer kit and the general information as well as instructions for the user.

It is assumed that users of Emtrion Linux developer kits are already familiar with U-Boot, Linux, Yocto and creating and debugging applications. General Linux and programming knowledge are out of the scope of this document. Emtrion is happy to assist you in acquiring this knowledge. If you are interested in training courses or getting support, please contact the Emtrion sales department.

Please understand we can not go into more details about Yocto Project inside the limited scope of this documentation, because Yocto/OpenEmbedded is a powerful but also complex build system. Emtrion offers paid support for problems regarding the developer kit. The official documentations can be referred to, however if that is not sufficient, we also offer training sessions about Yocto/OpenEmbedded.

The important resources that can be accessed for further in-depth knowledge are as follows:

1. Yocto Manual: https://www.yoctoproject.org/docs/3.1.7/ref-manual/ (any question related to Yocto has some reference here)
2. Openembedded: http://www.openembedded.org/ (information regarding openembedded layers for Linux development)
3. STM: https://wiki.st.com/stm32mpu/wiki/ (information about the open source STM MPU development tools and boards)
4. Yocto repositories: https://git.yoctoproject.org/ (with the main
5. Yocto repo: poky and other supplementary ones)
6. Openembedded repositories : https://git.openembedded.org/ (with the required meta-openembedded layer and other supplementary)
7. STM repositories : https://github.com/STMicroelectronics (STM repositories for linux, U-Boot and Yocto layers)
8. emtrion's github : https://github.com/emtrion (emtrion repos for linux, U-Boot and tf-a)

2 Terms and definitions

Term	Definition
Target	Module: emSBC-Argon
Host	Workstation, Developer PC
Toolchain	Compiler, Linker, etc.
RFS	Root file system, contains the basic operating system
Console	Text terminal interface for Linux
NFS	Network File System, which can share directories over network
NFS_SHARE	Location that is exported by the NFS for the purpose of updating and booting by using NFS
U-Boot	Bootloader, hardware initialization, updating images, starting OS
YP	Yocto Project
INST_DIR	Location where Yocto and the meta-layers are installed
MACHINE	Specifies the target device for which the image is built. The machine is named **emsbc-argon** for this kit
BUILD_DIR	Machine dependent build directory
BSP	Board Support Package
SDK	Software Development Kit

Table 2.1: *Terms and definitions*

3 Scope of the delivery

Other than before, this kit doesn't come with a prepared Virtual Machine. License issues are the reasons. Subsequently, emtrion provides the corresponding software and prebuilt binaries via cloud. Buyers will get a cloud link. This link points to a directory which includes the gz-Archiv **emsbc-argon_dunfell.tar.gz**, including a defined directory structure as below. The **gz-archiv** contains several images and the meta-layer for building them.

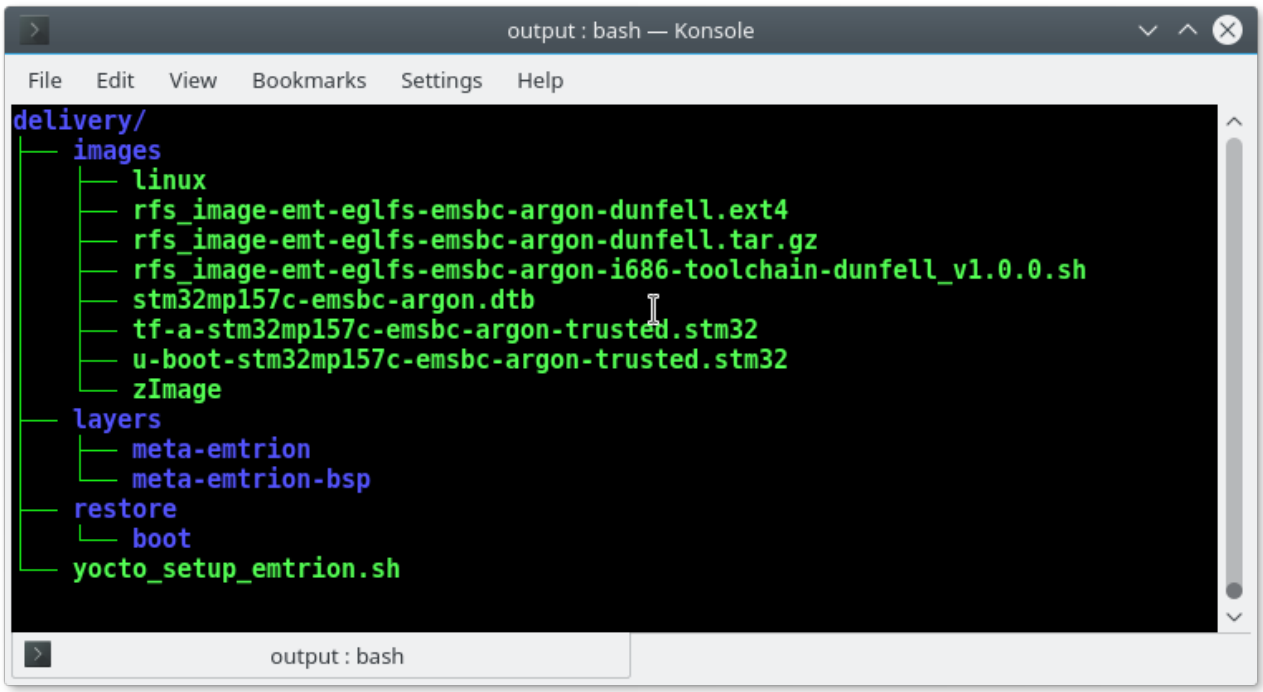

Figure 3.1: *layout delivery*

1. **layers**
   a) **meta-emtrion**
   Contains the general recipes and files required for the majority of the products in the Emtrion product range.
   b) **meta-emtrion-bsp**
   Contains the directory corresponding to the machine under consideration, with machine specific configuration files, recipes and patches.
      i. **meta-emtrion-emsbc-argon/yocto_emtrion_setup.sh**
      This script can be utilized to setup the entire environment for Yocto development.
      ii. **meta-emtrion-emsbc-argon/setup_emsbc-argon.txt**
      This input file contains all the variables and layers depending on which the environment is created. This file is user editable, while keeping in mind the mentioned
      iii. **meta-emtrion-emsbc-argon/Readme**
      The readme file contains more information about the setup script, providing instructions for the user.

Emtrion recommends reading this before starting with the installation, to have a clear idea about the modifiable variables and layers and to avoid problems in the later stages.

2. **images**
   a) **rfs_image-emt-eglfs-emsbc-argon-dunfell.ext4**
   Linux image file for the rootfs system created for the development kit.
   b) **rfs_image-emt-eglfs-emsbc-argon-dunfell.tar.gz**
   This archive file can be extracted and used booting the device by nfs boot.
   c) **zImage**
   Kernel image pertinent to the developer kit.
   d) **stm32mp157c-emsbc-argon.dtb**
   Device tree for the emsbc-argon board, developed for the linux kernel.
   e) **linux**
   fitImage including zImage and device tree
   f) **tf-a-stm32mp157c-emsbc-argon-trusted.stm32**
   trusted firmware, first stage bootloader (FSBL)
   g) **u-boot-stm32mp157c-emsbc-argon-trusted.stm32**
   U-Boot image, second stage bootloader (SSBL)
   h) **rfs_image-emt-eglfs-emsbc-argon-i686-toolchain-dunfell_v1.0.0.sh**
   SDK installer
3. **restore\boot**
   Files of emtrion's update mechanism emPURS for updating the target's RFS
   a) **linux**
   see 2e
   b) **uboot_script**
   Implements the U-Boot commands
   c) **emPURS_plat**
   Implements the emPURS process
   d) **ramdisk-emsbc-argon.rootfs.cpio.gz**
   Implements the initramfs of emPURS for updating

4 Host Development Machine

Due to not any prepared VM is available, you have to setup your own linux distribution before working with the kit. However, setting up the linux distribution is not so much work. Below is a short guideline.

