Real-Time and Certification on Multi-Core systems

Torrents Workshop 2012

Bertrand Marquis (bertrand.marquis@sysgo.com)
Summary

• Why using Multi Core CPUs?
• Software Multi Core overview
• Hardware Considerations
• Multi Core OS Design
• Certification Considerations
• PikeOS Virtualization Platform
• PikeOS Multi-Core Support
• Multi Core System and Application Design
Why using Multi-Core CPUs?
Why using Multi Core CPUs?

- Since 2004 processor frequency is limited around 4GHz
  - Physic limitation (propagation, temperature etc)
  - IBM research: 500 GHz !!!! but only at -296°C
Why using Multi Core CPUs?

- **Number of transistor still evolving**
  - Follow the Moore Law

- **Power and capacity increase achieved through:**
  - Improvement of cores themself
    - Instructions: SSE, MMX, Altivec …etc
    - Pipeline: bigger, branch prediction …etc
  - Improvement of the Cache (number, performance)
  - **Multiply number of Cores and Threads per Cores**
Why using Multi Core CPUs?

• Embedded applications require more and more power
  • Not possible by increasing CPU speed anymore
  • New compilers/solutions to have parallel computing

• Platform aggregation
  • IMA in avionics
  • Android/Windows in parallel of Real-Time or secure application

• Single core CPUs will disappear
  • Long-Term maintenance problem
  • SMP processors already used in certification (Freescale 8572DS)
    • Only use one core
    • Waste of Power
Multi Core OS Overview

Multi-Core Operating System Models

Asymmetric Multi Processing
- Each Core runs a different uni-processor OS
- Loosely coupled through IPI and shared memory
- Both operating systems need to be fully trusted

Semi Symmetric Multi Processing
- Each Core runs an instance of the same uni-processor OS
- Loosely coupled through IPI and shared memory
- Operating system needs to be fully trusted

Symmetric Multi Processing
- All cores are controlled by a single SMP operating system
- Closely coupled through resource locks and IPI synchronization
- Multi-processor support on application level
- Operating systems need to be fully trusted
# Multi Core OS Overview

## Comparing AMP and SMP

### AMP

**Pro**
- Simple system software design
- Concurrent execution of different uni-processor operating systems

**Contra**
- All operating systems need to be fully trusted
- External synchronization required to access shared resources
- No support for multi-processing within a single application
- Difficult to manage with more than 2 cores
- Distributed configuration

### SMP

**Pro**
- Only one trusted system software layer
- Better control of CPU activities
- Support for multi-processing on application level
- Simpler synchronization between partitions
- Homogenous configuration

**Contra**
- Increased complexity in SMP OS
- Performance decrease compared to AMP for loosely coupled applications
- Cache coherency required
Hardware Considerations
Hardware Considerations

CPU and Platform Considerations

- **Minimum CPU Features (AMP and SMP)**
  - Separate CPU, FPU and MMU
  - Separate L1 (ideally also L2) Data and Instruction Cache
  - Inter Processor Interrupt (IPI)

- **Extended Features required for SMP**
  - Cache and TLB Coherency Protocol
  - Coherency Sub-Domains (for mixed SMP / AMP configurations)
  - Global Time Base (to avoid clock synchronization)
  - Interrupt Routing
Hardware Considerations
Multi Core based Processing Module

![Diagram of a Dual-Core CPU with IPI inter-processor interrupts and a core communication bus. The diagram includes L1 I-Cache, L1 D-Cache, L2 Cache, Coherency Module, Memory Controller(s), DRAM, FLASH, NVRAM, PCI Controller(s), PCI-Bus(es), Special IO (Serial, DIO, CAN), and ARINC 664, ARINC 429.]
Hardware Considerations

Typical problems

• Performance degradation due to cache / TLB coherency protocol
• Implicit device sharing due to missing separation of I/O addresses
• Shared Processor, Memory and PCI Bus
• Shared Interrupts (typical problem on x86 platforms)
• Shared IO Devices (AFDX, A429, CAN, DIO, Frame Buffer, ...)

