XCS for NGRG2

The XCS (auXiliary Control System) is a component based platform for Radio S/W application development.

This document gives an overview of the components used and their purpose in the system.

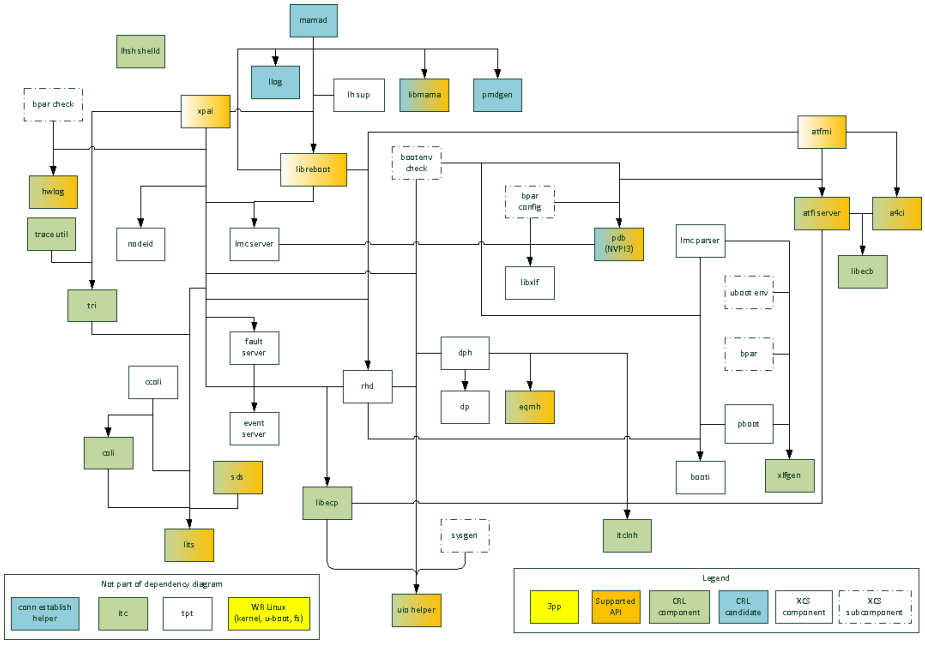

Figure 1: XCS dependency chart

Component overview

Some components are developed specifically for the radio, but many of the more complex modules are shared between the DU, XMU03 and Radio products through the CRL (Common Radio Linux) project.

The following components have publically exposed API:s available to the Radio application:

- **a4ci** Legacy interface for communicating with support units on the ecbus.
- **atfi** Legacy interface for managing links to units on antenna ports and the ecbus.
- **atfmi** Starts a4ci and atfi services depending on radio configuration.

- **eqmh** The equipment handler interface describes how to load, start, stop and dump device processors.
- **hwlog** The hardware log is kept in flash, and this daemon manages access for reading and writing to this log. The design rule 39/102 60-CSX 101 09 written for the DU originally is followed.
- **libmama** Mama handles the starting and monitoring of Linux processes.
- **libreboot** This library handles rebooting the board, either managed or
- **lits** The LITS module allows OSE applications to compile and run on Linux.
- **pdb** The parameter database lets the user read and write board parameters.
- **sds** An addition to LITS supporting additional OSE functionality.
- **XPAI** is the legacy XPP interface for the application. It wraps many of the component functionality for the application. For details see the IWD 1/155 19-CEH 101 90/1. immediately.

The rest of the components are considered implementation details of XCS:

- **bootenv check** This application makes sure the boot environment is sane during boot.
- **booti** This interface describes the reset register, describing reset cause and boot instructions to the boot loaders.
- **bpar** This is an example board parameter file used for XCS validation and development.
- **bpar check** This application makes sure mandatory board parameters are available during boot.
- **bpar config** This daemon configures the databases and provides access to them from flash.
- **ccoli** A front end for coli commands.
- **colid** A daemon for legacy OSE coli command support.
- **dp** This component contains cortex m4 code which is the device processor side of dph.
- **dph** This daemon implements the EQMH interface for managing the device processors (arm cortex m4:s) on Xenon.
- **dtb** This component is the XCS contribution to the system level dtb-file describing the hardware to the Linux kernel.
- **event server** A daemon supporting legacy event handling.
- **fault server** A daemon supporting legacy fault handling.
- **itc** This is an implementation of the OSE signaling mechanism on Linux.
- **itclnh** This component implements the standard link handler protocols RLNH and HDLC using ITC.
- **libecb** Library for accessing the ecbus and antenna ports.
- **lh sup** The link supervision daemon, making sure the Radio has a connection to the DU.

