Enabling New Hardware in U-Boot

Jon Mason, Broadcom Ltd.

About me

Jon Mason is a Software Engineer in Broadcom Ltd's CCX division. Jon's day job consists of enabling, bug fixing, and upstreaming the Linux and u-boot software for Broadcom's ARM/ARM64 iProc SoCs (StrataGX). Outside of work, Jon maintains NTB and a few other drivers in Linux.

About Broadcom

Broadcom Limited is a leading designer, developer and global supplier of a broad range of digital and analog semiconductor connectivity solutions that serve the wired infrastructure, wireless communications, enterprise storage and industrial markets.

About my group

Broadcom's family of communications processors features state-of-the-art functions and performance.

The StrataGX™ BCM587xx, BCM586xx, BCM585xx and BCM5301x Series of communications processors are based on powerful 64-bit and 32-bit ARM® CPU cores and are designed to enable higher performance and lower power. They target a broad range of enterprise networking applications, including control plane processing, virtual CPE (vCPE) and ONFV appliances, service routers and gateways, and small and medium business (SMB) networking devices. Broadcom's StrataGX™ SoCs offer server class networking capabilities with virtualization, stateless, and packet processing capabilities.

Broadcom’s StrataGX SoCs are offered with multicore options (single, dual, quad and octal) with operating frequencies of up to 3GHz and in a variety of packages. The StrataGX SoCs are supported by a full software suite based on enterprise network class open source Linux software and are offered pre-integrated with enterprise and service provider switch drivers. The software includes support for open source projects and development models such as KVM, DPDK, and ODP™.

The Northstar family of SoCs

Also known as StrataGX
Part of the iProc family of SoCs

Northstar, aka BCM470X/5301X
- ARM Cortex A9 based (without VFP)
- Used in many home routers

Northstar Plus, aka BCM58[6/5]2X
- ARM Cortex A9 based (with NEON)
- Used in NASes, routers, switches

Northstar 2, aka BCM5871X
- ARM Cortex A57 based
- Used in a variety of applications

Enough Marketing!

What is a bootloader?

What is a bootloader?

Used to put the hardware into a state suitable to boot an operating system, load the OS into memory, and start the OS booting.

Level 1 – Puts HW into a stable state and finds the bootable media

Level 2 – Loads the OS into memory and starts the boot

It can be both

What is u-boot?

Features and uses of u-boot

Actually called “Das u-boot, the Universal Boot Loader"

I’m going to call it “u-boot” throughout this presentation, don’t shoot me

GPLv2

Features of u-boot

Bootable from onboard storage: SD card, SATA, NAND, SPI, USB, etc.

Bootable File Systems: ext2/3/4, FAT, JFFS2, etc.

Network boot: TFTP, NFS

U-boot alternatives

Coreboot/UEFI/Tiano – Intel - Mixed – used primarily for x86 and ARM servers (for ACPI)

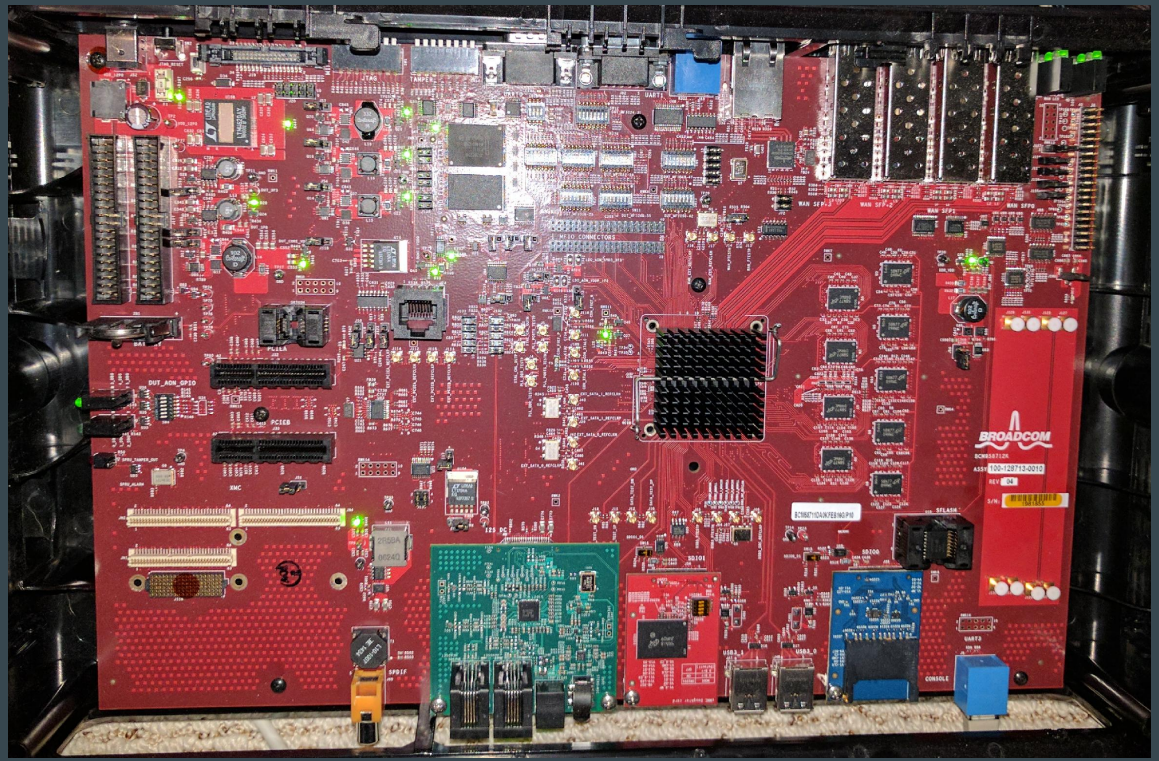
It seems like everyone and their brother is doing a bootloader, and to prove it Broadcom decided to have 2 other ones