© SYSGO AG
Hardware Considerations

Cache Coherency

• Ensures consistent cache content across cores
• Cache coherency protocol has large performance impact
• “False Sharing” of the cache

Example:

```c
int loop_cnt[NUM_CORES];

/* Loop executed on each core */
void thread_loop(int my_core_id)
{
    for (; ; ) {
        loop_cnt[my_core_id] += 1;
    }
}
```
Concurrent Data Access
Access do disjoined data regions

Intel (Copy Back Mode) / Write Non-Shared

AMD (Copy Back Mode) / Write Non-Shared

Intel (Copy Back Mode) / Read Non-Shared

AMD (Copy Back Mode) / Read Non-Shared
Concurrent Data Access
Access to shared data regions

Intel (Copy Back Mode) / Write Shared

AMD (Copy Back Mode) / Write Shared

Intel (Copy Back Mode) / Read Shared

AMD (Copy Back Mode) / Read Shared
Multi Core OS Design
Multi Core OS Design

Parallel operation on OS level

• **Design**
  • Each core runs its own OS instance (AMP)

• **Pro**
  • Single core OS design
  • Cache coherency problem only for shared memory

• **Contra**
  • Bus sharing difficult to control
  • Device sharing difficult to implement (e.g. AFDX)
  • No support for multi-core applications
  • Distributed and inhomogeneous configuration
  • Complexity increases while efficiency decreases with number of cores
Multi Core OS Design

Parallel operation on partition level

• **Design**
  • One OS manages all cores (SMP)
  • Each partition has its dedicated core
  • Each partition provides single core runtime environment

• **Pro**
  • Supports full control of all cores
  • Supports shared access to I/O devices
  • Supports ARINC-653 scheduling through core synchronization

• **Contra**
  • Increased complexity in the OS
  • Potential interference through shared OS resources
Multi Core OS Design

Parallel operation on application level

• Design
  • Based on SMP approach
  • Supports multiple cores per partition
  • A partition may provide a multi-core runtime environment (e.g. POSIX)
  • Code and data segment as well as the heap are shared between cores

• Pro
  • Increased processing bandwidth for a single application

• Contra
  • Increased risk of false sharing of the cache
Multi Core OS Design

Parallel operation on code block level

• **Design**
  - Based on SMP approach
  - Portions of one logical execution thread are executed in parallel (OpenMP Application Program Interface)
  - The compiler actually allocates the code to CPU cores

• **Pro**
  - Fine grain core allocation

• **Contra**
  - High risk of false sharing
Certification considerations
Certification Considerations

AMP Concept

• Concept similar to multiple Single Core platforms
  • More interference channels between cores due to stronger coupling of the cores (memory bus, coherency protocol, etc) compared to dedicated communication buses in federated architecture

• Partitioning is distributed across multiple OS instances
  • Correct spatial and temporal separation depends on an heterogenous system software architecture
  • Cores may even run instances of different operating systems
  • Execution on both cores is completely asynchronous

• Separate virtual address spaces
  • No need for MMU synchronization
  • Memory allocation based on OS instantiations
  • Only dedicated memory regions shared between cores for communication
Certification Considerations

SMP Concept

• SMP concept is new for certification
  • SMP operating system design needs to assure that there are no interference channels between partitions due to cross CPU locking mechanisms.

• Partitioning is controlled by a single OS instance
  • One time partition scheme with core synchronization on window boundaries
  • Critical parts of the time frame can even be executed in “single processor mode”

• Single virtual address spaces
  • Requires MMU synchronization upon mapping changes
  • Memory allocation based on resource partitions
Certification Considerations
Concerns related to the Use of Multiple Cores

• Hardware Interference Channels
  • Shared caches
    • Typically one L1 cache per CPU
    • L2 cache shared on some CPU (e.g. Intel, MPC 8572D)
    • L3 cache typically shared
  • Cache coherency protocol
    • Problem grows with number of cores
    • Configurable on some CPUs through core cross bar
    • Global / Local cache flush and invalidate
  • Shared buses (Core Connection, Processor, Memory, PCI)
  • Shared Interrupts
  • Shared devices (Memory, Timer, I/O)
  • Shared peripherals (AFDX ES, A429, GPU, DIO, Ethernet)

• All this is a major problem mainly for the WCET
Certification Considerations
Concerns related the COTS Multi Core CPUs

