DOCUMENTATION

emtrion
embedded systems

emSBC-Argon Yocto Manual

Yocto Based BSP Manual

Rev. 2 / 05.11.2019
Author: Saptarshi Mitra
Last Change: November 5, 2019

Revision: Rev. 2 / 05.11.2019

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

Contents

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 Thud (2.6)**, launched in November 2018. 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**, sourced from the STM Linux repo and the version is **4.19.49**. The developer kit is managed by the specific kernel configuration file and the device tree file, along with patches required for the correct operation of the associated peripherals.

The U-boot system implemented in the board is also based on the STM modified repository, using u-boot version **v2018.11**. Additional patches and have been provided in the meta-emtrion-bsp layer 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 Mainline	v4.19.49
Yocto	Thud v2.6.2
Bitbake	v1.40.0
Gcc version	8.2.0
Openssh	v7.8
Busybox	v1.29.3
Initscripts	v1.0
Netbase	v5.4
Qt	v5.11.2
Gstreamer	v1.0
Mesa	v18.1.9
Graphics support	OpenGL_ES 2.0
Graphics driver	Gcnano v6.2.4

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/2.6/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)

2 Terms and definitions

Term	Definition
Target	Module: emSBC-Argon
Host	Workstation, Developer PC
Toolchain	Compiler, Linker, etc.
RootFS	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 Linux Virtual Machine

3.1 Test

The set up environment demonstrated or referred to in the entire document is a virtual machine running Linux having the following settings. However, it is expected that the discussion holds true for other comparable systems.

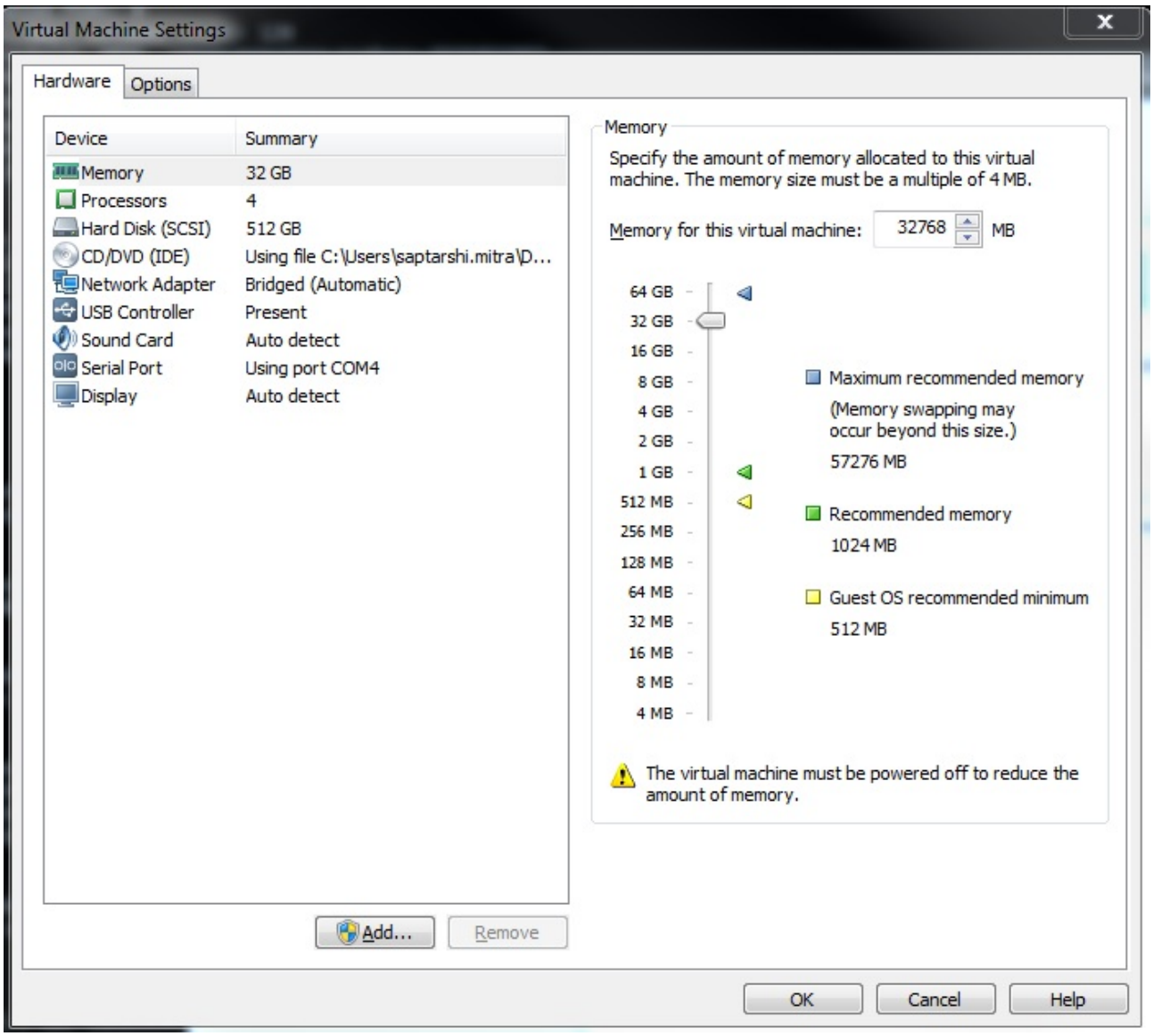

Figure 3.1: *Virtual Machine settings*

However, the settings are strongly dependent on the PC and have to be adjusted later on your PC. A powerful PC will cut down time spent on debugging as well as image creation.