4.1 Linux distribution

The Yocto Project supports several linux distributions and what are the requirements. For Yocto dunfell, please look at https://docs.yoctoproject.org/3.1.6/ref-manual/ref-system-requirements.html to meet the requirements. Please note, the linux distribution **Debian Buster** is used by this kit and so we prefer this for your system, too.

4.2 Further requirements

4.2.1 Predefined directories

This document references to some predefined directories. Following this document we recommend you to provide this predefined directories.

Placeholder	Assignment	Remark
HOME_DIR	/home/hico	home directory of user hico (Replace it with the corresponding home directory on your system)
DWL_DIR	<HOME_DIR>/Downloads	the user's download directory
NFS_SHARE	<HOME_DIR>/nfs	location exported by the NFS
NFS_ROOTFS	<NFS_SHARE>/rootfs	nfs share for booting the RFS by using NFS
NFS_RESTORE	<NFS_SHARE>/restore	location to put the emPURS files for updating the RFS
SDK_DIR	<NFS_SHARE>/sdk	installation directory for the SDK
INST_DIR	<HOME_DIR/yocto/build	location where Yocto and all the meta-layers will be stored to
BUILD_DIR	<INST_DIR>/YoctoBuildDirectory/emtrion/machines/<MACHINE>	location of the build system
BUILD_DWNL	<INST_DIR>/YoctoBuilddirectory/downloads	location of the fetched downloads while the build process
BUILD_SSTATE	<INST_DIR>/YoctoBuildDirectory/sstate-cache	location of the sstate-cache while the build process

Table 4.1: *Predifined directories*

4.2.2 Providing the delivery

Download the gz-Archiv **emsbc-argon_dunfell.tar.gz** from the cloud and save it to the folder <**DWL_DIR**>. Then decompress it in this folder by

tar xf **emsbc-argon_dunfell.tar.gz**

There will be a directory layout created as shown in section 3 on page 8

4.2.3 Installation emtrion's yocto layers

Create the directory <**INST_DIR**> and copy the layer directories **meta-emtrion** and **meta-emtrion-bsp** from <**DWL_DIR**>/delivery/layers to it.

4.2.4 Installation emtrion's yocto setup script

Copy the setup script **yocto_emtrion_setup.sh** from <**DWL_DIR**>/delivery to <**INST_DIR**>.

4.2.5 Preparing NFS server

Please look for this work in section 7.4 on page 28

4.2.6 Preparing the prebuilt target RFS for booting by NFS

Create the directory <**NFS_ROOTFS**> and unpack the RFS-archiv **rfs_image-emt-eglfs-emsbc-argon-dunfell.tar.gz** in <DWL_DIR>/delivery/images to it.

tar xf <**DWL_DIR**>/delivery/images/rfs_image-emt-eglfs-emsbc-argon-dunfell.tar.gz –C <**NFS_ROOTFS**>

4.2.7 Preparing the SDK

The SDK is useable for application development outside of the yocto build environment **BUILD_DIR**> and is delivered by an installer. The SDK's installation directory is default

/opt/emtrion/emsbc-argon/dunfell_v1.0.0

However, we recommend the location within the <**NFS_SHARE**>, to make the access of the included RFS possible by NFS. For that we suggest to create the directory <**SDK_DIR**>. Then install the the SDK as described in section 9.

4.2.8 Preparing emtrion's update mechanism emPURS

Create the directory <**NFS_RESTORE**> and copy the emPURS files from <**DWL_DIR**>/delivery/restore to it.

cp <**DWL_DIR**>/delivery/restore/* <**NFS_RESTORE**>/

4.2.9 Serial port

The USB serial converter will appearing at **/dev/ttyUSBn**

4.2.10 Serial terminal

We assume the terminal program **picocom** for connecting to the target.

4.3 Device Startup

Connect the developer kit to the serial port attached to your system and to your network. Open a console in the linux and open a serial terminal by entering:

sudo picocom -b 115200 /dev/ttyUSBn

ttyUSBn has to be replaced with the device assigned to the connected USB serial adapter eg. **ttyUSB0**.

The developer kit is ready to be powered on. The terminal should then shwo messages for tf-a, U-Boot and Linux kernel.

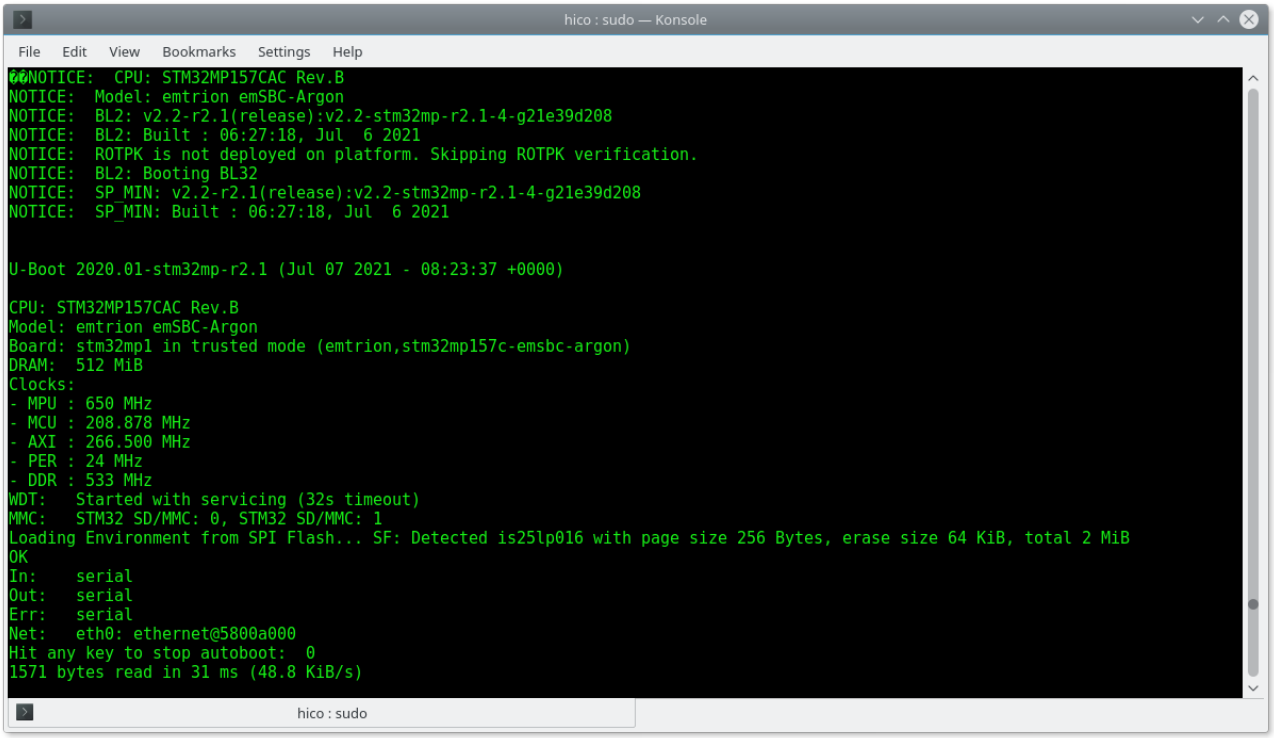

Figure 4.1: *booting*

After the developer kit is booted, the prompt is showed. Due to automatic login, no login is required.

