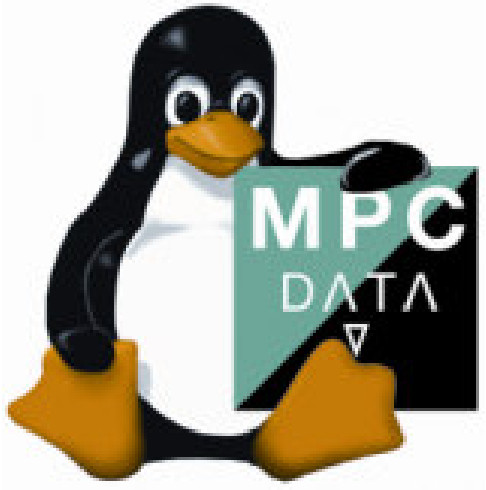
Porting uClinux to a new processor architecture

Embedded Linux Conference 2008 Europe

Peter Griffin

Embedded Software Engineer

MPC Data Ltd



Overview

Porting uClinux to a new processor architecture

- What is uClinux (NOMMU Linux)?
- What's different?

Porting to a new arch

- Chip and architecture overview
- Our approach to the project, what we did
- Porting uClinux 2.6 kernel to a new arch
- Porting uClibc and elf2flt to a new arch

War Stories

- Problems , bugs, and debugging methods

Performance enhancements *(XIP user space, relax)*

SiTel platform now

What is uClinux?

- Linux for systems **without a Memory Management Unit (MMU)**
- First ported to the Motorola MC68328 (1998 Linux 2.0) now supports most architectures (ARM, MIPS, x86, sh, powerpc, CRIS, NIOSII, m68k, blackfin and probably more) to varying degrees.
- NOMMU support has enabled many more devices to run Linux (low end 32bit processors, and even high end 16bit ones)
- Now NOMMU support is in the mainline kernel, but kernel.org only gets you to init!
- The uClinux distribution (www.uclinux.org) provides a whole framework for building software for MMUless Linux (lots of userland stuff).
- When we started our port uClinux stable dist was at 2.6.19

What is different?

- Most of the kernel is exactly the same! (networking, filesystems, user / kernel separation etc)
- However No MMU means **no Virtual memory (VM)** and **no memory protection**, therefore the main changes are to the memory management subsystem.
- No virtual memory means : -
  - No on demand loading (applications must fit wholly in RAM / ROM if XIP)
  - No Fork system call (have to use vfork)
  - No dynamic stack (fixed at compile time)
  - RAM fragmentation can be a problem (memory free but can't allocate as it is not continuous).
- Special 'light weight' binary format called "bin flat"
  - **No dynamic stack (**fixed at compile time)
  - Programs need to be 'relocated' to get a unique address space at runtime
    - Blackfin & frv archs support FDPIC (elf for NoMMU) binary format (needs PIC toolchain).

Examples

POS

Opengear KVM

Blackfin DSP's

PMP

Snapgear router

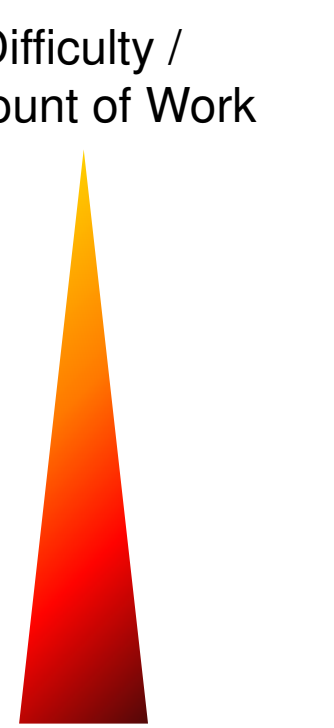
Samsung's Miniket

Pictures from www.linuxdevices.com

Linux porting

Generally **three** types of porting: -

1. **Board port**
   - •E.g. Porting from an existing reference BSP to some custom hardware for a customer.
2. **CPU Port**
   - •E.g. Typically porting from an existing similar CPU implementation, to a new processor. Maybe new on chip peripherals to support / differences to Timers / UART / Clocks / etc
3. **Architecture Port – What we will be looking at today**
   - •. All assembler and arch specific code has to be written. Normally start by copying a similar architecture.



