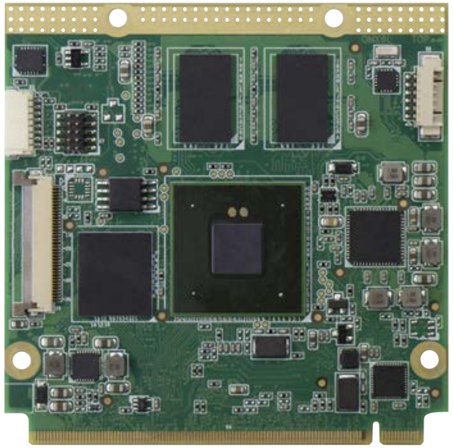


Qseven® conga-QMX6/UMX6

NXP® i.MX6 ARM® Cortex A9 processor with Ultra Low Power Consumption

Software User's Guide

Revision 1.0

Revision History

Revision	Date (yyyy.mm.dd)	Author	Changes
0.1	2013.08.12	AEM	• Preliminary release
0.2	2014.02.28	AEM	• Added Yocto and Android sections. • Added section 5.2 "IOMUX Configuration". • Restructured and updated the whole document.
1.0	2017.05.15	BEU	• Official release

Preface

This user's guide provides information on how to set up and install the congatec Linux BSP on the conga-QMX6/UMX6. It is one of seven documents that should be referred to when designing an i.MX6 based Qseven® application for the conga-QMX6/UMX6. The other reference documents that should be used include the following:

conga-QMX6/UMX6 Hardware User's Guide
Qseven® Design Guide
Qseven® Specification
i.MX6 Applications Processor Reference Manual (available at www.nxp.com)
congatec AN33 Installation and Update of NXP MFGTool and congatec Bootloader Profiles
congatec CTN-20120906-001

The links to these documents can be found on the congatec AG website at www.congatec.com. For the list of sources of information, see section 9 "Sources of Information".

Disclaimer

The information contained within this user's guide, including but not limited to any product specification, is subject to change without notice.

congatec AG provides no warranty with regard to this user's guide or any other information contained herein and hereby expressly disclaims any implied warranties of merchantability or fitness for any particular purpose with regard to any of the foregoing. congatec AG assumes no liability for any damages incurred directly or indirectly from any technical or typographical errors or omissions contained herein or for discrepancies between the product and the user's guide. In no event shall congatec AG be liable for any incidental, consequential, special, or exemplary damages, whether based on tort, contract or otherwise, arising out of or in connection with this user's guide or any other information contained herein or the use thereof.

Intended Audience

This user's guide is intended for technically qualified personnel. It is not intended for general audiences.

Lead-Free Designs (RoHS)

All congatec AG designs are created from lead-free components and are completely RoHS compliant.

Electrostatic Sensitive Device

All congatec AG products are electrostatic sensitive devices and are packaged accordingly. Do not open or handle a congatec AG product except at an electrostatic-free workstation. Additionally, do not ship or store congatec AG products near strong electrostatic, electromagnetic, magnetic, or radioactive fields unless the device is contained within its original manufacturer's packaging. Be aware that failure to comply with these guidelines will void the congatec AG Limited Warranty.

Symbols

The following symbols are used in this user's guide:

Warning

Warnings indicate conditions that, if not observed, can cause personal injury.

Caution

Cautions warn the user about how to prevent damage to hardware or loss of data.

Note

Notes call attention to important information that should be observed.

Copyright Notice

Copyright © 2013, congatec AG. All rights reserved. All text, pictures and graphics are protected by copyrights. No copying is permitted without written permission from congatec AG.

congatec AG has made every attempt to ensure that the information in this document is accurate yet the information contained within is supplied "as-is".

Trademarks

Product names, logos, brands, and other trademarks featured or referred to within this user's guide, or the congatec website, are the property of their respective trademark holders. These trademark holders are not affiliated with congatec AG, our products, or our website.

Warranty

congatec AG makes no representation, warranty or guaranty, express or implied regarding the products except its standard form of limited warranty ("Limited Warranty") per the terms and conditions of the congatec entity, which the product is delivered from. These terms and conditions can be downloaded from www.congatec.com. congatec AG may in its sole discretion modify its Limited Warranty at any time and from time to time.

The products may include software. Use of the software is subject to the terms and conditions set out in the respective owner's license agreements, which are available at www.congatec.com and/or upon request.

Beginning on the date of shipment to its direct customer and continuing for the published warranty period, congatec AG represents that the products are new and warrants that each product failing to function properly under normal use, due to a defect in materials or workmanship or due to non conformance to the agreed upon specifications, will be repaired or exchanged, at congatec's option and expense.

Customer will obtain a Return Material Authorization ("RMA") number from congatec AG prior to returning the non conforming product freight prepaid. congatec AG will pay for transporting the repaired or exchanged product to the customer.

Repaired, replaced or exchanged product will be warranted for the repair warranty period in effect as of the date the repaired, exchanged or replaced product is shipped by congatec, or the remainder of the original warranty, whichever is longer. This Limited Warranty extends to congatec's direct customer only and is not assignable or transferable.

Except as set forth in writing in the Limited Warranty, congatec makes no performance representations, warranties, or guarantees, either express or implied, oral or written, with respect to the products, including without limitation any implied warranty (a) of merchantability, (b) of fitness for a particular purpose, or (c) arising from course of performance, course of dealing, or usage of trade.

congatec AG shall in no event be liable to the end user for collateral or consequential damages of any kind. congatec shall not otherwise be liable for loss, damage or expense directly or indirectly arising from the use of the product or from any other cause. The sole and exclusive remedy against congatec, whether a claim sound in contract, warranty, tort or any other legal theory, shall be repair or replacement of the product only.

Certification

congatec AG is certified to DIN EN ISO 9001:2008 standard.

Technical Support

congatec AG technicians and engineers are committed to providing the best possible technical support for our customers so that our products can be easily used and implemented. We request that you first visit our website at www.congatec.com for the latest documentation, utilities and drivers, which have been made available to assist you. If you still require assistance after visiting our website then contact our technical support department by email at support@congatec.com

Terminology

Term	Description
PCI Express (PCIe)	Peripheral Component Interface Express – next-generation high speed Serialized I/O bus
PCI Express Lane	One PCI Express Lane is a set of 4 signals that contains two differential lines for transmitting and two differential lines for Receiving. Clocking information is embedded into the data stream.
LTIB	Linux Target Image Builder
PCI Express Mini Card	PCI Express Mini Card add-in card is a small size unique form factor optimized for mobile computing platforms.
eMMC	Embedded Multi Media Card is a non-volatile memory system, which frees the processor from low level flash memory management.
SDIO card	SDIO (Secure Digital Input Output) is a non-volatile memory card format developed for use in portable devices.
USB	Universal Serial Bus
SATA	Serial AT Attachment: serial-interface standard for hard disks
HDA	High Definition Audio
HDMI	High Definition Multimedia Interface. HDMI supports standard, enhanced, or high-definition video, plus multi-channel digital audio on a single cable.
BSP	Board Support Package
OTP	One Time Programmable
USB OTG	USB On-The-Go. A USB specification that allows USB devices to act as host.
SPI Bus	Serial Peripheral Interface is a synchronous serial data link standard named by Motorola that operates in full duplex mode.
IOMUX	Input Output Multiplexer
GbE	Gigabit Ethernet
LVDS	Low-Voltage Differential Signaling

Contents

1 Introduction

Qseven® Concept

The Qseven® concept is an off-the-shelf, multi vendor, Single-Board-Computer that integrates all the core components of a common PC and is mounted onto an application specific carrier board. Qseven® modules have a standardized form factor of 70mm x 70mm and a specified pinout based on the high speed MXM system connector. The pinout remains the same regardless of the vendor. The Qseven® module provides the functional requirements for an embedded application. These functions include, but are not limited to graphics, sound, mass storage, network interface and multiple USB ports.

A single ruggedized MXM connector provides the carrier board interface to carry all the I/O signals to and from the Qseven® module. This MXM connector is a high speed signal interface connector that is commonly used for high speed PCI Express graphics cards in notebooks.

Carrier board designers can utilize as little or as many of the I/O interfaces as deemed necessary. The carrier board can therefore provide all the interface connectors required to attach the system to the application specific peripherals. This versatility allows the designer to create a dense and optimized package, which results in a more reliable product while simplifying system integration.

The Qseven® evaluation carrier board provides carrier board designers with a reference design platform and the opportunity to test all the Qseven® I/O interfaces available and then choose what are suitable for their application. Qseven® applications are scalable, which means once a carrier board has been created there is the ability to diversify the product range through the use of different performance class Qseven® modules. Simply unplug one module and replace it with another; no need to redesign the carrier board.

This document describes the features available at congatec module based on NXP's i.MX6 ARM Cortex A9 processor.

Board Support Package

congatec AG provides developers with various BSPs as startup framework for building applications that run on conga-QMX6/conga UMX6. The BSPs offered are Linux, android and Windows Embedded Compact. The Linux and android BSPs are provided directly by congatec while the Windows Embedded Compact is provided via Witekio. The Windows Embedded Compact BSPs and documentation can be obtained from Witekio at https://witekio.com/cpu/conga-qmx6.

Software Distribution

There are two channels for distributing the software and documentation for the congatec i.MX6 based products:

The product specific download page

The product specific download pages for i.MX6 based products are:

- conga-QMX6: http://www.congatec.com/de/produkte/qseven/conga-qmx6.html
- conga-UMX6: http://www.congatec.com/de/produkte/qseven/conga-umx6.html

The product specific download pages provide:

- binary tools
- readme files
- operating system specific sample images
- product manuals and datasheets

The congatec git-server

The congatec git-server provides:

- source code distribution of bootloader, kernel and BSPs (board support packages)

The public repositories on the congatec git-server can be reached via https://git.congatec.com/public

In order to fetch from the public git server repositories, use the "git clone" command on your development workstation.

For example, to fetch the "qmx6_uboot" repository (the bootloader repository for conga-QMX6 and conga-UMX6), enter:

```
~$ git clone https://git.congatec.com:arm/qmx6_uboot.git
```

2 Setting Up the Host System

2.1 Overview

Software development is usually not performed at the target system. Most development tasks are handled at a dedicated development system, called host. Depending on the task, either a Windows or Linux based host will be required. In most cases, the first task is to set up both host systems.

2.2 Requirements

Requirements for the set up of the standalone cross-development environment:

- x86 host system (64-bit)
- recommended free disk space: 25 GB
- recommended memory size: 8 GB
- Ubuntu 14.04 (64-bit)
- Yocto toolchain (http://downloads.yoctoproject.org/releases/yocto/yocto-2.1/toolchain/x86_64/poky-glibc-x86_64-core-image-sato-armv7a-neon-toolchain-2.1.sh)

2.3 Setting Up the Standalone Cross-Development Environment

A suitable cross-development toolchain is required to develop software for an ARM target system at a x86 host (development) system. In order to develop Linux software on the basis of the provided Yocto based Linux BSP, use the pre-built toolchain installer provided by the Yocto Project.

1. Perform a standard Ubuntu 14.04 (64-bit) installation.

2. Install additional packages:

```
ssh
git
gitk
lzop
libncursesw5-dev
```

3. Install the toolchain:

```
$ chmod a+x poky-glibc-x86_64-core-image-sato-armv7a-neon-toolchain-2.1.sh
$ ./poky-glibc-x86_64-core-image-sato-armv7a-neon-toolchain-2.1.sh
```

4. Create the "sourceme" file (simplifies the setup of the environment):

```
$ cat > sourceme << EOF
> export ARCH=arm
> export SUBARCH=arm
> export CROSS_COMPILE=arm-poky-linux-gnueabi-
> ./opt/poky/2.1/environment-setup-armv7a-neon-poky-linux-gnueabi
> EOF
```

2.4 Serial Port Terminal (Serial Console)

The initial start up code (bootloader) limits the hardware initialization to a minimum. Video interfaces and other interfaces like the keyboard are not fully initialized. The operating system initializes the hardware later on. Therefore, the bootloader program and the operating system kernel redirect their output to a specified serial port and a serial connection is required to read it. The serial connection is also required to determine or influence the bootloader's behavior via command prompt.

2.4.1 Setting up the Hardware

UART2 is the specified serial port for conga-QMX6/UMX6. On the conga-QMX6, this port is available via the X6 on-module connector and edge connectors (multiplexed). On the conga-UMX6, this port is only available via the Qseven edge connector (multiplexed with JTAG signals).

Note

The UART signal level at the Qseven edge connector is 3.3V. The signal level of the X6 on-module connector (conga-QMX6) conforms to the RS232 interface specification

Connect the serial port UART2 with the host system as shown in one of the following tables:

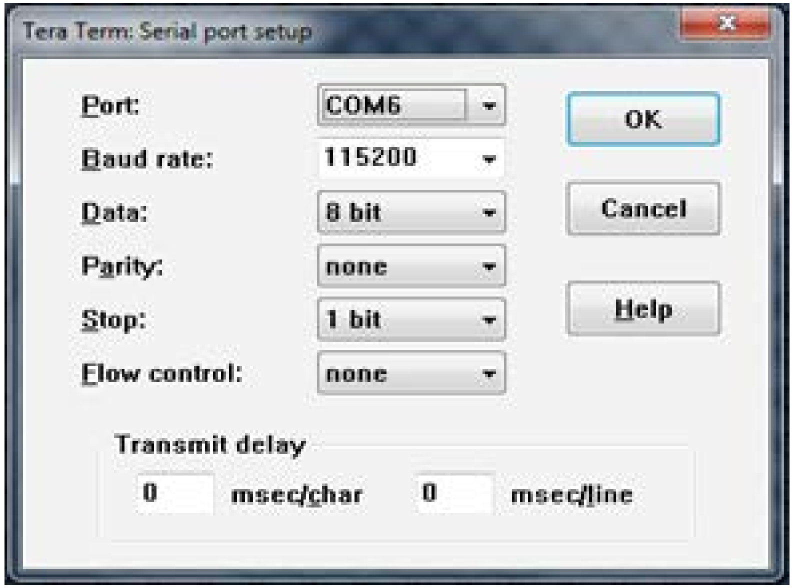
X6 On-Module Connector (conga-QMX6)	Host System (DTE), 9 Pol. DSUB
Pin 3 (GND)	Pin 5 (GND)
Pin 4 (UART2, TX)	Pin 2 (RxD)
Pin 5 (UART2, RX)	Pin 3 (TxD)

Qseven Edge Connector	Host System (DTE), 9 Pol. DSUB
GND	Pin 5 (GND)
Pin 209 (UART2, TX)	Pin 2 (RxD)
Pin 208 (UART2, RX)	Pin 3 (TxD)

2.4.2 Setting Up the Software

A serial terminal application is required to connect to the target system's serial console. congatec recommends Tera Term (Windows) or minicom (Linux) with the following serial port configuration: 115200 8N1; flow control: none.

1. Set up the serial port as shown in the screenshot below:



Port: COM6

Baud rate: 115200

Data: 8 bit

Parity: none

Stop: 1 bit

Flow control: none

Transmit delay

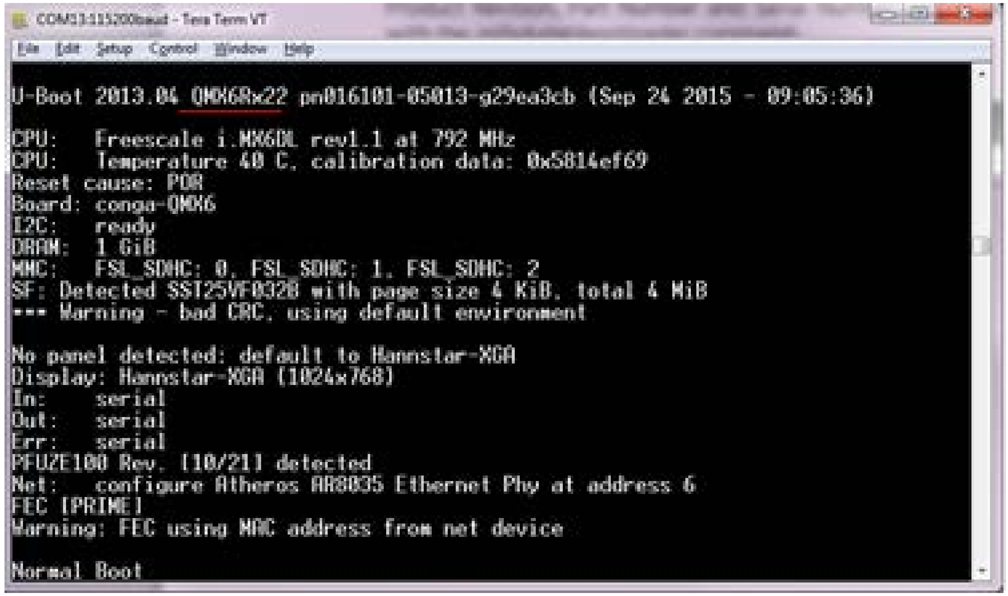
0 msec/char 0 msec/line

Note

Ensure to select the right port.

