Embedded Linux Conference Europe 2014

Porting Linux to a New Architecture

Marta Rybczyńska

October 15, 2014

AGILE
PERFORMANCE

Different Types of Porting

- New board
- New processor from existing family
- New architecture

New Architecture: What it Means?

- Processor instruction set
  - Compile
  - Write the assembly parts
- Memory map: different peripherals
  - Configure drivers
  - Write new drivers
- Optimizations
  - New opportunities
  - Write optimized code

Porting Linux: Basic Elements

- Build tools
  - Gcc, binutils...
- The kernel
  - Core code
  - Drivers
- Libraries
  - Libc, libm, pthread, ...
- User space
  - Busybox, applications

One day..

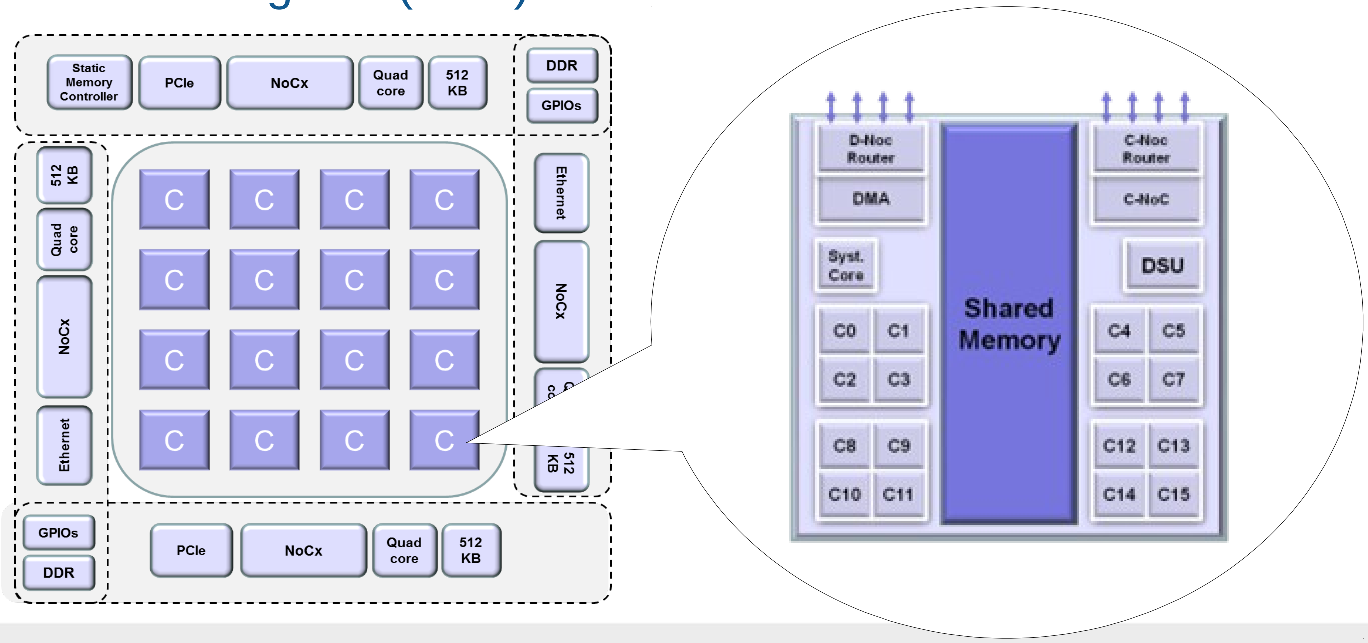
you have a new architecture

First MPPA®-256 Chips with TSMC 28nm CMOS
256 Processing Engine cores + 32 Resource Management cores

