Multicore Application Development with Zephyr RTOS

Alexey Brodkin,
Engineering manager, Synopsys
ELCE 2019, October 28, 2019
Alexey Brodkin
Engineering manager @ Synopsys

Open Hub says:
• Most experienced in C
• First commit about 7 years ago
• Most recent commit about 1 month ago
• Has made 642 commits
• Most contributions to:
  – U-Boot
  – Linux kernel
  – Buildroot
  – Yocto Project / OpenEmbedded
  – Zephyr
  – uClibc
  – etc
Agenda

• Use-cases for multicore embedded processors
• Multicore support models review: AMP vs SMP
• AMP & SMP in Zephyr
• Challenges with SMP & Zephyr
• Performance scaling on SMP system
Why go multicore?

Increase performance & responsiveness, decrease power consumption

**Challenges**

- Power consumption increases dramatically with increase of clock rate
- Limited MHz budget
  - Limited clock rate for a given tech process
- Multiple critical or resource-hungry task
- Very specific tasks (DSP, CNN, etc.)

**Solution**

- Use multiple CPUs or CPU cores
  - Scale MHz budget
  - Schedule critical tasks on dedicated CPU cores
  - Really parallel execution, not pseudo
- Use separate “accelerator” core(s)
  - Offload specific tasks
  - Use very specialized accelerators
    - DSP
    - Crypto-processor
    - Vector-processor
    - Application-specific instruction-set processor (ASIP)

Increase performance & responsiveness, decrease power consumption
Examples of multicore solutions

A lot of real-life use-cases

- 5G/LTE modems
- Audio & video DSPs
- AI & vision processors
- Human-interface devices
Modern heterogeneous SoC

Combination of multiple different processors

- **Vision processor with CNN**: ARC EV74
- **General purpose CPU cluster**: ARC HS48
- **DSP**: ARC EM11D
- **Sensor hub**: ARC EM4

**Data bus**
- UART
- USB
- Ethernet
- DDR

**Clusters**
- **SMP cluster**
- **AMP cluster**

**Connectivity**
- Shared memory
- Interrupts
- Mailbox

---

© 2019 Synopsys, Inc.
Multicore support models: AMP

Asymmetric multiprocessing

- Completely custom system design
  - Each core might be unique
  - Multiple SW/API standards
- Wide variety of hardware IPCs
  - Shared memory
  - Proprietary mailboxes
  - Cross-core interrupts
  - HW semaphores etc
- Manual SW partitioning
- Non-scalable
  - There’s no simple way to add yet another core
Multicore support models: SMP

Symmetric multiprocessing

- Strong requirements for system design
- CPU cores are equal from SW stand-point*
- Standard HW IPCs, typically
  - Shared memory
  - Coherent caches**
  - Cross-core interrupts
- Complex run-time scheduling
  - Load-balancing
  - Task pinning
- Scalable
  - Just add another core in cluster

* Except early start when there’s a dedicated master and slaves are all the rest
** If CPUs have caches
Why use operating system

Significantly simplifies SW development

Abstractions of hardware

• Processor abstraction
  – Processes
  – Threads

• Peripherals abstraction
  – Complex subsystems
  – Device drivers

Implementation of complex services

• Crypto
• File systems
• Communication
  – Ethernet
  – USB
  – CAN
AMP in Zephyr
Available from the very first commit

- Arduino/Genuino 101*
  - Intel Quark SE SoC
    - Intel x86 core
    - ARC EM4 core
  - Very limited IPCs
    - Shared memory: 80KiB @ 0xA8000000
    - ARC EM start/stop signals via Quark’s SS_CFG regs
- NXP LPCXPRESSO54114
  - Arm Cortex M4F & dual Arm Cortex-M0
- ST STM32H747I Discovery
  - Arm Cortex-M7 & Arm Cortex-M4
- PSoC6 WiFi-BT Pioneer Kit
  - Arm Cortex-M4 & Arm Cortex-M0

* Arduino 101 support got dropped from Zephyr in v2.0.0
** Nano- & microkernels replaced by unified kernel in v1.14
Simplest AMP example
Arduino/Genuino 101