- Memory

- Number of processors
- Network adapter
- Serial port

3.2 Virtual Machine settings

System requirements
As Linux Distribution in the system, **Debian GNU/Linux 9 (stretch)** is used. Please note that if using Debian, Yocto Thud used for this device, is supported by **Jessie and Stretch**. The other possible development environments are listed in the Yocto reference manual. The packages mentioned below are only indicative of the packages that may be required for other systems. The distribution was set up with general Yocto Project system requirements described in the chapter "1. System Requirements" of the Reference Manual Yocto Project 2.6.2 Release.

http://www.yoctoproject.org/docs/2.6/ref-manual/ref-manual.html

Further components are included

- Serial ports added to the virtual machine appear at /dev/ttySn, USB serial converters at **/dev/ttyUSBn**.
- A NFS server exporting the nfs share **/home/hico/nfs**.
- The serial terminal program **picocom** for connecting to the target.
- The Yocto-Layers **meta-emtrion** and **meta-emtrion-bsp**.
- Packages needed to build an image are listed in the following section. Some of the packages may already be installed in the system, but ensuring the presence of all will avoid any unforeseen problems during the build.

```
$ sudo apt-get install sed wget curl cvs subversion git-core coreutils
 unzip texi2html texinfo docbook-utils gawk python-pysqlite2 diffstat
help2man make gcc build-essential g++ desktop-file-utils chrpath
libxml2-utils xmlto docbook bsdmainutils iputils-ping cpio python-wand
 python-pycryptopp python-crypto libsdl1.2-dev xterm corkscrew nfs-
common nfs-kernel-server device-tree-compiler mercurial u-boot-tools
libarchive-zip-perl ncurses-dev bc linux-headers-generic gcc-multilib
libncurses5-dev libncursesw5-dev lrzsz dos2unix lib32ncurses5 repo
libssl-dev

```

Listing 3.1: *Recommended Packages from STM*

3.3 Starting the VM

The VM is a compressed ZIP archive. Changing settings and starting the VM is used by the VMware Player or VMware Workstation. Here is the link for downloading the VMware Player.

https://www.vmware.com/go/downloadplayer

The corresponding manual is available in the following link.

https://docs.vmware.com/en/VMware-Workstation-Pro/15.0/workstation-pro-15-user-guide.pdf

After decompressing of the VM and importing it by the VMware Player, first check if the settings above are fit to your PC. If not, adjust the settings by reading the corresponding chapters of the specified manual.

Please note, the size of the VM will increase up to 128 GB while you are working with it.

3.4 Preconfigured variables

Within the VM there are used some predefined locations. In the scope of this document the locations are assigned to specified placeholders. They are listed in the table below.

Placeholder	Assignment	Remark
MACHINE	emsbc-argon	machine name
DISTRO	emt-eglfs	distro name created for emSBC-Argon
HOME_DIR	/home/hico	home directory of user hico (Replace it with the corresponding home directory in your PC)
NFS_SHARE	<HOME_DIR>/nfs	Location exported by the NFS
NFS_ROOTFS	<NFS_SHARE>/<MACHINE>/rootfs	nfs share for booting the root file system by using NFS
INST_DIR	<HOME_DIR/Yocto_sbc-emstamp-argon	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
HOME_DWNL	<HOME_DIR>/Downloads	Location of the pre-built images, SDK
SDK_DIR	/opt/emtrion/emsbc-argon/thud_v1.0.0)	SDK with tools, sources and libraries

Table 3.1: *Preconfigured Variables*

4 Installation

4.1 Emtrion development kit package

4.1.1 Package Contents

Emtrion software package for emSBC-Argon development kit comes as an archive file, which on extraction reveals the following directories and files:

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.
2. **images**
   a) **ramdisk-emsbc-argon.cpio.gz**: This image is required for the EMPURS script, for updating and restoring the system.
   b) **rfs_image-<DISTRO>-<MACHINE>-<DISTRO_CODENAME>-<DISTRO_VERSION>-{yyyymmddhhmmss}.rootfs.ext4**: Linux image file for the rootfs system created for the development kit.
   c) **rfs_image-<DISTRO>-<MACHINE>-<DISTRO_CODENAME>-<DISTRO_VERSION>-{yyyymmddhhmmss}.rootfs.tar.gz**: This archive file can be extracted and the device can be booted and accessed using nfs boot.
   d) **zImage**: Kernel image pertinent to the developer kit.
   e) **stm32mp157c-emsbc-argon.dtb**: Device tree for the emsbc-argon board, developed for the linux kernel.
   f) **u-boot-spl.stm32-stm32mp157c-<MACHINE>-basic**: U-boot spl i.e. first stage bootloader (FSBL)
   g) **u-boot-stm32mp157c-<MACHINE>-basic.img**: U-boot image, second stage bootloader (SSBL)
3. **yocto_emtrion_setup.sh**: This script can be utilized to setup the entire environment for Yocto development.
4. **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 instructions.
5. **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.