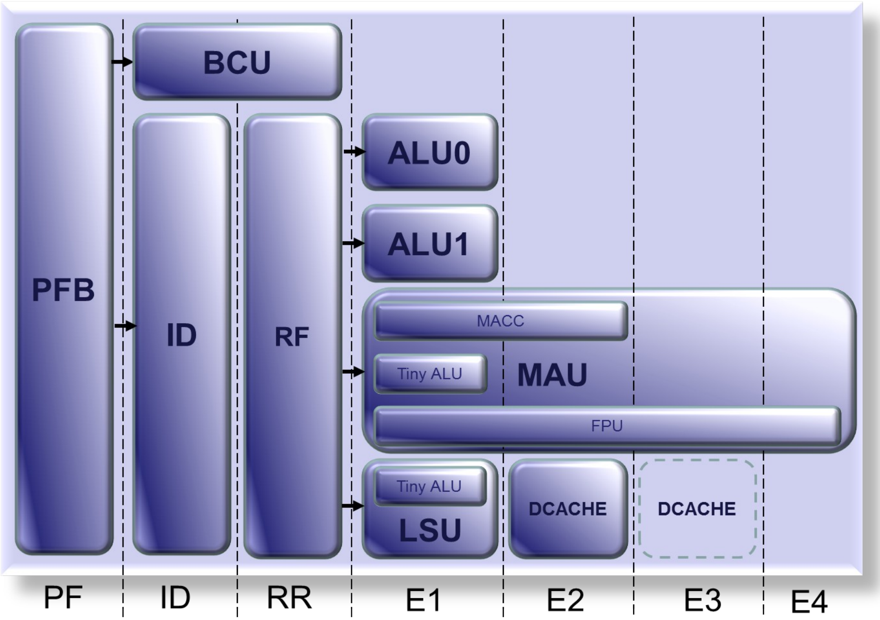
- 256 (+32) user-programmable, generic cores
- Architecture and software scalability
- High processing performance
- High energy efficiency
- Execution predictability
- PCIe Gen3, Ethernet 10G, NoCX

The MPPA-256 Processor

- Compute cluster includes:
  - 16+1 cores
  - Shared memory
  - Network-on-Chip Interfaces
  - Debug unit (DSU)
- IO cluster includes:
  - 4 cores
  - Shared memory
  - Peripherals



The MPPA-256 Processor Core ISA



- Same on IO and compute cluster
- 5-issue Very Long Instruction Word (VLIW)
- DSP instructions
- Advanced bitwise instructions
- Hardware loops
- MMU
- Idle modes
- 32/64-bit IEEE 754 floating point unit

```
mkdir linux/arch/k1
```

The Initial Files: Less Than You Expect

- Processor startup
  - Configure the core
- Memory map
  - Initialize the memory allocators
  - Configure memory zones
- Processor mode change
  - Interrupts and traps
  - Clock interrupt
  - Context switch
- Device tree and KConfig
- Console (printk)

How To Write It?

- Read documentation
- Copy & paste
- Understand & write

Assembly vs C code

- K1 core is a VLIW: multiple instructions (one bundle) per cycle

- High performance gain
  - GCC handles it well
  - Manual bundling OK for short code, hard for longer ones

- Result
  - Preferring built-ins over asm inlines
  - Less assembly in the code

```
__mcount:
        add $r53 = $r33, 16
        copy $r40 = $r33
        get $r41 = $sr0
        ;;
#ifdef CONFIG_K1_TRACES
# Generate HW trace with 2x32 bit values
# args: r40, r41
__mcount_tracepoint:
        get $r38 = $pcr
        make $r35 = 0x1 ## tracepoint name
        make $r34 = 136
        ;;
        insf $r34 = $r35, 31, 16
        extfz $r38 = $r38, 15, 11
        ;;
        srl $r35 = $r35, 16
        insf $r34 = $r38, 12, 8
        ;;
        make $r33 = 0
        copy $r32 = $r40
        copy $r38 = $r41
        make $r40 = 1879588896
        ;;
        copy $r39 = $r33
        slld $r32:$r33 = $r32:$r33,16
        or $r36 = $r34, 5
        ;;
```