• SoC / MPSoC Design
  • Lot of multi-core processors come as “System on Chip” devices which requires additional certification activities
  • EASA has issued specific CRIs (CRI-F08 for A400M Project) which address the use of complex electronic devices.
  • Currently a proposal for a Certification Memorandum (CM SWCEH 001) has been released which addresses “Complex Electronic devices”, “Complex COTS Microcontrollers” and “Highly Complex COTS Microcontrollers”
  • Multi Core SoCs are considered by EASA as Highly Complex COTS Microcontrollers
Certification Considerations

Concerns related the COTS Multi Core CPUs

- **Main challenges to comply with CM SWCEH – 001**
  - Leak of processor design documents may lead to undetected interference channels
  - In Service History
    - There is not much in service history available for multi-core based designs in avionics applications
  - Availability of Safety Features
    - Sufficient robustness against SEU events (radiation, electro-magnetic)
    - Detection and correction of single and multi bit errors
  - Determinism
    - Random behavior e.g.
      - Cache and TLB refill algorithms
      - Bus arbitration
    - Undocumented interference channels
PikeOS Virtualization Platform
Company History

Foundation as RTOS Services Company

1st Embedded Linux project

Distributor for Safety-Critical RTOS

1st DO178B DAL A Certification

PikeOS Market introduction

ELinOS Market introduction

PikeOS Research Project

ELinOS Product of The Year

New Office In Paris

New Office In USA

Tier-1 Airbus supplier

2011

2010

2009

2008

2005

2004

2000

1999

1997

1996

1992

1991
Markets

We consider our target markets to be all industries related to Embedded Systems in which safety, security and more generally complete reliability play a major role.
SSV – Safe and Secure Virtualization
SSV – Safe and Secure Virtualization

**Safe**
- Address all applicable safety standards (DO-178B, IEC 61508, EN 50128)
- Guarantee determinism with respect to timing and resource consumption

**Secure**
- Support evaluation according to Common Criteria
- Prohibit any information flow between VMs
- Comply to security design patterns (MILS, SKPP)
- Support run-time auditing
- Minimize size of trusted components

**Open**
- Support open standards like POSIX, ARINC-653, Linux, ADA, Java
- Open System Interfaces
- Open verification procedures

**Transparent**
- Safety and security features may only rely on verifiable features
- Extensive use of hardware protection means may be problematic due to lack of HW design documents
- The certification process needs to be transparent down to the source code level

**Extensible**
- Multi Core, Hardware Virtualization, IOMMU
- New communication protocols, e.g. AFDX
- Future API extensions

**Flexible**
- Flexible CPU time, memory allocation
- Simple communication redirection
- Easily portable to new platforms
PikeOS: Safe & Secure Virtualization RTOS

- Designed from ground-up for safety & security
  - Modularity and compactness
  - MILS and ARINC653 compliant architecture
- Genuine virtualization for embedded/real-time
  - By design (no hypervisor add-on)
  - POSIX, ARINC-653, Linux, Android, RTEMS, Autosar etc.
- Same core technology for all application domains
  - No mix of old RTOS and new hypervisor add-on
  - Safety and security attributes available to all
- Scalable and flexible
  - Can be used just as small and fast RTOS
  - Multi-core support ranging from AMP to SMP
  - Widest range of supported API’s in the market
- Hardware independant
  - x86, PowerPC, ARM, MIPS, SPARC/Leon, v850, SH-4,...
- Certification for safety & security
  - DO-178B, EN50128, IEC61508, CC’s EAL, MILS...
PikeOS - Architecture Overview

**Application Layer**
- Standard Partitions based on different API (PikeOS Native API, ARINC 653, POSIX, etc.)
- System Partitions based on PikeOS Native API

**PikeOS Virtualization Platform**
- Configuration
- Partition Management
- Partition Communication
- Health Monitoring
- Device Drivers
- OS primitives
- Platform Support Package
- Low Level Drivers

---

**PikeOS System Software**

- **PikeOS Microkernel**
- **Driver** (File API)
- **Driver** (Port API)
- **Low Level Driver**
- **PikeOS PSP**