2. Turn on the module.

3. Tera Term will show the bootloader messages in its main window as shown in the screenshot below:


COM13:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
U-Boot 2013.04 QMX6Rx22 pn016101-05013-g29ea3cb (Sep 24 2015 - 09:05:36)

CPU:   Freescale i.MX6DL rev1.1 at 792 MHz
CPU:   Temperature 40 C, calibration data: 0x5814ef69
Reset cause: POR
Board: conga-QMX6
I2C:   ready
DRAM:  1 GiB
MMC:   FSL_SDHC: 0, FSL_SDHC: 1, FSL_SDHC: 2
SF: Detected SST25VF032B with page size 4 KiB, total 4 MiB
*** Warning - bad CRC, using default environment

No panel detected: default to Hannstar-XGA
Display: Hannstar-XGA (1024x768)
In:    serial
Out:   serial
Err:   serial
PFUZE100 Rev. [10/21] detected
Net:   configure Atheros AR8035 Ethernet Phy at address 6
FEC [PRIME]
Warning: FEC using MAC address from net device

Normal Boot
```

2.5 Updating the Bootloader (NXP MFGTool / cgtMFGui)

The NXP manufacturing tool (MFGTool) enables communication with a conga-QMX6/UMX6, even without a working bootloader in the SPI-flash. For this purpose, connect the host and target system via USB and set the i.MX6 powered congatec design into Serial Downloader Mode (SDM).

The communication is done by special protocols, called Serial Download Protocol (SDP) and Update Transfer Protocol (UTP). In SDM, the CPU module acts as a USB client and fetches the bootloader from a host computer via USB.

The manufacturing software environment consists of two main components:

- MFGTool
- MFG Profiles

Note

congatec recommends a serial terminal application, as described in section 2.4 "Serial Port Terminal (Serial Console)", to observe the update progress.

2.5.1 Use Cases

- Burning bootloader program to SPI-flash
- Updating SPI-resident bootloader program
- Module recovery:
  - Recovery from corrupt bootloader
  - Recovery from wrong bootloader image burned

2.5.2 Download, Installation and Update Procedure (NXP MFGTool and cgtMFGui)

Releases used to be distributed via the congatec website. MFGTool and profiles were split up into several archive files. Now, latest MFGTool releases are distributed via the congatec git server (https://git.congatec.com/imx6-mfg-tool/mfgtool). The MFG Profiles are linked to the MFGTool repository by means of git submodules.

In order to obtain the MFGTool, follow the steps described in *AN33 Installation and Update of NXP MFGTool and congatec Bootloader Profiles,* available on the congatec website www.congatec.com.

2.5.3 Configuring the NXP MFGTool

In order to burn or update a module's bootloader with the MFGTool, configurations are required in advance. You can edit the configuration file with a suitable text editor or use a GUI application, called cgtMFGui.

2.5.3.1 GUI Based Configuration (cgtMFGui)

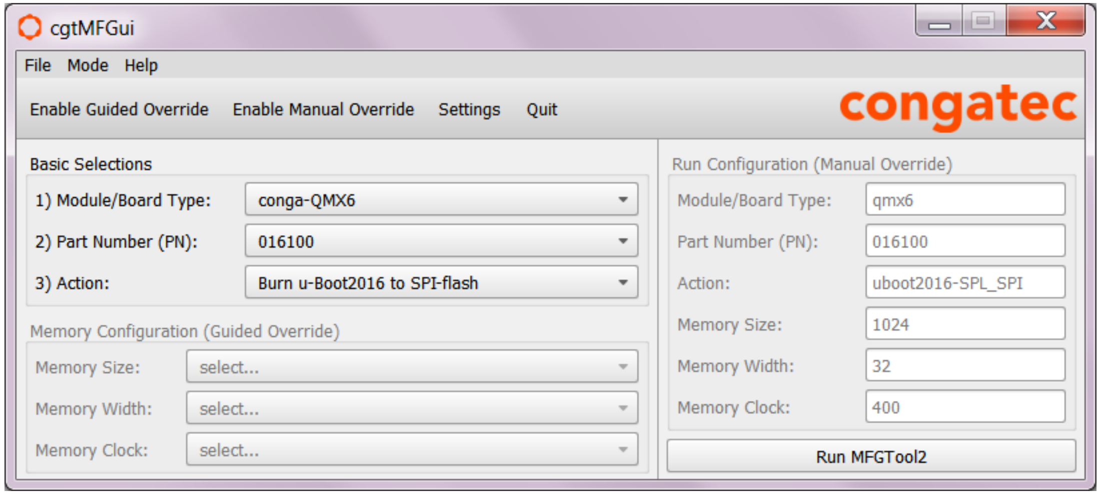
cgtMFGui is a GPLv3 licensed, open source application. It makes using the NXP's MFGTool2 easier by eliminating the need of manual editing of configuration files for common use cases. cgtMFGui's main concept is configuring and executing NXP's MFGTool2 via appropriate command-line parameters - the cgtMFGui selections are mapped to suitable MFGTool2 command-line arguments.

If one follows the instructions given in section 2.5.2 "Download, Installation and Update Procedure (NXP MFGTool and cgtMFGui)", cgtMFGui is automatically downloaded. There is a cgtMFGui subfolder within the MFGTool distribution containing the cgtMFGui executable.

Note

congatec's cgtMFGui is a GPLv3 licensed, open source application. The sources are available at the congatec public git server https://git.congatec.com/imx6-mfg-tool/cgtMFGui

The following screenshot shows cgtMFGui's main window:



Basic Selections

1) Module/Board Type: conga-QMX6

2) Part Number (PN): 016100

3) Action: Burn u-Boot2016 to SPI-flash

Memory Configuration (Guided Override)

Memory Size: select...

Memory Width: select...

Memory Clock: select...

Run Configuration (Manual Override)

Module/Board Type: qmx6

Part Number (PN): 016100

Action: uboot2016-SPL_SPI

Memory Size: 1024

Memory Width: 32

Memory Clock: 400

Make three selections:

- Module board type
- Part number of the module/board to be updated (the part number is printed at the module's/board's barcode label)
- Action to perform, e.g. burning bootloader program to target system

Proceed as described in section 2.5.4 "Usage".

2.5.3.2 Manual Configuration (cfg.ini)

Specify the following information to set up the MFGTool:

- Module type: at the moment, there are two distinct module types (qmx6 and umx6)
- Software selection/action to perform: one has to select the bootloader/firmware image to burn to the target module
- Part Number (PN) of the product:
  - Each product type is identified by a unique number, called PN.
  - MFGTool uses the PN to identify the type of the module in use. This information is optional in some cases.

- Memory configuration:
  - The specification of a suitable memory configuration is mandatory for the entire update process.
  - Unless a proper memory configuration is set up, the update process will not work as expected or will even fail.

The cfg.ini consists of the following sections:

- [profiles]
  - Specifying the module type
- [platform]
  - Not used yet
- [LIST]
  - Specifying software/action to perform (e.g. updating u-boot 2016).
- [variable]
  - Specifying the PN.
  - Specifying memory configuration.

All configuration options are predefined but commented out. Excluded configuration options start with a single semicolon ";". Uncomment a configuration option to include it. The following example sets the module type to conga-UMX6:

```
;chip = qmx6
chip = umx6
```

More details about configuration options are embedded in the cfg.ini configuration file.

2.5.4 Usage

The steps below describe how to update a module with the help of the NXP MFGTool:

1. Configurate the NXP MFGTool as described in section 2.5.3 "Configuring the NXP MFGTool"
2. Establish a USB host/client connection between the Windows based host system and the USB OTG port of the carrier board / i.MX6 powered design. If the conga-QEVAL/Qseven 2.0 evaluation carrier board is used, a USB 2.0 A to USB micro-B cable is required. Connect the USB A connector to the host system and the USB micro-B connector to the carrier board (connector X53).

3. Set the USB OTG port of the module to client mode. The procedure depends on the carrier board in use. In case of the conga-QEVAL / Qseven 2.0 evaluation carrier board, please ensure the following jumper settings:

   – SW9: 1, 2 OFF; 3, 4 ON (sets X53 to client mode)

   – X37: 1-2 (sets USB_ID to floating)

   – SW1: 1..4 OFF

   – SW2: 1..4 OFF

   – SW3: 1 ON; 2..4 OFF

   If other jumpers/switches settings were changed, it may be required to go back to the default settings as described in *conga Tech Note CTN-20120906-001 Rev 1.12* or later.

4. Set the signal BOOT_ALT#_BPLANE to low. This forces the module to Serial Downloader Mode (alternative boot mode). The procedure depends on the carrier board in use. In case of the conga-QEVAL / Qseven 2.0 evaluation carrier board, the alternative boot mode is controlled by switch M13:

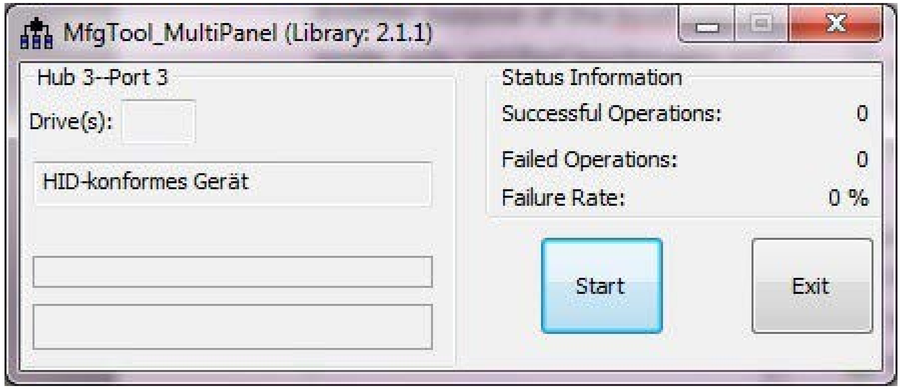
   – standard boot mode: M13 -> 1 OFF, 2 OFF

   – alternative boot mode: M13 -> 1 ON, 2 OFF

5. Run the serial terminal application, e.g. Tera Term, as described in section 2.4 "Serial Port Terminal (Serial Console)".

6. Run the MFGTool. If cgtMFGui is used, just click the "Run MFGTool2" button.

7. Power on the i.MX6 based module. If all steps and settings mentioned above are correct, the MFGTool reports a HID compliant device as shown below:



MfgTool_MultiPanel (Library: 2.1.1)

Hub 3--Port 3

Drive(s):

HID-konformes Gerät

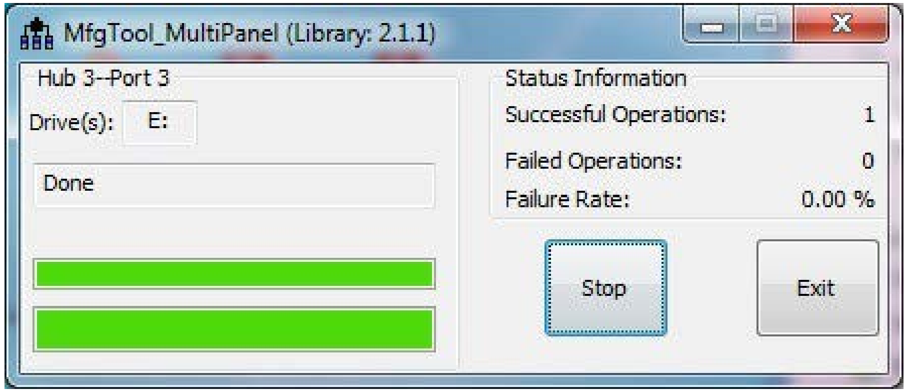
Status Information

Successful Operations: 0

Failed Operations: 0

Failure Rate: 0 %

8. Press 'Start' in order to launch the bootloader update process. The manufacturing bootloader, kernel and ramdisk are transferred to the target system. If everything works correctly, the progress of the update process can be observed via the serial console. After some time, the success of the update procedure is reported as shown below:



Hub 3--Port 3

Drive(s): E:

Done

Status Information

Successful Operations: 1

Failed Operations: 0

Failure Rate: 0.00 %

2.6 BSP-Setup

i.MX6 based congatec designs support the following operating systems:

- Linux (Yocto)
- Android
- Windows Embedded Compact
- ...

Each BSP requires an initial setup - please refer to instructions given by the appropriate section.

3 Yocto Based Linux BSP

A x86 based Linux system with installed cross compile toolchain is required to develop Yocto based software for i.MX6-based designs. It is possible to use a virtual machine that runs Linux but a dedicated system with Ubuntu is preferrable. Furthermore, the host should have a serial port to access the debug console and an SD card reader.

This guide shows the procedure for Yocto Project 2.1 (Krogoth) as an example. Use the latest Yocto release whenever possible. The available Yocto releases are available on the following website: https://git.congatec.com/yocto/cgt-bsp-manifest/branches

For instructions on how to build an image for older versions, refer to the readme files provided on the relevant project page of the congatec website www.congatec.com.

3.1 Setting Up the BSP

Follow the steps below to set up the development platform:

1. Install a fresh Ubuntu 14.04 64-bit operating system on your development host.
2. Update the host system's package repository list

   ```
   $ sudo apt-get update
   ```

3. Install Yocto dependencies – a clean Ubuntu 14.04 system requires the installation of the following dependencies:

   ```
   $ sudo apt-get install gawk wget git-core diffstat unzip texinfo build-essential chrpath libsdl1.2-dev xterm curl
   ```

4. Obtain the repo utility:

   ```
   $ mkdir ~/bin
   $ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
   $ chmod a+x ~/bin/repo
   ```

5. Complete your local git client configuration. For example:

   ```
   $ git config --global user.email "your@email"
   $ git config --global user.name "your name"
   ```

6. Create a working directory and obtain the congatec Yocto-based BSP from the congatec public git server:

   ```
   $ mkdir ~/yocto
   $ cd ~/yocto
   $ ~/bin/repo init -u https://git.congatec.com/yocto/cgt-bsp-manifest.git -b krogoth
   $ ~/bin/repo sync
   ```

3.2 Building a Root Filesystem Image

Follow the steps below to build the Yocto root filesystem (rootfs) for conga-QMX6:

1. Configure the build environment. This example is for conga-QMX6:

   ```
   $ cd ~/yocto
   $ MACHINE='cgtimx6' source setup-environment build
   ```

   Accepting the EULA is required before proceeding to the next step

2. Build the root filesystem image. The Yocto-Project provides various example recipes to create a root filesystem image. For example:

   a. fsl-image-machine-test: A console-only image that includes gstreamer packages, Freescale's multimedia packages (VPU and GPU) when available, test and benchmark applications.

   b. fsl-image-multimedia-full: A console-only image that includes gstreamer packages and Freescale's multimedia packages (VPU and GPU) when available for the specific machine extended with additional gstreamer plugins.

   c. core-image-sato: An image with Sato, a mobile environment and visual style for mobile devices. The image supports X11 with a Sato theme, Pimlico applications, and contains terminal, editor, and file manager.

   To build the fsl-image-machine-test image, use the following bitbake command:

   ```
   $ bitbake fsl-image-machine-test
   ```

This build may take hours. When the build is finished, the image will be located in ~/yocto/build/tmp/deploy/images/cgtimx6/

3.3 Deploying the Image

This section explains how to transfer the kernel and the root filesystem (rootfs) to the target system. You can load the root filesystem via network or locally from an SD card, eMMC, SATA or USB – depending on the used module and bootloader version.

3.3.1 Network Boot

Follow the steps below to configure the TFTP and NFS services on the host system to boot from the network and set up the target system for network boot:

1. Configure TFTP service on the host system. You need this service to transfer the kernel image, the device tree blob and the initial ramdisk (if used).

a. Install the tftpd-server and its dependencies:

```
$ sudo apt-get install xinetd tftpd tftp –y
```

b. Create a configuration file for the TFTP service:

```
$ sudo nano /etc/xinetd.d/tftp
```

c. Add the content below to the configuration file and save it:

```
service tftp
{
           protocol = udp
           port = 69
           socket_type = dgram
           wait = yes
           user = nobody
           server = /usr/sbin/in.tftpd
           server_args = var/lib/tftpboot -s
           disable = no
}
```

d. Create the directory (/var/lib/tftpboot) and change its ownership. This directory is the root of the tftpd server:

```
$ sudo mkdir /var/lib/tftpboot
$ sudo chown -R nobody:nogroup /var/lib/tftpboot
$ sudo chmod -R 777 /var/lib/tftpboot
```

e. Restart xinetd in order to start the tftp service:

```
$ sudo service xinetd stop
$ sudo service xinetd start
```

To boot from network, the kernel (uImage), the devicetree blob files (imx6q-qmx6.dtb and imx6dl-qmx6.dtb) and if used, the initial ramdisk, must be copied to the /var/lib/tftpboot directory.

Note

This directory is also used for exporting the root filesystem via NFS as described in the next section.

2. Configure NFS service on the host system. You need this service to access the root filesystem at the target system via network:

a. Install the required packages and dependencies on the development host:

```
$ sudo apt-get install nfs-kernel-server portmap
```

b. Create the directory (/var/lib/tftpboot/rootfs). This directory is the location of the exported root filesystem:

```
$ sudo mkdir /var/lib/tftpboot/rootfs
$ sudo chown -R nobody:nogroup /var/lib/tftpboot/rootfs
$ sudo chmod -R 777 /var/lib/tftpboot/rootfs
```

c. Open the file /etc/exports, add the lines below to the file and save it:

```
/var/lib/tftpboot/rootfs *(rw,sync,no_root_squash,no_subtree_check)
## Use the absolute path of the rootfs
```

d. Trigger an update of the exported NFS directories:

```
sudo exportfs -a
```

e. If the root filesystem image is available as a compressed ".bz2" file, extract it to the exported folder with the following command:

```
$ sudo tar –xjf ~/yocto/build_cgtqmx6/tmp/deploy/images/cgtqmx6/fsl-image-machine-test-cgtqmx6.tar.bz2 –C /var/lib/tftpboot/rootfs
```

f. Finally, copy the kernel and the devicetree blob files:

```
$ sudo cp ~/yocto/build_cgtqmx6/tmp/deploy/images/cgtqmx6/uImage /var/lib/tftpboot
$ sudo cp ~/yocto/build_cgtqmx6/tmp/deploy/images/cgtqmx6/uImage-imx6q-qmx6.dtb /var/lib/tftpboot
$ sudo cp ~/yocto/build_cgtqmx6/tmp/deploy/images/cgtqmx6/uImage-imx6dl-qmx6.dtb /var/lib/tftpboot
```

After that, the host system is prepared to provide the kernel and the root filesystem via network to the target system.

3. Configure the target system.

Adapt the following bootloader environment variables of the target system to your needs:

Variable	Description
ipaddr	The IP address of the target system
netmask	The netmask of the network
serverip	The IP address of the host system
bootfile	The name of the kernel image
fdt_file	The name of the device tree blob file
nfsroot	The path of the root filesystem

Note

Bootloader 2013 and later versions can boot kernels with device tree support. The environment variable "boot_fdt" controls whether the device tree blob file, specified by variable fdt_file (i.e imx6q-qmx6.dtb), is loaded or not.

An example of how to set up network boot at the u-boot console is shown below (u-boot version 2013.04 and kernel version 3.10.x):

```
setenv dyn_ip 'no'
setenv ipaddr '10.11.7.2'
setenv netmask '255.255.0.0'
setenv serverip '10.11.7.3'
setenv bootfile 'uImage'
setenv nfsroot '/var/lib/tftpboot/rootfs'
setenv netargs 'setenv bootargs console=${console},${baudrate} video=mxcfb0:dev=${vid_dev0} root=/dev/nfs ip=${ipaddr}:${serverip}:${gatewa
y}:${netmask} nfsroot=${nfsroot},v3,tcp'
saveenv
```

To boot a kernel that does not support device tree (for example kernel version 3.0.35), run the commands below at the u-boot command prompt:

```
setenv boot_fdt 'no'
saveenv
```

3.3.2 Micro-SD Card

1. Transfer the root filesystem with the following commands:

```
# Replace sdX / sdX1 with the appropriate device / partition with caution!
$ cd ~/yocto/build/tmp/deploy/images/cgtimx6/
$ sudo dd if=/dev/zero of=/dev/sdX count=1000 bs=512
$ sudo sfdisk --force -uM /dev/sdX <<EOF
10,,83
EOF
$ sudo mkfs.ext3 -j /dev/sdX1
$ sudo mkdir -p /mnt/imgprep
$ sudo mount /dev/sdX1 /mnt/imgprep
$ sudo tar -xjvf fsl-image-machine-test-cgtqmx6-xxxxxxxxxxxxxx.tar.bz2 -C /mnt/imgprep
$ sudo sync
$ sudo umount /dev/sdX1
```

Caution

Adapt sdX to whatever your device is recognized as. Otherwise, loss of data may occur.

2. Power on the target system and immediately press any key to enter the bootloader console prompt. To boot from a micro-SD card, modify the u-boot mmcdev environment with the commands below:

```
U-Boot> setenv mmcdev 0
U-Boot> setenv mmcroot '/dev/mmcblk0p1 rootwait rw'
U-Boot> saveenv
U-Boot> reset
```

3.3.3 eMMC

This requires a micro-SD card pre-installed with conga-QMX6 root filesystem. The micro-SD card acts as temporary bootmedia for the target system when the eMMC does not contain a valid root filesystem.

1. Follow section 3.3.2 "Micro-SD Card" to prepare the micro-SD card.

2. Transfer the tar.bz2 file to the micro-SD card

```
$ cd ~/yocto/build/tmp/deploy/images/cgtimx6/
$ sudo mkdir -p /mnt/imgprep
$ sudo mount /dev/sdX1 /mnt/imgprep
$ sudo cp fsl-image-machine-test-cgtqmx6-xxxxxxxxxxxxxx.tar.bz2 /mnt/imgprep
$ sync
$ sudo umount /dev/sdX1
```

Caution

Adapt sdX to whatever your device is recognized as. Otherwise, loss of data may occur.

3. Boot up the system. After system boot up, clear the master boot record on the eMMC of the target device, create and mount ext3 filesystem as shown below:

```
$ sudo dd if=/dev/zero of=/dev/mmcblk1 count=1000 bs=512
$ echo -e "o\nn\np\n1\n\n\nw\n" | fdisk /dev/mmcblk1
$ sudo mkfs.ext3 -j /dev/mmcblk1p1
$ sudo mount /dev/mmcblk1p1 /mnt/imgprep
$ sudo tar -xjvf /fsl-image-machine-test-cgtqmx6-xxxxxxxxxxxxxx.tar.bz2 -C /mnt/imgprep
$ sudo sync
```

4. Shut down the system and turn off the power supply. Afterwards, remove the micro-SD card.

5. Power on the system and immediately press any key to enter the bootloader console prompt. To boot from eMMC, modify the u-boot environment as follows:

```
U-Boot> setenv mmcdev 1
U-Boot> setenv mmcroot '/dev/mmcblk1p1 rootwait rw'
U-Boot> saveenv
U-Boot> reset
```

Note

conga-UMX6 does not provide a micro-SD card slot. The procedure is also applicable to SD cards but the device paths have to be changed appropriately.

4 Android

A x86 based Linux system with installed cross-compile toolchain / SDK Android development is required for i.MX6-based congatec designs. It is possible to use a virtual machine but a dedicated system is preferrable. Furthermore, the host should have a serial port to access the serial console and an SD card reader.

The i.MX6-based congatec designs require a 64-bit Ubuntu 14.04 system as the host for Android development. This chapter uses Android Lollipop 5.1.1 as an example but it is also applicable to Android Marshmallow 6.0.1 and possibly even later versions.

For instructions on how to build Android images for Lollipop 5.0.0 or older, see the readme files provided on the appropriate product page at the congatec website www.congatec.com.

4.1 Setting Up the Development System

1. Install the JDK:

```
$ sudo apt-get update
$ sudo apt-get install openjdk-7-jdk
```

2. Install required packages:

```
$ sudo apt-get install bison g++-multilib git gperf libxml2-utils make python-networkx zlib1g-dev:i386 zip uuid uuid-dev liblzo2-2 liblzo2-dev lzop git-core curl u-boot-tools mtd-utils gcc-multilib
```

4.2 Preparation: Required Sources and Files

Follow the steps below to obtain the required sources and files:

1. Obtain android source code:

```
$ cd ~
$ mkdir myandroid
$ mkdir bin
$ cd myandroid
$ curl http://commondatastorage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
$ ~/bin/repo init -u https://android.googlesource.com/platform/manifest -b android-5.1.1_r1
$ ~/bin/repo sync
$ cd ~/myandroid/prebuilts/gcc/linux-x86/arm
# This command loads the necessary repositories. Therefore, it can take several hours to load.
```

```
$ git clone https://android.googlesource.com/platform/prebuilts/gcc/linux-x86/arm/arm-eabi-4.6
$ cd arm-eabi-4.6
$ git checkout android-4.4.3_r1
```

2. Patch android source code:

```
$ cd ~
$ tar -zxvf android_L5.1.1_2.1.0-ga_core_source.tar.gz
$ cd android_L5.1.1_2.1.0_consolidated-ga_core_source/code
$ tar -zxf L5.1.1_2.1.0_consolidated-ga.tar.gz
$ cd ~/myandroid
$ source ~/android_L5.1.1_2.1.0_consolidated-ga_core_source/code/L5.1.1_2.1.0_consolidated-ga/and_patch.sh
$ c_patch ~/android_L5.1.1_2.1.0_consolidated-ga_core_source/code/L5.1.1_2.1.0_consolidated-ga imx_L5.1.1_2.1.0-ga
```

3. Obtain u-boot source code:

```
$ cd ~/myandroid/bootable/bootloader
$ git clone https://git.congatec.com/arm/qmx6_uboot.git uboot-imx
$ cd uboot-imx
$ git checkout cgt_imx_v2016.01_1.0.0
```

4. Obtain kernel source code:

```
$ cd ~/myandroid
$ git clone https://git.congatec.com/android/qmx6_kernel.git kernel_imx
$ cd kernel_imx
$ git checkout cgt-lp5.1.1-3.14.52
```

5. Obtain device BSP files:

```
$ cd ~/myandroid/device/fsl
$ git remote add cgt-lp5.1.1-2.1.0 https://git.congatec.com/android/device.git
$ git fetch cgt-lp5.1.1-2.1.0
$ git checkout --track cgt-lp5.1.1-2.1.0/cgt-lp5.1.1_2.1.0
```

6. Obtain hardware BSP files:

```
$ cd ~/myandroid/hardware/imx
$ git remote add cgt-lp5.1.1-2.1.0 https://git.congatec.com/android/hardware.git
$ git fetch cgt-lp5.1.1-2.1.0
$ git checkout --track cgt-lp5.1.1-2.1.0/cgt-lp5.1.1-2.1.0
```

7. Obtain android build files:

```
$ cd ~/myandroid/build
$ git remote add cgt-lp5.1.1-2.1.0 https://git.congatec.com/android/build.git
$ git fetch cgt-lp5.1.1-2.1.0
$ git checkout --track cgt-lp5.1.1-2.1.0/cgt-lp5.1.1-2.1.0
```

8. Obtain android system files:

```
$ cd ~/myandroid/system/core
$ git remote add cgt-lp5.1.1-2.1.0 https://git.congatec.com/android/system.git
$ git fetch cgt-lp5.1.1-2.1.0
$ git checkout --track cgt-lp5.1.1-2.1.0/cgt-lp5.1.1-2.1.0
```

4.3 Building the Image

Follow the steps below to build the image:

1. Prepare the environment:

```
$ export ARCH=arm
$ export CROSS_COMPILE=~/myandroid/prebuilts/gcc/linux-x86/arm/arm-eabi-4.6/bin/arm-eabi-
$ cd ~/myandroid
$ source build/envsetup.sh
$ lunch cgt_imx6-eng
# If the target system is conga-UMX6 PN 016203 or 016204 execute also the following command:
# export OPTION512M=yes
```

2. Build the system:

```
$ cd ~/myandroid/kernel_imx
$ make qmx6_android_defconfig
$ make
$ cd ..
$ make
```

Note

Two different files are provided for each module variant in the folder "~/myandroid/out/target/products/cgt_imx6/":

- *conga-QMX6: SPL-cgtimx6-2016.01-r0-cgtqmx6-2016.01-r0 and u-boot-cgtqmx6-2016.01-r0.img*
- *conga-UMX6: SPL-cgtimx6-2016.01-r0-cgtumx6-2016.01-r0 and u-boot-cgtumx6-2016.01-r0.img*

Substitute the bootloader on the target module with the bootloader from the manufacturing tool (MFGTool) package.

4.4 Deploying the Image

4.4.1 Micro-SD Card

After compilation, three images are generated under the folder "~/myandroid/out/target/product/cgt_imx6":

- boot.img
- system_raw.img
- recovery.img

1. Transfer these images to the micro-SD card:

```
$ sudo chmod +x ~/myandroid/device/fsl/common/tools/fsl-sdcard-partition.sh
$ sudo ~/myandroid/device/fsl/common/tools/fsl-sdcard-partition.sh /dev/sdX
$ cd ~/myandroid/out/target/product/cgt_imx6
$ sudo dd if=boot.img of=/dev/sdX1; sync
$ sudo if=system_raw.img of=/dev/sdX5; sync
$ sudo dd if=recovery.img of=/dev/sdX2; sync
```

Caution

Adapt sdX to whatever your device is recognized as. Otherwise, loss of data may occur.

2. Insert the micro-SD card into the module.

3. Power up the module and press any key to stop the autoboot.

4. At the u-boot command prompt, type the commands below:

```
$ setenv bootcmd “run bootcmd_android”
$ saveenv
$ boot
```

Note

There is no micro-SD card slot at conga-UMX6 modules. Please refer to section 4.6.2 "SD Card".

4.4.2 SD Card

1. Transfer the android image files to the SD card as described in section 3.3.2 "Micro-SD Card".
2. Insert the SD card.
3. Power up the module and press any key to stop the autoboot.
4. At the u-boot command prompt, type the commands below:

```
$ setenv mmcdev 2
$ setenv bootcmd “run bootcmd_android”
$ saveenv
$ reset
```

4.4.3 eMMC

You require an SD or micro-SD card with a Yocto image:

1. Download the Yocto sample image from the congatec website and transfer it to the SD card.

```
$ sudo mount /dev/sdX1 /mnt
$ sudo tar -zxvf cgt-imx6_yocto2.0_3.14.52_core-image-minimal_r110.tar.bz2 -C /mnt
```

2. Copy the Android images to the previously created Yocto SD card (card shall be mounted on /mnt).

```
$ sudo mkdir /mnt/android
$ sudo cp ~/myandroid/out/target/product/cgt_imx6/boot.img /mnt/android
$ sudo cp ~/myandroid/out/target/product/cgt_imx6/system_raw.img /mnt/android
$ sudo cp ~/myandroid/out/target/product/cgt_imx6/recovery.img /mnt/android
$ sudo cp ~/myandroid/device/fsl/cgt_imx6/eMMCtransfer.sh /mnt/android
$ sync
$ sudo umount /mnt
```

3. Insert the SD card.
4. Power up the module and press any key to stop the autoboot. At the command prompt, type the commands below:

```
$ env default –a
$ boot
```

5. Login as root user (username "root")
6. Make the `eMMCtransfer.sh` script executable:

```
$ cd /android
$ chmod +x eMMCtransfer.sh
$ ./eMMCtransfer.sh /dev/mmcblk1
$ sync
$ reboot
```

7. Press any key to stop the autoboot at u-boot console prompt (subsequent to reboot).

8. Adapt the `bootcmd` environment variable:

```
$ setenv mmcdev 1
$ setenv bootcmd “run bootcmd_android”
$ saveenv
$ reset
```

4.5 Updating Procedure: Sources

1. Update the source and config files:

```
$ cd ~/myandroid/kernel_imx
$ git pull
$ cd ~/myandroid/devices/fsl
$ git pull
$ cd ~/myandroid/hardware/imx
$ git pull
$ cd ~/myandroid/build
$ git pull
```

2. Prepare the environment:

```
$ export ARCH=arm
$ export CROSS_COMPILE=~/myandroid/prebuilts/gcc/linux-x86/arm/arm-eabi-4.6/bin/arm-eabi-
$ cd ~/myandroid
$ source build/envsetup.sh
$ lunch cgt_imx6-eng
```

3. Rebuild the system:

```
 $ cd ~/myandroid/kernel_imx
 $ make qmx6_android_defconfig
 $ make
 $ cd ..
 $ make
```

4. Transfer all the images to the micro-SD card as described above.

5 Boot Process

The Power on Reset (PoR) signal starts the boot process by executing the code in the boot ROM. In normal operation mode, the boot ROM uses the state of the BOOT_MODE register and the boot fuses to determine the boot device storing the bootloader code.

5.1 Boot fuses

The i.MX6 processor provides an array of One Time Programmable (OTP) registers, called boot fuses. The boot fuses store configuration and data permanently. They can be programmed to fetch the bootloader from different locations such as SPI-flash, eMMC, SATA or SD card. They are used for boot, security and MAC address configuration. They can only be programmed once. If a boot fuse is burned, it can not be restored to its original state.

On the conga-QMX6/UMX6, the boot fuses pre-configuration causes the boot ROM to fetch the bootloader from the onboard SPI-flash. The boot fuses can also be configured to determine whether the boot ROM boots in standard mode or in a secure mode, called High Assurance Boot (HAB). In secure boot mode, only certified boot images are accepted by the internal boot ROM. If you attempt to boot an uncertified image, the boot flow jumps to the serial downloader mode. In this mode, you have to pass certified boot images to the system via serial USB connection.

The hash keys are used in secure boot mode to authenticate a certified image. They are stored in an OTP boot fuse array. For more information about how to use and implement secure boot mode, refer to section 8 "High Assurance Boot (HAB)".

Note

In some cases, it may be desirable to fetch the bootloader from an interface (e.g. SD card) instead of the SPI-flash. To achieve this, you can either use a stub file in the SPI-flash or a customized conga-QMX6/UMX6 (with a customized setup of the boot fuses). For additional information about board customization, contact the congatec technical support.

The boot fuses on the conga-QMX6 are not write-protected. Therefore, the customer may choose application specific functions. If the boot fuses were write-protected, some functional decisions, e.g. enable or disable JTAG debugging, would be made in advance. This would reduce the functions available to the customer.

Caution

Altering settings of the boot fuses can make the module inoperable. Therefore, congatec recommends to write-protect the boot fuses against alteration in the customer’s final production. If the boot fuses are altered, the customer is solely responsible for any damage that occurs. Damage on the module due to improper handling, altering or configuring of the boot fuses, is not the responsibility of congatec.

5.2 IOMUX Configuration

For Linux kernel version 3.0.35, the conga-QMX6 IOMUX routing configuration can be found in the following kernel files:

```
/arch/arm/mach-mx6/board-mx6q_qmx6.c
/arch/arm/mach-mx6/board-mx6q_qmx6.h
/arch/arm/mach-mx6/board-mx6dl_qmx6.h
```

For Linux kernel versions 3.10.xx, 3.14.xx and 4.1.xx the conga-QMX6/UMX6 IOMUX routing configuration can be found in the following kernel device tree source files:

```
/arch/arm/boot/dts/imx6q-qmx6.dts
/arch/arm/boot/dts/imx6dl-qmx6.dts
/arch/arm/boot/dts/imx6qdl-qmx6.dtsi
/arch/arm/boot/dts/imx6qdl.dtsi
```

Note

Please contact your congatec support to receive a full overview of all i.MX6 to Qseven signal connections.

6 Bootloader (u-boot)

The u-boot bootloader is a GNU GPL licensed open source software. The u-boot input/output is redirected to one of the two onboard serial ports. In order to get access to the u-boot output, or the u-boot command-line interface in general, establish a serial connection between the host and target system.

The boot behavior is controlled via so called environment variables. They can be set with help of the u-boot command-line interface.

6.1 u-boot 2009.08

The source code for u-boot 2009.08 is provided on the congatec git server:

- https://git.congatec.com/arm/qmx6_uboot, branch

cgt_imx_3.0.35_1.1.0 conga-UMX6 is supported since git rev:

- https://git.congatec.com/arm/qmx6_uboot/commit/206c65d49f3e7bab0dd291377148e58dfcd2c9ff

It is suitable to boot i.MX6 Linux kernels without device tree support.

Note

Bootloader version 2009.08 is not recommended for new designs. If you must use it, start the development on top of the latest bootloader release.

6.1.1 Environment Variables

The behavior of the bootloader is controlled by environment variables. The bootloader binary serves a predefined default environment. The following table shows the environment variables of the standard bootloader version u-boot 2009.08:

Variable	Default	Description
bootdelay	3	The boot delay in seconds
baudrate	115200	The baudrate for the serial terminal connection
ipaddr	192.168.1.103	The ip address used for network communication
netmask	255.255.255.0	The netmask used for network communication
serverip	_SERVER_IP_ADDR_	The ip address of a remote server used for netboot

bootfile	_BOOT_FILE_PATH_IN_TFTP_	The name of the file that is requested from a remote server during netboot (e.g. via the dhcp or the tftp command)
nfsroot	_ROOTFS_PATH_IN_NFS_	The path to the NFS root filesystem used for netboot
loadaddr	0x10800000	The destination address in the memory the bootfile/bootscript is stored to
bootdelay	3	The boot delay in seconds
baudrate	115200	The baudrate for the serial terminal connection
ipaddr	192.168.1.103	The ip address used for network communication
netmask	255.255.255.0	The netmask used for network communication
serverip	_SERVER_IP_ADDR_	The ip address of a remote server used for netboot
bootfile	_BOOT_FILE_PATH_IN_TFTP_	The name of the file that is requested from a remote server during netboot (e.g. via the dhcp or the tftp command)
nfsroot	_ROOTFS_PATH_IN_NFS_	The path to the NFS root filesystem used for netboot
loadaddr	0x10800000	The destination address in the memory the bootfile/bootscript is stored to

Furthermore, the following bootloader scripts are defined:

Script	Description
bootcmd	The default boot command that will be executed during system boot. By default, this script executes the bootcmd_mmc script.
bootcmd_mmc	Sets the bootargs and tries to fetch and execute the bootscript (6q_bootscript) from onboard µSD, onboard eMMC or external SD card
bootcmd_net	Tries to get an ip address via dhcp and boots from network. Observe: In order to perform network boot, additional settings, such as nfsroot, serverip, etc. have to be adjusted accordingly.
bootargs_base	Used by various other scripts to set the basic boot parameters (such as console setting and the configuration of the video devices)
bootargs_mmc	Used by the bootcmd_mmc script to initialize the boot parameters for mmc boot
bootargs_nfs	Used by the bootcmd_net script to initialize the boot parameters for network boot
clearenv	The clearenv script is used to reset the environment settings to their default state
upgradeu	Tries to fetch and execute the upgrade script (6q_upgrade) from onboard µSD, onboard eMMC or external SD card

6.1.2 Version Specific Hints

- Booting from USB devices is not supported.
- In contrast to newer bootloader versions (e.g, u-boot 2013), the bootloader binary is zero padded. Zero padding means, that the first 1024 (0x400) bytes of the binary are filled with zeros (0x00). Because of this, the binary file (.bin) is suitable to be copied to SPI-flash or SD card without additional offset.
- Booting usually occurs by means of a bootscript (6q_bootscript) residing in the root of the boot partition.

6.1.3 Special Functionality

The mfgdump u-boot command is used to print the content of the congatec manufacturing area in human readable format.

6.1.4 Bootloader Scripts

Environment variables serve as storage for values and small scripts, which can be executed with the `run` command (e.g. variable `bootcmd_mmc`, `clearenv` and others).

Likewise, the u-boot scripts enable to load and execute scripts stored at storage media devices (e.g. SD card). For example, the `6q_bootscript` is loaded from an external storage device and determines the further boot sequence.

Such bootloader script files must be converted into a special binary format. This is done with the mkimage utility (part of the u-boot sources).

Example

Convert the 6q_bootscript script:

```
$ mkimage -A arm -O linux -T script -C none -a 0 -e 0 -n “boot script” -d 6q_bootscript.src 6q_bootscript
```

Use the u-boot commands extload and source in order to load and execute the resulting binary bootscript file (`6q_bootscript`):

```
conga-QMX6 U-Boot >  extload mmc 0:1 10008000 /6q_bootscript && source 10008000
```

In the command above, the bootscript (6q_bootscript) is loaded from the first EXT partition of mmc device 0 to the system memory (memory address 0x10008000) – this is done with the u-boot command `extload`. Afterwards, the bootscript is executed by means of the `source` command.

6.1.5 Runtime Configuration

The user interacts with the u-boot bootloader by means of a command-line interface, called hush shell, via a serial connection. Such a serial connection has to be established with help of a serial terminal application like Tera Term or minicom.

The hush shell provides a set of commands and simple scripting functionality. The `help` command gives a short overview of the available commands.

The boot sequence is controlled by a set of environment variables, simply called environment. The u-boot binary comes with a set of predefined variables, modelling commonly used bootmodes, called standard environment.

There are several commands in order to administrate environment variables. The following table shows an important subset:

Command	Description
setenv	Modifies the value of an environment variable.
saveenv	Saves the environment to SPI-flash.
help	Prints a help text for each command.
print	Prints a list of the current environment variables.

6.1.6 Restoring the Default Environment

Execute the `clearenv` script to restore the default environment settings:

```
conga-QMX6 U-Boot > run clearenv
```

Switch off the power directly after running the `clearenv` script to ensure the environment will not be modified (and stored) by subsequent actions (e.g. by executing scripts that contain `setenv` commands).

6.1.7 Selecting the Boot Device

6.1.7.1 Network Boot

If a DHCP server provides the network configuration parameters, enter `run bootcmd_net` in the bootloader console prompt to perform network boot.

The following table shows an example of a minimal configuration to boot from network (dynamic network configuration via DHCP):

Variable	Example Value
serverip	10.11.7.3
nfsroot	/tftproot/rootfs
nootfile	uImage
bootcmd_mfg	sets the bootargs for manufacturing and tries to boot the manufacturing system

The following table shows the whole set of environment variables required for network boot in case of a static network:

Variable	Example Value
ipaddr	10.11.7.2
serverip	10.11.7.3
ipaddr	10.11.7.2
netmask	255.255.0.0
nfsroot	/tftproot/rootfs
bootfile	uImage
bootargs_nfs	'setenv bootargs ${bootargs} root=/dev/nfs ip=${ipaddr}:${serverip}:${gateway}:${netmask} nfsroot=${nfsroot},v3,tcp'
bootcmd_net	'run bootargs_base; run bootargs_nfs; tftp $loadaddr uImage; bootm $loadaddr'

Note

Network boot requires additional server components and configuration, providing a kernel and a root filesystem to the target system via TFTP/NFS – please refer to section 3.3.1 "Network Boot".

6.1.7.2 eMMC

1. Format the eMMC to FAT or EXT2.

2. Modify bootcmd_mmc to boot from the onboard eMMC:

```
conga-QMX6 U-Boot > print bootcmd_mmc
bootcmd_mmc=run bootargs_base bootargs_mmc;for disk in 0 1 2; do mmc dev ${disk};for fs in fat ext2 ; do ${fs}load mmc ${disk}:1   10008000
/6q_bootscript && source 10008000 ; done ; done
conga-QMX6 U-Boot > setenv bootcmd_mmc 'run bootargs_base bootargs_mmc;for disk in 1 2 0; do mmc dev ${disk};for fs in fat ext2 ; do  ${fs}
load mmc ${disk}:1 10008000 /6q_bootscript && source 10008000 ; done ; done'
conga-QMX6 U-Boot > saveenv
```

In the example above, just the scan order of the different mmc devices is modified. The search order for the bootscript is: onboard eMMC, external SD card, onboard micro-SD.

Note

The root device entry in the kernel parameters of the bootscript (6q_bootscript) has to be set to the onboard eMMC as well.

6.1.7.3 SATA Device

Modify `bootcmd_sata` to boot from SATA:

```
conga-QMX6 U-Boot > setenv bootcmd_sata 'sata init; ext2load sata 0:1 10008000 /6q_bootscript && source 10008000'
conga-QMX6 U-Boot > saveenv
```

Note

The root device entry in the kernel parameters of the bootscript (6q_bootscript) has to be set to the SATA device as well.

6.1.8 Boot Loader Types

There are three types of bootloaders:

- Standard bootloader

  This is the standard bootloader for booting into a Linux based system (without device tree support).

- Bootloader for Android

  This bootloader is required for booting into an Android based system. Compared to the standard bootloader, it has enhanced capabilities and a different environment setup.

- Manufacturing bootloader

  This is a special version of the bootloader to be used with the NXP manufacturing utility (MFGTool) to bring up or update the module.

Note

The bootloader is tailored to the part number of the congatec module. Use matching bootloader and part number for each module.

6.1.9 Build Process

The following steps describe the bootloader build process in a standalone environment using the standalone cross-development environment (see section 2.3 "Setting Up the Standalone Cross-Development Environment") for conga-QMX6 (part number 016103):

1. Fetch the source code and switch to correct branch.

2. Set up the build environment:

```
~/qmx6_uboot$ source ~/sourceme
```

Note

Follow section 2.3 " Setting Up the Standalone Cross-Development Environment" to set up your build environment and the stated sourceme file.

3. Select the matching configuration and build the bootloader:

```
~/qmx6_uboot$ make cgt_qmx6_config partnumber=016103
~/qmx6_uboot$ make
```

A bootloader binary (u-boot.pn016103.bin) is created in the build directory, respecting the part number naming scheme.

4. Flash the bootloader to the target system with the NXP MFGTool.

6.2 u-boot 2013.04

The source code for uBoot 2013.04 is provided on the congatec git server:

- https://git.congatec.com/arm/qmx6_uboot), branch cgt_imx_v2013.04_3.10.17_1.0.2

conga-UMX6 is supported since git rev:

- https://git.congatec.com/arm/qmx6_uboot/commit/ef1818de1e0a29d75927336c1c007a2e67645d22

It is suitable to boot i.MX6 Linux kernels with and without device tree support.

Note

Bootloader version 2013.04 is not recommended for new designs. If you must use it, start the development on top of the latest bootloader release.

6.2.1 Environment Variables

The behavior of the bootloader is controlled by environment variables. The bootloader binary serves a predefined default environment. The following table shows the environment variables of the standard bootloader version u-boot 2013.04:

Variable	Default	Description
baudrate	115200	The baudrate for the serial terminal connection
boot_fdt	try	Specifies if a kernel with separate devicetree blob file will be loaded. Possible values: yes, no, try
bootdelay	1	The boot delay in seconds
console	ttymxc1	The device for console output
ethact	FEC	Name of active ethernet interface
ethaddr	00:00:00:00:00:00	The ethernet MAC address: if specified, this value temporarily overwrites the MAC address that is provided by the OTP fuses
ethprime	FEC	Name of primary ethernet interface
fdt_addr	0x18000000	The destination address in the memory the fdt blobfile is stored to
fdt_file	imx6q-qmx6.dtb*	The name of the fdt blobfile (dependends on the type of module)
fdt_high	0xffffffff	Restricts the maximum address that the flattened device tree will be copied into upon boot. A value of 0xffffffff prevents copying the fdt at all.
initrd_addr	0x12C00000	The destination address in the memory the initial ramdisk is stored to (optional)
initrd_high	0xffffffff	Restricts the positioning of initrd images. A value of 0xffffffff prevents copying the ramdisk at all.
ip_dyn	yes	Specifies if the ip address should be assigned dynamically (via dhcp) or if a statically assigned ip address should be used
ipaddr		The static ip address used for network communication (not de-fined in default environment)
loadaddr	0x12000000	The destination address in the memory the bootfile/bootscript is stored to
mmcdev	0	The mmc device from which the bootscript/kernel/system is loaded (0: µSD, 1: external SD card, 2: onboard eMMC)
mmcpart	1	The partion number from which the bootscript/kernel/system is loaded
netmask		The static netmask used for network communication (not de-fined in default environment)
nfsroot		The path to the NFS root filesystem used for netboot (not de-fined in default environment)
mmcroot	/dev/mmcblk0p1 rootwait rw	The root device for mmcboot (can also be used to pass addi-tional kernel parameters, e.g. rootwait, etc.)
script	boot.scr	The name of the (optional) bootscript
serverip		The static ip address of a remote server used for netboot (not defined in default environment)
uimage	uImage	The name of the kernel image
vid_dev0	hdmi,1920x1080M@60,if=RGB24	The definition of the first video device, see sec-tion "configuration of video devices" for details
vid_dev1	ldb,LDB-XGA,if=RGB666	The definition of the second video device, see sec-tion "configuration of video devices" for details

Furthermore, the following bootloader scripts are defined:

Script	Description
bootcmd	The default boot command that will be executed during system boot. By default, this script tries to load and execute a bootscript/kernel from mmc. If this fails, netboot is performed.
bootcmd_mfg	Sets the bootargs for manufacturing and tries to boot the manufacturing system
loadbootscript	Loads the specified bootscript from mmc via ext2load
loadfdt	Loads the fdtblob file from mmc (/boot folder) via ext2load
loaduimage	Loads the kernel from mmc (/boot folder) via ext2load
mfgtool_args	Used by bootcmd_mfg script to initialize the boot parameters for the manufacturing environment
mmcargs	Used by the mmcboot script to initialize the boot parameters for mmc boot
mmcboot	Script for booting from mmc. This script initializes the boot parameters (via mmcargs), loads the fdtblob file (via loadfdt) and finally starts the kernel.
	Attention: the kernel must already be present in memory (i.e. previously loaded via loaduimage).
netargs	Used by the netboot script to initialize the boot parameters for network boot
netboot	Tries to negotiate an ip address (dhcp or static) and boots from network. Observe: In order to perform network boot, additional settings, such as nfsroot, serverip, etc. have to be adjusted accordingly.

6.2.2 Version Specific Hints

- Booting from USB devices is supported.
- In contrast to previous bootloader versions (e.g, u-boot 2009), the bootloader binary is NOT zero padded. Zero padding means, that the first 1024 (0x400) bytes of the binary are filled with zeros (0x00). Because there is NO zero padding, the binary file (.imx) has to be copied to SPI-flash or SD card with additional offset.

6.2.3 Special Functionality

The `mfgdump` u-boot command is used to print the content of the congatec manufacturing area in human readable format.

6.2.4 Runtime Configuration

The user interacts with the u-boot bootloader by means of a command-line interface, called hush shell, via a serial connection. Such a serial connections has to be established with help of a serial terminal application like Tera Term or minicom.

The hush shell provides a set of commands and simple scripting functionality. The `help` command gives a short overview of the available commands.

The boot sequence is controlled by a set of environment variables, simply called environment. The u-boot binary comes with a set of predefined variables, modelling commonly used bootmodes, named standard environment. There are several commands in order to administrate environment variables. The following table shows an important subset:

Command	Description
setenv	Modifies the value of an environment variable
saveenv	Saves the environment to SPI-flash
env default -a	Restore the default values of the entire environment
help	Prints a help text for each command
print	Prints a list of the current environment variables

6.2.5 Selecting the Boot Device

The u-boot bootloader version 2013.04 enables boot from MMC devices (SD, eMMC and micro-SD), SATA, USB and via network (TFTP/NFS). The boot command loads a bootscript or a kernel from the boot device. If unavailable, network boot is performed.

In case of MMC devices, the variable `${mmcdev}` specifies the boot device:

MMC Device	${mmcdev}	${mmcroot}
onboard micro-SD card	0	/dev/mmcblk0p1 rootwait rw
onboard eMMC	1	/dev/mmcblk1p1 rootwait rw
external SD card	2	/dev/mmcblk2p1 rootwait rw

Furthermore, the variable `${mmcroot}` is passed to the kernel in order to specify the location of the root filesystem, e.g. `/dev/mmcblk0p1 rootwait rw`.

6.2.5.1 Network Boot

If a DHCP server provides the network configuration, enter `run netboot` in the bootloader console prompt to perform neetwork boot. The following table shows an example of a minimal configuration to boot from network (dynamic network configuration via DHCP):

Variable	Example Value
serverip	10.11.7.3
nfsroot	/tftproot/rootfs

The following table shows the whole set of environment variables required for network boot in case of a static network:

Variable	Example Value
dyn_ip	No
ipaddr	10.11.7.2
serverip	10.11.7.3
netmask	255.255.0.0
nfsroot	/tftproot/rootfs
bootargs_nfs	'setenv bootargs ${bootargs} root=/dev/nfs ip=${ipaddr}:${serverip}:${gateway}:${netmask} nfsroot=${nfsroot},v3,tcp'

Note

Network boot requires additional server components and configuration, providing a kernel and a root filesystem to the target system via TFTP/NFS – please refer to section 3.3.1 "Network Boot".

6.2.5.2 Micro-SD Card

Adjust the environment to boot from a micro-SD card:

```
U-Boot > setenv mmcdev 0
U-Boot > setenv mmcroot /dev/mmcblk0p1 rootwait rw
U-Boot > saveenv
```

Note

There is no micro-SD card slot at conga-UMX6 modules.

6.2.5.3 eMMC

Adjust the environment to boot from the onboard eMMC:

```
U-Boot > setenv mmcdev 1
U-Boot > setenv mmcroot /dev/mmcblk1p1 rootwait rw
U-Boot > saveenv
```

6.2.5.4 SD Card

Adjust the environment to boot from an external SD card:

```
U-Boot > setenv mmcdev 2
U-Boot > setenv mmcroot /dev/mmcblk2p1 rootwait rw
U-Boot > saveenv
```

6.2.5.5 SATA Device

Adjust the environment to boot from the SATA device:

```
U-Boot > setenv bootcmd_sata 'sata init; run loadfdt; run loaduimage; run mmcargs; bootm ${loadaddr} - ${fdt_addr}'
U-Boot > setenv loadfdt 'ext2load sata ${mmcdev}:${mmcpart} ${fdt_addr} boot/${fdt_file}'
U-Boot > setenv loaduimage 'ext2load sata ${mmcdev}:${mmcpart} ${loadaddr} boot/${uimage}'
U-Boot > setenv mmcroot /dev/sda1 rootwait rw
U-Boot > setenv bootcmd run bootcmd_sata
U-Boot > saveenv
```

Note

This example assumes kernel and device tree file (fdt) are stored at an EXT filesystem. For FAT filesystems, please use fatload instead of ext2load.

6.2.5.6 USB Device

Adjust the environment to boot from the USB device:

```
U-Boot > setenv bootcmd_usb 'usb start; run loadfdt; run loaduimage; run mmcargs; bootm ${loadaddr} - ${fdt_addr}'
U-Boot > setenv loadfdt 'ext2load usb ${mmcdev}:${mmcpart} ${fdt_addr} boot/${fdt_file}'
U-Boot > setenv loaduimage 'ext2load usb ${mmcdev}:${mmcpart} ${loadaddr} boot/${uimage}'
U-Boot > setenv mmcroot /dev/sda1 rootwait rw
U-Boot > setenv bootcmd run bootcmd_usb
U-Boot > saveenv
```

Note

This example assumes kernel and device tree file (fdt) are stored at an EXT filesystem. For FAT filesystems, please use fatload instead of ext2load.

6.2.6 Configuring the Video Devices

In order to set up the video devices, the bootloader refers to two environment variables:

- `vid_dev0` controls the first kernel framebuffer device
- `vid_dev1` controls the second kernel framebuffer device

In general, the configuration of a framebuffer device via the kernel command-line follows the definition:

```
video=mxcfbX:dev=device,mode,interface[,options]
```

The following table describes the command one by one:

Variable	Description
X	The number of the framebuffer device, usually 0 or 1. This is the number of the device, not the number of the framebuffer itself. During startup, the kernel enumerates all the framebuffers and usually assigns fb0 and fb1 to the first device (respectively the background and the foreground framebuffer). Therefore, mxcfb0 is usually assigned to fb0 and fb1, mxcfb1 is usually assigned to fb2.
device	Specifies the video device, usually hdmi or ldb (LVDS display bridge).
mode	Specifies the video mode, e.g. LDB-XGA (in recent kernels, this entry is ignored for device ldb and the DTS configuration is used instead)
interface	Specifies the interface pixel format, e.g. if=RGB666 or if=RGB24

Additional options are available based on the device type, e.g. `fbpix=BGR32` specifies the framebuffer layout. Find additional information about framebuffer configuration in the documentation of the kernel source, e.g. in file Documentation/devicetree/ bindings/fb/fsl_ipuv3_fb.txt

Examples

The typical kernel command-line configuration for a 1920x1080 full HD display connected via HDMI is:

```
video=mxcfb0:dev=hdmi,1920x1080M@60,if=RGB24
```

In u-boot, set the environment variable `vid_dev0` as follows:

```
setenv vid_dev0 hdmi,1920x1080M@60,if=RGB24
```

Please ensure, that `${vid_dev0}` is correctly referenced at the kernel command-line, e.g.:

```
[…] video=mxcfb0:dev=${vid_dev0} […]
```

The typical kernel command-line configuration for a 1024x768 18-bit XGA display connected via the LVDS display bridge is:

```
video=mxcfb1:dev=ldb,LDB-XGA,if=RGB666
```

In u-boot, set the environment variable `vid_dev1` as follows:

```
setenv vid_dev1 ldb,LDB-XGA,if=RGB666
```

Please ensure, that `${vid_dev1}` is correctly referenced at the kernel command-line, e.g.:

```
[…] video=mxcfb1:dev=${vid_dev1} […]
```

The examples mentioned before describe the default configuration of the bootloader: framebuffer device 0 is assigned to a 1920x1080 full HD device connected via hdmi; framebuffer device 1 is assigned to a 1024x768 18-bit LVDS display connected via the LVDS display bridge.

6.2.7 Boot Loader Types

There are two types of bootloaders:

- Standard bootloader

  This is the standard bootloader for booting into a Linux based system (with or without device tree support).
- Bootloader for Android

  This bootloader is required for booting into an Android based system. Compared to the standard bootloader, it has enhanced capabilities and a different environment setup. Building a conga-QMX6/UMX6 bootloader for manufacturing purposes is not supported for bootloader version 2013.04

Note

The bootloader is tailored to the part number of the congatec module. Use matching bootloader and part number for each module.

6.2.8 Build Process

The following steps describe the bootloader build process using the standalone cross-development environment (see section 2.3 "Setting Up the Standalone Cross-Development Environment") for conga-QMX6 (part number 016103):

1. Fetch the source code and switch to correct branch:

```
~$ git clone https://git.congatec.com/arm/qmx6_uboot.git
~$ cd qmx6_uboot
~/qmx6_uboot$ git checkout -b cgt_imx_v2013.04_3.10.17_1.0.2 origin/cgt_imx_v2013.04_3.10.17_1.0.2
```

2. Set up the build environment:

```
~/qmx6_uboot$ source ~/sourceme
```

Note

Follow section 2.3 "Setting Up the Standalone Cross-Development Environment" *to set up your build environment and the stated sourceme file.*

3. Select the matching configuration and build the bootloader:

```
~/qmx6_uboot$ make cgt_qmx6_pn016103_config
~/qmx6_uboot$ make
```

   As a result, a part number based bootloader binary has been created in the build directory, e.g. u-boot.pn016103.imx

4. Flash the bootloader to the target system with the NXP MFGTool.

6.3 u-boot 2016.01

The source code for uBoot 2016.01 is provided on the congatec git server:

- https://git.congatec.com/arm/qmx6_uboot), branch cgt_imx_v2016.01_1.0.0

conga-UMX6 is supported since git rev:

- https://git.congatec.com/arm/qmx6_uboot/commit/ef1818de1e0a29d75927336c1c007a2e67645d22

It is suitable to boot i.MX6 Linux kernels with and without device tree support.

The Secondary Program Loader (SPL) implementation enables a common u-boot build (which is divided into two binaries: SPL and u-boot. img) for all product variants:

- the bootloader has to be built just once per product group (e.g. for conga-QMX6 or conga-UMX6), instead of individual builds for each variant
- no part number specific configuration required
- the two resulting binaries (SPL and u-boot.img) have to be flashed to distinct locations in the SPI-flash

Note

Bootloader version 2016.01 is recommended for new designs. Start the development on top of the latest bootloader release.

6.3.1 Environment Variables

The behavior of the bootloader is controlled by environment variables. The bootloader binary serves a predefined default environment. The following table shows the environment variables of the standard bootloader version u-boot 2016.01:

Variable	Default	Description
baudrate	115200	The baudrate for the serial terminal connection
board_rev	MX6Q (resp. MX6DL)	Depending on the type of i.MX6 CPU, this variable contains MX6Q or MX6DL
boot_fdt	try	Specifies if a kernel with separate devicetree blob file will be loaded. Possible values: yes, no, try
bootdelay	3	The boot delay in seconds
bootm_size	0x10000000	This variable defines the size of the region allowed for use by the bootm command
console	ttymxc1	The device for console output
ethact	FEC	Name of active ethernet interface
ethaddr	00:00:00:00:00:00	The ethernet MAC address: if specified, this value temporarily overwrites the MAC address provided by the OTP fuses

ethprime	FEC	Name of primary ethernet interface
fdt_addr_r	0x18000000	The destination address in the memory the fdt blobfile is stored to
fdt_file	undefined	The name of the fdt blobfile (dependend from the type of module)
image	uImage	The name of the kernel image
ip_dyn	yes	Specifies whether the ip address is assigned dynamically (via dhcp) or a statically assigned ip address is used
ipaddr		The static ip address used for network communication (not defined in default environment)
loadaddr	0x12000000	The destination address in the memory the bootfile/bootscript is stored to
mmcdev	0	The mmc device from which the bootscript/kernel/system is loaded (0: µSD, 1: external SD card, 2: onboard eMMC)
mmcpart	1	The partion number from which the bootscript/kernel/system is loaded
baudrate	115200	The baudrate for the serial terminal connection
mmcroot	/dev/mmcblk0p1 rootwait rw	The root device for mmcboot (can also be used to pass addi-tional kernel parameters, e.g. rootwait, etc.)
nfsroot		The path to the NFS root filesystem used for netboot (not de-fined in default environment)
script	boot.scr	The name of the (optional) bootscript
serverip		The static ip address of a remote server used for netboot (not defined in default environment)
vid_dev0	hdmi,1920x1080M@60,if=RGB24	The definition of the first video device, see section "configuration of video devices" for details
vid_dev1	ldb,LDB-XGA,if=RGB666	The definition of the second video device, see section "configuration of video devices" for details

Furthermore, the following bootloader scripts are defined:

Script	Description
bootcmd	The default boot command that will be executed during system boot. By default, this script locks the SPI-flash, tries to load and execute a bootscript/kernel from mmc. If this fails, netboot is performed.
bootcmd_android	An alternate boot command that can be used to boot into an Android based operating system
bootscript	Just executes an already sourced script via the source command
dfu_alt_info	
dfu_alt_info_img	
dfu_alt_info_spl	
dfu_spi	
findfdt	Sets the correct value of variable ${fdtfile} according the value of variable ${board_rev}. Usually, the script findfdt has to be executed before loading the fdtfile via loadfdt.
loadbootscript	Noads the specified bootscript from mmc via ext2load
loadfdt	Loads the fdtblob file from mmc (/boot folder) via ext2load
loadimage	Loads the kernel from mmc (/boot folder) via ext2load
mfgtool_args	Used by bootcmd_mfg script to initialize the boot parameters for the manufacturing environment
mmcargs	Used by the mmcboot script to initialize the boot parameters for mmc boot

mmcargs_android	Used by the bootcmd_android script to initialize the boot parameters for booting into android.
mmcboot	Script for booting from mmc. This script initializes the boot parameters (via mmcargs), loads the fdtblob file (via loadfdt) and finally starts the kernel. Note: The kernel must already be present in memory (i.e. previously loaded via loadimage).
netargs	Used by the netboot script to initialize the boot parameters for network boot
netboot	Tries to negotiate an ip address (dhcp or static) and boots from network. Observe: In order to perform network boot, additional settings, such as nfsroot, serverip, etc. have to be adjusted accordingly.
spilock	Protects the MFG area in the SPI-flash for beeing erased/corrupted by mistake (the MFG area, which will be initialized during production, contains important data that should not be destroyed).
update_sd_firmware	

6.3.2 Version Specific Hints

The congatec u-boot 2016.01 is based on the SPL framework to unify all existing variants:

- in the past (u-boot 2013 and before), there was a dedicated bootloader binary for each module variant. Main reason for this was the memory configuration which is different for each module variant (due to memory size, DDR clock frequency, density, etc.).
- a bootloader based on SPL is divided into two parts: a small binary (SPL) that will be loaded into onchip static RAM (OCRAM) in order to perform the DDR memory setup and the actual bootloader image (uboot.img) which will be loaded to DDR memory once SPL has finished the memory configuration. Two parts of the bootloader have to be flashed to distinct offsets of the SPI-flash.
- the SPL binary has to be flashed to offset 0x400
- the uboot.img binary has to be flashed to offset 0x10000
- u-boot 2016.01 supports booting from USB devices. See section 6.3.5.6 "USB Device" in order to perform USB boot.

6.3.3 Special Functionality

The SPL uses a mechanism to configure the onboard memory (basic setup of geometry, timings as well as calibration data) for each module variant. This code performs an automatic detection of the module variant and configures the memory accordingly. The build process generates a unified bootloader (consisting of SPL and uboot.img binaries) which works with all module variants. In previous implementations, each variant required an individual bootloader.

6.3.4 Runtime Configuration

The user interacts with the u-boot bootloader by means of a command-line interface, called hush shell, via a serial connection. Such a serial connection has to be established with help of a serial terminal application like Tera Term or minicom.

The hush shell provides a set of commands and simple scripting functionality. The `help` command gives a short overview of the available commands.

The boot sequence is controlled by a set of environment variables, simply called environment. The u-boot binary comes with a set of predefined variables, modelling commonly used bootmodes, called standard environment.

There are several commands in order to administrate environment variables. The following table shows an important subset:

Command	Description
setenv	Modifies the value of an environment variable
saveenv	Saves the environment to SPI-flash
env default variable	Restores the default value of a variable
env default -a	Restore the default values of the entire environment
help	Prints a help text for each command
print	Prints a list of the current environment variables

6.3.5 Selecting the Boot Device

The u-boot bootloader version 2016.01 supports boot from MMC devices (SD, eMMC and micro-SD), SATA, USB and via network (TFTP/NFS). The boot command loads a bootscript or a kernel from the boot device. If unavailable, network boot is performed. In case of MMC devices, the variable `${mmcdev}` specifies the boot device:

MMC Device	${mmcdev}	${mmcroot}
onboard micro-SD card	0	/dev/mmcblk0p1 rootwait rw
onboard eMMC	1	/dev/mmcblk1p1 rootwait rw
external SD card	2	/dev/mmcblk2p1 rootwait rw

Furthermore, the variable `${mmcroot}` is passed to the kernel in order to specify the location of the root filesystem, e.g. `/dev/mmcblk0p1 rootwait rw`.

6.3.5.1 Network Boot

If a DHCP server provides the network configuration, enter `run netboot` in the bootloader console prompt to perform network boot.

The following table shows an example of a minimal configuration to boot from network (dynamic network configuration via DHCP):

Variable	Example Value
serverip	10.11.7.3
nfsroot	/tftproot/rootfs

The following table shows the whole set of environment variables required for network boot in case of static network:

Variable	Example Value
dyn_ip	No
ipaddr	10.11.7.2
serverip	10.11.7.3
netmask	255.255.0.0
nfsroot	/tftproot/rootfs
bootargs_nfs	'setenv bootargs ${bootargs} root=/dev/nfs ip=${ipaddr}:${serverip}:${gateway}:${netmask} nfsroot=${nfsroot},v3,tcp'

Note

Network boot requires additional server components and configuration, providing a kernel and a root filesystem to the target system via TFTP/NFS – please refer to section 3.3.1 "Network Boot".

6.3.5.2 Micro-SD Card

Adjust the environment to boot from a micro-SD card:

```
U-Boot > setenv mmcdev 0
U-Boot > setenv mmcroot /dev/mmcblk0p1 rootwait rw
U-Boot > saveenv
```

Note

There is no micro-SD card slot at conga-UMX6 modules.

6.3.5.3 eMMC

Adjust the environment to boot from the onboard eMMC:

```
U-Boot > setenv mmcdev 1
U-Boot > setenv mmcroot /dev/mmcblk1p1 rootwait rw
U-Boot > saveenv
```

6.3.5.4 SD Card

Adjust the environment to boot from an external SD card:

```
U-Boot > setenv mmcdev 2
U-Boot > setenv mmcroot /dev/mmcblk2p1 rootwait rw
U-Boot > saveenv
```

6.3.5.5 SATA Device

Transfer the root filesystem to the SATA device and adjust the environment:

```
U-Boot > setenv bootcmd_sata 'sata init; run loadfdt; run loaduimage; run mmcargs; bootm ${loadaddr} - ${fdt_addr}'
U-Boot > setenv loadfdt 'ext2load sata ${mmcdev}:${mmcpart} ${fdt_addr} boot/${fdt_file}'
U-Boot > setenv loaduimage 'ext2load sata ${mmcdev}:${mmcpart} ${loadaddr} boot/${uimage}'
U-Boot > setenv mmcroot /dev/sda1 rootwait rw
U-Boot > setenv bootcmd run bootcmd_sata
U-Boot > saveenv
```

Note

This example assumes kernel and device tree file (fdt) are stored at an EXT filesystem. For FAT filesystems, please use fatload instead of ext2load.

6.3.5.6 USB Device

Transfer the root filesystem to the USB device and adjust the environment:

```
U-Boot > setenv bootcmd_usb 'usb start; run loadfdt; run loaduimage; run mmcargs; bootm ${loadaddr} - ${fdt_addr}'
U-Boot > setenv loadfdt 'ext2load usb ${mmcdev}:${mmcpart} ${fdt_addr} boot/${fdt_file}'
U-Boot > setenv loaduimage 'ext2load usb ${mmcdev}:${mmcpart} ${loadaddr} boot/${uimage}'
U-Boot > setenv mmcroot /dev/sda1 rootwait rw
U-Boot > setenv bootcmd run bootcmd_usb
U-Boot > saveenv
```

Note

This example assumes that kernel and device tree file (fdt) are stored at an EXT filesystem. For FAT filesystems, please use fatload instead of ext2load.

6.3.6 Configuring the Video Devices

In order to set up the video devices, the bootloader refers to two environment variables:

- `vid_dev0` controls the first kernel framebuffer device.
- `vid_dev1` controls the second kernel framebuffer device.

In general, the configuration of a framebuffer device via the kernel command-line follows the definition:

```
video=mxcfbX:dev=device,mode,interface[,options]
```

The following table describes the command one by one:

Variable	Description
X	The number of the framebuffer device, usually 0 or 1. This is the number of the device, not the number of the framebuffer itself. During startup, the kernel enumerates all the framebuffers and usually assigns fb0 and fb1 to the first device (respectively the background and the foreground framebuffer). Therefore, mxcfb0 is usually assigned to fb0 and fb1, mxcfb1 is usually assigned to fb2.
device	Specifies the video device -usually hdmi or ldb (LVDS display bridge).
mode	Specifies the video mode; e.g. LDB-XGA (in recent kernels, this entry is ignored for de-vice ldb and the DTS configuration is used instead)
interface	Specifies the interface pixel format, e.g. if=RGB666 or if=RGB24

Additional options are available based on the device type e.g. `fbpix=BGR32` specifies the framebuffer layout.

Find additional information about framebuffer configuration in the documentation of the kernel source, i.e. in file Documentation/devicetree/bindings/fb/fsl_ipuv3_fb.txt

Examples

The typical kernel command-line configuration for a 1920x1080 full HD display connected via HDMI is:

```
video=mxcfb0:dev=hdmi,1920x1080M@60,if=RGB24
```

In u-boot, set the environment variable `vid_dev0` as follows:

```
setenv vid_dev0 hdmi,1920x1080M@60,if=RGB24
```

Please ensure, that `${vid_dev0}` is correctly referenced at the kernel command-line, e.g.:

```
[…] video=mxcfb0:dev=${vid_dev0} […]
```

The typical kernel command-line configuration for a 1024x768 18-bit XGA display connected via the LVDS display bridge is:

```
video=mxcfb1:dev=ldb,LDB-XGA,if=RGB666
```

In u-boot, set the environment variable `vid_dev1` as follows:

```
setenv vid_dev1 ldb,LDB-XGA,if=RGB666
```

Please ensure, that `${vid_dev1}` is correctly referenced at the kernel command-line, e.g.:

```
[…] video=mxcfb1:dev=${vid_dev1} […]
```

The examples mentioned before describe the default configuration of the bootloader: framebuffer device 0 is assigned to a 1920x1080 full HD device connected via hdmi; framebuffer device 1 is assigned to a 1024x768 18-bit LVDS display connected via the LVDS display bridge.

6.3.7 Boot Loader Types

There are three distinct bootloader types:

- Standard bootloader (SPL build)
- HAB (High Assurance Boot) enabled bootloader (non-SPL build)
- MFG (Manufacturing) bootloader (non-SPL build)

Beginning with u-boot 2016.01, the term "standard bootloader" refers to unified SPL-enabled bootloader builds. Such a SPL-enabled bootloader, is suitable for all the module variants of one product group, due to integrated memory detection and calibration routines. The HAB and MFG bootloaders are non-SPL builds. A non-SPL bootloader is unable to detect and calibrate the memory by itself. Therefore, the source code contains different defconfig files for each distinct memory configuration – please pick the one fitting to your particular module.

6.3.8 Build Process

The following steps describe the build process of a standard bootloader using the standalone cross-development environment (see section 2.3 "Setting Up the Standalone Cross-Development Environment").

In the build process, the following configuration targets (standard bootloader, SPL build) are defined:

Defconfig File	Description
cgtqmx6eval_defconfig	Configuration for a conga-QMX6 standard bootloader
cgtumx6_defconfig	Configuration for a conga-UMX6 standard bootloader

Execute the following steps to build the standard bootloader for conga-QMX6 (all variants) :

1. Fetch the source code and switch to correct branch:

```
~$ git clone https://git.congatec.com/arm/qmx6_uboot.git
~$ cd qmx6_uboot
~/qmx6_uboot$ git checkout -b cgt_imx_v2016.01_1.0.0 origin/cgt_imx_v2016.01_1.0.0
```

2. Set up the build environment:

```
~/qmx6_uboot$ source ~/sourceme
```

Note

Follow section 2.3 "Setting Up the Standalone Cross-Development Environment" *to set up your build environment and the stated sourceme file.*

3. Select the matching configuration and build the bootloader. This is an example for conga-QMX6:

   ```
   ~/qmx6_uboot$ make cgtqmx6eval_defconfig
   ~/qmx6_uboot$ make
   ```

   The bootloader binaries (SPL and u-boot.img) will be created in the build directory.

4. Flash the bootloader to the target system with the NXP MFGTool.

7 Falcon Mode (u-boot)

7.1 Overview

A standard u-boot build consists of two image files: u-boot.img and SPL. The bootrom is loading the SPL image which performs some basic/initial configuraion tasks. Afterwards, SPL loads the u-boot.img which loads device-tree and kernel image files. Falcon mode means, enabling SPL to load/execute the kernel image directly. This accelerates boot time but requires special u-boot configuration as well as a special SD card/eMMC setup. congatec's falcon mode implementation enables SPL to:

- Load the kernel image directly from an arbitrary MMC device (micro-SD, SD, EMMC)
- Select the boot target (u-boot.img/kernel) depending on GPIO-level or environment-settings
- Load the bootloader image file (u-boot.img) from an arbitrary MMC device. Please note, the environment still has to be stored at the SPI-flash.

The following sections describe the necessary tasks to create a falcon mode enabled SPL image and show how to perform a boot device setup.

7.2 Requirements

A Linux-based x86 host system with working cross-compiler setup is required - refer to section 2 "Host System Setup".

7.3 Setting Up the Bootloader

congatec provides falcon mode enabled u-boot sources (branch cgt_imx_v2016.01_1.0.0) for all i.MX6 based designs (conga-QMX6/UMX6), starting with u-boot version 2016.01, commit 2a24305.

7.3.1 Downloading Sources

Clone the latest u-boot 2016.01 sources (commit 2a24305 or newer) from the congatec public git server:

```
$ cd /PATH/TO/YOUR/PLAYGROUND
$ git clone https://git.congatec.com/arm/qmx6_uboot.git
# [...]
$ cd qmx6_uboot
$ git checkout remotes/origin/cgt_imx_v2016.01_1.0.0 -b cgt_imx_v2016.01_1.0.0
```

7.3.2 Configuration

1. Select the matching basic default configuration (defconfig) depending on the module-type in use:

```
# If target system is conga-QMX6:
$ make cgtqmx6eval_defconfig
# If target system is conga-UMX6:
$ make cgtumx6_defconfig
```

2. In order to enable falcon mode, special configuration is required at compile-time. The congatec falcon mode implementation provides the essential configuration options via Kconfig:

```
$ cd /PATH/TO/YOUR/PLAYGROUND/qmx6_uboot
$ make menuconfig
```

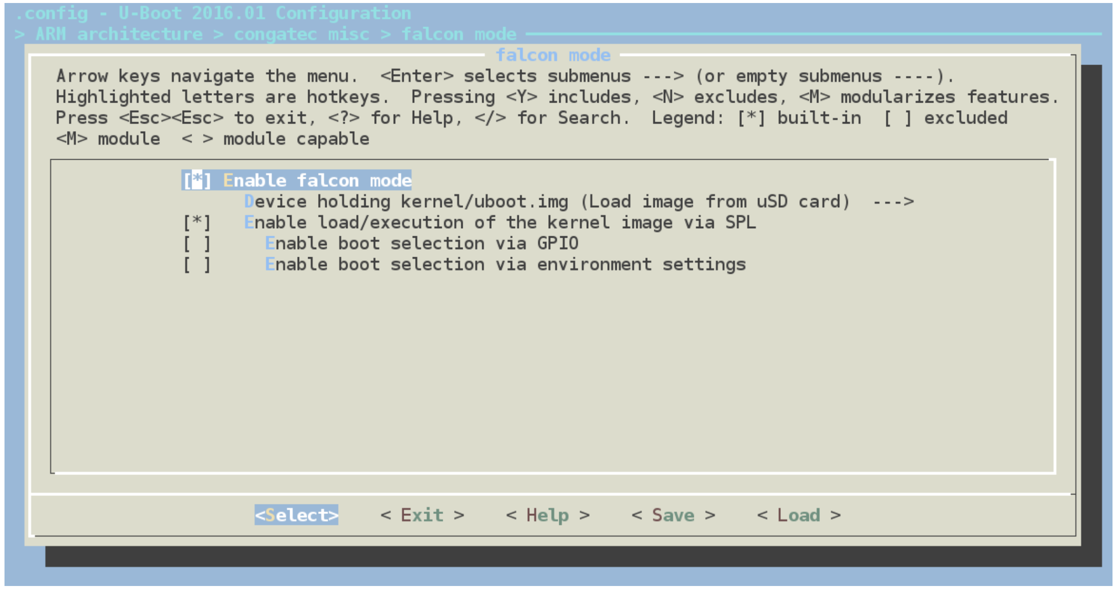
3. Navigate to the congatec falcon mode submenu and perform the configuration as follows:

```
ARM architecture
    -> congatec misc
        -> falcon mode
```

As mentioned before, the main use-case of the congatec falcon mode implementation is reducing boot time. Furthermore, the falcon mode implementation enables to store (load) the u-boot.img image file at (from) an arbitrary MMC device.

Use-Case I) Load Kernel Image Directly (Quickboot)

A minimal configuration, enabling SPL to directly load the Linux kernel image from a distinct MMC device, is shown below:



```
.config - U-Boot 2016.01 Configuration
> ARM architecture > congatec misc > falcon mode
                                falcon mode
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.
  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded
  <M> module  < > module capable

            [*] Enable falcon mode
                  Device holding kernel/uboot.img (Load image from uSD card)  --->
            [*]   Enable load/execution of the kernel image via SPL
            [ ]     Enable boot selection via GPIO
            [ ]     Enable boot selection via environment settings
```

Note

Falcon mode extends/changes boot order from SPI (1) to MMC (1), SPI (2).

If "Enable load/execution of the kernel image via SPL" is set, each device is searched for the kernel image first, if there is no kernel image, they are searched for the bootloader image (u-boot.img) afterwards.

The configuration options:

```
[ ]     Enable boot selection via GPIO
[ ]     Enable boot selection via environment settings
```

are optional but recommended.

The former configuration option enables selection of the boot image (kernel/u-boot.img) via GPIO. By default, this is mapped to the LID button; the mapping is adaptable via the BOOT_MODE_BTN C-preprocessor constant.

The latter enables boot image selection via the `boot_os` environment variable. If `boot_os` is set, the kernel image is directly loaded by SPL.

Both configuration options are combinable.

Note

If boot_os is set to 1, SPL always loads the kernel image directly. There are two ways to get back to the u-boot command prompt:

- *Ejecting MMC:*

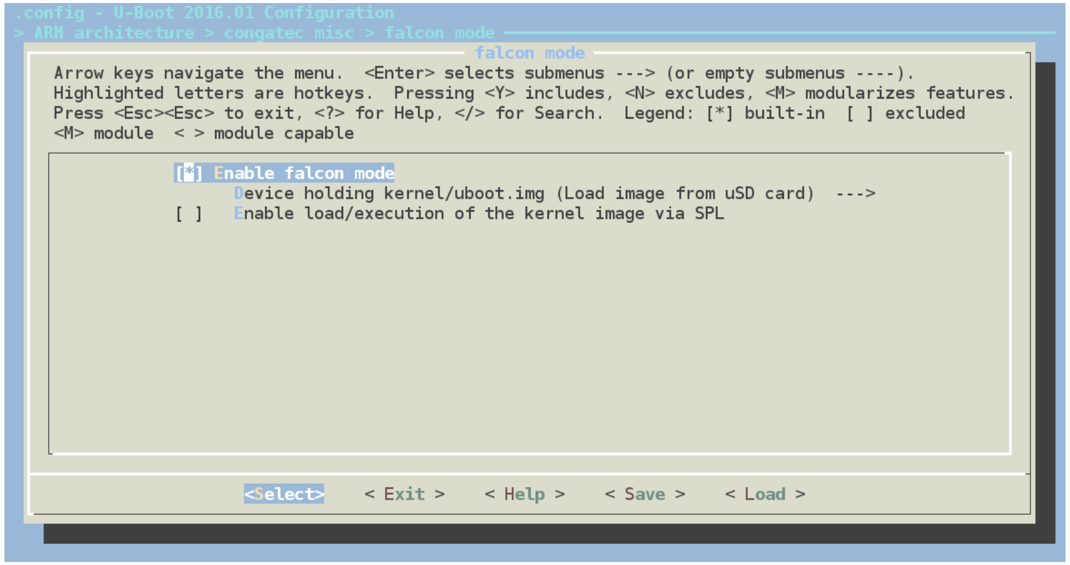
  If there is no valid kernel image at the specified MMC device or there is no mmc device, u-boot.img is loaded as a fallback.

- *GPIO override:*

  If "Enable boot selection via GPIO" is also enabled, loading u-boot.img can be forced via GPIO.

Use-Case II) Load Bootloader Image (u-boot.img) from MMC device

In order to enable SPL to load the bootloader image (u-boot.img) from an MMC device instead of SPI-flash, perform the u-boot configuration as shown below:

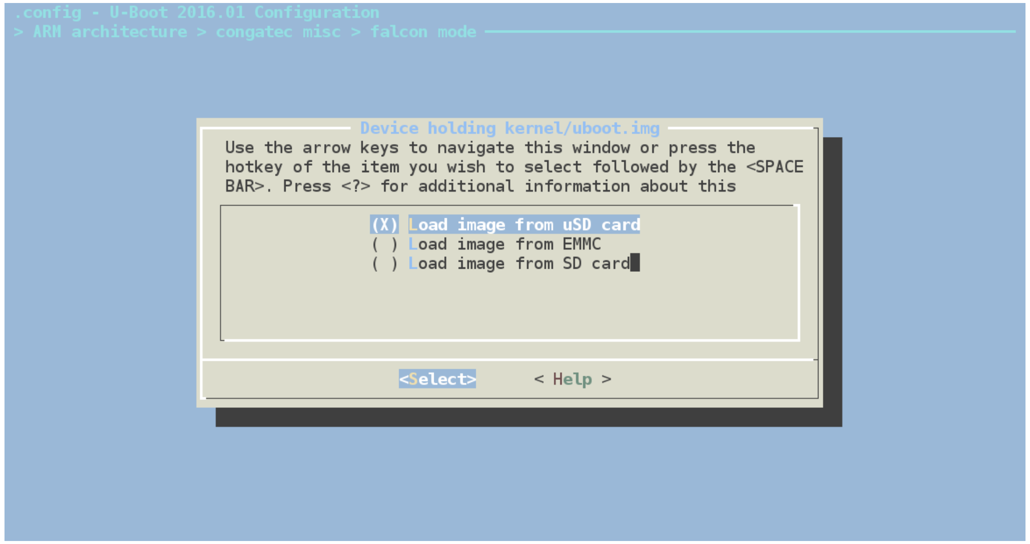


```
.config - U-Boot 2016.01 Configuration
> ARM architecture > congatec misc > falcon mode ─────────────────────────────
                              falcon mode
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.
  Press <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded
  <M> module  < > module capable

            [*] Enable falcon mode
                Device holding kernel/uboot.img (Load image from uSD card)  --->
            [ ]   Enable load/execution of the kernel image via SPL

