ARM Linux specifics

Thomas Petazzoni / Michael Opdenacker
Free Electrons
http://free-electrons.com/

Created with OpenOffice.org 2.x

Rights to copy

creative commons
COMMONS DEED

Attribution – ShareAlike 3.0

You are free

- to copy, distribute, display, and perform the work
- to make derivative works
- to make commercial use of the work

Under the following conditions

Attribution. You must give the original author credit.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

License text: http://creativecommons.org/licenses/by-sa/3.0/legalcode

Document sources, updates and translations:
http://free-electrons.com/docs/arm-linux

Corrections, suggestions, contributions and translations are welcome!

Best viewed with...

This document is best viewed with a recent PDF reader or with OpenOffice.org itself!

- Take advantage of internal or external hyperlinks. So, don't hesitate to click on them!
- Find pages quickly thanks to automatic search
- Use thumbnails to navigate in the document in a quick way

If you're reading a paper or HTML copy, you should get your copy in PDF or OpenOffice.org format on http://free-electrons.com/training/devtools!

Training contents

- Floating point and ABIs
  - Floating point support on ARM Linux
  - Different ABIs
- Thumb mode
  - Introduction to Thumb
  - Thumb and ARM code together
  - Interworking on your system
- Other ARM extensions

Floating point and ABIs

Floating point support (1)

- Many ARM platforms do not have an hardware floating point unit
- Two solutions exists to emulate floating point
  - Hard float: let userspace binaries use floating point instructions, and emulate them in the kernel using the “illegal instruction” exception
  - Soft float: add the emulation code in userspace at compile time, using gcc `-msoft-float` option
- The solution traditionally used in Linux is hard float, with FPA instructions
- However, hard float is very slow due to the exception handling and context switch.

Floating point support (2)

- In the Linux kernel, two floating point emulators are available
  - NWFPE, NetWinder Floating Point Emulator
    `CONFIG_FPE_NWFPE`
  - FastFPE, faster that NWFPE, but not fully complete and not recommended for scientific applications
    `CONFIG_FPE_FASTFPE`
- Support for VFP is also available
  - VFP is a coprocessor extension for floating point computations available in ARM10, ARM11 and Cortex processor families

Mixing hard and soft float

- Due to ABI calling conventions differences, it was not possible with the traditional ABI to mix hard and soft float code in userspace
- An application and all its libraries have to be compiled either for hard float or soft float
- One of the reason for which floating point emulation in the kernel was preferred over soft float
  - Binaries could take advantage of floating point capable hardware immediately, with no recompiling.
- Fortunately, the new EABI solves this issue

EABI (1)

- EABI is a new standardized ABI for the ARM platforms
- It has several advantages
  - Ability to mix hard and soft float code, so that general code can be compiled with soft float and several versions of optimized libraries can be provided using hard float
  - Allows to link with code generated by other compilers and provided by other vendors
  - Has integrated support for Thumb interworking

EABI (2)

Other changes coming from EABI

- Structure packing and alignment rules change : no minimum alignement in structures
- Stack alignment on function entry is 8 bytes instead of 4 bytes
- Alignment of 64 bits types is 8 bytes instead of 4
- System call interface
  - The system call number was passed as part of the `swi` instruction
  - The kernel had to read and decode the `swi` instruction, polluting the data cache with instructions
  - Now, the system call number is passed in `r7`
  - 64 bits function arguments are aligned to an even-number register instead of using the next available pair