CFE – Broadcom – Proprietary License – used primarily for home routers (and MIPS based STB)

BOLT – Broadcom – Proprietary License – used primarily for STB

How to enable new hardware in u-boot

New Hardware



What is the primary goal?

- Getting the Operating System/Linux up and running as quickly as possible
- You are gating all of the Linux kernel developers, device driver developers, diagnostic developers, and everyone else. This might also include customers that somehow got HW the same time as developers
- How to get out of the hot path?

Step 1 – Get Memory working

- RAM PHYs need to be setup, RAM needs to be setup, ECC RAM needs to be init'ed
- Without memory, we're not going to be able to do anything ☺

- It might be possible to get around this if there is a large enough SRAM built-in to the SoC
- It might also be possible to use L2 cache, if large enough and configured appropriately
- SRAM must be large enough to contain u-boot, kernel, etc

Step 2 – Get Serial working

- Without any output, how do you know it is working?
- It is possible to debug this by inspecting the log buffer in u-boot memory (via JTAG), and seeing how far you got

Step 3 – Get Networking working

- Netbooting a Linux kernel will allow all of the PITA developers from the previous slide to get working, thus removing you/your team from being the bottleneck
- Ethernet driver and Ethernet PHY driver are needed here
- You might be lucky and have a driver for a previous generation of SoC (or shared IP) that needs minimal modifications to work

Pro Tip

If you can get PCI working, you can use an existing driver in u-boot to Netboot Linux, etc

This assumes the SoC in question has PCI, and that the PCI driver is less complex than writing/fixing/debugging the Ethernet present in the SoC. Also, there may be PHY drivers necessary for the PCI

But Jon, my SoC doesn't have Ethernet

Or, Ethernet is not working in A0...

