FreeBSD on latest ARM Processors

EABI, Toolchain

Vassilis Laganakos

ARM Ltd.
vasileios.laganakos@arm.com

9th of October, 2010

Why all this fuss?

Well it is a combination of reasons:

- Been using FreeBSD for a long time and love its stability and robustness :o) <- Need to contribute!
- Want to learn more about the FreeBSD structure (in order to contribute :o))
- Found the ARM architecture unique, and wanted to get BSD on it ;o)
- Embedded systems are cool - why not enhance them with BSD ;o)

Why all this fuss?

Well it is a combination of reasons:

- Been using FreeBSD for a long time and love its stability and robustness :o) <- Need to contribute!
- Want to learn more about the FreeBSD structure (in order to contribute :o))
- Found the ARM architecture unique, and wanted to get BSD on it ;o)
- Embedded systems are cool - why not enhance them with BSD ;o)

Why all this fuss?

Well it is a combination of reasons:

- Been using FreeBSD for a long time and love its stability and robustness :o) <- Need to contribute!
- Want to learn more about the FreeBSD structure (in order to contribute :o))
- Found the ARM architecture unique, and wanted to get BSD on it ;o)
- Embedded systems are cool - why not enhance them with BSD ;o)

Why all this fuss?

Well it is a combination of reasons:

- Been using FreeBSD for a long time and love its stability and robustness :o) <- Need to contribute!
- Want to learn more about the FreeBSD structure (in order to contribute :o))
- Found the ARM architecture unique, and wanted to get BSD on it ;o)
- Embedded systems are cool - why not enhance them with BSD ;o)

OABI

Noticed that the ARM ports in buildtree are using OABI.
And also that the toolchain version (GCC 4.2.1) does not support ARMv6 and ARMv7.

Ports

Couldn't find any recent board ports in there (e.g. BeagleBoard, iMX/51, Tegra-2).

Interesting...

There are currently some very interesting ARM-based platforms.
So let's do something about this!

OABI

Noticed that the ARM ports in buildtree are using OABI.
And also that the toolchain version (GCC 4.2.1) does not support ARMv6 and ARMv7.

Ports

Couldn't find any recent board ports in there (e.g. BeagleBoard, iMX/51, Tegra-2).

Interesting...

There are currently some very interesting ARM-based platforms.
So let's do something about this!

ARM info pt.1

- ARM is a 32bit load/store architecture
- 32bit ARM Instruction Set - instructions conditionally executed
- 16bit Thumb Instruction Set - simple branches conditionally executed
- Thumb-2 Instruction Set : extension to Thumb, does not depend on ARM - A mixture of 16 and 32 bit instructions. Small as Thumb and fast as ARM code. :o)
- Has different modes of operation:
  - User unprivileged mode
  - SVC, FIQ, IRQ, Abort, Undef and System privileged modes
- The Core is configurable by the vendor

ARM info pt.2

- Support for Floating Point operations - requires the FP coprocessor
- NEON : wide SIMD extension for the latest ARMv7-A cores - requires the Vector NEON Unit
- Multiple-stage pipeline - size variable depending on implementation

ARMv4T

- Thumb Instruction Set

ARMv5TE

- Improved ARM-Thumb interworking
- DSP multiply-accumulate instr.

ARMv6

- SIMD Instructions
- Unaligned data support
- v6 Memory Architecture
- H/W Options: Thumb-2, Multicore, TrustZone

ARMv7

- Thumb-2
- NEON
- Architecture profiles: Hardware Divide, Thumb-2 only

ARM EABI is the way forward

Some of the benefits

- A more efficient syscall convention (used in GNU EABI)
  - Put syscall number to r7 & use `svc 0` calling the kernel
- More compatibility with various tools
- Structure packing is not as painful as it used to be
  - No minimum size - packing determined by types size
- ARM-Thumb interworking is mandatory - function-level granularity
- Better Floating point performance
  - with or without an FPU is much faster
  - makes mixing soft and hardfloat code possible
- Compatible with all ARM architectures greater than ARMv4T

GNU EABI for Linux is a sub-variant of ARM EABI

Parts of the standard - Pt.1

Register Conventions

r0	argument/result/scratch register
r1	argument/result/scratch register
r2	argument/scratch register
r3	argument/scratch register
r4	variable register
r5	variable register
r6	variable register
r7	variable register
r8	variable register
r9	platform register
r10	variable register
r11	variable register
r12	IP call register
r13	the stack pointer
r14	the link register
r15	the program counter

Parts of the standard - Pt.2

Some more Conventions...