EABI in gcc and Linux

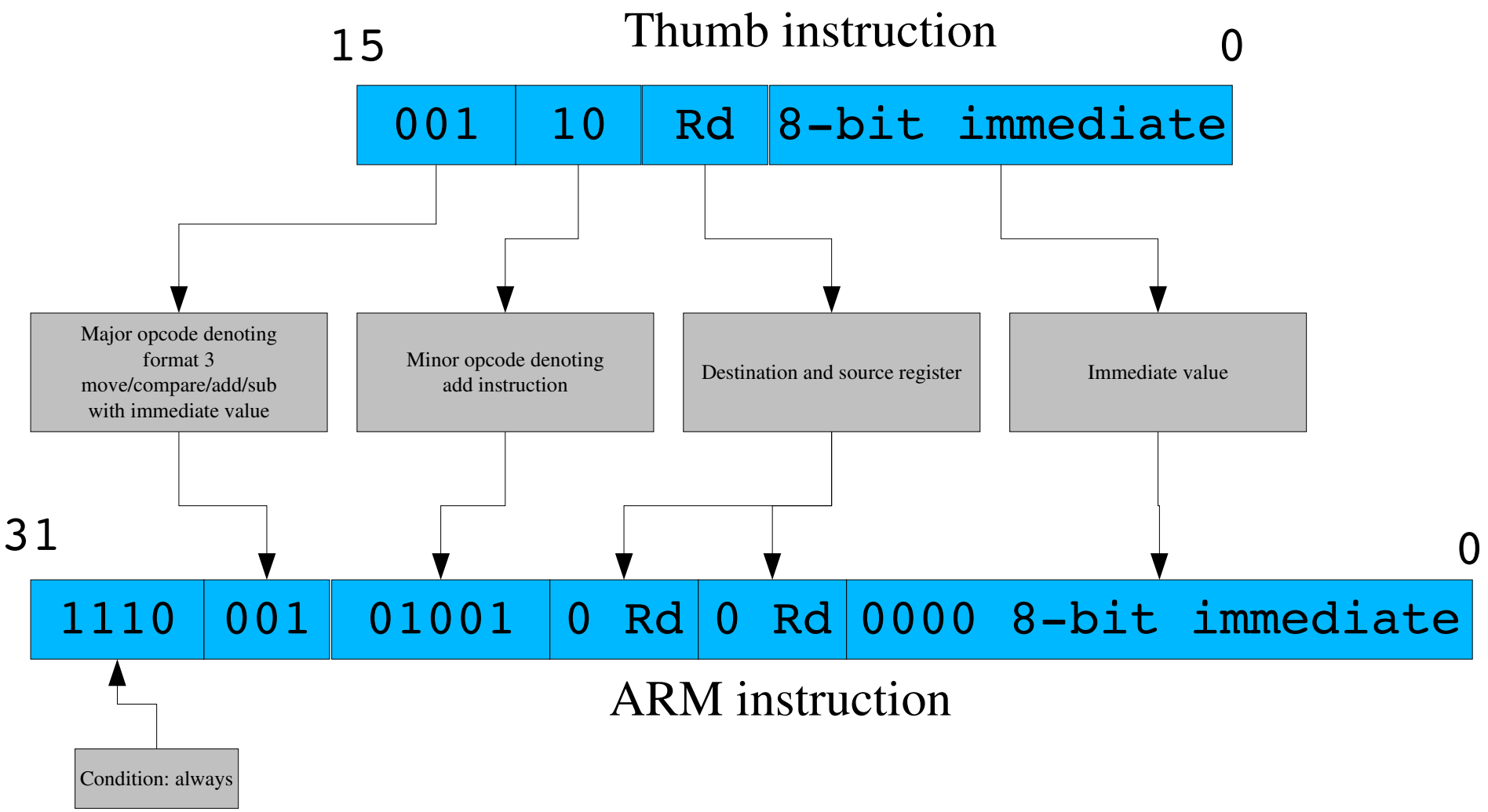
- Support for EABI was added in GCC 4.1.0
  - Buildroot allows to select the target ABI of the toolchain
- Support for EABI was added in Linux 2.6.16
  - `CONFIG_AEABI`
    - Compiles EABI support in the Linux kernel, so that applications can be compiled with the new EABI
  - `CONFIG_OABI_COMPAT`
    - In an EABI-able kernel, provides compatibility with old ABI userspace binaries
    - Works only for non-Thumb binaries
- Running an EABI binary on a non-EABI kernel doesn't work

Introduction to Thumb

Two instruction sets

- Default mode on ARM : instructions on 32 bits
- With the ARMV4T ISA, a new execution mode is added, with 16 bits instructions : Thumb mode
  - ARMV4T ISA is used in ARM7TDMI, ARM9TDMI, ARM7x0T, ARM9xxT
- In the ARMV5TE ISA, improvements to ease the switch between ARM and Thumb modes
- 16 bits instructions can be useful to increase code density, and decrease the overall code size