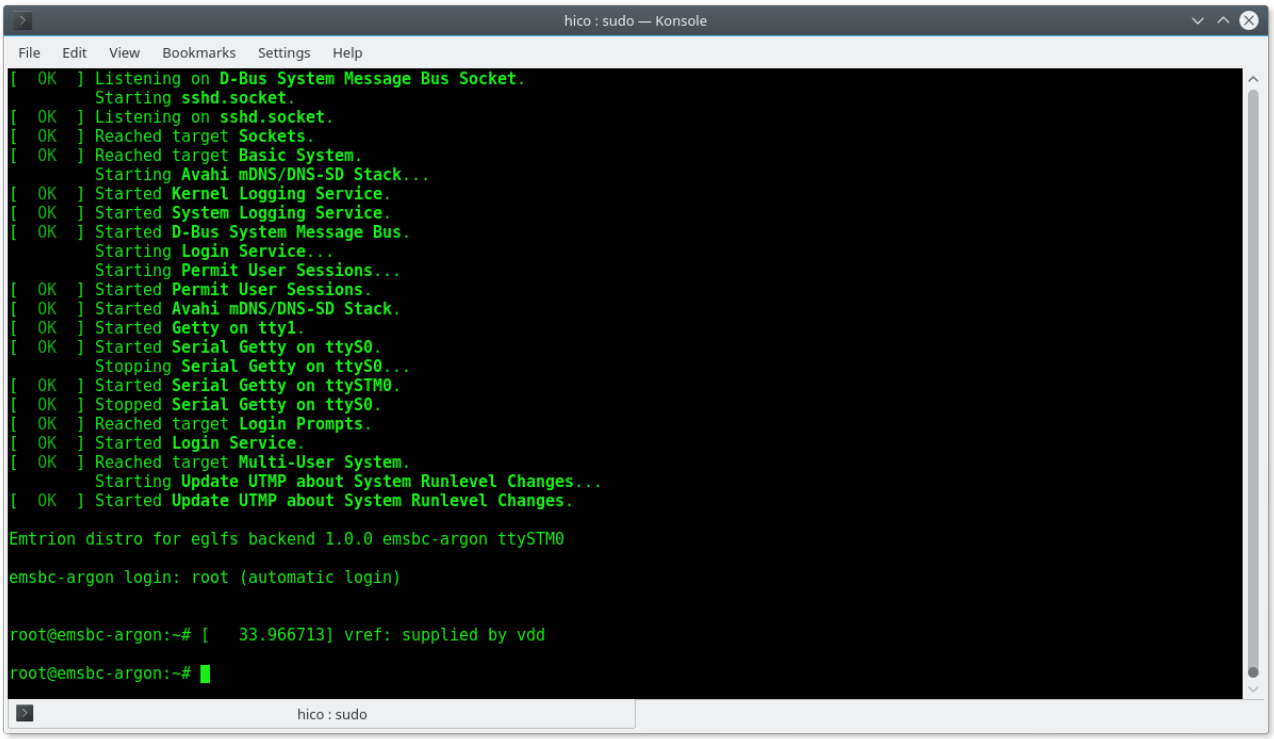

Figure 4.2: *prompt*

4.4 Device Network Setup

By default the developer kit is setup to use a DHCP server. This is configurable by a bootloader environment variable "ip-method". This variable can have the values "dhcp" or "static".

You can check if there is a valid IP address with the command "ifconfig" or "ip addr show eth0".

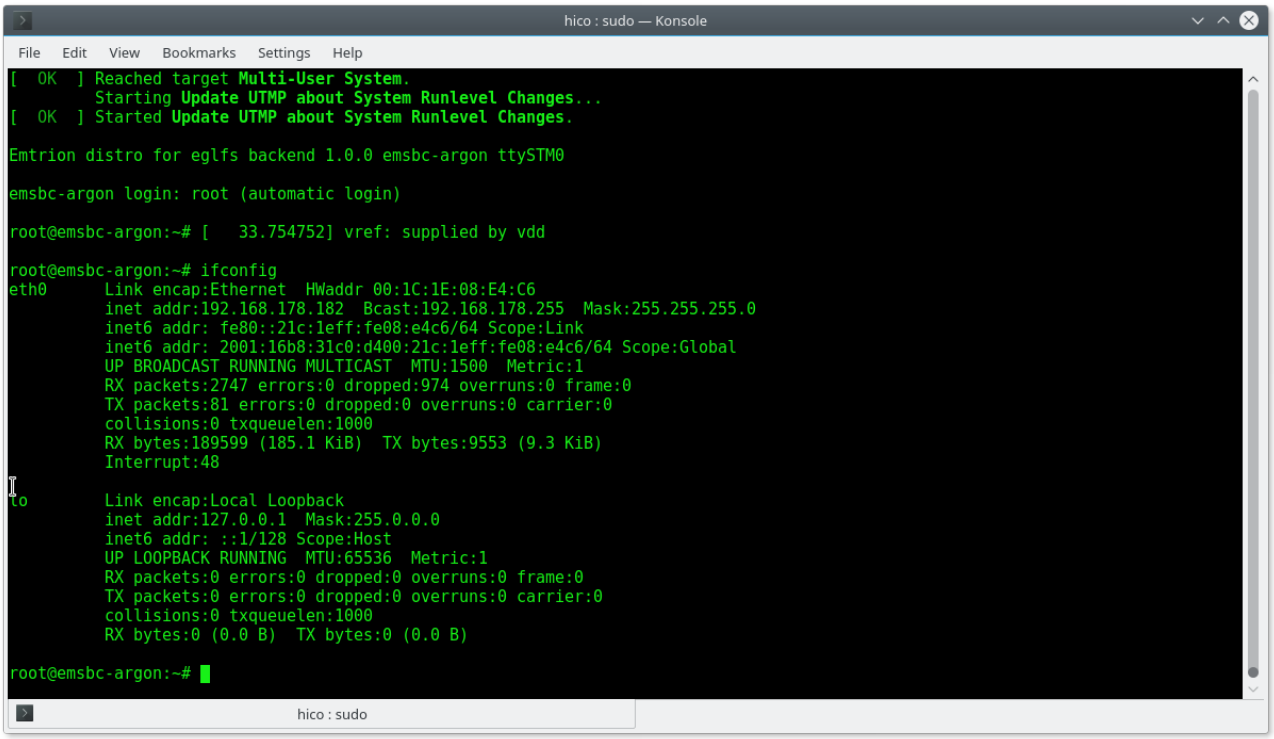

Figure 4.3: *network*

If the setup is not correct you have to do it manually. Please check the description of the bootloader configuration on how to set up the variable "ip-method".

4.5 Prebuilt images installations

As mentioned before, the kit is provided by various prebuilt binaries for downloading by a cloud link. The various binaries are assigned at specific subdirectories.

In order to create the images by your own, please follow the steps described in chapter 5.

4.5.1 Target Root Filesystem

The target root filesystem for the emsbc-argon is located in images.

5 Setting up Yocto Buildsystem

At this chapter we assume the emtrion yocto layers and setup script are installed as described in 4.2.3 and 4.2.4. Setting up the buildsystem is done by the setup script.

5.1 Setting up the machine specific build directory

While setting up a machine, Yocto sets up a build directory inside its directory structure as default. However, it is recommended to leave this directory clean and create a new one, especially if you are working on several machines.

The setup script considers this issue and automates the setup process. By default, it creates a directory structure with the top level directory YoctoBuildDirectory inside of <INST_DIR>, including

- a machine dependent build directory (<**BUILD_DIR**>)
- a common downloads directory (<**BUILD_DWNL**>)
- a common sstate-cache directory (<**BUILD_SSTATE**>)

The **downloads directory** can also be modified or shared on user discretion using the setup file.

The newly created directory structure in INST_DIR is as follows:

Location	Remarks
YoctoBuildDirectory/	
|– **downloads**	stores fetched data required for the build process
|– **machines**	stores build directories of various machines
| |–**emsbc-argon**	build directory of emsbc-argon
| |– **conf**	contains local.conf, bblayers.conf
| |– **tmp-glibc**	contains all the build outputs
|– **sstate-cache**	contains the build states created during the build process

Table 5.1: *Installation Directory structure*

Second, the required layers will be updated or installed, dependent if they exist or not. Then the defined YP release is checked out. These layers are also added as separate directories in the <**INST_DIR**>. The default layers are **poky, meta-st-stm32mp, meta-st-openstlinux, meta-openembedded** and **meta-qt5**.