- If you are passing more arguments that can fit in the first 4 registers, they will be placed on the stack
- Need to have 64-bit alignment. I.e.

  for a 64-bit value, `r0-r1` *or* `r2-r3` *must be used, but not* `r1-r2`

- enums can have variable size type

Conventions

Similar things hold for the VFP and Advanced SIMD Standard Variants of the EABI. Won't go into this now :o)

A project?

Yes! - It is project "Prometheus":

Trying to learn more for FreeBSD internals by getting first an EABI, ARMv7 supporting toolchain, which will then be imported to the build system (that's the 'learn internals part of the story' :o)), and then used to set-up a kernel for an ARMv7 arch platform.
And then going to see if we can boot the platform, in order to test whether the application we built is actually working. :o)
Iterate as appropriate!

Oh! Also chose to work on FreeBSD-Current. (9.0 at the time)

Note!

This is – as most of the times, work-in-progress :o)

Please note

Not using the FreeBSD build system, yet!

The cross-building toolchain is built on its own, outside the build system.

Getting the goodies!

- get binutils 2.20.1

  `wget http://ftp.gnu.org/gnu/binutils/binutils-2.20.1.tar.gz`
- get GCC 4.4 branch

  `svn co svn://gcc.gnu.org/svn/gcc/branches/gcc-4_4-branch/ gcc`
  - Picked 4.4 branch because I want to have Thumb-2 and ARMv7 Arch support.

Building binutils - Pt.1

Configure:

```
./configure --prefix=$HOME/work/out  --target=arm-unknown-freebsd9.0
```

Build:

```
gmake
```

Install:

```
gmake install
```

Building binutils - Pt.2

And this will deploy things in a layout, like things would look in a live embedded system.

The result will look like:

```
[vassilis@prometheus ~/work/out]$ ls
arm-unknown-freebsd9.0  include  lib  man
bin                     info     libexec  share
```

Errors in building

Need to use gmake, or you get errors like the following:

```
flat_bl.m:2: error: expected identifier or '(' before '%' token
*** Error code 1

Stop in /prometheus/work/arm/binutils-2.20.1/gprof.
*** Error code 1

Stop in /prometheus/work/arm/binutils-2.20.1/gprof.
*** Error code 1

Stop in /prometheus/work/arm/binutils-2.20.1/gprof.
*** Error code 1

Stop in /prometheus/work/arm/binutils-2.20.1.
*** Error code 1

Stop in /prometheus/work/arm/binutils-2.20.1.
```

GCC Pre-requisites

Get GMP and MPFR as you need them for the compilation of the Cross-Toolchain.

The stuff you get from the respective ports seem to be just fine.

Or if you prefer building them from source for some reason, grab them from:

`http://www.mpfr.org/mpfr-current/mpfr-3.0.0.tar.bz2`
`ftp://ftp.gmplib.org/pub/gmp-4.3.2/gmp-4.3.2.tar.bz2`

No obvious reason why you should do this though... :o)

Important

Make sure...

- ...you use gmake for building GCC
- ...you build it out of its source tree
- ...before `config` and `gmake`, add the path to the built and installed binutils early at your `PATH` env var

Or else you will get quite a few weird warnings and artefacts!

Prepare the required headers

Need to copy some header files where the GCC building system expects them to be: in `sysroot` related path

```
mkdir -p $HOME/work/out/sysroot/usr/include/sys
cp /usr/src/sys/arm/include/_types.h $HOME/work/out/sysroot/usr/include/sys/
cp /usr/src/sys/sys/cdefs.h $HOME/work/out/sysroot/usr/include/sys/
...
```

Some duplicates...(?)

```
cd $HOME/work/out/sysroot/usr/include
ln -s /prometheus/work/out/sysroot/usr/include/sys/errno.h
ln -s /prometheus/work/out/sysroot/usr/include/sys/sched.h
...
```

And some more not included above

```
cp /usr/src/include/pthread.h $HOME/work/out/sysroot/usr/include/
...
```

Or else...