•Usually have to do anything **above you** in this list. E.g. architecture port requires CPU and board support as well.
•Board and CPU ports are a far more common task, than architecture. Although arch ports are getting more common with FPGA soft cores running Linux.

Delivering Software Innovation

The Chip

The SiTel SC14450 Baseband processor

240 MIPS SC14450 CAT-iq baseband processor

- **CompactRISC CR16C 16-bit microcontroller**
  - 2 user-programmable Gen2DSPs
  - Peripheral interfaces including master / slave PCM, UART, SPI, 3 x Timers, dual-access bus and USB 2.0.
  - Power and battery management options
  - White LED driver
  - Integrated class D amplifier

The Product

- VOIP, DECT phones
- High performance, low cost

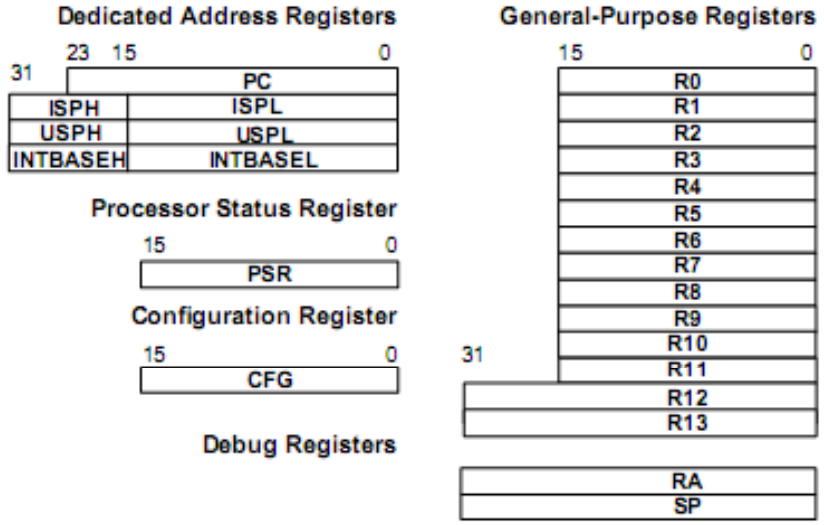
The CPU: SiTel SC14450

The Board: SC14450 SDK

CR16C Architecture

- 82Mhz 16 bit MCU
  - 16 GPR's, 4 double registers
  - Processor status & configuration registers
  - Variable instruction length (16, 32, or 48 bits)
  - 3 stage pipeline (IF, ID, EX)
  - 5 addressing modes: register, immediate, absolute, relative and indexed
- 16Mbytes of program space (24bit PC)
- 4Gb Data space
- Supervisor and user modes for OS support
- 3 Stacks (user, supervisor and interrupt)
- 32kb cache
- JTAG debug connection (one wire jtag)

Register Set




```
/*Doubling 16bit
registers*/
movd   $DEADBEAF, (r1,r0)
loadd  $4(r1,r0), (r13)
```

What we started with

- New gcc & binutils port for CR16C+ architecture
- New SC14450 SDK hardware
- JTAG debugger and initialisation script

Our Approach

Phase 1

1. **Initial feasibility investigation**
   - Test tools (debugger, compiler, linker, hardware)
     - Revealed no 64bit support in toolchain

Phase 2

2. **Port uClinux kernel with minimal device drivers**

Phase 3

3. **C library port (uClibc)**
4. **Elf2flt binary conversion tool**

Phase 4

5. **Lots of debugging & testing (busybox, uClibc test apps)**
6. **Performance enhancements (relaxing, PIC, XIP)**