In a further step the configuration files **bblayers.conf** and **local.conf** will be modified and copied to the build directory. At last the build environment of the created build directory is set.

Setting up the machine dependent build directory is done by sourcing the setup script from <**INST_DIR**>

. ./yocto_setup_emtrion.sh -f ./meta-emtrion-bsp/meta-emtrion-emsbc-argon/setup_emsbc-argon.txt

After sourcing the script, the prompt automatically changes to the build directory, where images can be created using bitbake command.

5.2 Yocto Setup Script

To get the initial build setup please execute the following command in the INST_DIR:

. ./yocto_setup_emtrion.sh -i/-f -o -b -h

Info

Please note that the **script has to be sourced** from the directory mentioned and not directly executed. Doing otherwise will lead to an error while setting up the environment for building images. The script requires at least either f or i argument to run successfully.

5.2.1 Parameters

- **-i**: Interactive mode. This mode will query the user to input information for the machine, parallelism, repository details.
- **-f**: Specifies the input file to be parsed which represents the repositories to be configured. If this option is used in conjunction with the -i option then the setting in the input file are used first and then the user is prompted for additional repositories to configure.
- **-o**: Specifies the output file to save the repository configurations to. This allows the user to share the settings they enter with interactive mode with others.
- **-b**: This optional directory will be the location of the base directory where the build will be configured. The default is the current directory.
- **-h**: The help message

5.2.2 Input File Guidelines

Input files are expected to have the following format:
name,repo uri,branch,commit[,layers=layer1:layer2:...:layerN]

The first 4 values are required, whereas the layers= value is optional. By default all layer.conf files found in a repository will be selected unless the layers= option is set to limit the layers being used. The settings for the layers option should be the path from the base of the repository to the directory containing a conf directory with the layer.conf file. For example, when configuring openembedded-core the following can be used:
openembedded-core,git://github.com/openembedded/oe-core.git,master,HEAD This would select both the meta and meta-skeleton layers.

or, to limit to only the meta layer you could use the syntax
openembedded-core,git://github.com/openembedded/oe-core.git,master,HEAD,layers=meta

or, to explicitly set the meta and meta-skeleton layers use
openembedded-core,git://github.com/openembedded/oe-core.git,master,HEAD,layers=meta:meta-skeleton

It is needed to add the following variables to set up the environment
MACHINE = Target machine for which the project is being developed (e.g. emsbc-argon)
tnum = Parallelism options during build and compile (e.g. 4)
DOWNLOAD_LOC = Location of the downloads folder, can be shared with other projects. By default it is created as one of the 3 folders in YoctoBuildDirectory.

5.2.3 Functions

In the **file mode (f)**, the script is associated with the setup_emsbc-argon.txt file (provided as well by Emtrion), containing all the default parameters required for a particular Emtrion board. Some configuration variables are provided to be modified by the user, if required. The MACHINE, tnum, DOWNLOAD_LOC and the repo details can be checked in this file. If required, the user can add extra lines to provide the Yocto build with more layers than the default options.

The **interactive mode (i)** gives the user the flexibility to go step by step through the environment set up, by prompting the user to enter details in the terminal.

The **base directory (b)** containing the repositories and the YoctoBuildDirectory can be dictated by the argument. By default the location of the setup script is selected.

The **output file (o)** option can be utilized to save the value of either the interactive or the file mode in a text file, which can be reused by the user in the future.

For development, with Emtrion git repo access the Emtrion layers can be added by the script, otherwise 2 layers (meta-emtrion and meta-emtrion-bsp) and 2 files(yocto_setup_emtrion and setup_emsbc-argon.txt) will be provided in an archive to the user.

The script automatically checks out the correct versions of the other layers which are needed for building. After execution of the script you are in the build environment under the directory <BUILD_DIR>. The user also has the flexibility to add whatever layer is deemed to be relevant for the build by appending the test file in the aforementioned process.

It creates the bblayers.conf and local.conf from a sample provided with each hardware. The bitbaking environment is also readied by the script.

The successful execution of the script will create directories for each repository requested by the user. The YoctoBuildDirectory is the other directory created, with 3 sub directories for machine, sstate-cache and downloads. The user will end up in BUILD_DIR, where the execution of bitbake of the images are possible.

The script setup is useful for an already setup system as well updating or modifying the environment. The behavior of the setup script in the two different instances are listed in the table below.

Item under consideration	Initial Run	Rerun	Remarks
meta-layers	fetched	updated (if any)	
<BUILD_DIR>	created	obtained or created (if deleted)	
configuration file	copied or modified	copied or modified	local.conf (default stored in meta-emtrion-bsp layer)
configuration file	copied or modified	copied or modified	bbayers.conf (default stored in meta-emtrion-bsp layer)
<BUILD_DWNL>	created or obtained	obtained	
<BUILD_SSTATE>	created	obtained	
Asking confirmation for deleting build system	no	yes	delete process can take several minutes

Table 5.2: *Behavior of the setup script*

5.3 Emtrion's Yocto Layers

The Yocto layers provided with the software package have been described in the following section, with the associated directory structure.

Location	Remarks
meta-emtrion	Emtrion's basic meta layer for general Yocto development
|– conf | |– layer.conf	Configuration file for this meta layer
|– recipes	Recipes created by emtrion
| |– empurs-scripts | | |– empurs-scripts_0.1.bb | | |– files | | | |– empurs.service | | | |– devtmpfs | | | |– emPURS | | | |– emPURSinit.sh	Recipes and scripts for system update and restore
| |– emtrion-config | | |– emtrion-config_0.2.bb | | |– files | | | |– profile.qt | | | |– telnetd.sh	Basic configuration files
| |– gpio-utils | | |– files | | | |– gpio-utils.tar.bz2 | | |– gpio-utils_0.1.bb	Example of using GPIOs
| |– images	Recipes for various images offered by Emtrion
| | |– core-image-opengles.bb	Image with opengles support (not needed in this board)
| | |– core-image-purs.bb	Recipe for ramdisk image creation
| | |– emtrion-devkit-image.bb	Recipe for image of the board with rootfs
|– recipes-connectivity	
| |– openssh | |– files | | |– sshd_config | |–openssh_%.bbappend	Recipes for openssh to use ssh login in the boards
|– recipes-core	
| |– busybox | | |– busybox_%.bbappend | | |– files | | | |– busybox-udhcpc | | | |– defconfig	Recipe for busybox, providing a host of tools for linux
| |– initscripts | | |– files | | | |– umountfs | | |– initscripts_1.0.bbappend	Recipes for initscripts, used during system boot
| |– netbase | |– netbase-6.1 | | |– interfaces | |– netbase_6.1.bbappend	Recipes for netbase, important for TCP/IP communications
|– recipes-kernel	

|– gator	
| |– gator-daemon	
| | |– makefile_21.patch	
| |– gator-daemon_5.21.bb	Recipes not tested for this board
| |– gator-driver	
| | |– emtrion_activate_ETM_fwding_21.p	
| |– gator-driver_5.21.bb	
|– linux	
| |– files	Include file containing various linux parameters
| | |– linux.inc	
|– recipes-qt	
|– images	
| |– qt5-image-slim.bb	Packagegroups and images required for Qt5 support
|– packagegroups	
| |– packagegroup-qt5-slim.bb	
|– recipes-support	
|– devmem2	Recipes for devmem, used for reading and writing to different memory locations
| |– devmem2	
| | |– devmem2-fixups-2.patch	
| |– devmem2.bb	