More information about the layers are available in the Emtrion's Yocto layers section. The setup script is described in details in the Yocto setup script section.

4.1.2 Installation steps

The installation of the Emtrion layers has to be performed in order of the following steps.

1. Download the Yocto package for the emSBC-Argon board from the Emtrion website to **HOME_DWNL**.
2. Extract the contents to any directory and name it suitably for future reference.

3. Move **meta-emtrion** and **meta-emtrion-bsp** directories either to an already existing or intended **<INST_DIR>** of Yocto. (For those with direct access to the Emtrion git repositories, this step in unnecessary, as the setup text file can be modified to install these)
4. The setup script and the text file must also be placed in the **<INST_DIR>** directory.
5. Set up of the machine dependent build directory by sourcing the corresponding setup script from the **<INST_DIR>**. e.g. ***. ./yocto_emtrion_setup.sh -f setup_emsbc-argon.txt***

4.1.3 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
|– sstate-cache	contains the build states created during the build process

Table 4.1: *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.

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

Normally, one each of the download and sstate-cache directories exist for each <MACHINE>. in order to save space and time, the setup script creates common downloads and sstate-cache directories. Setting up a machine with the machine specific setup script supports sharing of these directories, and allows executing of more than one build processes at the same time.

4.2 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.

4.2.1 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

4.2.2 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.

4.2.3 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 4.2: *Behavior of the setup script*

4.3 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 | | | |– 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_1.29.3.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-5.4 | | |– interfaces | |– netbase_5.4.bbappend	Recipes for netbase, important for TCP/IP communications
|– recipes-kernel	

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

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 | |– layer.conf	Configuration and distro for this meta layer
| |– machine | | |– emsbc-argon.conf | | |– include | | | |– gpu_add.inc	Machine specific configuration file, containing important information regarding the build and graphics support addition
|– default-config | |– bblayers.conf | |– local.conf	Default files required for proper environment set up (may or may not be modified by the user while using the setup script for installation)
|– recipes	Recipes created or modified by emtrion for the specific board
| |– empurs-scripts | | |– empurs-scripts_0.1.bbappend | | |– files | | | |– emPURS	Recipe for update or restore of system
| |– emtrion-config | | |– emtrion-config_0.2.bbappend | | |– files | | | |– emPURS_plat | | | |– pointercal | | | |– uboot_script	Recipe involving uboot parameters, pointer calculation as well as platform specific parameters for recovery and update
| |– 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	Recipe for creation and installation of emtrion blinking LED demo using m4 co-processor
| |– files	
| | |– emtm4demo.elf	
| | |– fw_cortex_m4.sh	
|– recipes-bsp	Recipes related to the particular bsp sourced from STM
|– u-boot	
|– u-boot-stm32mp	
| |– 0001-Ethernet-reset-pin-initialization.patch	
| |– 0002-Makefile-modification-to-build-devicetree.patch	
| |– 0003-Add-ISSI-spi-support.patch	
| |– 0004-U-boot-script-modifications-for-emtrion-boards.p	
| |– arch	Patches, device trees, configuration file and recipes responsible from creation of the u-boot from the STM layer
| | |– arm	
| | | |– dts	
| | | | |– stm32mp157c-emsbc-argon.dts	
| | | | |– stm32mp157c-emsbc-argon-u-boot.dtsi	
| | | | |– stm32mp157c-emstamp-argon.dts	
| |– configs	
| | |– emsbc-argon_defconfig	
|– u-boot-stm32mp1-emsbc-argon_2018.11.inc	
|– u-boot-stm32mp_%.bbappend	
|– recipes-core	
|– psplash	
| |– files	
| | |– 0001_psplash.c.patch	Recipe for splash screen image display(optional)
| | |– 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	
| |– linux-stm32mp	
| |– 0001-Makefile-modifications-to-build-devicetrees.patch	
| |– 0002-Build-deb-modifications-for-emtrion-board.patch	Patches and recipes responsible from creation of the kernel from the STM developed linux with the relevant device tree and defconfig files
| |– arch/arm/boot	
| | |– dts	
| | | |– stm32mp157c-emsbc-argon.dts	
| | | |– stm32mp157c-emstamp-argon.dts	
| | |– configs	
| | | |– emsbc-argon_defconfig	
|– linux-stm32mp_4.19.bb	
|– recipes-multimedia	Recipes for multimedia frameworks
|– gstreamer	Recipes for gstreamer and video4linux offering multimedia support sourced from poky and openembedded layers
|– v4l2apps	
|– recipes-qt	
|– packagegroups	
| |– nativesdk-packagegroup-qt5-toolchain-host.bbappend	Package groups for QT5 used in image creation
| |– packagegroup-qt5-qtcreator-debug.bbappend	
| |– packagegroup-qt5-toolchain-target.bbappend	