                <Select>    < Exit >    < Help >    < Save >    < Load >
```

Note

The u-boot environment is still stored at the SPI-flash; the current implementation does not enable to store the u-boot environment at a different device. Falcon mode extends/changes boot order from SPI (1) to MMC (1), SPI (2). If "Enable load/execution of the kernel image via SPL" is unset, SPL tries to load the u-boot.img from the selected MMC device first. If there is no u-boot.img, SPL tries to load the u-boot.img from the SPI-flash.

4. Select the desired boot device by entering the "Device holding kernel/u-boot.img" submenu:



```
.config - U-Boot 2016.01 Configuration
> ARM architecture > congatec misc > falcon mode ─────────────────────────────

                 Device holding kernel/uboot.img
  Use the arrow keys to navigate this window or press the
  hotkey of the item you wish to select followed by the <SPACE
  BAR>. Press <?> for additional information about this

               (X) Load image from uSD card
               ( ) Load image from EMMC
               ( ) Load image from SD card

                   <Select>      < Help >
```

5. Finally, save and exit the graphical configuration.

Note

The graphical configuration provides extensive built-in help messages for each falcon mode related configuration option.

7.3.3 Build Falcon Mode Enabled Bootloader

Execute the following command in order to create the falcon mode enabled u-boot.img and SPL image:

```
$ make
```

7.3.4 Update Target System

Use the MFGTool to transfer the falcon mode enabled bootloader images (SPL and u-boot.img) to the i.MX6 based target system.

In order to flash a custom bootloader, proceed as follows:

1. Copy both image files (SPL and u-boot.img) to Profiles\<module-type>\OS Firmware\update.devel

2. Adapt MFGTool's configuration file (cfg.ini) as follows:

```
name = uboot2016-devel-SPL_SPI
DEVEL2016_SPL_FILENAME = SPL
DEVEL2016_UBOOT_FILENAME = u-boot.img
```

Note

The section 2.3 "Setting Up the Standalone Cross-Development Environment" *describes the usage of the MFGTool in general.*

7.4 Setting Up Boot Device

The congatec u-boot falcon mode implementation supports to load the kernel image file from eMMC, SD card and micro-SD card. This needs a special boot device setup. The following sections describe the necessary steps to create a boot medium accomplishing the falcon mode requirements.

7.4.1 Use-Case I: Boot Kernel Image Directly (Quickboot)

If the u-boot config option "Enable load/execution of the kernel image via SPL" is set, the SPL tries to load a kernel image from the selected MMC device.

7.4.1.1 Setting Up Partition/Filesystem

SPL expects the following files at defined offsets of the selected MMC device:

File	Offset in sectors of 512 bytes (KiB)	Max. size in sectors of 512 bytes (KiB)	Notes
Argument File	0x800 (1024 KiB)	0x800 (1024 KiB)	Kind of u-boot internal device tree representation created with help of the u-boot command `spl export`
Kernel Image (uImage)	0x1000 (2048 KiB)	0x4000 (8192 KiB)	Actual maximum depends on offset of the first partition

In falcon mode, kernel image and argument file have to be written to the raw MMC device at fix addresses instead of putting them into a filesystem. As a result, the start offset of the first partition has to be moved. congatec recommends to move the start offset to byte 32768.

Option I) Boot from SD Card / micro-SD Card

1. Create a new partition layout at the micro-SD card as shown below:

```
# (1) Start fdisk;
$ sudo fdisk /dev/sdX
# (2) Create a new dos partition table
Command (m for help): o <ENTER>
# (3) Add a new partition
Command (m for help): n <ENTER>
# (4) Select primary partition
Command (m for help): p <ENTER>
# (5) Enter partition number
Command (m for help): 1 <ENTER>
# (6) Enter start sector offset (in sectors of 1 * 512 = 512 bytes)
Command (m for help): 32768 <ENTER>
# (7) Enter end sector (in sectors of 1 * 512 = 512 bytes)
Command (m for help): <ENTER>
# (8) Write the partition table to device
Command (m for help): w <ENTER>
# (9) Quit fdisk
Command (m for help): q <ENTER>
# (10) Create filesystem for rootfs at partition 1
$ sudo mkfs.ext3 /dev/sdX
```

Caution

Adapt sdX / sdX1 to whatever your device / partition is recognized as. Otherwise, loss of data may occur.

2. Decompress the root filesystem image to the recently created partition:

```
$ sudo mkdir -p /mnt/imgprep
$ sudo mount /dev/sdX1 /mnt/imgprep
$ sudo tar -xvjf your_yocto_image.tar.bz2 -C /mnt/imgprep
$ sudo sync
$ sudo umount /dev/sdX1
```

Option II) Boot from eMMC

3. Set up a micro-SD card - please refer to section Option I) Boot from SD Card / micro-SD Card.

4. Transfer the tar.bz2-compressed root filesystem image to the micro-SD card:

```
# (1) Clear the first 1000 512 byte sectors of the eMMC
$ sudo dd if=/dev/zero of=/dev/mmcblk1 count=1000 bs=512
 # (2) Start fdisk
$ sudo fdisk /dev/mmcblk1
 # (3) Create a new dos partition table
Command (m for help): o <ENTER>
# (4) Add a new partition
Command (m for help): n <ENTER>
# (5) Select primary partition
Command (m for help): p <ENTER>
# (6) Enter partition number
Command (m for help): 1 <ENTER>
# (7) Enter start sector offset (in sectors of 1 * 512 = 512 bytes)
Command (m for help): 32768 <ENTER>
# (8) Enter end sector (in sectors of 1 * 512 = 512 bytes)
Command (m for help): <ENTER>
# (9) Write the partition table to device
Command (m for help): w <ENTER>
# (10) Quit fdisk
Command (m for help): q <ENTER>
# (11) Create filesystem for rootfs at partition 1
$ sudo mkfs.ext3 -j /dev/mmcblk1p1
```

5. Decompress the root filesystem image to the recently created partition at the eMMC device:

```
$ sudo mkdir -p /mnt/imgprep
$ sudo mount /dev/mmcblk1p1 /mnt/imgprep
$ cd /
$ sudo tar -xjvf your_yocto_image.tar.bz2 -C /mnt/imgprep
$ sudo sync
```

7.4.1.2 Setting Up Raw MMC Device

The following steps will finalize the setup of the boot device and activate the falcon mode.

1. Power on the i.MX6 based target system

2. Press any key to enter u-boot command prompt

3. Build up kernel command-line arguments using the following commands at the u-boot command prompt:

```
$ run findfdt
# In case of boot device: micro SD card
$ setenv mmcdev 0
$ setenv mmcroot '/dev/mmcblk0p1 ro'
# In case of boot device: eMMC
$ setenv mmcdev 1
$ setenv mmcroot '/dev/mmcblk1p1 ro'
# In case of boot device: SD card
$ setenv mmcdev 2
$ setenv mmcroot '/dev/mmcblk2p1 ro'
# X depends on boot device: 0-> micro SD card; 1->eMMC ; 2-> external SD card
mmc dev X
$ run mmcargs
$ run loadimage
$ run loadfdt
```

4. Create the argument file and copy it to the selected MMC device at sector offset 0x800

```
$ spl export fdt ${loadaddr} - ${fdt_addr_r}
## Booting kernel from Legacy Image at 12000000 ...
   Image Name:   Linux-4.1.15-1.2.0_cgt-imx6+gd83
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    2941472 Bytes = 2.8 MiB
   Load Address: 10008000
   Entry Point:  10008000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 18000000
   Booting using the fdt blob at 0x18000000
   Loading Kernel Image ... OK
   Loading Device Tree to 1fff2000, end 1ffff4b9 ... OK
subcommand not supported
subcommand not supported
   Loading Device Tree to 1ffe1000, end 1fff14b9 ... OK
$ mmc write 0x1ffe1000 0x800 0x800
```

Note

Please look for the line "`Argument image is now in RAM: [...]`":

This line mentions the memory address of the generated argument file. The named address is used as source address of the subsequent `mmc write` invocation. This writes the argument file to the selected MMC device.

5. Copy the kernel image file to the selected MMC device at sector offset 0x1000

```
$ mmc write ${loadaddr} 0x1000 0x4000
```

6. Set the `boot_os` environment variable (optional)

```
$ setenv boot_os 1
$ saveenv
```

Note

If the u-boot configuration option "Enable boot selection via environment settings" has been set, it is required to set the environment variable `boot_os`. If `boot_os` is set, SPL will directly load the kernel image from the selected MMC device. Otherwise, SPL will load the u-boot.img image file.

7.4.2 Use-Case II: Load Bootloader Image (u-boot.img) from MMC Device

If the u-boot config option "Enable load/execution of the kernel image via SPL" is not set, the SPL tries to load the bootloader image (u-boot.img) from the selected MMC device.

The u-boot.img image file has to be written to the selected MMC device at offset 69K:

```
$ cd /PATH/TO/YOUR/PLAYGROUND/qmx6_uboot
$ sudo dd if=u-boot.img of=/dev/sdX bs=1k seek=69
```

Caution

Adapt sdX / sdX1 to whatever your device / partition is recognized as. Otherwise, loss of data may occur.

8 High Assurance Boot (HAB)

8.1 Overview

The High Assurance Boot (HAB) technology is currently **only supported under u-boot 2016.01 non-SPL builds**. It represents secure boot at NXP's i.MX6 CPU-family. HAB supports a wide variety of NXP processors but this chapter is limited to the i.MX6 SOC.

HAB enables a so called chain of trust. This term describes a setup where each software involved at the boot process has to be validated using the ROM embedded HAB library. Another option is the encrypted boot.

The subject of this chapter is limited to the topic of restricted boot. Follow the steps to restrict the bootup of your design to signed u-boot bootloader images.

For encrypted boot or a gapeless chain of trust setup, refer to the official NXP documentation.

8.2 Requirements

A working cross-compiler setup is required. The following steps describe the necessary preparations in order to sign u-boot images.

8.2.1 Download/Setup

1. Obtain cst-2.3.2.tar.gz from the NXP website www.nxp.com :

   ```
   $ cp cst-2.3.2.tar.gz /PATH/TO/YOUR/HAB/PLAYGROUND/.
   $ tar -xvf cst-2.3.2.tar.gz
   ```

2. Clone the latest u-boot sources with HAB support (2016.01) from the congatec public git server:

   ```
   $ cd /PATH/TO/YOUR/HAB/PLAYGROUND
   $ git clone https://git.congatec.com/arm/qmx6_uboot.git
   # [...]
   $ cd qmx6_uboot
   $ git checkout remotes/origin/cgt_imx_v2016.01_1.0.0 -b cgt_imx_v2016.01_1.0.0
   ```

3. Clone the latest version of the cgtIVThelper.py script (python3) from the congatec public git server:

   ```
   $ cd /PATH/TO/YOUR/HAB/PLAYGROUND
   $ git clone https://git.congatec.com/arm/imx_cgtIVThelper.git cgtIVThelper
   # [...]
   ```

8.2.2 Building u-boot with HAB Support Enabled

Currently, High Assurance Boot (HAB) is only supported by u-boot 2016.01 non-SPL builds. There are dedicated non-SPL u-boot default configurations with HAB support enabled by default:

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/qmx6_uboot
$ find . -iname "*cgt*hab*defconfig"
./configs/cgtumx6_hab_1024_64_528_defconfig
./configs/cgtumx6_hab_1024_64_400_defconfig
./configs/cgtumx6_hab_1024_32_400_defconfig
./configs/cgtqmx6eval_hab_1024_32_400_defconfig
./configs/cgtqmx6eval_hab_1024_64_528_defconfig
./configs/cgtqmx6eval_hab_1024_64_400_defconfig
./configs/cgtqmx6eval_hab_2048_64_400_defconfig
./configs/cgtqmx6eval_hab_2048_64_528_defconfig
./configs/cgtqmx6eval_hab_4096_64_528_defconfig
```

Note

The various defconfig files for a given design just differ in the memory configuration (e.g. 1024_64_528 → size_width_clock → size: 1024MB, width: 64-bit, clock: 528 MHz).

Building HAB-enabled u-boot for a 1024MB 64-bit 528MHz conga-QMX6 variant (e.g. PN016103):

```
$ export ARCH=arm
$ export CROSS_COMPILE=arm-poky-linux-gnueabi-
$ # $PATH adjustment is maybe also required
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/qmx6_uboot
$ make cgtqmx6eval_hab_1024_64_528_defconfig
$ make -j4 V=1
# [...]
Image Type:   Freescale IMX Boot Image
Image Ver:    2 (i.MX53/6/7 compatible)
Data Size:    466944 Bytes = 456.00 kB = 0.45 MB
Load Address: 177ff420
Entry Point:  17800000
HAB Blocks:   177ff400 00000000 0006fc00
```

Note

Please note the line "HAB Blocks", those three values are required later on.

For more information, refer to u-boot 2016.01 Documentation:

https://git.congatec.com/arm/qmx6_uboot/blob/cgt_imx_v2016.01_1.0.0/doc/README.mxc_hab

The latest mfg defconfig files also enable HAB support by default. In order to build a signed mfg u-boot, pick the matching one of the following default configurations:

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/qmx6_uboot
$ find . -iname "*cgt*mfg*defconfig"
./configs/cgtumx6_mfg_1024_64_400_defconfig
./configs/cgtumx6_mfg_1024_32_400_defconfig
./configs/cgtumx6_mfg_1024_64_528_defconfig
./configs/cgtqmx6eval_mfg_4096_64_528_defconfig
./configs/cgtqmx6eval_mfg_1024_32_400_defconfig
./configs/cgtqmx6eval_mfg_2048_64_528_defconfig
./configs/cgtqmx6eval_mfg_2048_64_400_defconfig
./configs/cgtqmx6eval_mfg_1024_64_528_defconfig
./configs/cgtqmx6eval_mfg_1024_64_400_defconfig
```

8.2.3 Setting Up Public Key Infrastructure (PKI)

1. Generate the PKI tree (CA, SRKs and certificates/keys) as follows:

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2
$ cd keys
$ ./hab4_pki_tree.sh
    +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    This script is a part of the Code signing tools for Freescale's
    High Assurance Boot.  It generates a basic PKI tree.  The PKI
    tree consists of one or more Super Root Keys (SRK), with each
    SRK having two subordinate keys:
        + a Command Sequence File (CSF) key
        + Image key.
    Additional keys can be added to the PKI tree but a separate
    script is available for this.  This this script assumes openssl
    is installed on your system and is included in your search
    path.  Finally, the private keys generated are password
    protectedwith the password provided by the file key_pass.txt.
    The format of the file is the password repeated twice:
        my_password
        my_password
    All private keys in the PKI tree are in PKCS #8 format will be
    protected by the same password.

    +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    Do you want to use an existing CA key (y/n)?: n
```

```
        Do you want to use Elliptic Curve Cryptography (y/n)?: n
        Enter key length in bits for PKI tree: 2048
        Enter PKI tree duration (years): 10
        How many Super Root Keys should be generated? 4
        Do you want the SRK certificates to have the CA flag set? (y/n)?: y
# [...]
```

For more information, refer to NXP AN4581 Rev. 1, 10/2015.

2. Generate the SRK table and SRK hash table:

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2
$ cd crts
$ ../linux64/srktool -h 4 -t SRK_1_2_3_4_table.bin -e SRK_1_2_3_4_fuse.bin -d sha256 -c ./SRK1_sha256_2048_65537_v3_ca_crt.pem,./SRK2_
sha256_2048_65537_v3_ca_crt.pem,./SRK3_sha256_2048_65537_v3_ca_crt.pem,./SRK4_sha256_2048_65537_v3_ca_crt.pem -f 1
```

Note

- *The SRK table will be part of the signature (binary CSF file)*
- *The padded binary CSF file will be appended to the (unsigned) u-boot image file*

  --> A signed u-boot image is simply a concatenation of an unsigned u-boot image and the padded binary CSF file
- *The SRK hash table contains the hash of the SRK table, which will be written to dedicated OTP registers (SRK0..SRK7, see section 8.3.2.1 "Fuse Overview")*

8.3 Secure Boot: Restricted Execution (Signed Bootloader)

NXP's High Assurance Boot (HAB) implementation enables a secure boot chain at the i.MX6 CPU-family. This section's main subject is to show the process of signing a given binary u-boot image file. Such a signed u-boot image file is suited for restricted execution. Restricted execution means that the execution of arbitrary bootloader software is prevented.

Encrypted boot (encryption of a given u-boot image file) is not subject of this section.

8.3.1 Signing Bootloader Image (u-Boot 2016.01, non-SPL)

The following section describes how to sign a given binary u-boot image file useable for restricted execution (secure boot) at NXP's i.MX6 CPU-family. There is a differentiation between the process of signing of normal u-boot image files (usually stored at SPI NOR-flash) and the signing of so called mfg u-boot image files, which are used for bootloader updates using the NXP MFGTool2.

8.3.1.1 Preparation

1. Set up an image specific signing area:

```
$ mkdir /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64/uboot-signing-area
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64/uboot-signing-area
```

2. Copy u-boot image file to sign to the previously created signing area:

```
$ cp -avr /PATH/TO/YOUR/UBOOT/SOURCES/qmx6_uboot/u-boot.imx ./u-boot.unsigned.imx
```

Note

Use separate signing areas for differing u-boot image files

3. Copy the cgtIVThelper.py script to the previously created signing area:

```
$ cp -avr /PATH/TO/YOUR/HAB/PLAYGROUND/cgtIVThelper/cgtIVThelper.py /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64/uboot-signing-area/.
```

Note

Alternatively, add /PATH/TO/YOUR/HAB/PLAYGROUND/cgtIVThelper/ to your $PATH environment variable

8.3.1.2 Signing

This section describes the signing of u-boot image files - it is differentiated between normal u-boot images and special mfg u-boot images, which are used for initial bootstrap in Serial Downloader Mode (MFGTool).

u-boot (non-SPL build)

1. Obtain the habblocks data from the original u-boot image file using cgtIVThelper.py (required later on):

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64/uboot-signing-area
$ python3 cgtIVThelper.py -f ./u-boot.unsigned.imx --get-habblocks
start-signing address : 0x177ff400
start-signing offset  : 0x0
signed-data length    : 0x6ec00
```

2. Create textual CSF description file cgt-qmx6-umx6-sample.csf and fill it with the content shown below:

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64/uboot-signing-area
$ nano cgt-qmx6-umx6-sample.csf
```

Note

The placeholders <start-signing address>, <start-signing offset> and <signed-data length> have to be changed to appropriate values obtained by using cgtIVThelper.py as shown in subitem 1.