You will get errors such as:

GCC: "...I can't find the headers man..."

```
In file included from ../../../gcc/libgcc/../gcc/tsystem.h:44,
                 from ../../../gcc/libgcc/../gcc/libgcc2.c:29:
/prometheus/work/arm/objdir/./gcc/include/stddef.h:59:24: error:
sys/_types.h: No such file or directory
In file included from ../../../gcc/libgcc/../gcc/libgcc2.c:29:
../../../gcc/libgcc/../gcc/tsystem.h:87:19: error: stdio.h: No such file or directory
../../../gcc/libgcc/../gcc/tsystem.h:90:23: error: sys/types.h: No such file or directory
../../../gcc/libgcc/../gcc/tsystem.h:93:19: error: errno.h: No such file or directory
../../../gcc/libgcc/../gcc/tsystem.h:100:20: error: string.h: No such file or directory
../../../gcc/libgcc/../gcc/tsystem.h:101:20: error: stdlib.h: No such file or directory
../../../gcc/libgcc/../gcc/tsystem.h:102:20: error: unistd.h: No such file or directory
../../../gcc/libgcc/../gcc/tsystem.h:108:18: error: time.h: No such file or directory
```

Build Cross-GCC

Need to perform 3 passes:

- 1st pass - Just C Language support
  - Don't build C library
- 2nd pass - Build C library
- 3rd pass - Build the rest of the toolchain

Configure GCC for cross-building

So, configure GCC accordingly:

```
../gcc/configure \
--target=arm-unknown-freebsd9.0 --host=i386-unknown-freebsd9.0 \
--build=i386-unknown-freebsd9.0 --prefix=$HOME/work/out/ \
--with-sysroot=$HOME/work/out/sysroot --enable-languages=c \
--disable-multilib --disable-libmudflap --disable-threads \
--disable-libgomp --disable-libssp
```

Notice that only C is enabled, since this will suffice for now. Disabled are *multilib* and *libmudflap*, as well as threads.
For more details, see
http://www.linuxfromscratch.org/lfs/view/6.6/chapter05/gcc-pass1.html

Configure GCC for cross-building

Also...

Edited the `freebsd.h` file to build for another, more recent architecture.

```
#define SUBTARGET_CPU_DEFAULT   TARGET_CPU_arm926ejs

#undef  TARGET_VERSION
#define TARGET_VERSION fprintf (stderr, " (FreeBSD/ARM926EJ-S ELF)");
```

Once...

...this gets in a reasonable state, will try moving to ARMv7

And build...

```
gmake
```

Wotchas!

What's that!?

```
In file included from tm.h:16,
                 from insn-recog.c:7:
../.././gcc/config/arm/arm.h:606:1: warning: "WCHAR_TYPE_SIZE" redefined
In file included from tm.h:15,
                 from insn-recog.c:7:
../.././gcc/config/arm/freebsd.h:61:1: warning: this is the location of the previous definition
In file included from tm.h:16,
                 from insn-recog.c:7:
../.././gcc/config/arm/arm.h:2249:1: warning: "HANDLE_PRAGMA_PACK_PUSH_POP" redefined
In file included from tm.h:12,
                 from insn-recog.c:7:
../.././gcc/config/freebsd.h:59:1: warning: this is the location of the previous definition
```

Well...

- `WCHAR_TYPE_SIZE` - should be 32bits
- `HANDLE_PRAGMA_PACK_PUSH_POP` - Source code says: */* Define this so we can compile MS code for use with WINE. */*

Done!

...and install it

gmake install

So...

We now have a toolchain that can compile and link programs that do not use C library ... Not that useful!

All we need now is the C Library!

More specifically:

- crt1.o – part of csu code
- libc.so

and as source code reveals:

"csu must be built before all shared libraries for ELF"

and

"libc must be built before all other shared libraries"

CSU

Built it from the main build tree -> got crt1.o which was missing from what we got from GCC so far.

However...