- **lhsh shelld** This daemon accepts connections over a link, and is used to communicate with the DU.
- **libxlf** This library provides functions for parsing xlf archives (LMCs).
- **llog** The logs crashes and restarts.
- **lmc parser** This component is called by u-boot to parse the lmc:s in flash and select one for boot.
- **lmc server** A daemon managing the LMC:s in flash, validating existing LMCs and storing new LMCs.
- **mamad** The mama daemon wraps the mama library and implements the callbacks required. Libmama makes the decisions, but mamad decides what to do when a decision is made.
- **nodeid** This daemon manages reading and writing the nodeid of the Radio.
- **pboot** This is the primary boot loader, selecting which bootimage and uboot enviroment to use while booting. It also loads them from flash and continues the boot process.
- **pmdgen** This hooks up in to Linux core dump generation, and generates a post mortem dump of a crashed process.
- **rhd** This component contains hardware drivers.
- **sysgen** This application generates run-time SYS environment variables during boot.
- **trace util** This component wraps the general linux tracing called lttng in legacy compatible commands such as "te log".
- **tri** This module implements the legacy tracing behavior on top of lttng.
- **uboot env* This is an example u-boot environment used during XCS validation and development.
- **uio helper** A utility library for managing hardware interrupts from user space in Linux.
- **xlfgen** This host application generates xlf archives (LMCs).

Boot sequence

The following diagram gives an overview of the boot process:

ROM boot loader

This component is not part of XCS, but is stored in ROM as part of the ASIC. In the flash boot mode, which we use, it downloads the XLF file stored in the beginning of flash to SRAM and executes it. In NGRG2 the XLF stored first in flash is Pboot.

On Xenon, the second core is already running during this stage. It is trapped in an initite loop waiting for an event until it gets a trampoline address to jump to. The second core is released when the ROM boot loader exits.