Instruction encoding

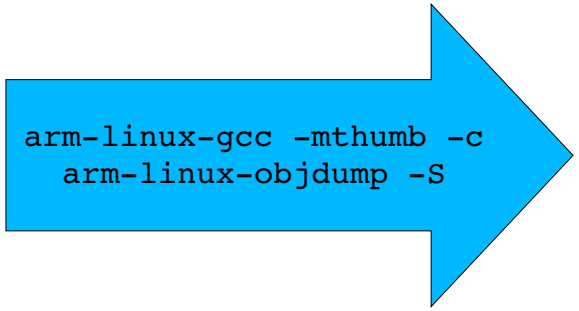


Compiling a thumb program

- Any ARM toolchain is able to produce binaries using the Thumb instruction set
- Using the `-mthumb` option of the GNU C Compiler

```
int bar(int c, int d)
{
    return c + d;
}

int foo(int a, int b)
{
    a += 3;
    b -= 2;
    return bar(b, a);
}
```




```
0000000 <bar>:
   0:   b580            push    {r7, lr}
   2:   b082            sub     sp, #8
   4:   af00            add     r7, sp, #0
   6:   1d3b            adds    r3, r7, #4
   8:   6018            str     r0, [r3, #0]
   a:   1c3b            adds    r3, r7, #0
   c:   6019            str     r1, [r3, #0]
   e:   1d3b            adds    r3, r7, #4
  10:   1c3a            adds    r2, r7, #0
  12:   6819            ldr     r1, [r3, #0]
  [...]
```


Branches on two instructions

- In Thumb mode, branch and link instructions take two instructions

```
0000000 <bar>:
   0:   b580            push    {r7, lr}
  [...]

00000020 <foo>:
  [...]
  4e:   f7ff fffe       bl      0 <bar>
  [...]
```

- A.7.1.17 BL, BLX instructions in Thumb mode
  « These Thumb instructions must always occur in the pairs described above »

Size gains

- Size gains on a small, non-representative example

```
int bar(int c, int d)
{
    return c + d;
}

int foo(int a, int b)
{
    a += 3;
    b -= 2;
    return bar(b, a);
}
```

arm-linux-gcc -c → test.arm.o

```
$ sizediff test.arm.o test.thumb.o
   text    data     bss     dec     hex filename
    124       0       0     124      7c test.arm.o
     96       0       0      96      60 test.thumb.o
    -28       0       0     -28     -1C +/-
```

arm-linux-gcc -c -mthumb → test.thumb.o

- 28 bytes reduction, 22% code size reduction

Thumb and ARM code together

Interworking

Thumb and ARM code together (2)

- For several reasons, one might need to combine ARM and Thumb code together
  - Performance-critical code in ARM
  - Libraries compiled in ARM mode
- The ARM achitecture provides instructions to switch back and forth
  - `bx` and `blx` instructions, the lowest bit of the address set allowing to select Thumb or ARM mode
  - `ldr` and `ldm` instructions that load the pc register can also be used
- T bit (bit 5) in the CPSR controls the mode

Interworking

- The GNU C Compiler provides a transparent mechanism called *interworking* to allow the mix of ARM and Thumb code
- *Interworking*-enabled code can be generated using `-mthumb-interwork`
- The toolchain must be *interwork* capable
  - `--enable-interwork` binutils configuration option
  - `--enable-interwork` gcc configuration option

Interworking (2)

```
000081c4 <main>:
    81c4:       b580            push    {r7, lr}
    81c6:       af00            add     r7, sp, #0
    [...]
    81cc:       f005 fb30       bl      d830 <__foo_from_thumb>
    [...]

00008220 <foo>:
    8220:       e1a0c00d        mov     ip, sp
    8224:       e92dd800        push    {fp, ip, lr, pc}
    [...]
    8254:       e12fff1e        bx      lr

0000d830 <__foo_from_thumb>:
    d830:       4778            bx      pc
    d832:       46c0            nop                     (mov r8, r8)

0000d834 <__foo_change_to_arm>:
    d834:       eaffea79        b       8220 <foo>
```

Function `main()`, compiled in Thumb, calls `foo()` in ARM mode.