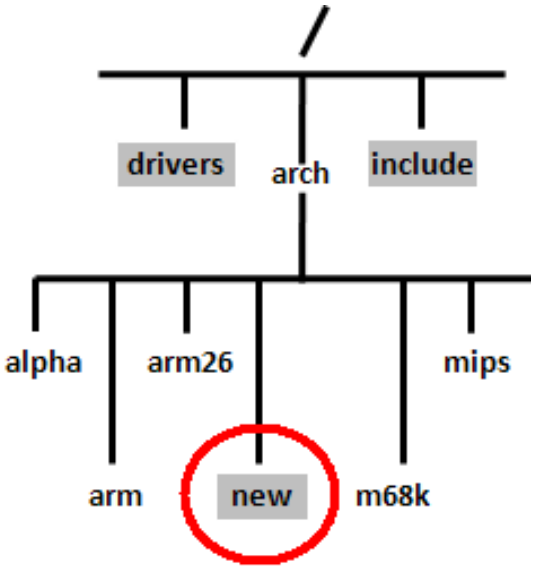
Approach to porting the kernel

1. Copy an existing similar architecture framework
   Time spent here can help you in the long run ☺

2. Think about the differences between the arch you copied and the arch your porting to. This can give you a good idea of where problems will arise. E.g. cache

3. Create entries for your arch, CPU and board, stub functions that need implementing (gives you a good idea of what you have to implement).

4. Get it to compile and link!



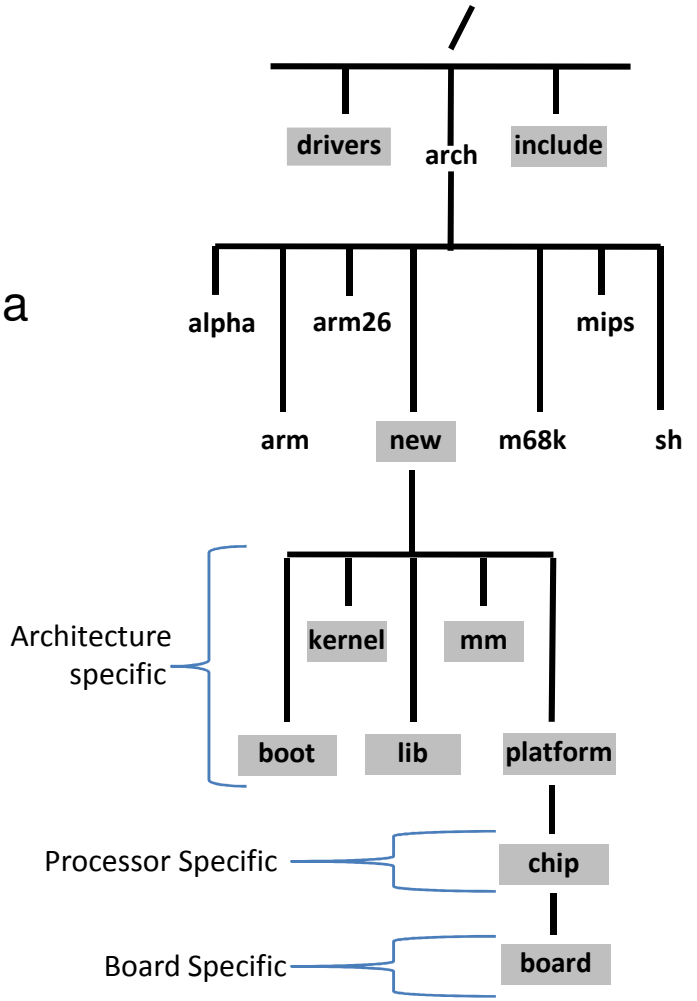
Kernel Porting

Files which will need implementing

1. Port startup assembler (crt0_ram.S / head.S)
   - Initialise the status registers and cache, turn interrupts off, setup the stack, generally get to a state where you can call C code (start_kernel)

2. Early printk – get your UART working!

3. setup_arch function (arch/kernel/setup.c)
   - Setup bootmem allocator - reserve the RAM where the kernel is running!