Location	Remarks
meta-emtrion-bsp	Board specific Emtrion meta layer
|– meta-emtrion-emsbc-argon	Particular board specific components, relevant to this board
|– conf	Configuration files
|– distro	
| |-emt.eglfs.conf	Configuration and distro for this meta layer
|– layer.conf	
|– machine	
| |– emsbc-argon.conf	Machine specific configuration file, containing important information regarding the build and graphics support addition
| |– include	
| | |– gpu_add.inc	
|– default-config	Default files required for proper environment set up (may or may not be modified by the user while using the setup script for installation)
|– bblayers.conf	
|– local.conf	
|– recipes	Recipes created or modified by emtrion for the specific board
|– empurs-scripts	
| |– empurs-scripts_0.1.bbappend	
| |– files	Recipe for update or restore of system
| | |– emPURS.service	
| | |– emPURS	
|– emtrion-config	
| |– emtrion-config_0.2.bbappend	
| |– files	Recipe involving uboot parameters, pointer calculation as well as platform specific parameters for recovery and update
| | |– emPURS_plat	
| | |– pointercal	
| | |– uboot_script	
|– images	Images relevant to the dev kit
| |– core-image-purs.bbappend	Ramdisk image creation for system update and core features
| |– emtrion-devkit-image.bbappend	Image for the actual kit, with all the necessary attributes

|– m4example	
| |– m4example.bb	
| |– files	
| | |– README	Recipe for creation and installation of emtrion blinking LED demo using m4 co-processor
| | |– rpmsg_client_sample.ko	
| | |– fw_cortex_raw_m4.sh	
| | |– fw_cortex_emt_m4.sh	
| | |– fw_cortex_emt_ca7.sh	
| | |– OpenAMP_raw.elf	
| | |– emtm4demo.elf	
|– recipes-bsp	Recipes related to the particular bsp sourced from STM
| |– trusted-firmware-fa	recipe responsible from creation of the trusted firmware based on the STM
| |– tf-a-stm32mp_2.2.bb	
| |– tf-a-stm32mp-common.inc	
| |– u-boot	recipe responsible from creation of the u-boot based on the STM
| |– u-boot-stm32mp-emsbc-argon_2020.01.inc	
| |– u-boot-stm32mp_2020.01.bbappend	
| |– u-boot-stm32mp.inc	
|– recipes-core	
| |– psplash	
| | |– files	Recipe for splash screen image display(optional)
| | | |– 0001_psplash.c.patch	
| | | |– 0002_psplash-colors.h.patch	
| | | |– Logo_emtrion_new.jpg	
| | |– psplash_git.bbappend	
|– recipes-graphics	Recipes related to the graphics support
| |– mesa	Mesa support for software graphics acceleration
| | |– mesa_%.bbappend	
|– recipes-kernel	
| |– linux	recipe responsible for creation of the kernel based on STM
| | |– linux-stm32mp-argon_5.4.bblinux-stm32mp	
| | |– linux-stm32mp.inc	
| | |– linux-stm32mp-archiver.inc	
|– recipes-qt	
| |– packagegroups	Package groups for QT5 used in image creation
| | |– nativesdk-packagegroup-qt5-toolchain-host.bbappe	
| | |– packagegroup-qt5-qtcreator-debug.bbappend	
| | |– packagegroup-qt5-toolchain-target.bbappend	
| |– qt5	Recipe extensions and set up script for Qt5 support with eglfs
| | |– files	
| | | |– qt-eglfs_add.sh	
| | |– openstlinux-qt-eglfs.bbappend	
| | |– qtbase_%.bbappend	
| | |– qtwebkit_git.bbappend	
| | |– qtmultimedia_git.bbappend	
|– recipes-st	
| |– images	Extension for recipes generating bootfs or userfs partition
| | |– st-image-bootfs.bbappend	
| | |– st-image-userfs.bbappend	

6 Image Creation

The next step after setting up the build system as instructed in the previous chapter Installation, is to start building recipes and images for the emsbc-argon.

As mentioned before, the layer meta-emtrion and meta-emtrion-bsp provides the required images for the device. You can start building an image by prompting bitbake following the name of the image recipe. Enter in the terminal of the build environment:

bitbake <name_of_ image_ recipe>

bitbake core-image-purs

Builds the *initramfs* that is used for Emtrion devices' update mechanism. As the image is included by the other image emtrion-devkit-image, the image is automatically built while bitbaking this image, but only if the image was still not yet built. For this reason the image has to build explicitly, if any changes were made before building one of the other images.

bitbake emtrion-devkit-image

Builds the image for emsbc-argon. It creates a root file system with all the required components as well as QT5 support. Additionally it includes the initramfs, the kernel and device tree.

6.1 Output files

During the build process various objects and images are created. However, the most relevant images are installed in: **<BUILD_DIR>/tmp-glibc/deploy/images/<MACHINE>** and **<BUILD_DIR> /tmp-glibc/deploy/sdk**.

The exact names of the images are listed below. Note: Some of them are symbolic links.

Images	Description
./kernel/**fitImage***	Kernel, comprises zImage + device tree
./kernel/**stm32mp157c-<MACHINE>.dtb***	Device tree
rfs_image-emt-eglfs-<MACHINE>-dunfell-<DISTRO_VERSION>-{yyyymmddhhmmss}.rootfs.tar.gz	RootFS Archive
rfs_image-emt-eglfs-<MACHINE>-dunfell-<DISTRO_VERSION>-{yyyymmddhhmmss}.rootfs.ext4	RootFS in ext4 image format
ramdisk-<MACHINE>.rootfs.cpio.gz	Ramdisk for update mechanism
tf-a-stm32mp157c-<MACHINE>-trusted.stm32	tf-a, first stage bootloader(fsbl)
./u-boot/**u-boot-stm32mp157c-<MACHINE>-trusted.stm32**	U-boot image, second stage bootloader(ssbl)
../../sdk/**rfs_image-emt-eglfs-<MACHINE>-i686-toolchain-dunfell_<DISTRO_VERSION.sh**	SDK installer

Table 6.1: *Build output description*

(*) means a symbolic link

6.2 Root File System

As shown in the list above, the output of the root file system is a tar.gz archive. You can decompress it by the tar command. For testing or development we recommend to decompress the archive to the <**NFS_SHARE**> and then starting the device via *nfs*. Navigate to the directory <**BUILD_DIR**> and call

sudo tar xvzf tmp-glibc/deploy/images/<MACHINE> /name_of_rootfs_archive –C <NFS_ROOTFS>

Don't forget "sudo" otherwise the kernel won't be able to modify the files during starting of the system.

6.3 Boot directory

The directory structure of the root file system includes a directory called boot. In addition to the fitimage **linux** and **ramdisk** (for update root file system) a file **uboot_script** is located there.

This file implements some U-Boot command sequences and has a central part at booting in any case. For that, the uboot_script will be loaded first from the corresponding medium e.g. eMMC, before any other software part.

In the case of updating and booting by NFS, the environment of the U-Boot has to be set up before. This is discussed in detail in the chapter concerning booting.

While updating, the **rootfs** archive can later be stored in this directory to flash eMMC using NFS in linux environment.

7 Booting, updating and restoring

The Emtrion emSBC-Argon development kit can be booted, updated or restored using various methods. The U-Boot is one of the most popular methods, where using the already integrated U-Boot image, the device can be controlled with a few user defined variables, during boot time. However, this can only be used in presence of a working U-Boot.

The embedded ROM in the STM micro-controller searches for the bootloaders, which is responsible for implementing the boot strategies for the device. It starts with the chip initialization and clock frequency detection. During the execution of the ROM boot, the device looks for a valid copy of the system boot in one of the external non-volatile memories (NVM) as selected by the boot pins. When this is available, it copies the relevant part to the internal SRAM and starts booting the device.

The development kit has 4 DIP switches acting as the boot pins; where the first 3 pins are to be controlled for different boot options. The relevant pin positions can be found in the hardware manual for the device.

7.1 Updating from Yocto thud to dunfell

Due to the Yocto dunfell package comes with a tf-a firmware and the bootloader was upgraded, both parts have to be updated on a board installed Yocto thud, before any booting.

To manage this step, please follow the guidline below.