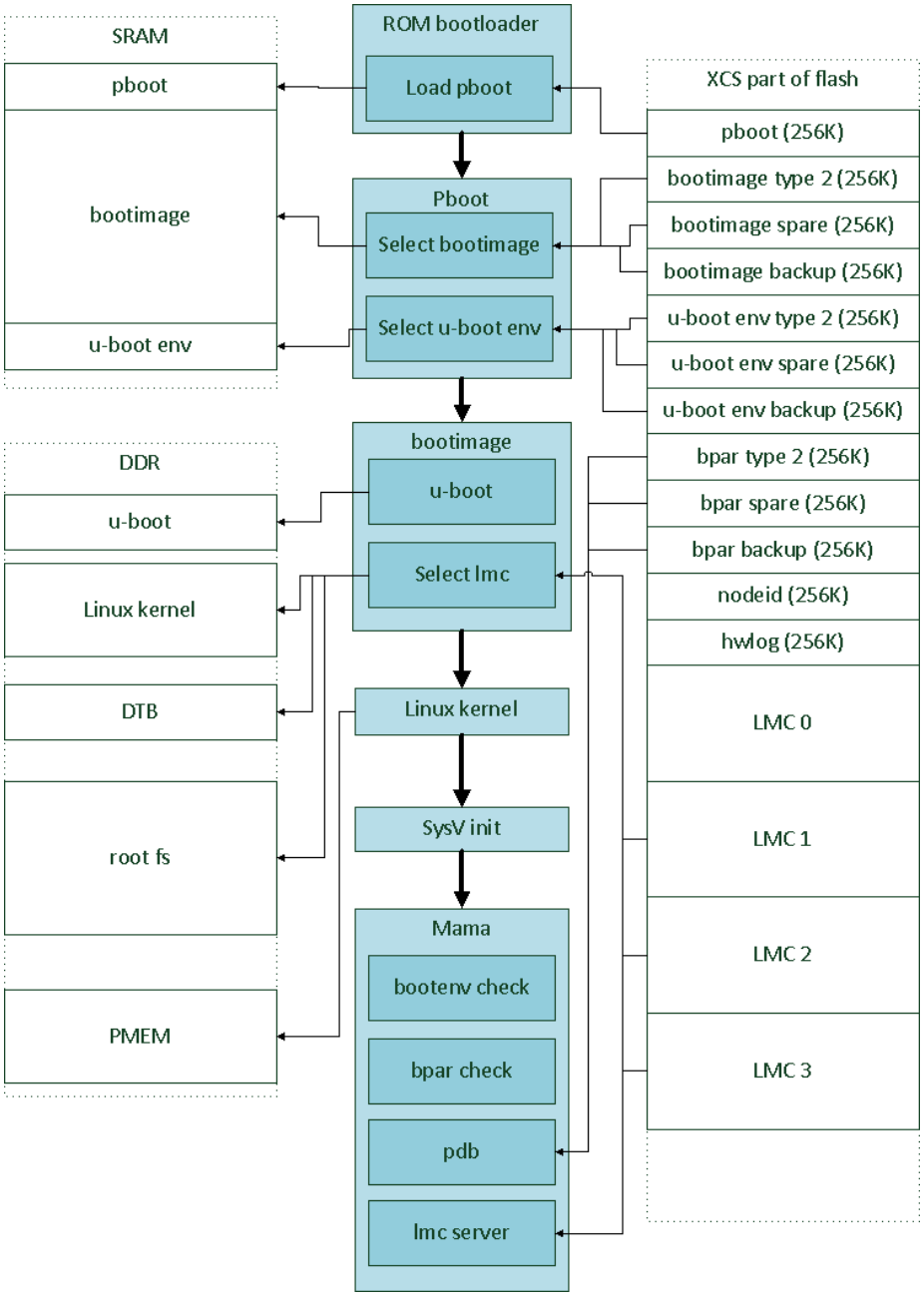

Figure 2: XCS boot sequence

Pboot

The purpose of pboot is to select which bootimage and u-boot environment to boot. This is necessary since XCS supports upgrades of both of these components.

It will also make sure that the u-boot environment chosen is compatible with the bootimage chosen.

On Xenon, the second core is running at this stage. Similar to the ROM boot, the second core is kept in an infinite loop during the Pboot execution, and released when it exits.

Bootimage

The bootimage has 2 components: u-boot and the lmc parser. The reason for dividing this in to 2 components is because of licensing. U-boot is under the GPL license, and parsing our LMC format in u-boot would expose this to the GPL.

U-boot sets up DDR and relocates itself from SRAM to DDR early during it's initialization. U-boot uses an environment kept in RAM for configuration, and has some basic scripting capabilities where variables in the environment can be executed as scripts.

In XCS there are 4 parts which add to the u-boot environment:

1. U-boot iself creates variables. It exports 2 additional variables in addition to the standard ones: the flash size and flash sector size. The lmc parser needs this information, which is why it is exported once detected.
2. The Radio application supplies a u-boot environment which configures XCS in general, not u-boot or Linux. See the table in section 5.2.1 of the XCS IP.
3. The bootimage has an internal u-boot environment which is appended after the previous 2 configurations, overriding any duplicates. This is to make sure that system cricial variables such as boot command and removal of the option to interrupt u-boot is removed in production. # The lmc parser generates variables for the Linux kernel command line and appends to it in the u-boot environment.

After sourcing the environment it copies the whole flash to RAM before launching the LMC parser. The reason for copying the whole flash is that the low level flash API in u-boot is not exposed to lmc parser. Because the lmc parser cannot access flash, it operates on RAM, and since it doesn't know which LMC to pick or the boundaries of these partitions, the whole flash is copied.

The lmc parser selects which LMC to boot and validates it before returning to u-boot which in turn boots Linux. The algorithm for selecting an LMC is not described here, but has not changed from legacy platforms.

On Xenon, the second core is running at this stage. Similar to the ROM boot, the second core is kept in an infinite loop during the Pboot execution, but it is *not* released when it exits. Instead the trampoline address is an interface between u-boot and the Linux kernel, as the Linux kernel requires the second core not to be running when it boots.

Linux kernel

The Linux kernel initializes itself and the device drivers it controls. Early on it enables the second core and adds it in SMP (Symmetric Multi Processor) mode, where the usage of more than 1 CPU core is transparent to the applications (user space). When initialization is complete, the kernel starts */sbin/init* as pid 1, which in our case is the SysV init.

SysV init

The SysV init is quite simple to understand. It executes all the scripts in a folder associated with the chosen runlevel in alphabetical order.