Kernel Porting

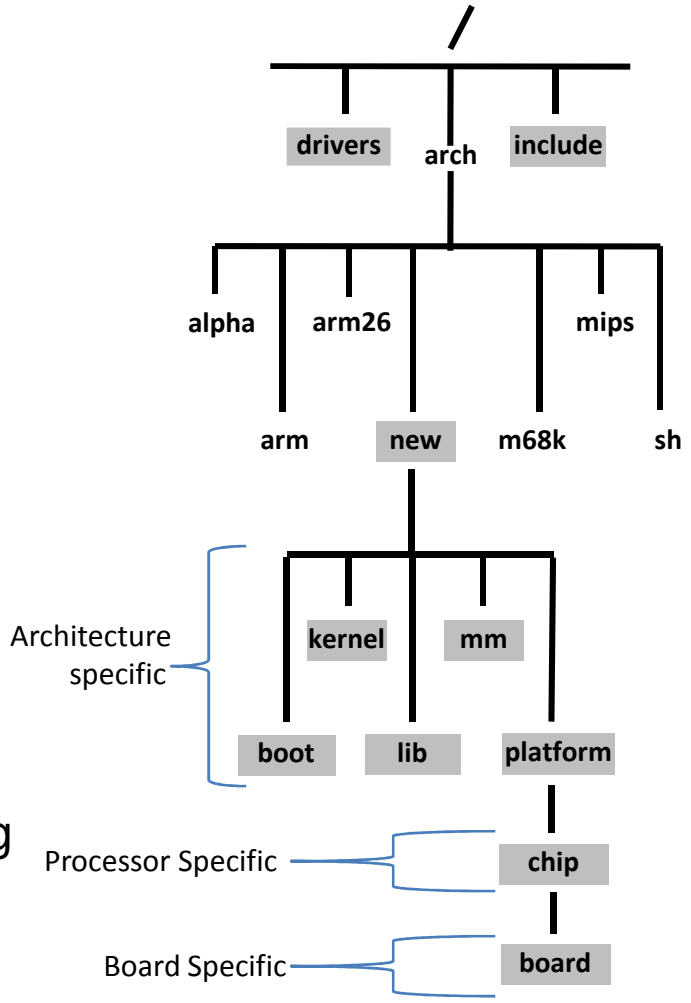
6. arch/kernel/entry.S
   - Assembler for system call and interrupt handling
   - Saving & restoring processor context
   - process switching (switch_to)

7. Process creation (arch/kernel/process.c)
   - copy_thread, exec, kernel_thread, clone, fork, start_thread

8. Signals (arch/kernel/signal.c)
   - More architecture specific than you might think!

9. Binflat loader (include/asm/flat.h)
   - Arch specific relocation types

Other CPU / board stuff to get a bare minimum working kernel
- Interrupt controller support
- Timer and serial port drivers



User Space

Two main things need porting for user space to work

1. C library – probably uClibc
2. Elf2flt tool – convert elf binaries to flat format

uClibc

- C library optimised for embedded systems
- arch specific bits need porting for a new architecture (setjmp, longjmp, clone, vfork system call parameter passing, semaphore lock) mainly in **uClibc/libc/sysdeps/arch** directory.
- Had some nasty bugs with fork and clone due to stupid assembler mistake.