Failed to execute /init

Kernel panic - not syncing. No init found

Time to Bring User Space Up

- Port libc (if not done already)
  - Which one? It depends...
  - For K1, we've ported uClibc
- First init can be statically linked
  - If not, dynamic loader needed first

Interface User<->Kernel (ABI)

- Program startup
  - Which values in which registers?
  - What is on the stack?
- Syscalls
- Signals

KALRAY

Instruction Set Simulator: Boot Process Debugging

```
>k1-cluster --mcluster=ioddr -- vmlinux
Compiled-in FDT at 0x8001a0a0
Linux version 3.10.0+ (mrybczyn@doros) (gcc version 4.7.4 20130620 (prerelease) [Kalray Compiler unknown af8028d-dirty] (GCC) ) #1 SMP Fri Aug 23 13:42:41 CEST 2013
Cpu clock: 400MHz
setup_memory: Memory: 0x80000000-0x84000000
bootmem::init_bootmem_core nid=0 start=802d1 map=802d1 end=84000 mapsize=7a8
bootmem::mark_bootmem_node nid=0 start=802d1 end=84000 reserve=0 flags=0
bootmem::__free nid=0 start=802d1 end=84000
bootmem::mark_bootmem_node nid=0 start=802d1 end=802d2 reserve=1 flags=0
bootmem::__reserve nid=0 start=802d1 end=802d2 flags=0
Reserved - 0x83fff960-0x000006a0
bootmem::mark_bootmem_node nid=0 start=83fff end=84000 reserve=1 flags=0
bootmem::__reserve nid=0 start=83fff end=84000 flags=0
bootmem::alloc_bootmem_bdata nid=0 size=80000 [128 pages] align=20 goal=80000000 limit
bootmem::__reserve nid=0 start=802d2 end=80352 flags=1
bootmem::alloc_bootmem_bdata nid=0 size=8 [1 pages] align=20 goal=80000000 limit=0
bootmem::__reserve nid=0 start=80352 end=80353 flags=1
bootmem::alloc_bootmem_bdata nid=0 size=600 [1 pages] align=20 goal=80000000 limit=0
bootmem::__reserve nid=0 start=80353 end=80353 flags=1
bootmem::alloc_bootmem_bdata nid=0 size=4d [1 pages] align=20 goal=80000000 limit=0
bootmem::__reserve nid=0 start=80353 end=80353 flags=1
bootmem::alloc_bootmem_bdata nid=0 size=4d [1 pages] align=20 goal=80000000 limit=0
bootmem::__reserve nid=0 start=80353 end=80353 flags=1
bootmem::alloc_bootmem_bdata nid=0 size=1000 [1 pages] align=20 goal=80000000 limit=0
bootmem::__reserve nid=0 start=80353 end=80354 flags=1
bootmem::alloc_bootmem_bdata nid=0 size=1000 [1 pages] align=20 goal=80000000 limit=0
bootmem::__reserve nid=0 start=80354 end=80355 flags=1
bootmem::alloc_bootmem_bdata nid=0 size=20000 [32 pages] align=1000 goal=80000000 limi
bootmem::__reserve nid=0 start=80355 end=80375 flags=1
bootmem::mark_bootmem_node nid=0 start=8035a end=8035d reserve=0 flags=0
bootmem::__free nid=0 start=8035a end=8035d
bootmem::mark_bootmem_node nid=0 start=80362 end=80365 reserve=0 flags=0
bootmem::__free nid=0 start=80362 end=80365
bootmem::mark_bootmem_node nid=0 start=8036a end=8036d reserve=0 flags=0
bootmem::__free nid=0 start=8036a end=8036d
bootmem::mark_bootmem_node nid=0 start=80372 end=80375 reserve=0 flags=0
bootmem::__free nid=0 start=80372 end=80375
PERCPU: Embedded 5 pages/cpu @80355000 s6304 r0 d14176 u32768
bootmem::alloc_bootmem_bdata nid=0 size=4 [1 pages] align=20 goal=80000000 limit=0
bootmem::__reserve nid=0 start=8035a end=8035b flags=1
bootmem::alloc_bootmem_bdata nid=0 size=4 [1 pages] align=20 goal=80000000 limit=0
bootmem::__reserve nid=0 start=8035b end=8035b flags=1
bootmem::alloc_bootmem_bdata nid=0 size=10 [1 pages] align=20 goal=80000000 limit=0
bootmem::__reserve nid=0 start=8035b end=8035b flags=1
bootmem::alloc_bootmem_bdata nid=0 size=10 [1 pages] align=20 goal=80000000 limit=0
bootmem::__reserve nid=0 start=8035b end=8035b flags=1
bootmem::alloc_bootmem_bdata nid=0 size=78 [1 pages] align=20 goal=80000000 limit=0
bootmem::__reserve nid=0 start=8035b end=8035b flags=1
