



High-Performance Real-Time Linux Solution for Xilinx Ultrascale+

Presented By

ENEAA

Patrik Strömlad
Senior System Architect
2018-12-10



Use case for “accelerating” Linux

on Xilinx Ultrascale+

- **Embedded application wants all of:**
 - Linux features and API:s
 - High performance
 - Very low jitter and response time to external events

- **Needs more control processing/calculation performance than the R5:s can provide**
 - Flexibility to use Cortex A53 cores for low overhead / real-time processing

- **Linux is not suitable for real-time**
 - Use preempt_rt patch?
 - Other ways to use parts of the A53 cluster for real-time processing?

Embedded Linux challenge:

**How to achieve the best of two worlds -
Needs Linux programming API, but also a hard real-time POSIX runtime**

What everyone want:

- **Standard Linux/POSIX API**
- **Portability and future proofness**
- **Hardware platform independence**
- **Independency of number of cores**
- **Deployment flexibility**
- **High performance, low OS overhead**
- **High determinism (low latency and jitter)**
- **Safety (robust, high availability)**
- **Security**

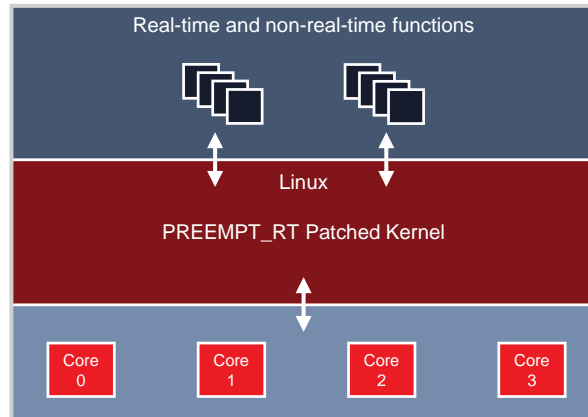
But which are the challenges?

- ✓ Linux cannot provide real-time characteristics
- ✓ Linux has a quite high overhead in scheduling and OS calls
- ✓ "Bare-metal" runtime feature level is very poor
- ✓ "Bare-metal" runtime debug support is non-existent
- ✓ Linux multicore scaling neither linear, nor deterministic
- ✓ Existing RTOS:es cannot compete with Linux eco-system
- ✓ (Most) existing RTOS:es are only single-core kernels
- ✓ Cache and Memory hierarchies will hit you hard in memory contention situations both on OS and application level

Alternative ways to improve Linux for real-time on multicore:

The PREEMPT_RT patch

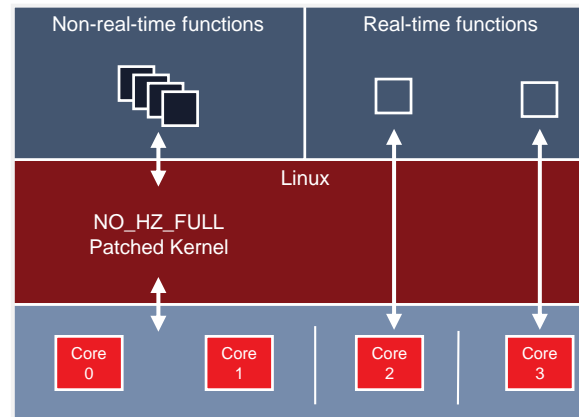
Modify Linux kernel
(code and configuration):



- PREEMPT_RT maintained by the Linux foundation.
- Might require significant changes compared to "standard" Linux. Increases overhead for context switches and system calls of around 10-50%.
- Less quality
- Offers full POSIX
- Suitable for low to moderate real-time requirements
- ~50-100 us worst case latency

User space Partitioning (Core isolation)