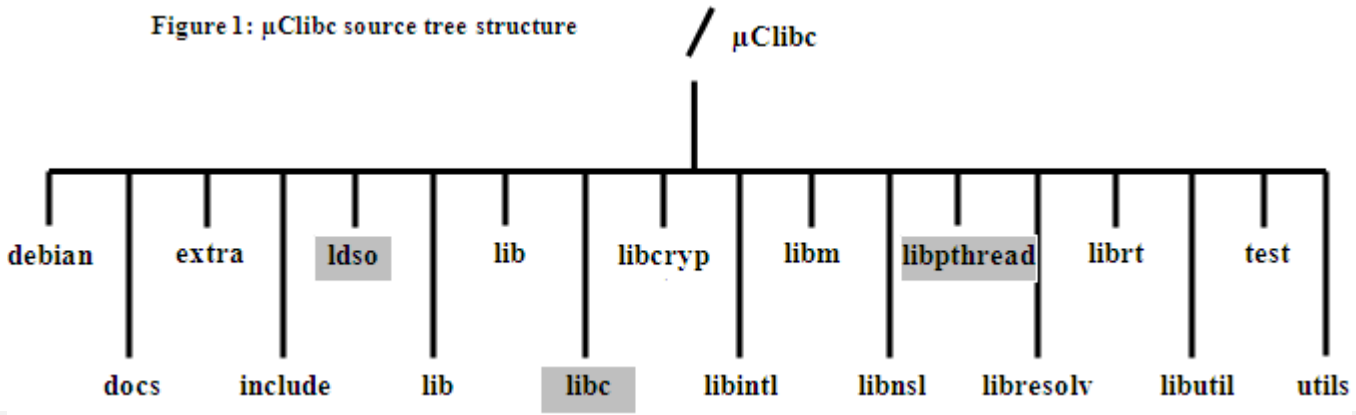
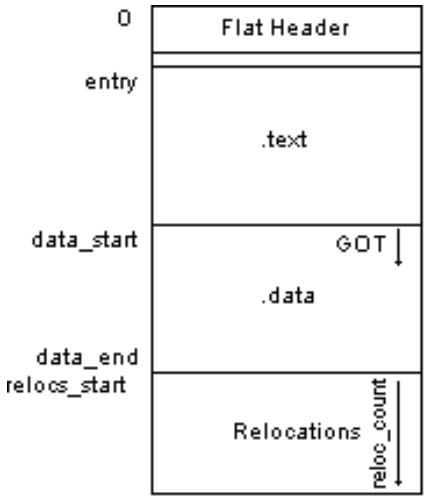


Figure 1: µClibc source tree structure

elf2flt and kernel binflat loader

- Tool to convert elf binaries to the NOMMU 'bin flat' format.
- Binflat format enables binaries to be 'relocated' at runtime into their own unique address space.
- Patches up PC relative address during the conversion.
- Makes an entry in relocation table for other address which need to be adjusted depending on where in memory binary is loaded.
- Must match up correctly to 'bin flat' kernel loader
- Turned out to be a big (& time consuming) part of the port (especially debugging!)
- Crucial for any user space apps!

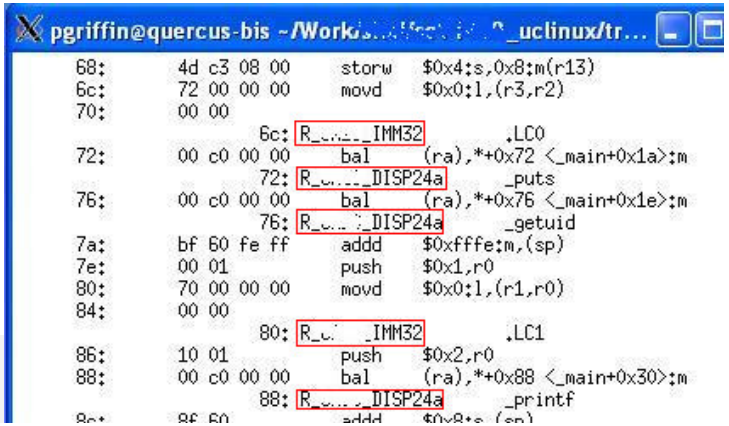
Bin flat file format



Example instruction encoding for 24 bit pc relative instruction

			16 0 word1 at addn		16 0 word2 at addn+2				16 0 word3 at addn+4	
format name	# wrds	oc size	byte 1	byte 0	byte 3		byte 2		byte 5	byte 4
escape3_20	3	16+4[5]	opcode16		p4_4	p3_20	p2_4	p1_4	p3_20	

Objdump disassembly showing reloc information



Major Project Milestones

1. The first (properly formatted) printk
2. The first kernel thread and switching between them successfully
3. Timer interrupts & BogoMIPS value (good performance)
4. Trying to execute init (kernels booting ☺)
5. First 'hello world' user application (uClibc & elf2flt kind of working a bit)
6. Busybox shell & pthread applications (Most stuff working, bug hunting)

Major kernel bugs / war stories

1. Signals
2. High resolution timer bug
3. Stack corruption under load
4. Out of Memory (OOM) bug

Also what caused them, and how we found them.

