Porting FreeBSD to AArch64

Andrew Turner – andrew@fubar.geek.nz

12 June 2015

About me

Source committer – focusing on ARM

Freelance Software Engineer

Status of arm64 (AArch64)

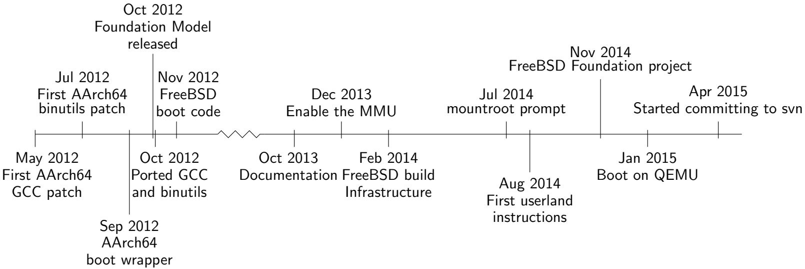
Support to boot in QEMU committed to subversion

Some support for Cavium ThunderX in subversion

Boots on Xen, with a few patches

Only single-core, DMA is assumed to be cache-coherent for now

Timeline



May 2012
First AArch64
GCC patch

Jul 2012
First AArch64
binutils patch

Sep 2012
AArch64
boot wrapper

Oct 2012
Foundation Model
released

Oct 2012
Ported GCC
and binutils

Nov 2012
FreeBSD
boot code

Oct 2013
Documentation

Dec 2013
Enable the MMU

Feb 2014
FreeBSD build
Infrastructure

Jul 2014
mountroot prompt

Aug 2014
First userland
instructions

Nov 2014
FreeBSD Foundation project

Jan 2015
Boot on QEMU

Apr 2015
Started committing to svn

History of FreeBSD on AArch64

4 phases:

1. Early experimentation
2. FreeBSD subversion project
3. FreeBSD Foundation project
4. Committed to HEAD (main development branch)

Early experimentation

My early work to learn the architecture

1. ARM boot code
2. Simple ELF loader
3. Early ASM
4. C code

ARM boot code

Provided by ARM to initialise the hardware

BSD Licensed

Simple ELF Loader

Reads enough of an ELF file to run it

9 instructions

Early ASM

Written to become locore.S – the initial kernel code

1. Puts the hardware in a known state
2. Builds the initial pagetable
3. Enables the MMU
4. Branches to a virtual address
5. Calls into C code

C code

Mostly for debugging

Could write to the UART

Early issues

- No documentation until September 2013
- No debugger in the Foundation Model
- Stopped when enabling the MMU

FreeBSD subversion project

Moved to a FreeBSD project branch in February 2014

- Update the build infrastructure
- Started with external gcc, quickly moved to clang/llvm 3.4
- Initial port of loader.efi
- Imported locore.S
- Stub the kernel → implement as they are hit

clang/llvm

Ported to FreeBSD/arm64

Based on the old, buggy AArch64 backend

Crashed when building some files

Port loader.efi

An EFI application to load the kernel

Provides runtime configuration

Loaded the kernel from the host

Make the kernel build

Started with stub functions to make it build

And atomic operations – all static inline

Then make the kernel run

Started with locore.S from GitHub

Use EARLY_PRINTF to watch the early boot progress

Add initial pmap handling

Faulted in functions as needed Implemented pmap from scratch

To the mountroot prompt

mountroot is when the kernel fails to mount the root fs

Need:

- Exceptions
- Thread/process creation
- User/kernel copy handling

But not devices

But we will need devices

Many parts are machine independent

Except the root device – nexus

Also need bus_space to talk to the devices

And handling device memory and interrupts

For arm64 we need and interrupt controller and timer device

What about userland?

Can run from a small in-kernel fs

The kernel needs to set up the cpu to run userland

Userland needs something to run – crt1.o, libc

Skipped dynamic linking – not needed for init

Need to find out if userland code is running, without syscalls

More pmap

Userland breaks pmap

As userland progresses so does pmap

Ported the amd64 pmap code – simpler than fixing my code

And syscalls

Syscalls are handled as exceptions

Userland and the kernel need to agree on the syscall convention

Which register to place the syscall ID – on FreeBSD x8 Need to

signal to userland when the syscall failed

FreeBSD subversion project – Completion

- Finished in November 2014
- UEFI loaded loader.efi
- The loader loaded the kernel
- The kernel could start userland from a memory filesystem
- Starting to run init