bootmem::alloc_bootmem_bdata nid=0 size=2c [1 pages] align=20 goal=80000000 limit=0
bootmem::__reserve nid=0 start=8035b end=8035b flags=1
bootmem::mark_bootmem_node nid=0 start=80353 end=80353 reserve=0 flags=0
bootmem::__free nid=0 start=80353 end=80353
bootmem::mark_bootmem_node nid=0 start=80354 end=80354 reserve=0 flags=0
bootmem::__free nid=0 start=80354 end=80354
Built 1 zonelists in Zone order, mobility grouping on.  Total pages: 16256
Kernel command line: dhash_entries=1024 ihash_entries=1024 bootmem_debug=1 init=/init
bootmem::alloc_bootmem_bdata nid=0 size=400 [1 pages] align=20 goal=80000000 limit=0
bootmem::__reserve nid=0 start=8035b end=8035b flags=1
```

```
        2042427:        0x800cd6a0:     lw $r8(0x80005150) = 44[$r12(0x8002bfa0)] [V@ 0x8002bfcc ; P
        2042428:        0x800cd6a8:     lw $r10(0x8028a374) = 16[$r12(0x8002bfa0)] [V@ 0x8002bfb0 ;
        2042429:        0x800cd6ac:     lw $r15(0x80016630) = 20[$r12(0x8002bfa0)] [V@ 0x8002bfb4 ;
        2042430:        0x800cd6b0:     ld $r16r17(0x8026605480266048) = 24[$r12(0x8002bfa0)] [V@ 0x8
        2042431:        0x800cd6b4:     set $ra(0x80005150) = $r8(0x80005150)
        2042431:        0x800cd6b4:     ld $r18r19(0x80315280) = 32[$r12(0x8002bfa0)] [V@ 0x8002bfc0
        2042432:        0x800cd6bc:     ret
        2042432:        0x800cd6bc:     add $r12(0x8002bfc8) = $r12(0x8002bfa0), 40
        2042432:        0x800cd6bc:     lw $r20(0x0) = 40[$r12(0x8002bfa0)] [V@ 0x8002bfc8 ; P@ 0x800
}}} __register_sysctl_paths
}}} register_sysctl_paths
}}} register_sysctl_table
        2042433:        0x80005150:     make $r0(0x0) = 0
        2042433:        0x80005150:     add $r12(0x8002bfd0) = $r12(0x8002bfc8), 8
        2042433:        0x80005150:     lw $r8(0x80001804) = 16[$r12(0x8002bfc8)] [V@ 0x8002bfd8 ; P
        2042434:        0x8000515c:     set $ra(0x80001804) = $r8(0x80001804)
        2042435:        0x80005160:     ret
}}} sysctl_init
}}} proc_sys_init
}}} proc_root_init
        2042436:        0x80001804:     call 2079364
{{{ rest_init
        2042437:        0x801fd288:     add $r12(0x8002bfc8) = $r12(0x8002bfd0), -8
        2042437:        0x801fd288:     make $r1(0x0) = 0
        2042437:        0x801fd288:     make $r2(0xa00) = 2560
        2042437:        0x801fd288:     make $r0(0x801fd308) = -2145398008
        2042438:        0x801fd29c:     get $r8(0x80001808) = $ra(0x80001808)
        2042439:        0x801fd2a0:     call -1882616
        2042439:        0x801fd2a0:     sw 16[$r12(0x8002bfc8)] = $r8(0x80001808) [V@ 0x8002bfd8 ; P
{{{ kernel_thread
        2042440:        0x800318a8:     copy $r3(0x0) = $r1(0x0)
        2042440:        0x800318a8:     copy $r4(0x801fd308) = $r0(0x801fd308)
        2042440:        0x800318a8:     or $r0(0x800b00) = $r2(0xa00), 8388864
        2042441:        0x800318b8:     copy $r2(0x0) = $r3(0x0)
        2042441:        0x800318b8:     make $r3(0x0) = 0
        2042441:        0x800318b8:     copy $r1(0x801fd308) = $r4(0x801fd308)
        2042442:        0x800318c4:     goto -652
        2042442:        0x800318c4:     copy $r4(0x0) = $r3(0x0)
{{{ do_fork
        2042443:        0x80031638:     get $r8(0x801fd2a8) = $ra(0x801fd2a8)
        2042443:        0x80031638:     add $r12(0x8002bfa0) = $r12(0x8002bfc8), -40
        2042444:        0x80031640:     and $r0(0x0) = $r0(0x800b00), 29, 28
        2042444:        0x80031640:     copy $r15(0x800b00) = $r0(0x800b00)
        2042444:        0x80031640:     sw 20[$r12(0x8002bfa0)] = $r15(0x80016630) [V@ 0x8002bfb4 ;
```