GCC generated wrappers around `foo()` to switch to ARM mode

Interworking (3)

- Two thumb “b” instructions
  - `lr = pc + (immediate << 12)`
  - `pc = lr + (immediate)`
    `lr = addr of next instruction | 1`

```
f005 fb30
bl d830 <__foo_from_thumb>
```

- Switch to ARM mode
  - `pc` has the lowest bit to 0, switch to ARM

```
4478 bx pc
```

- Call the correct function

```
eaffea79        b 8220 <foo>
```

- Return to Thumb mode at the calling site
  - `lr` has the lowest bit to 0, switch to Thumb

```
e12fff1e        bx lr
```

Interworking on your system

Several solutions

- ARM and Thumb mode of the kernel and userspace are independent
  - Can use a ARM kernel with a Thumb mode userspace, the system call ABI remains the same
- Full Thumb userspace, including the *libc*
  - *uClibc* doesn't seem to support Thumb mode correctly, at least gcc 4.2 is not able to compile it
- Thumb userspace, excluding the *libc*
  - The solution chosen for our experiments

Generating the toolchain

- Binutils and gcc
  - `--enable-interwork`
- Uclibc
  - `-mthumb-interwork`
  - `USE_BX` configuration option
- Automated using Buildroot
  - `BR2_INTERWORKING_SUPPORT`
  - Using a Free-Electrons contributed patch, not merged in the official Buildroot version yet

Compiling your applications

- Manually
  - Add the `-mthumb` option to the compilation command line
  - `CFLAGS+=-mthumb`
- Automated using Buildroot
  - `BR2_THUMB_BINARIES`
  - Using a Free-Electrons contributed patch, not merged in the official Buildroot version yet
- Using Scratchbox
  - Need to integrate the new toolchain inside Scratchbox
  - Follow http://www.scratchbox.org/wiki/ForeignToolchains