| |– qt5 | | |– files | | | |– qt-eglfs_add.sh | | |– openstlinux-qt-eglfs.bbappend | | |– qtbase_%.bbappend | | |– qtmultimedia_git.bbappend | | |– qtmultimedia_git.bbappend	Recipe extensions and set up script for Qt5 support with eglfs
|– recipes-st	
| |– images | | |– st-image-bootfs.bbappend | | |– st-image-userfs.bbappend	Extension for recipes generating bootfs or userfs partition

5 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.

5.1 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
zImage*	Kernel
stm32mp157c-<MACHINE>.dtb*	Device tree
rfs_image-<DISTRO>-<MACHINE>-<DISTRO_CODENAME>-<DISTRO_VERSION>-{yyyymmddhhmmss}.rootfs.tar.gz	RootFS Archive
rfs_image-<DISTRO>-<MACHINE>-<DISTRO_CODENAME>-<DISTRO_VERSION>-{yyyymmddhhmmss}.rootfs.ext4	RootFS in ext4 image format
ramdisk-<MACHINE>.cpio.gz	Ramdisk for update mechanism
u-boot-spl.stm32-stm32mp157c-<MACHINE>-basic	U-boot spl i.e. first stage bootloader(fsbl)
u-boot-stm32mp157c-<MACHINE>-basic.img	U-boot image, second stage bootloader(ssbl)
rfs_image-<DISTRO>-<MACHINE>-x86_64-toolchain-<DISTRO_CODENAME>_<DISTRO_VERSION.sh	SDK installer

Table 5.1: *Build output description*

(*) means a symbolic link

5.2 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/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.

5.3 Boot directory

The directory structure of the root file system includes a directory called boot. In addition to the **kernel image, device tree and ramdisk** (restore root file system) a file **uboot_script** is located there.

This file implements some U-Boot command sequences. It can be used for the purpose of updating and booting the RootFS by using NFS.

However, the environment of the U-Boot has to be set up before. This is discussed in detail in the chapter concerning booting.

Alternatively, **u-boot-spl, u-boot image and rootfs** archive can later be stored in this directory to flash NOR and eMMC using NFS in linux environment.

6 Device Startup

Connect the developer kit to the serial port attached to the virtual machine and your network. Open a console in the VM and open a serial terminal by entering

picocom –b 115200 /dev/ttyUSBx

In the case of using an USB serial adapter replace it by the corresponding **ttyUSBx**.

The developer kit is ready to be powered on. The terminal should then show messages for U-Boot-SPL, U-boot and Linux kernel.