1. Prepare the host system for booting NFS, as described in section 4.2.5 on page 11 and 4.2.6 on page 11.
2. Do further preparation of the U-Boot environment. Printenv the board specific U-Boot enviroment variables **hw_revison**, **hw_product_type**, **hw_serial_nr** and **ethaddr** to save them outside of U-Boot. Create and set the U-Boot environment variable and make them persistent
   setenv fs_type nfs && saveenv
3. Perform updating of the tf-a firmware and U-Boot as described in section 7.3.2.2 on page 26.
4. Reset the board and then stop booting
5. Restore the board specific U-Boot environment variables saved before and make them persistent.
   setenv hw_revison ... && setenv hw_product_type ... && setenv hw_serial_nr ... && setenv ethaddr ... && saveenv

7.2 Default boot and other

By default, the emSBC-Argon contains the bootloaders in the **NOR flash memory**. It is divided into 3 partitions. The first **2 partitions** are for the **first stage bootloader** (tf-a); at a time only one of them is active and the second one acts as a backup. The **third partition** contains the **second stage bootloader** (U-boot image). The boot pin for this boot should be "0110". After connecting to the host system via a USB to serial cable and pressing the reset switch should start the device as mentioned in the "Device Startup" chapter 4.3.

The **root file system** is stored in the **eMMC** present in the development kit. After the bootloaders are loaded, the default boot command from U-boot results in loading the **uboot_script** located in /boot from the eMMC to execute the corresponding command to start up the linux kernel.

The U-boot supports the possibility of running a RFS update command to fetch data from the NFS server and install a RFS to the flash memory connected to the board. It restores a system if something goes wrong with the rootfs.

The U-boot also has the capability to make the device look for the rootfs in the NFS server and boot from it, making it less cumbersome, to test and modify a system during development.

7.2.1 fitimage linux

The linux kernel is presented by the fitimage **linux**. The fitimage is located in **/boot** of the RFS. Besides the default boot configuration, the fitimage contains three additional configurations. Each of the configuration affects the rotation of the display.

7.2.1.1 Display rotation

The rotation of display is supported in 90°, 180° and 270°. Each rotation is presented by a different device tree in the fitimage. The configuration can be selected by the variable **configX** of the uboot_script.

configuration	device tree	configX set to
default	stm32mp157c-emsbc-argon.dtb	""
rotation 90°	stm32mp157c-emsbc-argon_display90.dtb	"_display90"
rotation 180°	stm32mp157c-emsbc-argon_display180.dtb	"_display180"
rotation 270°	stm32mp157c-emsbc-argon_display270.dtb	"_display270"

Table 7.1: *Boot configurations*

7.3 U-boot

The basic task of U-Boot is to load the operating system from bulk memory into RAM and then start the kernel. It can also be used to initiate an update of the RootFS. Furthermore, it can be configured to dictate from which medium the operating system is to be booted from, for example eMMC, SD-card or NFS.

7.3.1 Basic Uboot Operation

To work with U-Boot, first use a terminal program like picocom to connect to the serial line of the board. As soon as the U-Boot prompt appears in the terminal, U-Boot is ready to receive commands. The general U-Boot documentation can be found here: http://www.denx.de/wiki/U-Boot/Documentation

U-Boot has a set of environment variables which are used to store information needed for booting the operating system. Variables can contain details such as IP addresses, but they can also contain a whole script of actions to perform sequentially. By default the NOR flash is the location of storing the u-boot variables. The following commands explain the basic handling of environment variables:

U-boot command	Explanation
printenv [variable]	This shows the value of the specified variable. If no variable is specified, the whole environment is shown.
setenv [variable] [value]	Set a variable to a specific value. If no value is specified, the variable gets deleted.
editenv [variable]	Edit/modify a variable with an existing value.
saveenv	Make your changes permanent, so they remain after power off or reboot.The environemnt is written in the NOR flash

Table 7.2: *Uboot commands*

7.3.2 Using U-Boot to change boot device or update the system

This section describes how U-Boot has to be setup for updating and booting.

The variable serverip has to be set to the IP-address of the Host. You can get the IP-address by prompting

sudo **ip a** or sudo **ifconfig**

in the terminal of your Host.

Take the IP-address of the corresponding network adapter and assign it to the variable serverip in the U-Boot console. The format of [IP-address] is dot decimal notation.

STM32MP # setenv serverip <IP-address>

The **"flash_boot"** variable can be set to dictate the default command to be used, when the u-boot is started. This indicates the location of the root file system to start the kit. Some valid examples for **flash_boot are "run emmc_boot", "run sd_boot", or "run net_boot"**. run emmc_boot is the default one.

To use the NFS for easy update and boot procedures, the **NFS** server and the client has to be correctly set up, which is described in the section 7.4 on page 28.

7.3.2.1 Updating of the root file system and kernel

Due to the rootfs.tar.gz archive contains the root file system as well the kernel, updating and restoring is always the same process. While the update process is realized by emtrion's update mechanism emPURS following conditions has to be managed before.

1. Installation of emPURS as described in 4.2.8 on page 11
2. Setting up the U-boot environment as described in 7.4 on page 28

Then copy an image of the rootfs.tar.gz archive created with the Yocto from the **<BUILD_DIR>/tmp-glibc/deploy/images/<M** to the **<NFS_RESTORE>**/boot directory. Please note, the name of the rootfs.tar.gz archive is expected as **rfs_image-emt-eglfs-emsbc-argon-dunfell.tar.gz** and has possibly renamed.

Perform the update process by replacing of <NFS_RESTORE> on your conditions. Then paste and copy the following command sequence

setenv nfsroot <NFS_RESTORE> && run update_rootfs

to the U-Boot environment and execute it.

7.3.2.2 Updating of U-Boot and tf-a

As mentioned in 7.2 on page 23, the tf-a and U-Boot resides in the NOR-Flash at specified partitions. The layout is presented with table 7.3.

Partition	Image	Linux MTDx
1	tf-a, FSBL	mtd0
2	tf-a, FSBL	mtd1
3	U-Boot, SSBL	mdt2
4	uboot_env	mtd3

Table 7.3: *NOR-flash layout*

There are several mechanism to update U-Boot and the tf-a. However, updating of any of those images is not necessary in general. Be careful performing the update process. If anything goes wrong while updating, a normal boot is no longer possible.

1. One method to update U-Boot and tf-a is by U-Boot itself, loading the new U-Boot and tf-a from a NFS-Share to the RAM of the target and then flashing to the NOR-Flash. **This method is required if you have a Yocto version before dunfell**. Follow the steps below.
   a) Prepare the share <NFS_RESTORE> as described in 4.2.8 on page 11, if not yet done.
   b) Put the tf-a and U-Boot image to <NFS_RESTORE>/boot
   c) Import the uboot_script in to the U-boot environment by the following commando sequence at the U-Boot prompt. We assume using DHCP.
      dhcp && setenv nfsroot <NFS_RESTORE> && setenv serverip xxx.xxx.xxx.xxx
      && nfs ${loadaddr} ${nfsroot}/boot/uboot_script &&
      env import -t ${fileaddr} ${filesize}
   d) Start the update process by the command
      run update_uboot
   e) Set the default U-boot environment if a new U-boot was flashed by performing the command sequence
      env default -a -f && saveenv && res
2. Another one is from linux by booting NFS. Follow the guideline below.
   a) Installation of an image rootfs.tar.gz archive created with the Yocto as described in 4.2.6 on page 11, if not yet done.
   b) Copy the tf-a and U-Boot image to <NFS_ROOTFS>/home/root
   c) Setting up the U-boot environment as described in 7.4 on page 28
   d) Perform booting from NFS by replacing of <NFS_ROOTFS> on your conditions. Then paste and copy the following commando sequence
      setenv nfsroot <NFS_ROOTFS> && run net_boot
   to the U-boot environment and execute them.
   e) From the target's linux prompt, enter first the command for deleting and then flashing of the corresponding mtd partition. Please make this work carefully. Following the example of flashing U-Boot.
      flash_erase /dev/mtd2 0 0
      flashcp -v u-boot-stm32mp157c-emsbc-argon-trusted.stm32 /dev/mtd2
   Repeat the steps above to flash the tf-a on the other mtd partitions.
   f) Set the default U-boot environment if a new U-boot was flashed by performing the command sequence
      env default -a -f && saveenv && res

7.3.2.3 Booting

The default boot method is already mentioned in section 7.2 on page 23. However, there are alternate options for the system startup which is explained below.