Kernel Bugs

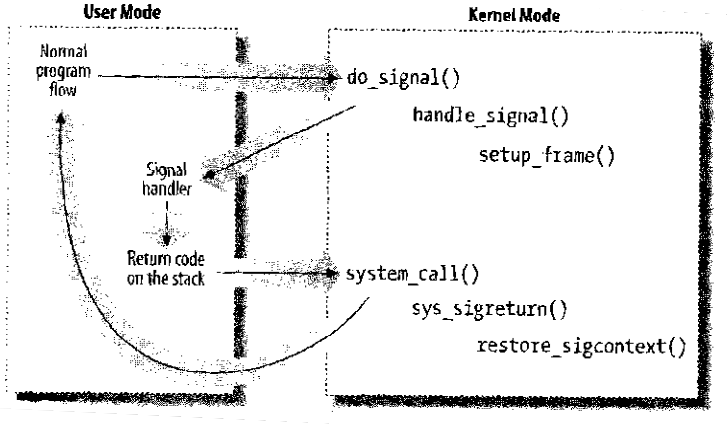
Debugging Linux Signals – The software Interrupt

Symptom: Undefined instruction, when running pthread applications.

Returning to the kernel after a Signal handler is implemented in Linux by inserting a trap instruction into the user tasks stack frame, and pointing the return address register at it.

Solution: Caused by cache incoherency , and incorrect cache flushing code,.

An interesting bug as JTAG debugger does not help you.....(much).




```
movd       $0x0, (r1,r0)
movd       __NR_sigreturn,(r1,r0)
excp       svc
```

```
put_user(0x5400,f->retcode + 0));
put_user(0x54b0,f->retcode + 2));
put_user((__NR_sigreturn & 0xff),
          (f->retcode + 4));
put_user(0x00c5,frame->retcode + 6));
```

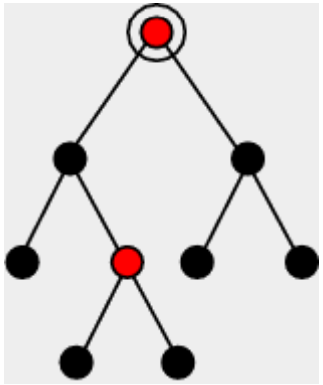

*Diagram from 'Oreilly Understanding the Linux Kernel '

Kernel Bugs

High resolution timers (hrtimer) bug

Symptom: When several user applications call ‘sleep’ system calls, and the system is under some load. System fails with an undefined instruction.

Debug methods: Mainly kernel ‘printk’ to isolate problem, with verbose debugging. JTAG watchpoints and breakpoints.



- hrtimers are stored in a ‘red & black binary tree’ (rbtree).
- The **colour** is stored in the lowest bit of the rb_node parent pointer.
- However kernel masks bottom 2 bits, when reading node colour.
- This masking changes the address of the structure on the stack by 2!

Kernel Bugs

Kernel Stack Corruption

Symptom: When system was under heavy load, kernel would crash with an undefined instruction.

Debug methods: printk, JTAG watchpoints and breakpoint, defensive programming

2 types of stack corruption

1. Not properly protecting the Interrupt stack pointer in the vmlinux.lds linker script.
2. A bug with the process switching assembler (switch_to assembler function) that swaps processor contexts. Not allocating enough room on the stack in the assembler function, causing corruption of local parameters in the schedule function. However, this corruption only caused a system failure under very heavy load, when a certain condition in the scheduler was met, which caused a goto branch to the top of the schedule function.

Kernel Bugs

Out of Memory (OOM) bug

Symptom: When OOM, system crashes with undefined instruction

Debug methods: Kernel printk and JTAG debugger.

With vfork system call, parent should remain suspended until child has called _exit or exec system call. Until this point they share the same stack.

Parent gets informed exec call has succeeded when it is still able to go wrong (memory allocation fails). Both parent and child return to the same stack.