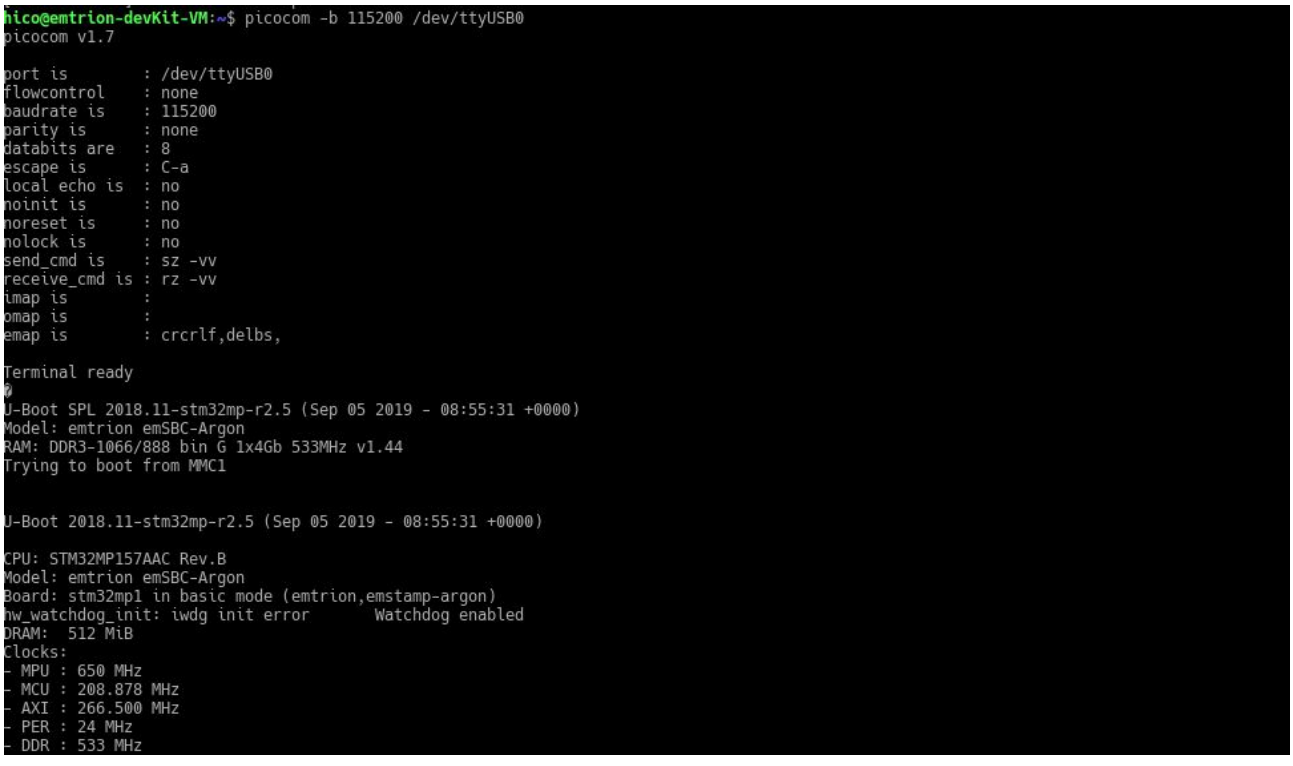

Figure 6.1: *Device Startup example*

After the developer kit is booted the user is prompted for login:

emsbc-argon **login: root**

```
INIT: Entering runlevel: 5
Configuring network interfaces... [    5.680585] Generic PHY stmmac-0:00: attached PHY driver [Generic PHY] (mii_bus:phy_addr=stmmac-0:00, irq=POLL)
[    5.701173] dwmac4: Master AXI performs any burst length
[    5.705045] stm32-dwmac 5800a000.ethernet eth0: No Safety Features support found
[    5.712611] stm32-dwmac 5800a000.ethernet eth0: IEEE 1588-2008 Advanced Timestamp supported
[    5.721883] stm32-dwmac 5800a000.ethernet eth0: registered PTP clock
[    5.727932] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
udhcpc: started, v1.29.3
udhcpc: sending discover
[    6.799123] stm32-dwmac 5800a000.ethernet eth0: Link is Up - 100Mbps/Full - flow control rx/tx
[    6.806296] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
udhcpc: sending discover
udhcpc: sending select for 172.26.4.84
udhcpc: lease of 172.26.4.84 obtained, lease time 3600
/etc/udhcpc.d/50default: Adding DNS 172.26.47.1
/etc/udhcpc.d/50default: Adding DNS 172.26.47.11
done.
Starting system message bus: dbus.
Starting random number generator daemon.
Starting OpenBSD Secure Shell server: sshd
/etc/ssh/sshd_config line 98: Deprecated option UsePrivilegeSeparation
done.
Starting rpcbind daemon...done.
Starting advanced power management daemon: No APM support in kernel
(failed.)
Starting bluetooth: bluetoothd.
Starting ntpd: done
Starting syslogd/klogd: done
 * Starting Avahi mDNS/DNS-SD Daemon: avahi-daemon

Emtrion distro for eglfs backend 1.0.0 emsbc-argon /dev/ttySTM0

emsbc-argon login: root
root@emsbc-argon:~#
```