```
# cgt-qmx6-umx6-sample.csf
[Header]
Version = 4.1
Hash Algorithm = sha256
Engine = ANY
Engine Configuration = 0
Certificate Format = X509
Signature Format = CMS

[Install SRK]
File = "../crts/SRK_1_2_3_4_table.bin"
Source index = 0          # Index of the key location in the SRK table to be installed

[Install CSFK]
# Key used to authenticate the CSF data
File = "../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"

[Authenticate CSF]

[Unlock]
Engine = CAAM
Features = RNG

[Install Key]
# Key slot index used to authenticate the key to be installed
Verification index = 0
# Target key slot in HAB key store where key will be installed
Target Index = 2
# Key to install
File= "../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"

[Authenticate Data]
# Key slot index used to authenticate the image data
Verification index = 2
#        Address                   Offset                  Length                    Data File Path
Blocks = <start-signing address>   <start-signing offset>  <signed-data length>      "/PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64/uboot-
signing-area/u-boot.unsigned.imx"
```

3. Generate the binary CSF file cgt-qmx6-umx6-sample.csf.bin:

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64
$ ./cst -i uboot-signing-area/cgt-qmx6-umx6-sample.csf -o uboot-signing-area/cgt-qmx6-umx6-sample.csf.bin
```

4. Add the required padding to the binary CSF file:

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64/uboot-signing-area
$ objcopy -I binary -O binary --pad-to 0x2000 --gap-fill=0x00 cgt-qmx6-umx6-sample.csf.bin cgt-qmx6-umx6-sample.csf.bin.padded
```

For more information, refer to u-boot 2016.01 documentation:

https://git.congatec.com/arm/qmx6_uboot/blob/cgt_imx_v2016.01_1.0.0/doc/README.mxc_hab

5. Sign the u-boot image file:

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64/uboot-signing-area
$ cat u-boot.unsigned.imx cgt-qmx6-umx6-sample.csf.bin.padded > u-boot.signed.imx
```

Manufacturing u-boot

1. Clear the DCD pointer of a copy of the u-boot image file:

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64/uboot-signing-area
$ cp u-boot.mfg.unsigned.imx u-boot.mfg.unsigned.cleared-dcd.imx
$ python3 cgtIVThelper.py -f ./u-boot.mfg.unsigned.cleared-dcd.imx --clear-dcdptr
```

Note

If the MFGTool is used, it extracts the DCD from the binary mfg u-boot image file to initialize the external memory. The external memory must not be initialized twice - that is the reason why the MFGTool modifies the u-boot binary internally, before it transfers the image to the target i.MX6 system. The MFGTool clears the DCD pointer - that means, it sets the pointer to 00000000. Therefore, the creation of the signature (binary CSF file) used for signing a mfg u-boot image file must reference a u-boot image file with cleared DCD pointer.

2 a) Obtain the habblocks data from the untouched u-boot image file using cgtIVThelper.py (required later on):

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64/uboot-signing-area
$ python3 cgtIVThelper.py -f ./u-boot.mfg.unsigned.imx --get-habblocks
start-signing address : 0x177ff400
start-signing offset  : 0x0
signed-data length    : 0x6ec00
```

Note

The habblocks data can also be obtained from the build process by simply passing V=1 to `make`.

2 b) Obtain the dcd-habblocks data from the untouched u-boot image file using cgtIVThelper.py (required later on):

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64/uboot-signing-area
$ python3 cgtIVThelper.py -f ./u-boot.mfg.unsigned.imx --get-dcd-habblocks
# DCD OFFSET | DCD LENGTH
0x0000002c     0x02f8
```

2 c) Create the textual CSF description file cgt-qmx6-umx6-mfg-sample.csf and fill it with the content shown below:

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64/uboot-signing-area
$ nano cgt-qmx6-umx6-mfg-sample.csf
```

Note

The placeholders <start-signing address>, <start-signing offset>, <signed-data length>, <dcd-offset> and <dcd-length> have to be substituted with the appropriate values obtained by using cgtIVThelper.py as shown in the subitems 2 a) and 2 b)

```
# cgt-qmx6-umx6-mfg-sample.csf
[Header]
Version = 4.1
Hash Algorithm = sha256
Engine = ANY
Engine Configuration = 0
Certificate Format = X509
Signature Format = CMS

[Install SRK]
File = "../crts/SRK_1_2_3_4_table.bin"
Source index = 0          # Index of the key location in the SRK table to be installed

[Install CSFK]
# Key used to authenticate the CSF data
File = "../crts/CSF1_1_sha256_2048_65537_v3_usr_crt.pem"

[Authenticate CSF]

[Unlock]
Engine = CAAM
Features = RNG

[Install Key]
# Key slot index used to authenticate the key to be installed
Verification index = 0
# Target key slot in HAB key store where key will be installed
Target Index = 2
# Key to install
File= "../crts/IMG1_1_sha256_2048_65537_v3_usr_crt.pem"

[Authenticate Data]
# Key slot index used to authenticate the image data
```

```
Verification index = 2
#         Address                  Offset                  Length                    Data File Path
Blocks  = <start-signing address> <start-signing offset> <signed-data length>      "/PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64/uboot-
signing-area/u-boot.mfg.unsigned.cleared-dcd.imx", \
0x00910000 <dcd-offset> <dcd-length> "/PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64/uboot-signing-area/u-boot.mfg.unsigned.cleared-dcd.imx"
```

3. Generate the binary CSF file cgt-qmx6-umx6-mfg-sample.csf.bin:

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64
$ ./cst -i uboot-signing-area/cgt-qmx6-umx6-mfg-sample.csf -o uboot-signing-area/cgt-qmx6-umx6-mfg-sample.csf.bin
```

4. Add the required padding to the binary CSF file:

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64/uboot-signing-area
$ objcopy -I binary -O binary --pad-to 0x2000 --gap-fill=0x00 cgt-qmx6-umx6-mfg-sample.csf.bin cgt-qmx6-umx6-mfg-sample.csf.bin.padded
```

For more information, refer to u-boot 2016.01 Documentation:

https://git.congatec.com/arm/qmx6_uboot/blob/cgt_imx_v2016.01_1.0.0/doc/README.mxc_hab

5. Sign the mfg u-boot image file:

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/linux64/uboot-signing-area
$ cat u-boot.mfg.unsigned.imx cgt-qmx6-umx6-mfg-sample.csf.bin.padded > u-boot.mfg.signed.imx
```

Note

The u-boot image file with cleared dcd pointer is just used for the signing process. In order to get a working signed mfg u-boot image, the untouched u-boot image file has to be concatenated with the padded binary CSF file. Do not use the u-boot image file with cleared DCD pointer for concatenation. The MFGTool needs the DCD pointer to locate the DCD structure. The DCD pointer of the signed image will be cleared by the MFGTool before the transfer begins. This is why the signature has to be created on basis of an image file with cleared DCD pointer.

For more information, refer to NXP AN4581 Rev. 1, 10/2015, pp.18sqq.

8.3.2 SOC-Configuration

The HAB configuration requires burning some One Time Programmable (OTP) registers (fuses).

That kind of register can be burned by using:

- u-boot's fuse command. For more information, refer to u-boot 2016.01 Documentation:
  - https://git.congatec.com/arm/qmx6_uboot/blob/cgt_imx_v2016.01_1.0.0/doc/README.fuse

  - https://git.congatec.com/arm/qmx6_uboot/blob/cgt_imx_v2016.01_1.0.0/doc/README.mxc_ocotp

- MFGTool/Linux pseudo filesystem. For more information, refer to i.MX 6 Linux® High Assurance Boot (HAB) User's Guide, Document Number: IMX6HABUG, Rev. L3.14.28_1.0.0-ga, 04/2015, p.9.
- fsdp6util, etc.

In the following example, we use the fuse command from the u-boot command prompt.

Command usage information:

```
fuse prog [-y] <bank> <word> <value>
```

- [-y] write without further enquiry (optional)
- <bank> and <word> refer to the addresses/offsets given by the fusemap
- <value> value to write to the OTP register at bank <bank> and word <word>

For more information, refer to u-boot 2016.01 Documentation:

- https://git.congatec.com/arm/qmx6_uboot/blob/cgt_imx_v2016.01_1.0.0/doc/README.fuse]
- https://git.congatec.com/arm/qmx6_uboot/blob/cgt_imx_v2016.01_1.0.0/doc/README.mxc_ocotp

8.3.2.1 Fuse Overview

OTP Fuse	Fusemap Offset / Bit	Bank	Word	Write-Command
SEC_CONFIG	0x460[1]	0	6	fuse prog 0 6 0x2
DIR_BT_DIS	0x460[3]	0	6	fuse prog 0 6 0x8
SRK_LOCK	0x400[14]	0	0	fuse prog 0 0 0x4000
SRK0	0x580	3	0	depends on your personal PKI
SRK1	0x590	3	1	depends on your personal PKI
SRK2	0x5A0	3	2	depends on your personal PKI
SRK3	0x5B0	3	3	depends on your personal PKI
SRK4	0x5C0	3	4	depends on your personal PKI
SRK5	0x5D0	3	5	depends on your personal PKI
SRK6	0x5E0	3	6	depends on your personal PKI
SRK7	0x5F0	3	7	depends on your personal PKI

For more information, refer to i.MX 6Dual/6Quad Applications Processor Reference Manual, Document Number: IMX6DQRM, Rev 2, 06/2014, pp.335sqq.

8.3.2.2 Burning SRK Hashes Into SRK OTP Registers

Refer to the creation of the SRK table and the SRK hash table as shown in section 8.3.2.1 "Fuse Overview".

1. Extract the SRK hashes to write to the SRK OTP fuses from the SRK hash table:

```
$ cd /PATH/TO/YOUR/HAB/PLAYGROUND/cst-2.3.2/crts
$ hexdump -e '/4 "0x"' -e '/4 "%X""\n"' SRK_1_2_3_4_fuse.bin
0x53C78AB7
0x96DE9CFD
0x50EF24F6
0x409FCB10
0x29ED70C7
0xE9864F28
0x3FBB5AA1
0xC50B3F39
```

   For more information, refer to NXP AN4581 Rev. 1, 10/2015.

2. Burn the SRK hash table to the dedicated SRK OTP fuses:

Caution

Do not write the SRK hash values from this example to the OTP registers or the module will become useless. Only write the SRK hashes extracted from your own SRK table.

In order to burn the SRK fuses, invoke the fuse command from the u-boot command prompt as follows:

```
> fuse prog -y 3 0 0x53C78AB7
> fuse prog -y 3 1 0x96DE9CFD
> fuse prog -y 3 2 0x50EF24F6
> fuse prog -y 3 3 0x409FCB10
> fuse prog -y 3 4 0x29ED70C7
> fuse prog -y 3 5 0xE9864F28
> fuse prog -y 3 6 0x3FBB5AA1
> fuse prog -y 3 7 0xC50B3F39
```

For more information, refer to NXP AN4581 Rev. 1, 10/2015.

8.3.2.3 Verifying the Signed u-boot Image File (hab_status)

```
> hab_status

Secure boot enabled

HAB Configuration: 0xcc, HAB State: 0x99
No HAB Events Found!

>
```

Caution

If `hab_status` *throws HAB events, the u-boot image file is probably not signed correctly.*

1. *Go back and double check all performed steps:*
   - *verify the signed u-boot image*
   - *verify the burned SRK hashes*
2. *Do NOT proceed as long as* `hab_status` *throws HAB events.*

8.3.2.4 Finalizing Lock

1. Burn the SRK_LOCK bit (locking SRK OTP fuses):

In order to burn the SRK_LOCK bit, execute the following command at the u-boot command prompt:

```
> fuse prog -y 0 0 0x4000
```

2. Burn the DR_BT_DIS bit:

In order to burn the DR_BT_DIS bit execute the following command at the u-boot command prompt:

```
> fuse prog -y 0 6 0x8
```

3. Burn the SEC_CONFIG bit (enabling secure boot / preventing execution of unsigned bootloader images):

In order to burn the SEC_CONFIG bit execute the following command at the u-boot command prompt:

```
> fuse prog -y 0 6 0x8
```

Caution

Verify the signed u-boot image file by using `hab_status` *before burning SEC_CONFIG, see section 8.3.2.3 "Verifying the Signed u-boot Image File (hab_status)".*

If SEC_CONFIG is set:

- the part will only execute properly signed u-boot image files
- the default MFGTool and MFG Profiles will not work anymore
- MFGTool usage demands the creation of a special signed mfg u-boot image, see section 8.3.1 "Signing the Bootloader Image (u-boot 2016.01, non-SPL)".

Burning SEC_CONFIG is always the last step in enabling secure boot on i.MX6 devices:

```
> fuse prog -y 0 6 0x2
```

4. After reboot, check the hab_status again:

```
> hab_status

Secure boot enabled

HAB Configuration: 0xcc, HAB State: 0x99
No HAB Events Found!

>
```

8.3.3 MFGTool and Locked Modules (SEC_CONFIG burned)

1. Download the latest version of the MFGTool and perform an update to the latest MFG Profiles.

2. Create a personal copy of the matching binary mfg u-boot image file from MFGTool2\Profiles\<MODULE_TYPE>\OS Firmware\mfg

3. Sign the selected binary mfg u-boot image file as described in *8.3.1.2 "Signing (Manufacturing u-boot)"*

4. Append the "-signed" string to the signed mfg u-boot version string (filename) as exemplarily shown below:

```
$ mv /YOUR/PERSONAL/COPY/OF/u-boot_mfg_1024_64_528__mu201601r003.imx /YOUR/PERSONAL/COPY/OF/u-boot_mfg_1024_64_528__mu201601r003-signed.imx
```

5. Store a copy of u-boot_mfg_1024_64_528__mu201601r003-signed.imx at MFGTool2\Profiles\<MODULE_TYPE>\OS Firmware\mfg

6. Adapt the MFGTool2\cfg.ini configuration file as exemplarily shown below:

```
;; cfg.ini
;; [...]

;QMX6_MFG_UBOOT_VER = _mu201601r003
QMX6_MFG_UBOOT_VER = _mu201601r003-signed

;; [...]
```

9 Sources of Information

For detailed information about the i.MX6 processor and the available software board packages/tools, consult the documents listed below. These documents are available at http://www.nxp.com. A registered account is required to download some of the files.

- i.MX 6Dual/6Quad Automotive and Infotainment Applications Processors (IMXDQAEC.pdf)
- i.MX 6Dual/6Quad Applications Processors for Consumer Products (IMXDQCEC.pdf)
- i.MX 6Dual/6Quad Applications Processors for Industrial Products (IMXDQIEC.pdf)
- i.MX 6Solo/6DualLite Automotive and Infotainment Applications Processors (IMX6SDLAEC.pdf)
- i.MX 6Solo/6DualLite Applications Processors for Consumer Products(IMX6SDLCEC.pdf)
- i.MX 6Solo/6DualLite Applications Processors for Industrial Products(IMX6SDLIEC.pdf)
- i.MX 6Dual/6Quad Applications Processors Reference Manual (IMX6DQRM.pdf)
- i.MX 6Solo/6DualLite Applications Processors Reference Manual (IMX6SDLRM.pdf)
- Chip Errata for the i.MX 6Dual/6Quad (IMX6DQCE.pdf)
- Chip Errata for the i.MX 6Solo/6DualLite (IMX6SDLCE.pdf)
- i.MX6 datasheets that covers all features and electrical characteristics of the processor
- NXP community at https://community.nxp.com

9.1 Industry Specification

The list below provides links to industry specifications that apply to congatec AG modules.

Specification	Link
Qseven® Specification	http://www.qseven-standard.org/
Qseven® Design Guide	http://www.qseven-standard.org/
Low Pin Count Interface Specification, Revision 1.0 (LPC)	http://developer.intel.com/design/chipsets/industry/lpc.htm
Universal Serial Bus (USB) Specification, Revision 2.0	http://www.usb.org/home
Serial ATA Specification, Revision 1.0a	http://www.serialata.org
PCI Express Base Specification, Revision 2.0	http://www.pcisig.com/specifications
NXP website	http://www.nxp.com