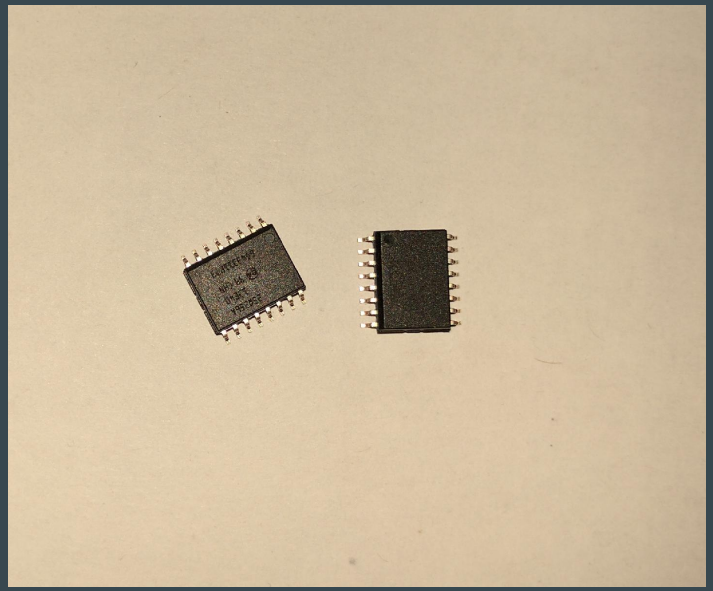
Or, TFTP is not allowed

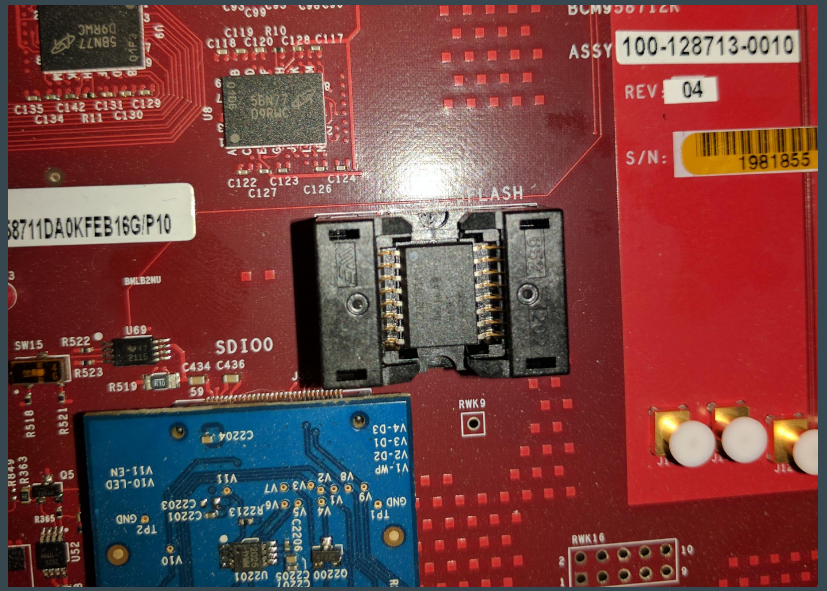
Or, there is no network

Option #1


BROADCOM
100000068787
DSTREAM
TRC CLK
FULL
DATA
TRIGGER
TRACING
DEBUG
TARGET
FLASH
STATUS

Option #2





SPI and NAND

Your SoC is already able to boot from SPI or whatever, but without the SPI or NAND drivers, you are not able to modify it from within u-boot. Getting this working can be vital in saving many footsteps between the lab and your desk.

Other peripherals

- PCI
- SATA
- USB
- MMC/SD

- If you don’t need to load Linux (or other OS) from it, you really don’t need to add a driver

Diagnostics

Test the hardware

Can be done prior to booting the OS, similar to x86 BIOS POST

Can also be used to stress the hardware in a less complicated environment that a full OS, or in HW validation

Caution – be careful of the size of u-boot

With all of the drivers, commands, diagnostics, etc., you might overflow the size of the u-boot partition on your SPI/NAND. Make sure it is generous or add some kind of check in your development environment.

How to upstream u-boot

Signup for the mailing list

It appears that the mailing list is moderated to only subscribers

Subscription appears to require a human to approve

Upstreaming approach

Given the Linux similarities, I would suggest following the Linux mantra of “Push early, push often”

- Submit small, easy to review patches
- Each patch should contain a logical change, but not more than one
- Run checkpatch.pl!
- Try to follow this model in-house to get used to this methodology (and make it easier to submit patches directly from internal git trees).

Customer demand for u-boot upstreaming?

Given the potential lack of customer requests, there might be little to no interest my corporate management to upstream u-boot. Why should you upstream?

- Upstreaming reviews result in superior code quality
- Having the ability in the future to base SDK releases on newer versions of u-boot
- Are you going to do future SoCs on an older version of u-boot? Really?
- Shared code base for all product lines results in less maintenance
- Modifications to the innards of u-boot will be made to the code you care about for you by the community
- Some customers really, really want to run on the latest version (or won't consider your product without it already being present upstream). So, you might sell more...

Upstreaming after-the-fact

Given that management might not have a desire to upstream during initial product development, you might have to upstream after the u-boot code is full featured.

Step 1 - Rebase

Rebase your code to the latest version (v2017.03-rc2)

- This might be as simple as `git rebase v2017.03-rc2`
- This might be so painful that you forego this and simply start from scratch

Step 2 - Squash

Squash the history of the files. No one externally cares, this is the first version they are seeing.

Step 2 – Carve into submittable chunks

Suggested cutting

1. Basic enablement (code sufficient to boot to u-boot prompt – UART, defconfig, headerfile, board file).
2. DT stuff
3. Each individual driver in a separate patch (I might need to add the basics on how we do this in Linux)
4. Diags and anything else. You might need get company permission

GPL Compliance

If this code has ever been given out to a customer or shipped in a product, you probably have to provide your code (as u-boot is a GPL v2 project). So any thing you might think is hidden (and thus you shouldn't upstream), isn't hidden. Just because you add a proprietary header to your source does not make it so!

Step 3 – Submit and rework

Do this serially. No one wants a huge dump of code to have to pick through with similar errors throughout.

If an issue is found in a prior submission, make the modification throughout the code base

Questions?

Request to u-boot maintainers

Please review my patch

http://lists.denx.de/pipermail/u-boot/2017-February/281891.html

Thanks!

Backup Slides

(ARM) Linux Boot Requirements

(from Documentation/arm/Booting)

Essentially, the boot loader should provide (as a minimum) the following:

1. Setup and initialize the RAM
2. Initialize one serial port
3. Detect the machine type
4. Setup the Device Tree or kernel tagged list
5. Load initramfs
6. Call the kernel image.