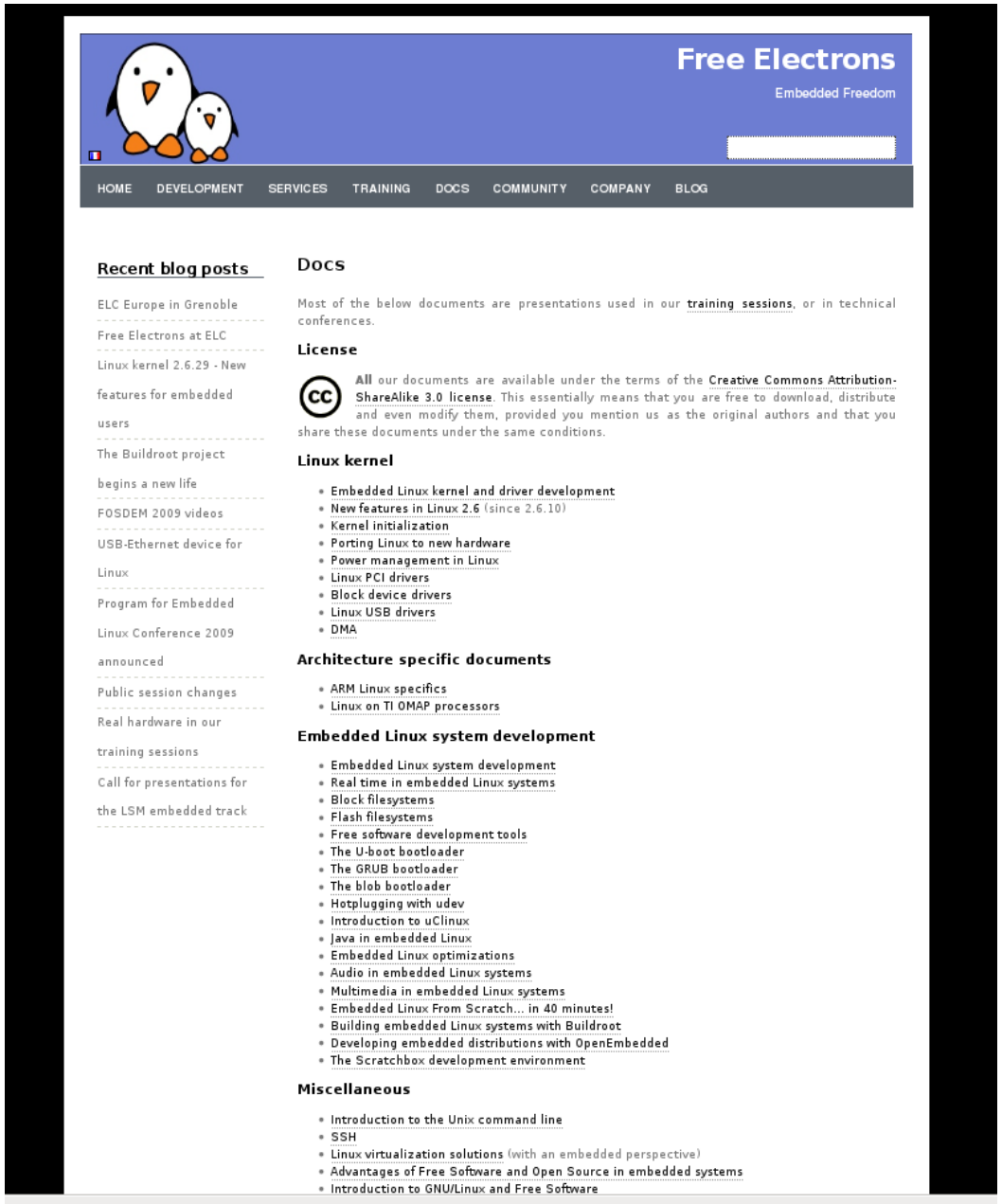
Jazelle and Thumb 2

- Jazelle, allows to execute some Java bytecode in hardware
  - Need a Jazelle-aware Java Virtual Machine
  - Support in ARM5vTEJ, ARMv6 and ARMv7
  - http://www.arm.com/products/esd/jazelle_home.html
- Thumb 2 extends the traditional 16 bits Thumb instruction set with 32 bits instruction
  - Goal is to achieve similar density as Thumb code with performance similar to ARM code
  - Support in ARMv6T2 (ARM1156T2) and ARMv7 (Cortex). Cortex-M3 has only Thumb 2 support.
  - http://www.arm.com/products/CPUs/archi-thumb2.html
  - Linux 2.6.26 adds support for Thumb 2 userspace.

ThumbEE

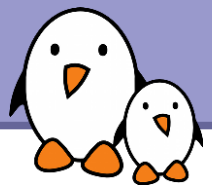
- ThumbEE stands for Thumb Execution Environment
  - Adds more instructions designed for runtime generated code, for example by JIT compilation (automatic null pointer checks or array boundary checks, branch to handlers, etc.)
  - http://www.arm.com/pdfs/JazelleRCTWhitePaper_final1-0_.pdf

Related documents



All our technical presentations on http://free-electrons.com/docs

- Linux kernel
- Device drivers
- Architecture specifics
- Embedded Linux system development



How to help

You can help us to improve and maintain this document...

- By sending corrections, suggestions, contributions and translations
- By asking your organization to order development, consulting and training services performed by the authors of these documents (see http://free-electrons.com/).
- By sharing this document with your friends, colleagues and with the local Free Software community.
- By adding links on your website to our on-line materials, to increase their visibility in search engine results.

Linux kernel

Linux device drivers
Board support code
Mainstreaming kernel code
Kernel debugging

Embedded Linux Training

All materials released with a free license!

Unix and GNU/Linux basics
Linux kernel and drivers development
Real-time Linux, uClinux
Development and profiling tools
Lightweight tools for embedded systems
Root filesystem creation
Audio and multimedia
System optimization

Free Electrons

Our services

Custom Development

System integration
Embedded Linux demos and prototypes
System optimization
Application and interface development

Consulting and technical support

Help in decision making
System architecture
System design and performance review
Development tool and application support
Investigating issues and fixing tool bugs