There are 6 runlevels, and the ones we care about are 3, 4 and 6.

Runlevel 3 is our default runlevel which launches the standard Linux daemons and then Mama, which in turn launches XCS and eventually RICR and the Radio Application.

Runlevel 4 is our *safe mode* and launching the same services as runlevel 3, except that it doesn't launch Mama. This runlevel allows you to bring up Linux without XCS, RICR or the Radio Application, which is useful for trouble shooting.

Runlevel 6 gets invoked when the system shuts down, e.g. using the *reboot* command. It is currently not invoked during our managed restarts, as it is unclear if runlevel 6 is guaranteed to terminate or if it can hang.

The scripts launched can be found under */etc/init.d/* on the target file system. Logical links are then created to each runlevel, allowing sharing of scripts between runlevels. Each script is expected to take one parameters which can have 3 values which are self describing: *start*, *stop* and *restart*. The *stop* command is used for runlevel 6, for the other runlevels *start* is passed to the script. The links determining which scripts are invoked for each runlevel is available under */etc/rc.d/* on the target file system.

A couple of the scripts executed sets up environment variables before launching Mama. In a POSIX system environment variables are inherited by the children, and for Mama to be able to pass on the environment to it's children they need to be available when Mama launches.

Mama

SysV init is sufficient for launching applications, but is lacking some features needed by a control system. There has been other attemps at solving this problem, but this is why we created Mama:

1. **Process supervision**. In a POSIX system, the parent always gets notified if a child dies (SIGCHLD). When SysV init is used, the parent (the script) dies, in which case the parent PID is reassigned to be PID 1 (*/sbin/init*), but XCS loses control over it. With Mama, XCS keeps track of all processes it launches.

2. **Easier process hang supervision**. Detecting whether or not a process hangs requires the process to periodically signal a central instance (server) to show that it is operating properly. Other attemps have used e.g. ITC to periodically deliver signals to a server monitoring processes, but this relies on ITC itself not hanging and requires more code and complexity in the client (hunt, monitor etc.). Mama takes advantage of the fact that each child has access to it's parent's pid. Therefore a simpler solution of sending a Linux real-time signal to the parent regularly can be used, relying on nothing but the Linux kernel features to work.

3. **Relaunch support**. With the increasingly aggressive requirements on restart times for auxiliary units, it may be advantageous to not restart the whole unit when an application crashes. This feature allows you to configure a child to be restarted if/when it crashes.

4. **Domain support**. There may be instances where it is advantageous to relaunch several children which have interdependencies. Mama supports a domain concept where a domain can be configured how many times it should be restarted, how long that is allowed to take etc. In this case, several processes (children) are terminated and then restarted as a group (domain).

5. **Faster boot**. SysV init is quite slow at launching processes. It invokes a shell everytime a script is started (which in turn launches the process), and even though the busybox shell is much (much) smaller than bash on a Linux desktop, it is still quite noticeable on a system launching many processes such as XCS.

6. **Deterministic boot**. SysV init launches scripts "as fast as it can", which admittedly can be slow as described above, but it is not deterministic. This means for example that if A is launched before B, and B requires A to make a mailbox available some time during it's initialization, SysV init cannot guarantee this. There is no mechanism for saying "my initialization is complete, processes depending on me can now launch". Mama solves this with a synchronization feature, which has to be explicitly turned on because it slows down the boot.

7. **Simple design**. Mama uses nothing from the rest of XCS, it depends only on the Linux kernel and C library. It is a single thread, servicing one event at a time. Since Mama is the single point of failure in the system (if Mama crashes, the board resets. But if it hangs, there is no recourse), it is important for it to be as simple and robust as possible.

Mama launches all of XCS, followed by RICR and lastly the Radio Application. Some are called out in the diagram because of their effect on boot or flash accesses:

- **bootenv_check** This component validates that the environment variables passed from u-boot are as supported, that the mtd partitions meets the requirements. It's a sanity check of the system during boot, needed because XCS doesn't control the end product and delivery.
- **pdb** The parameter database daemon selects which partition to use (algorithm not described here) and makes the parameters available to the rest of the system.
- **lmc server** This component scans the LMC partitions, removing corrupt LMCs and exposing APIs to the rest of the system for uploading new LMCs, rebooting on a specific LMC or just listing the available LMCs in the system.
- **hwlog** The hardware log owns the hardware log flash partitions where it writes faults during system operation.