Unfortunately building csu and libc is a hack for now

Moving stuff out of the build tree... :o(

Objective:

Need to familiarize more with the build system!

And this is the current stage in the project:
In the process of building libc...

Milestones; a Schedule or something...

1. Build libc and finalize the GCC cross toolchain
2. Find a better way of doing it and create a port
3. Work on getting support for a recent platform
4. List the changes required in the current kernel to support EABI (kind of critical ;o))
5. Userspace compilation...

- Update the Wiki: http://prometheus.wikidot.com
- Gradually push something in the project page as things are created: launchpad.net/prometheus
- Feedback would be most helpful! :o)

However:

- Not sure if dropping-in a later version of binutils in the buildtree is going to be accepted
- Same for the 4.4 GCC - as everything after version 4.2.1 is GPLv3

So:

- Perhaps this is destined to be a standalone port?
- Can't be sure yet...

However:

- Not sure if dropping-in a later version of binutils in the buildtree is going to be accepted
- Same for the 4.4 GCC - as everything after version 4.2.1 is GPLv3

So:

- Perhaps this is destined to be a standalone port?
- Can’t be sure yet...

Thoughts...

- Maybe switch to llvm entirely and forget the above work :o)

ARM have contributed some patches, regarding the EABI compatibility in the LLVM-backend

- http://llvm.org/viewvc/llvm-project?view=rev&revision=93884 – Optimise `~(~X >>s Y) --> (X >>s Y)`
- http://llvm.org/viewvc/llvm-project?view=rev&revision=97656 – Add framework for ARM builtins
- http://llvm.org/viewvc/llvm-project?view=rev&revision=103181 – Bug fix in ARMISelDAGToDAG.cpp
- http://llvm.org/viewvc/llvm-project?view=rev&revision=103876 – `powf/modf` fixes for Solaris build
- http://llvm.org/viewvc/llvm-project?view=rev&revision=103877 – `round()` fixes for Solaris build
- http://llvm.org/viewvc/llvm-project?view=rev&revision=109854 – Bug fix in `SelectionDAG/TargetLowering.cpp`
- http://llvm.org/viewvc/llvm-project?view=rev&revision=110072 – C-tor polymorphism in `lib/Analysis/DebugInfo.cpp`
- http://llvm.org/viewvc/llvm-project?view=rev&revision=114991 – Support for ARM Run-Time ABI (FP and Integerhelper functions)

EABI support is something ARM cares about -
Answering EABI/ARM specific questions in the mailing list :o)

Note:

llvm 2.8 - ARM NEON support was added to Clang.

Currently

- Investigating the existing infrastructure, and going through the instructions at: wiki.freebsd.org/FreeBSD/arm for the ports that are already there, and wiki.freebsd.org/FreeBSDArmBoards for adding new ports.
- The fact that the ARM cores are customizable, and the unique way that they are embedded in each different SoC, would make it useful to have a more generic way of "expressing" an ARM platform in the build system
- Had some experience adding support for ARM968 to FreeRTOS, so I hope that this is going to be helpful :o)

Currently

- Adding a new platform is of higher priority
- Need to check the existing syscalls and see if there are any changes required - I meant *how many* are required :o)

References

ARM EABI	ABI for the ARM Architecture
Debian ARM EABI	http://wiki.debian.org/ArmEabiPort
LLVM	http://llvm.org/docs/ReleaseNotes.html
FreeBSD ARM wiki	http://wiki.freebsd.org/FreeBSD/arm
EABI on FPU speed	Why-ARMs-EABI-matters
Linux from Scratch	Constructing a temporary system - GCC pass1
glibc -EABI syscall interface	http://sourceware.org/ml/libc-ports/2005-11/msg00028.html
ARM Tech Blogs	http://blogs.arm.com/software-enablement/

Acknowledgement:
The ever cute little Daemon is a registered trademark of Marshall Kirk McKusick :o)

Hoping that real-life / work-life is not going to interfere as much as it did last year with the project... :o)

Thank you for your time!

Any...

Questions?

Contact Details

email: vasileios.laganakos@arm.com

Prometheus Wiki http://prometheus.wikidot.com
Prometheus Project https://launchpad.net/prometheus

More than happy for any kind of feedback/information!