Intel Quark SE

```c
/* Start of the shared 80K RAM */
#define SHARED_ADDR_START 0xA8000000
#define ARC_READY   (1 << 0)
#define shared_data ((volatile struct shared_mem *) SHARED_ADDR_START)

struct shared_mem {
    u32_t flags;
};

int z_arc_init(struct device *arg) {
    /* Start the CPU */
    SCSS_REG_VAL(SCSS_SS_CFG) |= ARC_RUN_REQ_A;

    /* Block until the ARC core actually starts up */
    while ((SCSS_REG_VAL(SCSS_SS_STS) & 0x4000) != 0U) {};

    /* Block until ARC's quark_se_init() sets a flag
    * indicating it is ready, if we get stuck here ARC has
    * run but has exploded very early */
    while ((shared_data->flags & ARC_READY) == 0U) {};
}
```

ARC EM4

```c
static inline void quark_se_ss_ready(void) {
    shared_data->flags |= ARC_READY;
}

static int quark_se_arc_init(struct device *arg) {
    ARG_UNUSED(arg);
    quark_se_ss_ready();
    return 0;
}
```

```c
SYS_INIT(quark_se_arc_init, POST_KERNEL, CONFIG_KERNEL_INIT_PRIORITY_DEFAULT);
```
OpenAMP in Zephyr

Attempt to standardize interactions in HMP systems

- Life Cycle Management (LCM) – remoteproc
- Inter Processor Communications (IPC) – RPMSg
- Transport abstraction layer – Virtio
- Hardware abstraction layer (HAL) – libmetal
- Communications between different platforms
  - Linux
  - RTOS
    - FreeRTOS
    - NuttX
    - Zephyr
  - Bare-metal
- Currently supported: NXP LPC54114
- Linaro connect presentation
  HKG2018-411: OpenAMP Introduction
OpenAMP in Zephyr cont’d

RPMsg-Lite for devices with limited resources

• Smaller code size
• Simpler API
• May use “static API”
• Modular
• No remoteproc/LCM
• Currently supported: NXP LPC54114
• ELCE 2018 presentation by Diego Sueiro

“Linux and Zephyr “talking” to each other in the same SoC”
SMP in Zephyr: historical overview

Requires OS-wide support, thus only emerging lately

• February 2018
  – Initial support with ESP32 in v1.11.0
    https://github.com/zephyrproject-rtos/zephyr/pull/6061

• February 2019
  – Added x86_64 in v1.14.0
    https://github.com/zephyrproject-rtos/zephyr/pull/9522
  – “…Virtualized SMP platform for testing under QEMU…”

• September 2019
  – ARC supported in v2.0.0
    https://github.com/zephyrproject-rtos/zephyr/pull/17747
  – ARC nSIM instruction set simulator & ARC HS Development Kit board

* ESP32 only supports shared memory, no cross-core IRQs

---

* Author: Benoît Canet
Licensed under CC BY 3.0: https://creativecommons.org/licenses/by/3.0/
SMP in Zephyr: current status

Still in its early days

Ready

• Use of certain HW capabilities
  – Shared memory with coherent caches*
  – Cross-coreIRQs**
  – Cluster-wise wall-clock
  – HW atomic instructions***

• Threads pinning to CPU(s)
  – Via CPU affinity API:
    \( k\_thread\_cpu\_mask\_xxx() \)

Needs more work

• More architectures and platforms
• More tests & benchmarks
• Support for more cores in the cluster
  – Now up-to 4 cores
• Fancier scheduler
  – Account for CPU migration costs
• SMP-aware Zephyr debugging with OpenOCD/GDB

* If CPUs have caches
** For less latencies of premature thread termination
*** Used via compiler built-ins (CONFIG_ATOMIC_OPERATIONS_BUILTIN) or via hand-written implementation in assembly (ATOMIC_OPERATIONS_CUSTOM)
Changes required to support SMP in Zephyr

Most of changes done in architecture & platform agnostic code

- Extra preparations for slave cores
  - Setup idle threads: `init_idle_thread()`
  - Allocate per-core IRQ stacks: `Z_THREAD_STACK_BUFFER(_interrupt_stackX)`
  - Start the core: `z_arch_start_cpu()`

- `irq_lock()`
  - UP: `z_arch_irq_lock()`
  - SMP: `z_smp_global_lock() = z_arch_irq_lock() + spinlock on global_lock`

