Agenda

- Architecture
- Linux Port
  - Core
  - Peripherals
- Debugging
- Summary and Future Plans
Agenda

- Architecture
- Linux Port
  - Core
  - Peripherals
- Debugging
- Summary and Future Plans
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 (1)

- 16 compute clusters
- 4 Input/Output clusters with rich peripheral access
  - DDR, PCI Express, Ethernet, Flash, GPIO, SPI, I2C etc
- Connected by a Network-on-Chip (NoC)
The MPPA-256 Processor (2)

- 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)
- 32/64-bit IEEE 754 floating point unit
- DSP instructions
- Advanced bitwise instructions
- Hardware loops
- MMU
- Idle modes
Kalray Software Development Kit for MPPA-256 (1)

- Standard Embedded C/C++ Programming
  - GCC, GNU binutils, newlib, uClibc, ...
- Simulators, Profilers, Debuggers & System Trace
  - GDB
  - Cycle-based simulator
  - Hardware System trace
- Operating systems & Device Drivers
  - NodeOS on the compute clusters
    - One thread per core
  - RTEMS port on the IO clusters
  - PCI Express driver for the Linux Host
Kalray Software Development Kit for MPPA-256 (2)

- Programming Models
  - Dataflow Programming
    - FPGA Style
  - POSIX-level Programming
    - DSP Style
  - Streaming Programming
    - GPU Style
The MPPA POSIX Programming Model

- POSIX-like process management
  - Spawn 16 processes from the IO cluster
  - Process execution on the 16 clusters start with main(argc,argv) and environment
  - On each cluster, support to up to 16 concurrent threads

- Inter-process communication (IPC)
  - POSIX file descriptor operations on 'NoC Connectors'
  - Extension to the PCIe interface with the 'PCle Connectors'
  - Rich communication and synchronization

- Multi-threading inside clusters
  - Standard GCC/G++ OpenMP support
  - POSIX threads interface
The MPPA & Linux

- High-performance SMP on the IO clusters
- Device Drivers
  - SPI, I2C, GPIO (sensors, small peripherals)
- Full network stack
- Existing user libraries
- Rich configuration options
Agenda

- Architecture
- Linux Port
  - Core
  - Peripherals
- Debugging
- Summary and Future Plans
History

- 2.6.33-rc4
- 2.6.33
- 2.6.35
- 3.2
  - SMP, first specific drivers
- 3.10
  - tracing, MMU (in progress)
Features

- **The kernel**
  - Single core or SMP
  - Device-tree, generic headers
  - Supports userspace with FDPIC
  - Initial tracing support

- **Drivers**
  - Generic drivers tested
  - Specific drivers
    - PCI Express, console,...

- **Userspace: buildroot-based with custom toolchain & uClibc**
Agenda

- Architecture
- Linux Port
  - Core
  - Peripherals
- Debugging
- Summary and Future Plans
Optimization Case: Atomic Ops

- Single core case: disable/enable interrupts
  - Cheap on MPPA
- SMP case: compare-and-swap
  - Huge performance/code size impact
  - Last version uses implementation details of the caches and the write buffer
- Return from experience
  - Improved atomic ops for the next version of the core

```c
static inline void atomic_set(atomic_t *v, int i) {
    __k1_atomic_visibility_pre();
    __k1_umem_write32((void *) &v->counter, i);
    __builtin_k1_fence();
    __k1_atomic_visibility_post(&v->counter);
}
```
Lessons Learned 1: Read the Specs Carefully (1)

- SMP version started deadlocking
- Spinlock ticket value showed corruption
- Debugging
  - Spinlock ordering problem?
    - Careful platform code analysis
  - Enabled spinlock debug
  - Detailed simulator trace analysis
- Reason

```c
static inline
unsigned long __xchg_u32(volatile void *ptr, unsigned long x)
{
    unsigned long old;
    ...
    do {
        old = __k1_umem_read32((void *) ptr);
    } while (cmpxchg((unsigned int *) ptr, old, x) != old);

    return x; /* original */
    return old; /* fixed */
}
```
Lessons Learned 2: Use the GCC

- 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
Agenda

- Architecture
- Linux Port
  - Core
  - Peripherals
- Debugging
- Summary and Future Plans
Interrupt Controllers

- Two-level interrupt control
  - One private to each core (timers, NoC)
  - Shared between 4 cores (PCI Express, Ethernet, other peripherals)

- Basic operations on the core interrupt controller

- Multiple devices on the same line
  - Configuration issue: configuring lanes for devices
Memory Areas

- Two areas
  - Big DDR
  - Small internal shared memory
- Different latency, optimization opportunity
- Shared memory is visible through PCI Express
- Solution:
  - Allocation in the DDR by default
  - Separate allocator for the shared memory
The PCI Express Driver (1)

- PCI Express is the main interface to the MPPA
  - Host (Linux) driver is ~10KLOC
  - Boot, synchronization, DMA, message passing,...
  - Two interfaces per MPPA (visible as two PCI Express devices)

- Host-side framework in Linux is mature
- Less standard on the device side
  - Code reuse (protocols)
  - First synchronization in the bootloader
The PCI Express Driver (2)