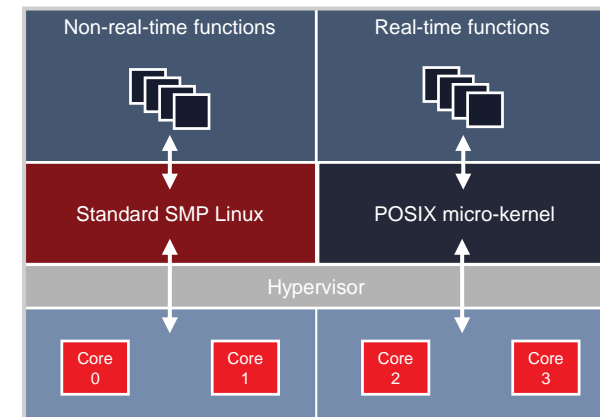
Vertically partition Linux
user-space in two domains:



- Isolate RT threads from non-RT threads.
- Complex configuration, needs patching (ex NOHZ_FULL) to remove ticks.
- Provides a deterministic bare-metal per core single-thread execution environment
- Suitable for single-threaded, polling applications pinned to a core.
- ~3-30 us worst case latency (poll)
- NOT suitable for embedded legacy, multithreaded RT POSIX applications that uses OS API
Using system calls is discouraged, will cause indeterminism and overhead!

Dual-OS partitioning (using hypervisor and uKernel)

Vertically partitioning on OS level using
type 1 hypervisor:



- Standard, unmodified SMP Linux
- A native SMP POSIX micro-kernel runs on a partitioned set of isolated cores
- An integrated OS platform with IPC, shared file system and debug console
- Suitable for legacy POSIX/RTOS customers that wants to migrate to Linux and still have very high realtime requirements
- <3 us worst case latency

Aspects of Linux behavior

- > Kernel Preemption model (server, desktop, LL desktop, RT)
 - >> Important for performance / quality / predictability tradeoff
 - >> Server – fastest, RT slowest (10-50%)
- > Scheduling model (ex: other, fifo, rr)
 - >> One of several system parameters, not the only!
 - >> Need careful consideration
- > RT Throttling (prevent fifo/rr to consume 100%)
 - >> Side effect: rt task level may be swapped out)
- > Load balancing
 - >> May cause unpredictable behavior – forces use of affinity
- > Power Save, frequency scaling
 - >> Enabling power save features often decreases real-time characteristics

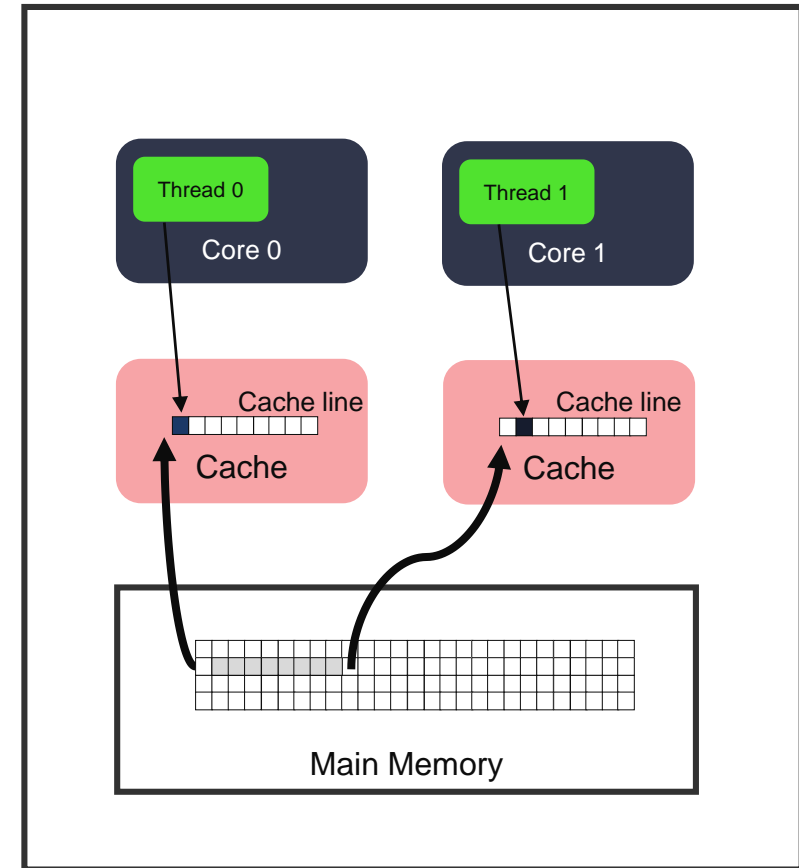
Improving control over Linux real-time capabilities:

- > Configure kernel for desired application profile:
 - >> Server throughput, or multithreaded performance?
 - >> Overall deterministic behavior on protocol level or on I/O event level?
 - >> Must-have debug and trace in field capability?
 - >> Need for low power or can we speed up?
- > If we run multicore, we have actually additional opportunities for partitioning.
- > Isolating an application to a set of cores
 - >> Disabling the load balancer to move to isolated core-set
 - >> Remove non-RT interrupt from isolated set causes jitter
- > Full dynamic ticks
 - >> Turns ticks off (low as 1Hz) if core single-threaded with no posix timers

Don't share writable states among cores!

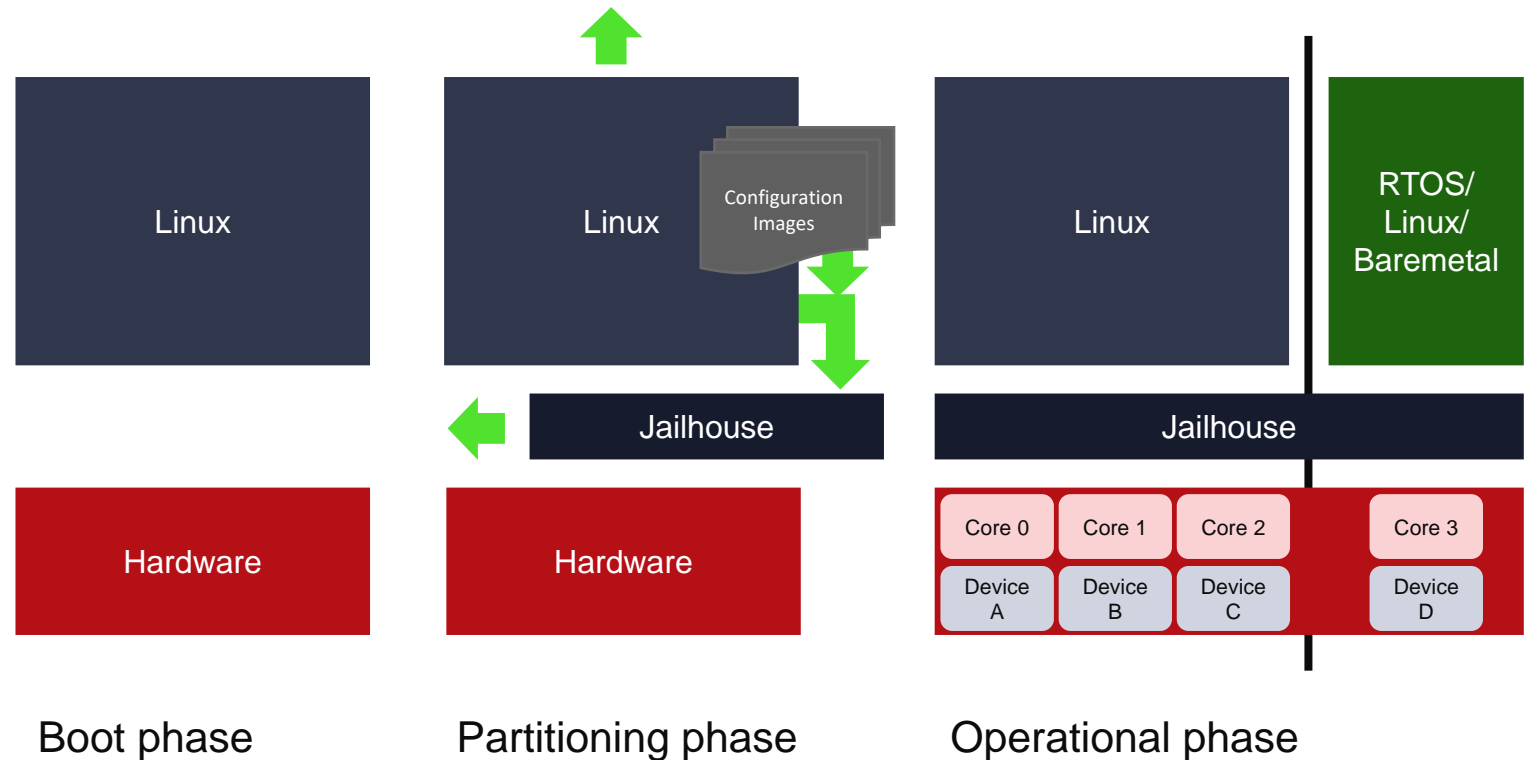
In neither OS nor application!