Solution: A bug in the ‘bin flat’ binary loader in the kernel. Fixed by sending a kill signal to the child if any of the memory allocations fail. Then only the parent returns.

Enhancements

Performance Enhancements

Performance good but only 16Mb program space available on this architecture. This must hold kernel, root filesystem, and running binaries.

Not very much free system RAM available when running kernel, shell, and VOIP application using fully relocatable flat binaries.

Toolchain improvements

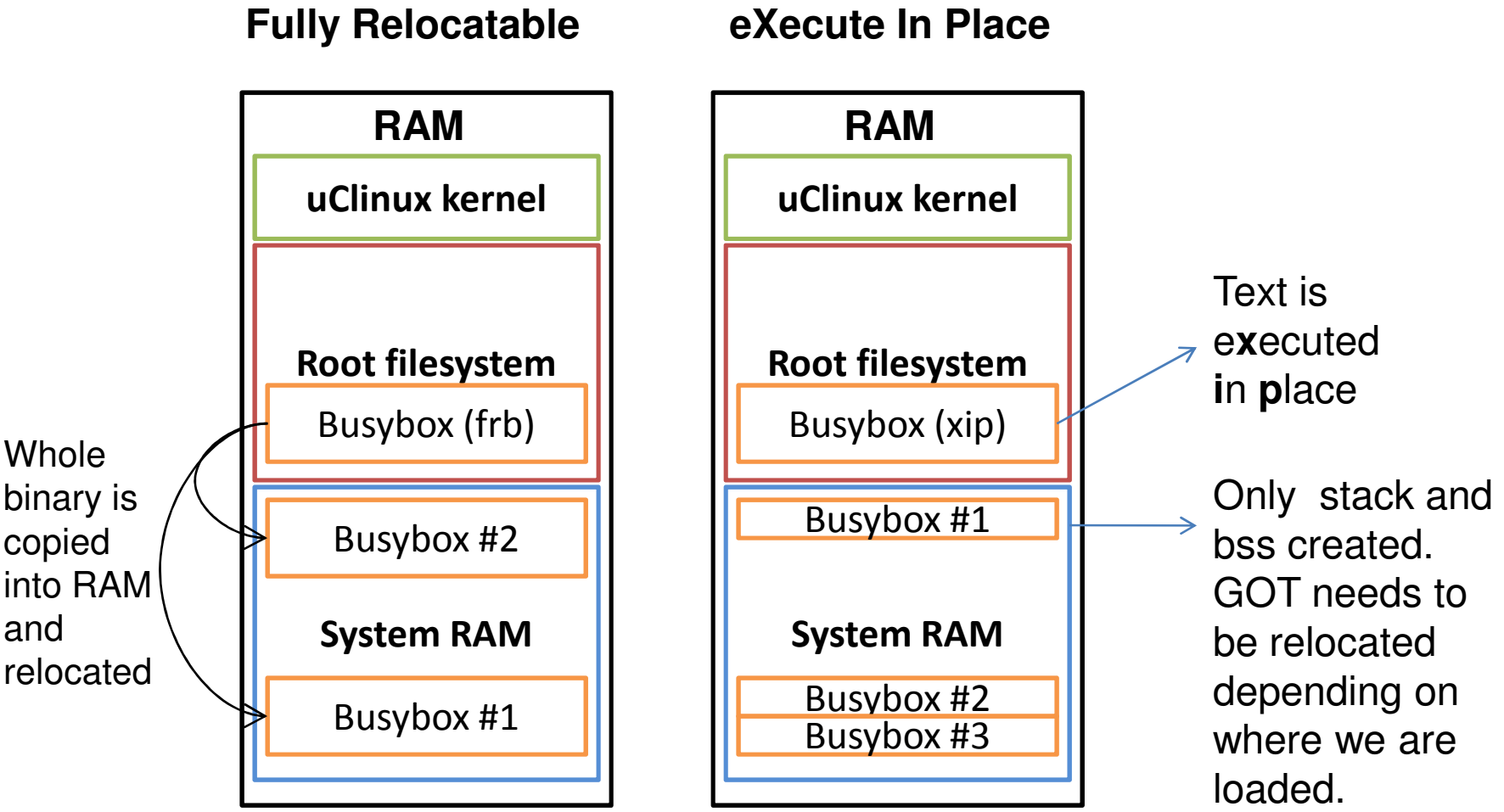
1. Better relaxation support in the linker to shorten branches and absolute memory addresses.
    - •CR16C core has 2, 4, or 6 byte instructions, so lots of scope for relaxing.
    - •Extra relocations types to implement in elf2flt (disp8, disp16)
    - •**8-10%** saving in binary size