---

**Hardware Platform**

---

**Custom Application**
- Level B
- Level C
- ARINC 653
- POSIX

**Custom Application**
- System Partition
- CBIT Health Mon. Logging

---

© SYSGO AG
PikeOS - Virtualization Platform
Real-Time Aspects

- PikeOS priority based FIFO scheduler
  - 256 Priority levels
  - Fixed time required to select next running thread
- Preemptive design
  - All services are preemptible
  - Shared data protected by critical sections
  - Duration of all critical sections evaluated during WCET analysis
- Time partitioning
  - All services can be preempted by time partition switch
  - Service provided to device drivers to prevent time partition switches within critical sections
PikeOS - Virtualization Platform
Resource Partitioning (I)

• Resource partitioning includes
  • Memory, memory mapped and port I/O partitioning,
  • Control access to files, device drivers, communication ports, privileged services
  • Control inter partition communication
• Implementation based on CPU privilege levels and MMU
• All partitions execute in user mode
• No error propagation throughout other partitions
• Static configuration of resource allocation
• Partition has guaranteed access to assigned resources
• Support for selective partition shutdown / restart
• Support for multiple resource partitions per time partition
• Partition payload independently linked and loaded
PikeOS - Virtualization Platform
Resource Partitioning (II)

• Assign a resource to one partition
• Share a resource using driver
domains in SSW
• Ensure availability of assigned
resources
Advanced Time Partitioning

Time partition principles

- **ARINC 653 compliant**
  - Static configuration of execution order and duration
- **Future ARINC 653 requirements**
  - Support for multiple scheduling schemes
  - Scheduling schemes can be switched during runtime
- **Partition ‘0’ offers additional functionality**
  - Threads with high priority can preempt active partition
  - Threads with low priority can act as global idle-job
Advanced Time Partitioning

Use of Background Time Partition to Optimize CPU Load

- More CPU time can be allocated to critical applications as a buffer to cope with worst case scenarios
- Unused CPU time can be provided to applications without hard real-time constraints

Traditional ARINC 653 Scheduling

Scheduling with Background Time Partition

- 4 ms additional buffer for critical applications
- 6 ms additional buffer for non-critical partition in the normal case
Advanced Time Partitioning

Use of Background Time Partition to Reduce Latency

• Handlers for exceptional conditions have a better response time
• No need to allocate time for exception handling application
• Interruption of running partition acceptable since fault will typically require change of operation mode

Traditional ARINC 653 Scheduling

--> Large delay for fault handler due to time partitioning

Scheduling with Background Time Partition

--> Minimal delay for fault handler
--> Running partition is preempted
--> Fault handler would typically switch to different operating mode

---

© SYSGO AG
PikeOS - Virtualization Platform
Inter Partition Communication

• **ARINC – 653 Compliant Communication Ports**
  • Statically configured communication endpoints (Queuing & Samplingports)
  • A partition has access to its own ports only
  • Access through PikeOS Port API
  • Statically configured comm. channels
  • Preemptible read / write access ports to minimize partition jitter
  • Messages buffered in trusted PSSW domain

• **Shared Memory Objects**
  • Support for transfer of large amount of data with minimum overhead
  • Access through PikeOS File API
Inter Partition Communication

Advanced

Communication Services

PikeOS Virtualization Platform

Hardware
PikeOS Multi-Core Support
PikeOS Multi-Core Support

Asymmetric Multi Processing

- PikeOS design fully supports the AMP concept
  - Adaptations are limited to the PikeOS PSP
  - PikeOS ROM file system can be used to store image to be executed on other cores
  - PikeOS can coexist with other operating systems or executives
  - Multiple instances of PikeOS can run in parallel
PikeOS Multi-Core Support

Symmetric Multi Core Support (PikeOS 3.3)

- Compatible with Single-Core PikeOS
  - Same API for single and multi core PikeOS applications
  - Cross CPU communication through IPC and Events
- Multiple cores managed by one OS instance
- Cores are statically allocated to resource partitions
  - One core can be allocated to multiple partitions
  - One partition can have multiple cores assigned
- CPU Affinity
  - A PikeOS task has assigned a set of CPU cores (CPU allocation mask)
  - A thread can only run on a core from its task’s CPU allocation mask
  - A thread can decide on which CPU it wants to run (CPU affinity mask)
  - The task’s CPU allocation mask is distributed to child tasks like any other task right (parent – child principle)
  - During IPC one IPC partner can be moved to the other partner’s CPU in accordance with its affinity mask
PikeOS Multi-Core Support
Symmetric Multi Core Support (PikeOS 3.3)