- > Avoid memory contention!
 - >> Avoid using shared data, even if it is not protected by lock!
 - >> Avoid locating unrelated data on the same cache line!
- > Memory contention causes a huge coherency traffic
 - >> 'Cache thrashing', or 'cache line ping-pong', severely degrades performance as the number of cores grow!
- > Taking a spinlock may add a large and unpredictable penalty that increases per core as cores are added!
- > With more than 4-8 cores, frequent memory contention may rapidly degenerate overall performance and increase latency!
- > *Example: the use of an atomic operation for a statistical global counter may hide a very long non-deterministic stall due to cache line 'ping-ping' storm!*
 - >> Use local counters instead!



The Jailhouse Architecture

- > Build static partitions on SMP systems
 - >> Flexible partitioning, runtime controlled
- > Use hardware assisted virtualization
 - >> Supports ARMv8 and Ultrascale+
- > Does not schedule VM on cores
 - >> Very thin hypervisor layer (low overhead)
 - >> 1:1 device assignment, memory mapped
- > Splits up running Linux systems
 - >> Starts native, "migrates" to be virtualized after boot
- > Simplicity over features
 - >> <10k lines of code (kernel module)
 - >> Assumes multicore, one guest per core



Accelerated Linux on Xilinx Ultrascale+

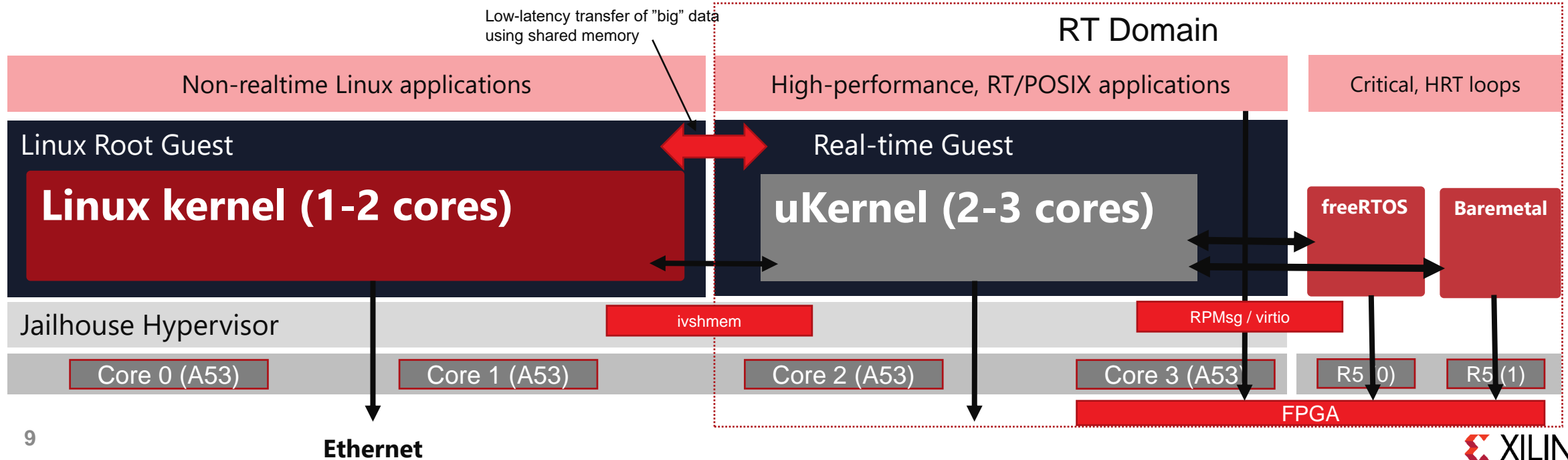
using Jailhouse Hypervisor

Linux:

- > Runs any standard yocto SMP Linux
- > Includes Jailhouse hypervisor
 - >> Guest Management (load, start, stop, restart)

Realtime Accelerator domain:

- > Run SMP POSIX ukernel
- > FPGA SDK (XIL Library)
- > OpenAMP to R5: Remoteproc/RPMsg



Accelerated Linux on Xilinx Ultrascale+

Important features:

- > Network O&M services:
 - >> Enea IPC (Linx) between Linux and RT domain
 - >> TCP/IP connectivity over ptp Ethernet (TAP)
- > Common O&M services:
 - >> Shared file system (Linux FS mounted as POSIX file system on RT side)
 - >> System debug tools of RT domain: Rumode gdb, trace, dump, profiling
- > High-bandwidth data transfer to/from RT domain:
 - >> Shared pool of very large buffers. Passing pointer objects to buffers over IPC
 - >> Uses shared, cache-coherent memory to copy data
- > Access to Xilinx Ultrascale+ devices:
 - >> openAMP/RPMsg IPC to R5
 - >> XIL Library access to FPGA

Jailhouse versus Xen and native benchmark setup

- >> Native OSE microkernel:
 - OSE 5.9 for ARMv8 set up for SMP 4 cores A53

- >> Enea Accelerated Linux with Jailhouse
 - PetaLinux 2017.2 (set up for core 0)
 - Jailhouse ver 0.7 (configured for 1 root cell (1 core) and one guest cell (3 cores))
 - OSE 5.9 for ARMv8 set up for SMP 3 cores A53
 - Alt 1: direct access to GICv2 (gic paravirtualization, bypass hypervisor mode to guest)
 - Alt 2: unmodified bsp)

- >> Linux + Xen
 - PetaLinux 2017.1 (set up for core 0)
 - Xen ver 4.9 (patched for "null" scheduler)
 - OSE 5.9 for ARMv8 set up for SMP 3 cores A53



Jailhouse versus Xen benchmarks

- > **Benchmarks for OSE running on a Zynq Ultrascale+ board for the following scenarios**
 - >> OSE standalone (or OSE running as Jailhouse guest when GICv2 paravirtualized)
 - >> OSE running as Jailhouse Hypervisor guest (unmodified)
 - >> OSE running as Xen guest, using the null scheduler (Xen v 4.9 or later)
 - >> OSE running as Xen guest, using credit2 scheduler

	Minum Latency	Average Latency	Maximum Latency
Native OSE /Jailhouse	0.9 μ s	~0.9 μ s	1-2 μ s
Jailhouse Hypervisor	1.5 μ s	~1.5 μ s	2-3 μ s
Xen, null scheduler	2.8 μ s	~2.9 μ s	3-5 μ s
Xen, credit2 scheduler	2.7 μ s	~3 μ s	5-7 μ s

Jailhouse benchmark Conclusions

- >> The cyclictst benchmark indicates that worst case task response latency in RT domain well below 3 us (not counting core 0) even when A53 cores are under load
 - ***will meet RT requirements for 5G L1/L2 baseband control & radio!***
- >> Average time overhead for latency is very small compared to Linux
 - ***more time spent in application processing!***
- >> OS overhead for scheduling and timer handling is around 10-15x smaller than in Linux
 - ***more time spent in application processing!***
- >> The Jailhouse hypervisor adds almost no overhead for a guest
 - ***the RTOS kernel guest has almost same latency & performance as if running bare-metal!***

Demo

The logo consists of a red chevron pointing right, followed by the letters 'XDF' in a white, bold, sans-serif font.

XILINX
DEVELOPER
FORUM