init started: BusyBox v1....

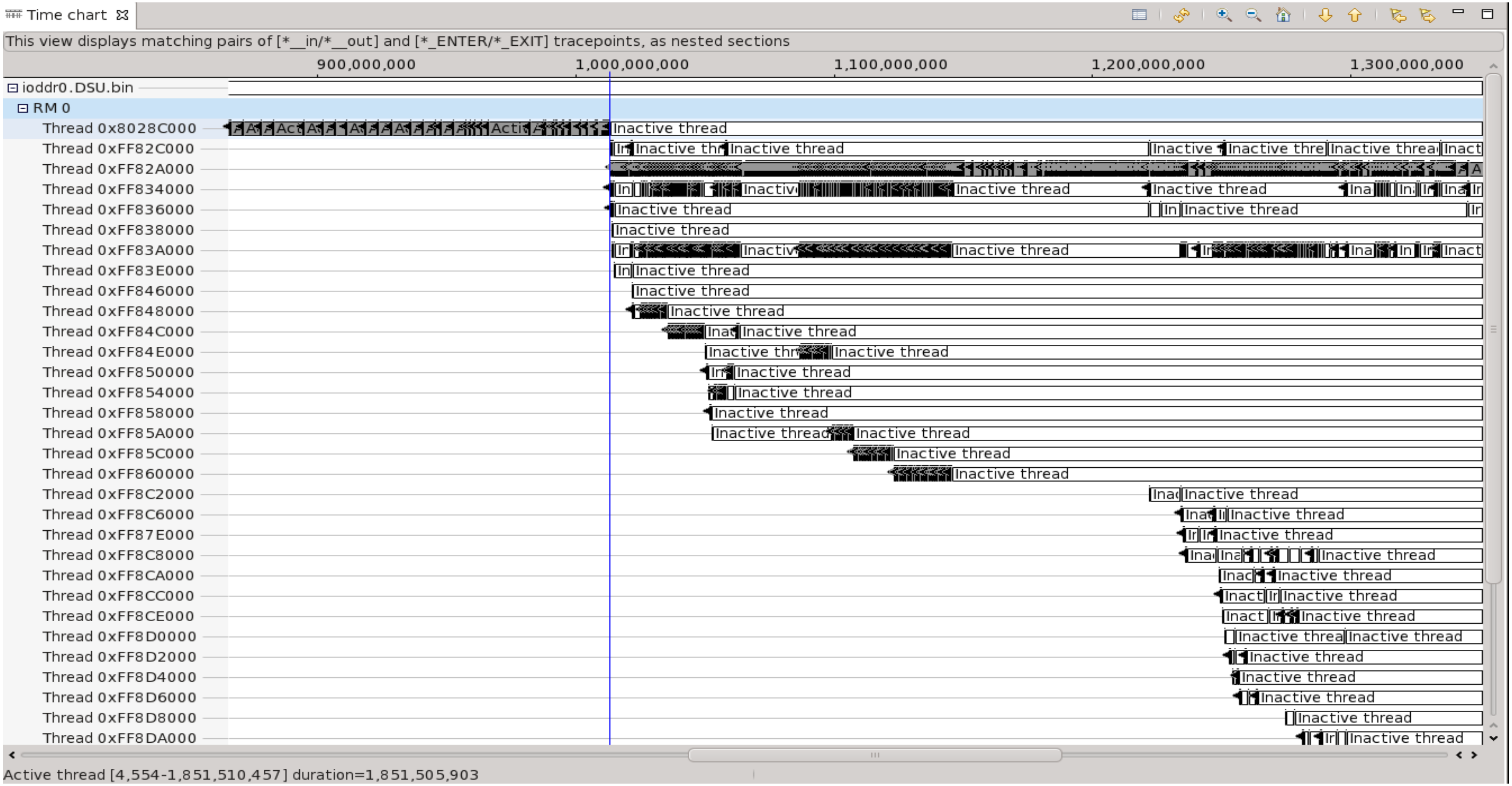
First Executables

- First static libraries
- Then dynamic loader
- And some drivers

Early Testing

- Unit tests for the kernel space
  - Complicated build
- Debugging ease is important
  - Best if run in simulator
- “Test” init
  - Basic tests of all main functionalities in an “init”

Traces: Visualization



Later Testing

- “Do it yourself”
  - Too much work
  - What is the expected behaviour?
- Use existing testsuites
  - For K1, we use LTP (Linux Testing Project)
  - Active, big number of tests at different level

```
open("/lib/libm.so.0", O_RDONLY) = 3
```

Enabling New Functions

- Examples
  - Traces
  - New file system
  - New device type
- New functionality requires
  - New kernel options
  - Support in kernel headers
  - Support in libc
- Try Test-Driven-Development

KALRAY

New Functionality Example: Strace and Ptrace (1)

- Strace
  - See syscalls run by a program
  - Shows both parameters and results
  - Useful for debugging errors
- Implementation
  - Ptrace calls
  - Signals

New Functionality Example: Strace and Ptrace (2)

- Unit tests
  - Available in LTP
- Strace implementation
  - The code compiles but...
  - Defines in the code

Supporting your hardware well

Special Cases

- SMP
- MMU
- Network-on-Chip
- Multiple address spaces
  - Device-tree

Building a distribution

Distribution Choices

- Do-it-yourself
- Buildroot
- Yocto

Summary: Lessons Learned (1)

- Divide the port in stages
- Test early

Summary: Lessons Learned (2)

- Use generic functionality if possible
- Keep the coding style

Summary: Lessons Learned (3)

- Use panic() and exit()
- Prefer code that doesn't compile if architecture unknown

Summary: Lessons Learned (4)

- Use and develop advanced debugging techniques
- Read documentation
- Read other platforms code

Questions?

Marta Rybczynska
marta.rybczynska@kalray.eu

http://www.kalray.eu