• Time Partitioning
  • All CPU cores share the same time partition
  • Cores are synchronized on time partition switches
  • Multiple resource partitions can share the same time partition
  • Concurrent execution of resource partitions on different cores
  • A strict ARINC 653 scheduling scheme can be configured for parts of the major time frame

• Full Support for PikeOS Time Partition 0
  • Background and high priority tasks can run on dedicated core
PikeOS Multi-Core Support

PikeOS SMP Scheduling Example
PikeOS Multi-Core Support

Implementation Details

• **Time Management**
  • Dedicated hardware timer per CPU or emulation through IPI
  • Time will be synchronized across CPUs

• **Interrupt Management**
  • Kernel binds the interrupt to the CPU where the handling thread is running on
  • Shared interrupts will be bound to the CPU with first-runable handler thread
Multi-Core Systems and Applications
Multi-Core Software Design

Certification Aspects

• **Support static partitioning of CPU cores**
  • Load balancing algorithms may not be used during execution of critical applications

• **Control access to shared resources**
  • Avoid parallel execution of different criticality levels

• **Eliminate “False Sharing” between partitions**
  • Avoid Multi-Core application design for safety critical applications
  • Ensure proper data alignment inside shared software components

• **Eliminate interference via OS internal synchronization objects**
  • Avoid module global locks
SMP Scheduling Example

Configuration

Execution
Use Cases

AMP - PikeOS + AFDX

• **Features**
  • MPC 8572D CPU based board
  • Core A running PikeOS UMP with ARINC 653 Personality
  • Core B running a software AFDX using 2 of the 4 TSEC controllers
  • Communication and synchronization through shared memory and inter processor interrupts
  • Suitable for certification safety critical applications

• **Certification Considerations**
  • Independent execution of CPU cores and TSEC required
  • L2-Cache need to be partitioned by software
  • Worst case interference on memory bus need to be evaluated
  • Impact of coherency protocol need to be evaluated
Use Cases

SMP - High Performance IMA Platform

• Assumptions
  • Platform connected to several Avionics busses
  • Some applications with high CPU demands
  • Some applications with hard real-time demands
  • Multiple applications need to be connected to communication devices
  • Inter-Partition communication needs

• Design Considerations
  • Interference between cores needs to be eliminated for hard real-time application
  • Configuration need to support "single processor" environment while hard real-time applications are running to eliminate interference channels introduced by shared HW and SW resources
  • Software architecture needs to address access to shared platform resources, e.g. AFDX
PikeOS on P4080 Platform

P4080 WCET considerations

• **Design independent application domains**
  • Allocate cores for I/O processing to reduce concurrent access to low bandwidth buses/devices
  • Use P4080 Sub-Domain feature to reduce interference through cache coherency protocol between independent domains

• **Resource allocation profile**
  • Generate application models which address memory and I/O utilization over time and use this information for module configuration
  • Use debug unit to monitor and potentially restrict bus allocation
  • Determine application domain behavior under worst-case scenarios (maximum bandwidth used by other domains)
  • Avoid to mix time-critical and non-trusted domains
Conclusion
Conclusion

• **Going to SMP for certification or real-time is mandatory**
  - Soon there will be almost no Mono-core CPUs
  - Safety and Security applications complexity is increasing

• **Some problems can be solved in software or by analysis**
  - Statistic WCET and its application in certification may be the only solution seeing the complexity of the hardware
  - Interference channels is now the biggest problem
Conclusion

• Hardware could help reducing certification complexity
  • Introduce QoS controlled by Software on some channels:
    • RAM: per CPU and per device (DMA) min/max bandwith
    • IO: per CPU and per Device min/max Bandwith
    • Cache: reduce coherency interferences, P4080 domains is a first approach to this
  • Para-virtualization and Certified OS like PikeOS may be mandatory
    • Not possible and probably never be to have different certification levels on different Cores without an OS at an higher level underneath
    • Hardware independance will reduce costs
      • No standard available for hardware and COTS CPU are the only choice to keep a reasonable cost
Any Questions?