Figure 6.2: *Device login example*

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 **ip**.

```
root@emsbc-argon:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:1C:1E:08:E4:3C
          inet addr:172.26.4.87  Bcast:172.26.255.255  Mask:255.255.0.0
          inet6 addr: fe80::21c:1eff:fe08:e43c/64 Scope:Link
          inet6 addr: 2003:5a:a012:1:21c:1eff:fe08:e43c/64 Scope:Global
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:318 errors:0 dropped:0 overruns:0 frame:0
          TX packets:46 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:32756 (31.9 KiB)  TX bytes:8130 (7.9 KiB)
          Interrupt:60

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

Figure 6.3: *Device Network example*

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

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 Default boot

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** (U-boot SPL); 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 connection to the workstation via a USB to serial cable and pressing the reset switch should start the device as mentioned in the "Device Startup" chapter.

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 to a search for a valid Rootfs system in this eMMC to start up the linux kernel.

Emtrion has scripts integrated in the image which can be used to **update the Kernel, U-boot or the Rootfs** systems of the kit. Similarly there is a possibility of running a system restore command in U-boot to fetch data from the NFS server and push boot images to the flash memories 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 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 it to initiate an update of the kernel, the RootFS and of U-Boot itself. 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.

7.2.1 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 envi-ronemnt is written in the NOR flash

Table 7.1: *Uboot commands*

7.2.2 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 VM. You can get the IP-address by prompting

ip a or **ifconfig**

in the terminal of your VM.

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 **"bootcmd"** variable can be set in the environment variable 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 **bootcmd are "run emmc_boot", "run sd_boot", "run net_boot" or "run bootcmd_pxe".**

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 7.3**.

7.2.3 Updating, booting and restoring the system

7.2.3.1 Updating of the system (root file system, u-boot and kernel)

As the image archive contains the root file system as well as the kernel, updating of the system affects always both.

Before performing the update process, following steps have to be done.

On VM

Copy the rootfs tar.gz archive from the **<BUILD_DIR>** to the **<NFS_ROOTFS>** directory. Extract the archive. Ensure that the rootfs contains all the required images for the update. The required images are located in the boot directory inside rootfs. The minimum expected files are:

emsbc-argon device tree emPURS_plat script ramdisk-emsbc-argon.cpio.gz uboot_script zImage or the Kernel

On U-Boot

On starting the Uboot, the NFS has to be set up properly. The process is mentioned in details in the Section 7.3. When this is successfully completed, the following commands can be executed.

To update kernel:

STM32MP # run update_kernel

To update rootfs:

Copy an image of the rootfs.tar.gz archive created with the Yocto from the <**BUILD_DIR**> to the <**NFS_ROOTFS**>/boot directory in the VM. The files mentioned as minimum requirements should also be present in the directory. Then run the following command.

STM32MP # run update_rootfs

This loads the rootfs to the eMMC of the development kit. This starts the update process. Please be patient as the process of fetching the root file system image via network and decompressing it to the flash storage can take a few minutes.

To update uboot:

Copy images of the u-boot-spl and u-boot created with the Yocto from the <**BUILD_DIR**> to the <**NFS_ROOTFS**>/boot directory in the VM. The files mentioned as minimum requirements should also be present in the directory. Then run the following command.

STM32MP # run update_uboot

This loads the bootloaders to the NOR flash.

7.2.3.2 Booting

The default boot method is already mentioned in the Section 7.1. However, there are alternate options for the system startup which is explained in this section.

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 (VM). 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. sudo sgdisk -o /dev/sdX
2. 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
3. sudo dd if=<BUILD_DIR>/tmp-glibc/deploy/images/<MACHINE>/u-boot-spl.stm32-stm32mp157c-emsbc-argon-basic of=/dev/sdX1
4. sudo dd if=<BUILD_DIR>/tmp-glibc/deploy/images/<MACHINE>/u-boot-spl.stm32-stm32mp157c-emsbc-argon-basic of=/dev/sdX2
5. sudo dd if=<BUILD_DIR>/tmp-glibc/deploy/images/<MACHINE>/u-boot-stm32mp157c-emsbc-argon-basic.img of=/dev/sdX3

6. sudo dd if=<BUILD_DIR>/tmp-glibc/deploy/images/<MACHINE>/rfs_image-<DISTRO>-<MACHINE>-<DISTRO_CODENAME>-<DISTRO_VERSION>-{yyyymmddhhmmss}.rootfs.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 bootcmd run sd_boot

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

Set up the NFS system as mentioned in Section 7.3 with all the required directories. In order to boot from the NFS sever, the command is:

STM32MP # run net_boot

Booting from TFTP server using PXE

PXELINUX is a Syslinux derivative, for booting from a network server using a network ROM conforming to the Intel PXE (Pre-Execution Environment) specification.

1. The system is based on a tftp server. The following has to be done to set up the server side i.e. the host computer.

```
1. sudo apt-get install xinetd tftpd tftp
2. sudo nano /etc/xinetd.d/tftp
Add the following text to the file:
service tftp
{
protocol        = udp
port            = 69
socket_type     = dgram
wait            = yes
user            = nobody
server          = /usr/sbin/in.tftpd
server_args     = /home/hico/tftpboot
disable         = no
}

3. mkdir ~/tftpboot
4. sudo chmod -R 777 ~/tftpboot
5. sudo chmod -R 777 ~/tftpboot
6. sudo chown -R nobody ~/tftpboot
7. sudo /etc/init.d/xinetd stop
8. sudo /etc/init.d/xinetd start


```

Listing 7.1: *TFTP server setup*

2. In target system, get into u-boot and obtain the MAC address. pri ethaddr The MAC address for example is of the following format AA:BB:CC:DD:EE:FF
3. In the tftpboot folder create a subfolder pxelinux.cfg.
4. Inside this folder create a file with the name as follows: AA-BB-CC-DD-EE-FF-GG i.e. 01- and - separted MAC address. The contents of the file for example dictates what to boot and from where. e.g for a NFS boot it can be as follows:

```
1   #bootfile for emsbc-argon
2   menu title Select the boot mode
3   DEFAULT nfs_boot
4   TIMEOUT 20
5   LABEL nfs_boot
6   KERNEL zImage
7   FDT stm32mp157c-emsbc-argon.dtb
8   APPEND root=/dev/nfs nfsroot={serverip}:<NFS\_ROOTFS> rootwait rw
    earlyprintk console=ttySTM0,115200 ip=dhcp
9
10
```

Listing 7.2: *PXE config file*

5. Copy the contents of the boot directory from the rootfs system to the tftpboot folder as this can be used int he future for various boot methods.
6. Once the server side is completely ready, continue with u-boot in the target system. Set the serverip which is the IP of the host system.
   STM32MP # setenv serverip xxx.xxx.xxx.xxx
7. In u-boot configuration of the STM32MP157c microcontroller the option for PXE boot already exists.
   STM32MP # run bootcmd_pxe
   The system shows for example the following set of messages and should boot from the dictated system.

7.2.3.3 Restoring

There is a script attached to the uboot which can be used to restore a system from a NFS system, provided that the Uboot has loaded successfully. The <NFS_ROOTFS>/boot must contain the files mentioned in Section 7.2.3.1 for updating rootfs. The command is:

U-Boot # run restore_sys

This script is responsible for initiating the emPURS_plat script of Emtrion. It loads the new device tree, kernel and rootfs, after deleting the existing partitions in the flash memories. The boot partitions are formatted and renewed with the images inside the NFS server. The new rootfs is installed and realtime clock is reset from NTP servers. Finally the system is rebooted.

7.3 NFS Setup

Update of the u-boot, kernel 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 the VM 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
   /home/hico/nfs/emsbc-argon/rootfs *(rw,sync,no_subtree_check,crossmnt,no_root_squash)
3. Unpack rfs_image-emsbc-argon.rootfs.tar.gz in the folder mentioned above. Ensure that the boot folder contains the required images.
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 for the VM)
   U-Boot # setenv nfsroot /home/hico/nfs/emsbc-argon/rootfs
   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 # saveenv (to save the added variables)
   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 already contains some default applications to test the M4 functions. To use the examples, the user has to follow either one set of the mentioned instructions.

Start from Linux

Method 1

1. Add m4-m4projects-stm32mp1-userfs package to the image
2. cd /usr/local/Cube-M4-examples/STM32MP157C-DK2/Applications/OpenAMP/OpenAMP_TTY_echo/
3. ./fw_cortex_m4.sh start Linux console messages:

```
[  194.055928] remoteproc remoteproc0: powering up m4
[  194.060352] remoteproc remoteproc0: Booting fw image OpenAMP_TTY_echo.elf, size 217044
[  194.067543]  m4@0#vdev0buffer: assigned reserved memory node vdev0buffer@10044000
[  194.076477] virtio_rpmsg_bus virtio0: rpmsg host is online
[  194.076666] virtio_rpmsg_bus virtio0: creating channel rpmsg-tty-channel addr 0x0
[  194.081412]  m4@0#vdev0buffer: registered virtio0 (type 7)
[  194.091927] rpmsg_tty virtio0.rpmsg-tty-channel.-1.0: new channel: 0x400 -> 0x0 : ttyRPMSG0
[  194.094418] remoteproc remoteproc0: remote processor m4 is now up
[  194.103448] virtio_rpmsg_bus virtio0: creating channel rpmsg-tty-channel addr 0x1
[  194.120031] rpmsg_tty virtio0.rpmsg-tty-channel.-1.1: new channel: 0x401 -> 0x1 : ttyRPMSG1

```

Listing 8.1: *Linux console messages*

4. cat /sys/kernel/debug/remoteproc/remoteproco/traceo

5. Steps to test the echo messaging are as follows:

```
root@emsbc-argon:~# stty -onlcr -echo -F /dev/ttyRPMSG0
root@emsbc-argon:~# cat /dev/ttyRPMSG0 &
root@emsbc-argon:~# stty -onlcr -echo -F /dev/ttyRPMSG1
root@emsbc-argon:~# cat /dev/ttyRPMSG1 &
root@emsbc-argon:~# echo "Hello Virtual UART0" >/dev/ttyRPMSG0
root@emsbc-argon:~# echo "Hello Virtual UART1" >/dev/ttyRPMSG1

```

Listing 8.2: *Testing the OpenAmp_tty_echo application*

Method 2

1. echo -n <firmware_path> > /sys/module/firmware_class/parameters/path
2. echo -n OpenAMP_TTY_echo.elf > /sys/class/remoteproc/remoteproco/firmware
3. echo start >/sys/class/remoteproc/remoteproco/state Linux console messages:

```
[ 1723.078223] remoteproc remoteproc0: powering up m4
[ 1723.086088] remoteproc remoteproc0: Booting fw image OpenAMP_TTY_
echo.elf, size 217044
[ 1723.092984]  m4@0#vdev0buffer: assigned reserved memory node vdev0
buffer@10044000
[ 1723.100419] virtio_rpmsg_bus virtio0: rpmsg host is online
[ 1723.101884] virtio_rpmsg_bus virtio0: creating channel rpmsg-tty-
channel addr 0x0
[ 1723.105637]  m4@0#vdev0buffer: registered virtio0 (type 7)
[ 1723.118556] remoteproc remoteproc0: remote processor m4 is now up
[ 1723.123812] rpmsg_tty virtio0.rpmsg-tty-channel.-1.0: new channel:
0x400 -> 0x0 : ttyRPMSG0
[ 1723.135165] virtio_rpmsg_bus virtio0: creating channel rpmsg-tty-
channel addr 0x1
[ 1723.146040] rpmsg_tty virtio0.rpmsg-tty-channel.-1.1: new channel:
0x401 -> 0x1 : ttyRPMSG1

```

Listing 8.3: *Linux console messages*

4. echo stop >/sys/class/remoteproc/remoteproco/state
   Linux Console messages:

```
[ 1797.699072] rpmsg_tty virtio0.rpmsg-tty-channel.-1.0: rpmsg tty
device 0 is removed
[ 1797.707004] rpmsg_tty virtio0.rpmsg-tty-channel.-1.1: rpmsg tty
device 1 is removed
[ 1798.217738] remoteproc remoteproc0: warning: remote FW shutdown
without ack
[ 1798.223258] remoteproc remoteproc0: stopped remote processor m4


```

Listing 8.4: *Linux console messages*

Start from Uboot

In linux console do the following to copy the firmware:

1. cd /usr/local/Cube-M4-examples/STM32MP157C-DK2/Applications/OpenAMP/OpenAMP_TTY_echo/lib/firmware/
2. cp OpenAMP_TTY_echo.elf /boot
3. sync

In Uboot:

1. ext4load mmc 0:4 $kernel_addr_r /boot/OpenAMP_TTY_echo.elf
2. rproc init
3. rproc load 0 $kernel_addr_r $filesize
4. rproc load_rsc 0 $kernel_addr_r $filesize
5. rproc start 0
6. run bootcmd

   Boot messages:

```
[    1.731359] mmcblk0: mmc0:0007 SD8GB 7.21 GiB
[    1.734692] usbhid: USB HID core driver
[    1.744181] stm32-ipcc 4c001000.mailbox: ipcc rev:1.0 enabled, 6 chans, proc 0
[    1.759920]  mmcblk0: p1 p2 p3 p4
[    1.761368] stm32-rproc m4@0: wdg irq registered
[    1.766593] remoteproc remoteproc0: m4 is available
[    1.771351] remoteproc remoteproc0: powering up m4
[    1.776023] remoteproc remoteproc0: Synchronizing with early booted co-processor
[    1.783663]  m4@0#vdev0buffer: assigned reserved memory node vdev0buffer@10044000
[    1.791184] virtio_rpmsg_bus virtio0: rpmsg host is online
[    1.791326] virtio_rpmsg_bus virtio0: creating channel rpmsg-tty-channel addr 0x0
[    1.796384]  m4@0#vdev0buffer: registered virtio0 (type 7)
[    1.804018] virtio_rpmsg_bus virtio0: creating channel rpmsg-tty-channel addr 0x1
[    1.809309] remoteproc remoteproc0: remote processor m4 is now up
[    1.823019] rpmsg_tty virtio0.rpmsg-tty-channel.-1.0: new channel: 0x400 -> 0x0 : ttyRPMSG0
[    1.823300] rpmsg_tty virtio0.rpmsg-tty-channel.-1.1: new channel: 0x401 -> 0x1 : ttyRPMSG1
```

Listing 8.5: *Linux console messages*

9 SDK

In order to develop applications outside of the Yocto build system the host development system needs to be set up. For this purpose the YP offers several installation methods. One of the methods for creating an SDK is by using the build directory. Use the following command for performing.

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/

Installing the SDK

While running the SDK installer, the user is asked for the installation directory. The default location is **/opt/emtrion/emsbc-argon/thud_v1.0.0**. The default can be left as it is and confirmed. From inside the location **<BUILD_DIR>** start the installer as follows.

./tmp-glibc/deploy/sdk/rfs_image-emt-eglfs-emsbc-argon-x86_64-toolchain-thud_v1.0.0.sh

Default installation location: **/opt/emtrion/emsbc-argon/thud_v1.0.0** Requires a confirmation for installation from user. On confirmation, the script will start extracting the SDK and then show the message "Setting it up....Done", indicating that the script has finished successfully.

Setting up the SDK environment

Before one 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.

9.1 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 5.11.2** with GCC in a 64 bit machine:

1. Open the "Tools", "Options..." menu and select the **Build & Run** section
2. Use **/opt/emtrion/emsbc-argon/thud_v1.0.0/sysroots/x86_64-emt_eglfs-linux/usr/bin/qmake** as qmake location in **Qt versions** tab. The version for development is **5.11.2**, 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 **/opt/emtrion/emsbc-argon/thud_v1.0.0/sysroots/x86_64-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 **/opt/emtrion/emsbc-argon/thud_v1.0.0/sysroots/x86_64-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 **/opt/emtrion/emsbc-argon/thud_v1.0.0/sysroots/x86_64-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 **/opt/emtrion/emsbc-argon/thud_v1.0.0/sysroots/x86_64-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 **/opt/emtrion/emsbc-argon/thud_v1.0.0/sysroots/x86_64-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.