- Prepare scheduler for SMP
  - `z_sched_abort()`

- `z_arch_switch(new_thread, old_thread)` instead of `z_swap()`
  - Lower-level
  - Scheduler unaware
  - No spinlocks inside architecture-specific assembly code
Hardware requirements for SMP

ARC HS primer

- Same ISA & functionally of all cores
- Shared memory
  - External RAM
  - SRAM
  - DDR
  - On-chip memory
    - Closely/Tightly-Coupled Memories are PRIVATE
    - ARC: ICCM/DCCM
    - ARM/RISC-V: ITCM/DTCM
- Flexible IRQ management
  - Inter-Core Interrupt Unit (ICI)
  - Interrupt Distribution Unit (IDU)
- Cluster-wise clock source
  - Global Free-Running Counter (GFRC)
Challenges designing application for SMP system

Software peculiarities

Task/thread scheduling

• Scheduler type
  – SCHED_DEADLINE
  – SHED_DUMB
  – SCHED_SCALABLE
  – SCHED_MULTIQ

• Migration between CPU cores
  – Automatic core selection (by default)
  – Task/thread pinning
    SCHED_CPU_MASK – only for SHED_DUMB

Shared resources...

• Clock source
• Caches (while managing them)
• Peripherals
  – Serial port
  – SPI, I2C, Ethernet, etc

...add overhead for access serialization

• IRQ masking
  irq_lock() → z_smp_global_lock()
• Spin-locks
• Additional SW/HW barriers
SMP benefits easily available with Zephyr

For supported architectures and platforms/boards

- Build SMP application in Zephyr
  - Change Zephyr configuration via `kconfig`
    - Enable `CONFIG_SMP` by `make menuconfig`
    - SMP tests will do it for you via `tests/kernel/smp/prj.conf`
    - Optionally set `CONFIG_MP_NUM_CPUS=x`
    - Recompile with `make`
- Get performance scaling
Utilization of multiple cores in SMP system is easy

SMP Pi example: https://github.com/zephyrproject-rtos/zephyr/pull/18849

```c
#define THREADS_NUM 16
#define STACK_SIZE XXX

static K_THREAD_STACK_ARRAY_DEFINE(tstack, THREADS_NUM, STACK_SIZE);
static struct k_thread tthread[THREADS_NUM];

void pi_thread(void *arg1, void *arg2, void *arg3) {
    ... calculate Pi value ...
}

void main(void) {
    ...
    for (i = 0; i < THREADS_NUM; i++) {
        k_thread_create(&tthread[i], tstack[i], STACK_SIZE,
                         (k_thread_entry_t)pi_thread,
                         NULL, NULL, NULL,
                         K_PRIO_COOP(10), 0, K_NO_WAIT);
    }
    ...
}
```

SMP Pi example: https://github.com/zephyrproject-rtos/zephyr/pull/18849
Utilization of multiple cores in SMP system is easy
Evaluation of SMP benefits for real applications

Scaling depends on workload

### Zephyr, SMP Pi, 16 threads

- 240 digits
- 120 digits

### Linux, EEMBC’s “rotate-color1Mp”

- Number of workers on quad-core CPU

---

**Speed-up, normalized**

<table>
<thead>
<tr>
<th>CONFIG_MP_NUM_CPUS</th>
<th>Zephyr, SMP Pi, 16 threads</th>
<th>Linux, EEMBC’s “rotate-color1Mp”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
Conclusion

Zephyr RTOS allows utilization of powerful multi-core systems

• Depending on use-case and system configuration choose
  – SMP for a capable SoC
  – AMP for any HW
  – Heterogeneous multiprocessing (HMP) as a combination of AMP & SMP

• Align HW & SW design
  – Configure HW according to requirements of SW stack
  – Make sure given HW configuration is supported in SW

• SMP allows for easy scaling
  – Depending on use-case might scale better or worse
Participate

We need your help!

- Get sources on GitHub: https://github.com/zephyrproject-rtos/zephyr
- Report bugs report & propose enhancements via GitHub issues https://github.com/zephyrproject-rtos/zephyr/issues
- Contribute fixes & improvements via pull-requests https://github.com/zephyrproject-rtos/zephyr/pulls
Thank You