Booting from SD card

The user has the flexibility to create an entirely **portable image** for starting the device using an SD card. The SD card is divided into 4 minimum partitions i.e. **fsbl1, fsbl2, ssbl and rootfs**. The images created from Yocto can now be added to the SD card. Insert an SD card into a card reader on the Linux based development system. It will show up as /dev/sdb or /dev/sdc or similar. We will use /dev/sdX as an example. If it gets automounted (the command mount will give you a list of mounted devices and their mount point), first unmount it. You need to have root rights to do the following steps to create such an SD card as listed below:

1. Install the partitioning tool sgdisk, if not yet installed
   sudo apt-get update
   sudo apt-get install gdisk
2. sudo sgdisk -o /dev/sdX
3. sudo sgdisk –resize-table=128 -a 1 -n 1:34:545 -c 1:fsbl1 -n 2:546:1057 -c 2:fsbl2 -n 3:1058:5153 -c 3:ssbl -n 4:5154: -c 4:rootfs -p /dev/sdX
4. sudo dd if=<BUILD_DIR>/tmp-glibc/deploy/images/<MACHINE>/tf-a-stm32mp157c-emsbc-argon-trusted.stm32 of=/dev/sdX1
5. sudo dd if=<BUILD_DIR>/tmp-glibc/deploy/images/<MACHINE>/tf-a-stm32mp157c-emsbc-argon-trusted.stm32 of=/dev/sdX2
6. sudo dd if=<BUILD_DIR>/tmp-glibc/deploy/images/<MACHINE>/u-boot/u-boot-stm32mp157c-emsbc-argon-trusted.stm32 of=/dev/sdX3
7. sudo dd if=<BUILD_DIR>/tmp-glibc/deploy/images/<MACHINE>/rfs_image-emt-eglfs-emsbc-argon.ext4 of=/dev/sdX4 bs=4M

Selecting the boot pins as **"0100"**, the development kit will boot from the SD card. Stop auto-boot by pressing any key during the system startup. Use the following commands to load the rootfs from the SD card.

For a single use:

STM32MP # run sd_boot

For changing the default boot device:

STM32MP # setenv flash_boot run sd_boot
STM32MP # saveenv

The integrated NOR flash and eMMC will be totally ignored and the system starts independently from the SD card in the process.

Booting from NFS server

1. Install the prebuild rootfs.tar.gz or a new created one with Yocto as described in 4.2.6 on page 11
2. Setting up the U-boot environment as described in 7.4 on the following page
3. Perform booting from NFS by replacing of <NFS_ROOTFS> on your conditions. Then copy and paste the following commando sequence
   setenv nfsroot <NFS_ROOTFS> && run net_boot
   to the U-boot environment and execute it.

7.4 NFS Setup

Update of the U-Boot or RootFS is done as mentioned earlier, using NFS. Avoiding the update process for test purpose after each modification while developing, it is recommended to use NFS for booting, too.

For that a NFS-Server must be available on the Host and a <NFS_SHARE> has to be exported. However, setting up a NFS-Server and exporting a NFS-share can be different from the linux distribution. Be sure to make this work correct, please inform yourself how this work has to be done at your distribution. For Debian buster used in this example, the following steps are advisable.

1. Get nfs server in the host system
   sudo apt-get install nfs-kernel-server
2. Edit in etc/exports
   <**NFS_SHARE**> *(rw,nohide,insecure,no_subtree_check,async,no_root_squash)
3. Unpack rfs_image-emt-eglfs-emsbc-argon-dunfell.tar.gz to the folder <NFS_ROOTFS>.
4. Restart server in host after nfs addition
   sudo service nfs-kernel-server restart

In the target system connected via picocom:

1. Stop autoboot by pressing a key during the startup
2. Run the following Uboot commands:
   U-Boot # setenv serverip xxx.xxx.xxx.xxx (obtain host system IP)
   U-Boot # setenv ip-method static/dhcp (either static or dhcp)
   U-Boot # setenv ipaddr xxx.xxx.xxx.xxx (set a target IP ,only for static)
   U-Boot # setenv netmask xxx.xxx.xxx.xxx (only for static)
   U-Boot # printenv (to verify that the changes have been updated in the Uboot environment)

In general the <NFS_SHARE> has pointed to the unpacked RootFS. While booting or updating, the directory boot of the RootFS takes a central position. It includes all the needed components used by the different processes. Due to of the central position of the directory “boot”, the need of an unpacked RootFS is not necessary while updating. In this case the <NFS_SHARE> must only contain a sub-directory boot which includes the required files mentioned in previous sections. If you want to update the RootFS using its archive, you also have to locate the archive there.

The basic structure of the <NFS_SHARE> looks like as following. **<NFS_SHARE>/boot**

8 Using ARM Cortex M4 processor

The STM32MP157 being an MPU gives the user the flexibility to use the M4 coprocessor in conjuction with the A/ processor. The application has to be built to be suitable for the controller, along with the firmware. More details for coprocessor development guidelines can be found in https://wiki.st.com/stm32mpu/wiki/STM32CubeMP1_development_guidelines.

The Yocto image includes two examples, emtm4demo and OpenAMP_raw. They are located in the directory

/usr/lib/emtm4demos

8.1 emtm4demo

This demo exchanges messages between the A7 and M4 processor. The activities between both processors are showing on blinking led in changing the colours red and yelloc from default blinking green.

Starting the example is supported by two scripts. One for loading the program to the M4 side and the other starting the ping test by using rpmsg channel at the A7 side.

1. Navigate to the directory of the demo
   /usr/lib/emtm4demos/emt4demo
2. Start loading the program to the M4 side by performing the script
   ./fw_cortex_emt_m4.sh start
3. Loading the program you will see the messages below

```
1        fw_cortex_m4.sh: fmw_name=emtm4demo.elf
2        [ 2493.086503] remoteproc remoteproc0: powering up m4
3        [ 2493.090592] remoteproc remoteproc0: Booting fw image emtm4
    demo.elf, size 197980
4        [ 2493.097994]  remoteproc0#vdev0buffer: assigned reserved
    memory node vdev0buffer@10044000
5        [ 2493.107295] virtio_rpmsg_bus virtio0: creating channel
    rpmsg-tty-channel addr 0x0
6        [ 2493.113977] virtio_rpmsg_bus virtio0: rpmsg host is online
7        [ 2493.123254]  remoteproc0#vdev0buffer: registered virtio0 (
    type 7)
8        [ 2493.128326] rpmsg_tty virtio0.rpmsg-tty-channel.-1.0: new
    channel: 0x400 -> 0x0 : ttyRPMSG0
9        [ 2493.136352] remoteproc remoteproc0: remote processor m4 is
    now up
10       root@emsbc-argon:/usr/lib/emtm4demos/emtm4demo#
11
```

Listing 8.1: *Linux console messages*

4. Start the ping test **./fw_cortex_emt_ca7.sh**
5. The led switches from blinking green to blinking changing the colours red and yellow

8.2 OpenAMP_raw

This sample also transfers messages between the M4 and A7 processor, however by using the rpmsg example of the kernel. Starting the example requires the following steps.