- PCI Express allows memory-mapping on Host of memory zones of the device
- Hardware resources
  - Doorbell registers (shared, in BAR0)
  - Shared memory accessible by the Host directly (BAR2)
  - Both DDR and shared memory accessible by DMA
- Cache effects
  - Shared zones negotiated (especially shared memory)
Boot

- Using custom bootloaders at the moment
- Boot of IO cluster by PCI Express
  - Initiated by the Host
  - First code in shared memory
  - Then complete image in shared memory+DDR
- Boot of clusters from the IO
  - Cluster executables in IO cluster memory
  - Also transferred by PCI Express
Network-on-Chip

- The way to communicate between clusters
  - High performance interface
  - Shared resources

- Used in different places
  - Boot, drivers in the kernel space
  - User code (IPC)
  - No Linux kernel API to reuse
The Ethernet Device Drivers

- Up to 8×10Gbit/s
  - MAC controller, PHY
  - Uses NoC

- Distributed network stack
  - Packet dispatch to different clusters
  - Potentially applications using different protocols in different clusters
Agenda

- Architecture
- Linux Port
  - Core
  - Peripherals
- Debugging
- Summary and Future Plans
Traces

This view displays matching pairs of [*_in/*_out] and [*_ENTER/*_EXIT] tracepoints, as nested sections.
printf(), printk() and GDB

End of assembler dump.
(gdb) s
0x000f58c in __mcount ()
(gdb) x/20x0 0x00f59c in __mcount ()
(gdb) s
0x000f59c in __mcount ()
(gdb) s
Single stepping until exit from function __mcount, which has no line number information.
_switch_to (prev=0xff28000, next=0xff282a0) at arch/kl/kernel/switch.c:15
(gdb) s
in arch/kl/kernel/switch.c
(gdb) s
__knlb () at arch/kl/kernel/switch.c:15
(gdb) s
in arch/kl/kernel/switch.c
(gdb) bt
__knlb () at arch/kl/kernel/switch.c:15
__switch_to (prev=0xff28000, next=0xff282a0) at arch/kl/kernel/switch.c:15
0x08231800 in context_switch (next=0x8f282a0, prev=0x8f28000, rq=0x800f756) at kernel/sched/core.c:2018
__schedule () at kernel/sched/core.c:3019
0x800f211b0 in sched_submit_work (tsk=0x8f28000) at kernel/sched/core.c:3046
schedule () at kernel/sched/core.c:3045
0x8023880c in schedule_timeout (timeout=0x8224000) at kernel/trigger/c:1445
0x80231400 in do_wait_for_common (state=0x8224000, timeout=0x8240000) at kernel/sched/core.c:3311
0x800f211b0 in _wait_for_common (state=0x8224000, timeout=0x8224000) at kernel/sched/core.c:3359
0x800f211b0 in kthread_create_on_mode (thread=0x800f2478, smp_thread_fn, data=0x8002210, name='smp_thread', softirq=0x800f215) at kernel/kthread.c:270
0x800f211b0 in kthread_create_on_cpu (thread=0x800f2478, smp_thread_fn=0x800f2210, cpumask='smp_thread', data=0x800f2210, cpu=0x800f2210, name='smp_thread') at kernel/kthread.c:325
0x800f211b0 in smp_thread_create (ht=0x800f28ac, cpu=0x82473420, smp_thread_fn=0x800f2210) at kernel/smpboot.c:1324
0x800f211b0 in smpboot_register_per_cpu_thread (percpu_thread=0x800f2210, kernel/smpboot.c:284
0x800f211b0 in smpboot_register_per_cpu_thread (percpu_thread=0x800f2210, kernel/smpboot.c:284
0x800f211b0 in do_one_initcall (fn=0x800f21e0, smp=0x800f2210) at kernel/init/main.c:666
0x800f211b0 in do_one_initcalls () at kernel/init/main.c:787
0x800f211b0 in kernel_init_freeable () at kernel/init/main.c:874
0x800f211b0 in kernel_init() (unused=optimized out) at kernel/init/main.c:813
0x800f211b0 in ret_from_kernel_thread () (gdb)
The Simulator
Agenda

- Architecture
- Linux Port
  - Core
  - Peripherals
- Debugging
- Summary and Future Plans
Lessons Learned 3: What was Difficult

- Documentation missing
  - With great exceptions!
- Need to read other platform code
  - No examples (yet) for our architecture
- Reaching limits of the existing software
  - Not always a bug in the Linux port
- Trying to be mainline-compatible
- Initial port hard to split between developers
- Time-consuming, needed alternative for early testing (RTEMS port)
Lessons Learned 4: What was Easy

- Debugging
  - Mixed mode depending on the testcase: simulator, gdb, FPGA
  - Simulator trace postprocessing
- Generic headers cover a good part of the code
- Recently merged architectures give good examples
Future Plans

- Complete driver support (Ethernet, NoC, others...)
- Optimized MMU
- Replace generic implementations with optimized ones
- Public release
- Mainlining
Marta Rybczynska
marta.rybczynska@kalray.eu

http://www.kalray.eu