FreeBSD Foundation project

Moved to a FreeBSD Foundation GitHub repository

Allowed collaboration with Ed Maste and Semihalf

Build with in-tree Clang and external binutils

Funded by:

- The FreeBSD Foundation
- ARM
- Cavium

More loader.efi

We cleaned up the kernel interface

And the UEFI interface

Could get device description from UEFI

Static userland

Added enough code to libc for sh and ls

And more pmap

Could run all from a 4MiB in-kernel fs

Moved to virtio-mmio when available

Userland finds bugs in the kernel

Shows missing tlb invalidation

And pmap – implement more stubbed functions

Handling of unmapped buffers

Dynamic linking

Handle program start – like crt1

Calls into the common rltd code

Need to understand the relocation types

Can be lazy and ignore lazy relocations

Faulted in more pmap code

Can we run make buildworld?

Port enough libraries – may disable some when not ready

Not all programs would build

And build with make -k – keep going

Fewer issues over time

Some are just stubs – libkvm

VFP – Floating-point

We got surprisingly fat into userland without VFP support

Needed a driver to handle context store/restore

- Enables the VFP unit when accessed
- Then saves VFP registers on context switch
- Restores registers on next access – only if current values are stale

Committed to HEAD

Moved to the main FreeBSD development branch

Only QEMU support to begin with – Cavium to follow patches as reviewed

Built as part of the Jenkins continuous building

Included in the regular snapshots

Build with in-tree Clang and external binutils

Review

Cleaned up code to get it ready for review

FreeBSD uses a Phabricator instance to review changes

Pushed patches to build for and boot on QEMU
(too many to list here)

Merged x86 and ARM efi loaders

Add Cavium ThunderX support

Added by Semihalf – zbb@

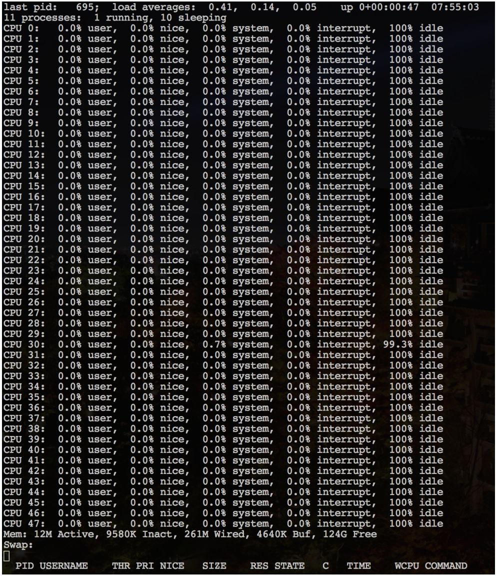
- Busdma
- GICv3 – interrupt controller
- ITS – for MSI/MSI-X
- ThunderX drivers

More changes

- ACPI
- hwpmc
- DTrace
- gem5 simulator support

ACPI by me, the rest from br@

FreeBSD on Cavium TunderX



```
last pid:   695;  load averages:  0.41,  0.14,  0.05    up 0+00:00:47  07:55:03
11 processes:  1 running, 10 sleeping
CPU 0:   0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 1:   0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 2:   0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 3:   0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 4:   0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 5:   0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 6:   0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 7:   0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 8:   0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 9:   0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 10:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 11:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 12:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 13:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 14:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 15:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 16:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 17:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 18:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 19:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 20:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 21:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 22:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 23:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 24:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 25:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 26:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 27:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 28:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 29:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 30:  0.0% user,  0.0% nice,  0.7% system,  0.0% interrupt, 99.3% idle
CPU 31:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 32:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 33:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 34:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 35:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 36:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 37:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 38:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 39:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 40:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 41:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 42:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 43:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 44:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 45:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 46:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
CPU 47:  0.0% user,  0.0% nice,  0.0% system,  0.0% interrupt,  100% idle
Mem: 12M Active, 9580K Inact, 261M Wired, 4640K Buf, 124G Free
Swap:

  PID USERNAME    THR PRI NICE   SIZE    RES STATE   C   TIME    WCPU COMMAND
```

Thanks to zbb@

Demo on QEMU

Thank You

Thank you to:

ARM:

- Andy Wafaa
- Mark Rutland
- Robin Randhawa
- Vassilis Laganakos

Cavium

The FreeBSD Foundation:

- Ed Maste

Semihalf

Questions?

andrew@FreeBSD.org

FreeBSD ARM resources:

Email: freebsd-arm@FreeBSD.org
IRC: #freebsd-arm64 and #bsdmips on EFnet
https://wiki.freebsd.org/arm64