1. Load the kernel module
   modprobe rpmsg_client_sample
2. Navigate to the directory of the demo
   /usr/lib/emtm4demos/OpenAMP_raw
3. Start loading the program to the M4 side by performing the script
   ./fw_cortex_raw_m4.sh start
4. Loading the program you will see the messages below. The last message will confirm the success by the message goodbye!

```
1       fw_cortex_m4.sh: fmw_name=OpenAMP_raw.elf
2       [ 1013.049070] remoteproc remoteproc0: powering up m4
3       [ 1013.053136] remoteproc remoteproc0: Booting fw image
    OpenAMP_raw.elf, size 197976
4       [ 1013.060723]  remoteproc0#vdev0buffer: assigned reserved
    memory node vdev0buffer@10044000
5       [ 1013.069628] virtio_rpmsg_bus virtio0: rpmsg host is online
6       [ 1013.069790] virtio_rpmsg_bus virtio0: creating channel
    rpmsg-client-sample addr 0x0
7       [ 1013.073810]  remoteproc0#vdev0buffer: registered virtio0 (
    type 7)
8       [ 1013.084386] rpmsg_client_sample virtio0.rpmsg-client-sample
    .-1.0: new channel: 0x400 -> 0x0!
9       [ 1013.092912] remoteproc remoteproc0: remote processor m4 is
    now up
10      root@emsbc-argon:/usr/lib/emtm4demos/OpenAMP_raw#
11      [ 1013.111462] rpmsg_client_sample virtio0.rpmsg-client-sample
    .-1.0: incoming msg 1 (src: 0x0)
12      [ 1013.118813] rpmsg_client_sample virtio0.rpmsg-client-sample
    .-1.0: incoming msg 2 (src: 0x0)
13      [ 1013.126936] rpmsg_client_sample virtio0.rpmsg-client-sample
    .-1.0: incoming msg 3 (src: 0x0)
14      [ 1013.135347] rpmsg_client_sample virtio0.rpmsg-client-sample
    .-1.0: incoming msg 4 (src: 0x0)
15      [ 1013.143562] rpmsg_client_sample virtio0.rpmsg-client-sample
    .-1.0: incoming msg 5 (src: 0x0)
16      [ 1013.151963] rpmsg_client_sample virtio0.rpmsg-client-sample
    .-1.0: incoming msg 6 (src: 0x0)
17      [ 1013.160448] rpmsg_client_sample virtio0.rpmsg-client-sample
    .-1.0: incoming msg 7 (src: 0x0)
18      [ ..........]
19      [ ..........]
```

```
20          [ ..........]
21          [ 1013.912254] rpmsg_client_sample virtio0.rpmsg-client-sample
    .-1.0: incoming msg 96 (src: 0x0)
22          [ 1013.920774] rpmsg_client_sample virtio0.rpmsg-client-sample
    .-1.0: incoming msg 97 (src: 0x0)
23          [ 1013.929141] rpmsg_client_sample virtio0.rpmsg-client-sample
    .-1.0: incoming msg 98 (src: 0x0)
24          [ 1013.937625] rpmsg_client_sample virtio0.rpmsg-client-sample
    .-1.0: incoming msg 99 (src: 0x0)
25          [ 1013.946044] rpmsg_client_sample virtio0.rpmsg-client-sample
    .-1.0: incoming msg 100 (src: 0x0)
26          \textbf{[ 1013.954524] rpmsg_client_sample virtio0.rpmsg-
    client-sample.-1.0: goodbye!}
27
```

Listing 8.2: *Linux console messages*

9 SDK

As mentioned in 4.2.7 on page 11 the SDK supports application development outside the Yocto build system. You can either use the prebuilt one or a new created one. Creating an SDK is performing by the command

bitbake emtrion-devkit-image –c populate_sdk

The result is a SDK installer containing the toolchain and the sysroot which includes and matches the target root file system. The installer is stored in

<BUILD_DIR>/tmp-glibc/deploy/sdk/

9.1 Installing the SDK

While running the SDK installer, the user is asked for the installation directory. The default location is **/opt/emtrion/emsbc-argon/dunfell_v1.0.0**. However we recommend to change the location to **<SDK_DIR>**. From inside the location **<BUILD_DIR>** start the installer as follows.

./tmp-glibc/deploy/sdk/rfs_image-emt-eglfs-emsbc-argon-i686-toolchain-dunfell_v1.0.0.sh

The script will start extracting the SDK and then show the message "Setting it up....Done", indicating that the script has finished successfully.

9.1.1 Setting up the SDK environment

Before you can start developing applications, the environment for the terminal or the application has to be set up. For that purpose a script is installed during the installation process of the SDK. The script is stored in the SDK's directory of **<SDK_DIR>**.

Performing the setup procedure, the script has to be sourced as follows.

source <SDK_DIR>/environment-setup-cortexa7t2hf-neon-vfpv4-emt_eglfs-linux-gnueabi

The environment is only valid in the context of the terminal where this script has been called.

9.2 Qt5 with Yocto development

To run Qt5 with eglfs support, the following two scripts have to be executed before starting with the development.

1. cd /etc/profile.d
2. ./qt-eglfs.sh
3. ./qt-eglfs_add.sh

The following steps are to be followed for configuring the SDK in Qt Creator. This is a general method which is shown as an example and the user may have make adjustments depending on the system and the build environment.

The following is only an example on how to setup a Yocto Project standard SDK built using Debian distribution for a Linux x86_64 host in Qt Creator 4.2.0 based on **Qt 5.14.2** with GCC in a 64 bit machine:

1. Open the "Tools", "Options..." menu and select the **Build & Run** section
2. Use <**SDK_DIR**>**/sysroots/i686-emt_eglfs-linux/usr/bin/qmake** as qmake location in **Qt versions** tab. The version for development is **5.14.2**, the latest supported by product. Add a name parameter to the Qt version for easy identification in the future e.g. qt5 sdk.
3. Use <**SDK_DIR**>**/sysroots/i686-emt_eglfs-linux/usr/bin/arm-emt_eglfs-linux-gnueabi/arm-emt_eglfs-linux-gnueabi-g++** as C++ compiler path and select ABI arm-linux-generic-elf-32bit in the Compilers tab. Add a name to the manually added compilers.
4. Use <**SDK_DIR**>**/sysroots/i686-emt_eglfs-linux/usr/bin/arm-emt_eglfs-linux-gnueabi/arm-emt_eglfs-linux-gnueabi-gcc** as C compiler path and select ABI arm-linux-generic-elf-32bit in the Compilers tab
5. Use <**SDK_DIR**>**/sysroots/i686-emt_eglfs-linux/usr/bin/arm-emt_eglfs-linux-gnueabi/arm-emt_eglfs-linux-gnueabi-gdb** as debugger path in Debuggers tab
6. If CMake is required, use <**SDK_DIR**>**/sysroots/i686-emt_eglfs-linux/usr/bin/cmake** as cmake path in CMake tab
7. Open the **Devices** section
8. In the **Devices** tab create a new device of type "Generic Linux Device", specify IP address and authentication details
9. Return to the **Build & Run** section
10. Create a new kit with name "**emsbc-argon**" selecting the configurations <**SDK_DIR**>**/sysroots/i686-emt_eglfs-linux** as sysroot path.
11. Remove any **Qt mkspec** if it has been added by default.
12. Press Apply and exit Qt Creator
13. At this point the environment set up script has to be already executed in a terminal.
14. Start Qt Creator from the current command line, where the environment has been modified. Use the location where Qt has been installed in the host PC.
15. Another method is to copy the entire environment set up file and add it to the **Environment** variable in QT options. This gives the Qt creator the flexibility to be started from any shortcuts, not only the actual modified terminal.
16. Create a new project otherwise build and compile an existing project like "Qt Quick Demo- Clocks" from the Examples in QT creator.
17. By running the successfully built program on the target device emSBC-Argon, the clocks should be visible in the screen attached the board.

Alternately, the Qteverywheredemo from Qt is also built in the image rootfs and located under the directory /usr/share. This demo can also be executed to see the results in a supported screen.

10 Further information

Online resources

Further information can be found on the Emtrion support pages.

www.support.emtrion.de

We support you

Emtrion offers different kinds of services, among them Support, Training and Engineering. Contact us at sales@emtrion.com if you need information or technical support.