XIP

2. Position Independent Code (PIC) support in the toolchain makes XIP possible.

•XIP allows several copies of a binary to run without duplicating the text segment
•Good for programs where there might be multiple instances running (e.g. Busybox)
•Allows impressive RAM savings

•Implemented using David Howells NOMMU mmap patches to romfs and mtd subsystem
    •Mtdram driver
    •Romfs root filesystem (so files are continuous)



With XIP

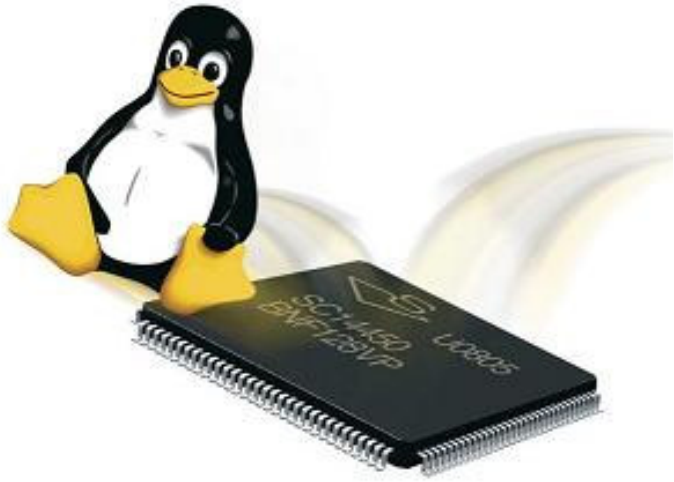
- Many more instances can be run before RAM is exhausted
- No copying of text from root filesystem
- Slight decrease in performance due to GOT indirection.

Delivering Software Innovation

The platform

The platform now

Uboot bootloader
uClinux 2.6 kernel
Alsa sound drivers, SPI, MTD flash, LCD, USB
SIP stack, and other libraries
SiTel VOIP app
Full documentation



Performance figures

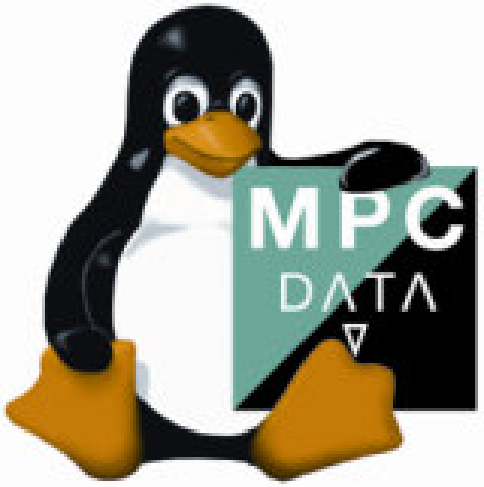
- 56 Dhrystone MIPS from user space (after relax)
- 47 Dhrystone MIPS from user space running XIP
- 5% CPU utilisation during a SIP call
- 12-14% CPU utilisation when doing the conferencing
- 56 milli amps during a voip phone call
- 1.5mb (kernel & file system), 836K compressed

Please visit us at the technical showcase (Kernhem 4) for DECT / VOIP demonstrations!

Q & A

Thank you for your attention

Peter Griffin
pgriffin@mpc-data.co.uk



MPC Data Limited
County Gate, County Way, Trowbridge, Wiltshire BA14 7FJ, United Kingdom

Phone: +44 (0) 1225 710600
Fax: +44 (0) 1225 710601
Web: www.mpc-data.co.uk