Loughborough University

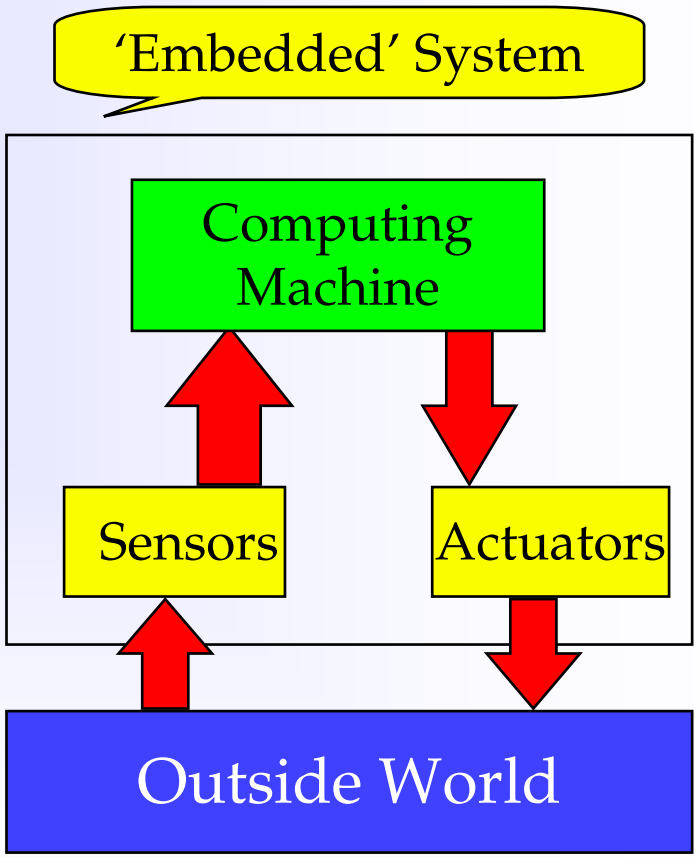
A Co-Processor Scheduler for Embedded Real-Time Systems

Andrew Fox & Jim Cooling

Dept. of Electronic and Electrical Engineering,
Loughborough University, UK

Introduction

- What is real-time embedded systems?
- Not a Computing Machine, per se.
- Computer Seen As Component Of The Larger System.
- RTES response dictated by equipment, not computer.



Introduction (2)

- Broadly categorise into hard and soft.
- 'Hard' systems may malfunction if timing constraints not met.
- 'Soft' systems may suffer performance degradation.
- 'Hard' real-time not necessarily 'fast' real-time.
- Hard-fast real-time difficult to provide.

	Hard	***Soft***
Fast	Aircraft Autopilot	Database Manager
Slow	Chemical Plant	Remote Terminal

The Real-Time Executive

- Embedded system frequently required to perform concurrent tasks.
- True concurrency not possible on uni-processor.
- Concurrency 'emulated' by ***Real-Time Executive*** (RTEX).
- Provides 'primitives' necessary for real-time operation.
- RTEX is software, execution time is an overhead.

The Real-Time Executive (2)

- Ease of use.
- No need to write & support your own RTOS.
- Reduced code & maintainance cost.
- Non-trival systems usually require it.
- Improved reliability.
- Modular, extensible (e.g. networking).
- RTEX is a software task.
- Execution time is an overhead.
- In hard-fast systems, use of RTEX can lead to performance problems.

Software Executives

- Manage a set of concurrent tasks.
- Which task next? *Scheduling*
- Install selected task? *Dispatching*
- Necessary to save/restore task state - *the context* - unavoidable overhead without processor support.
- Other functions provided:
  - Communications, synchronisation
  - Management functions etc.

Proposed Solution

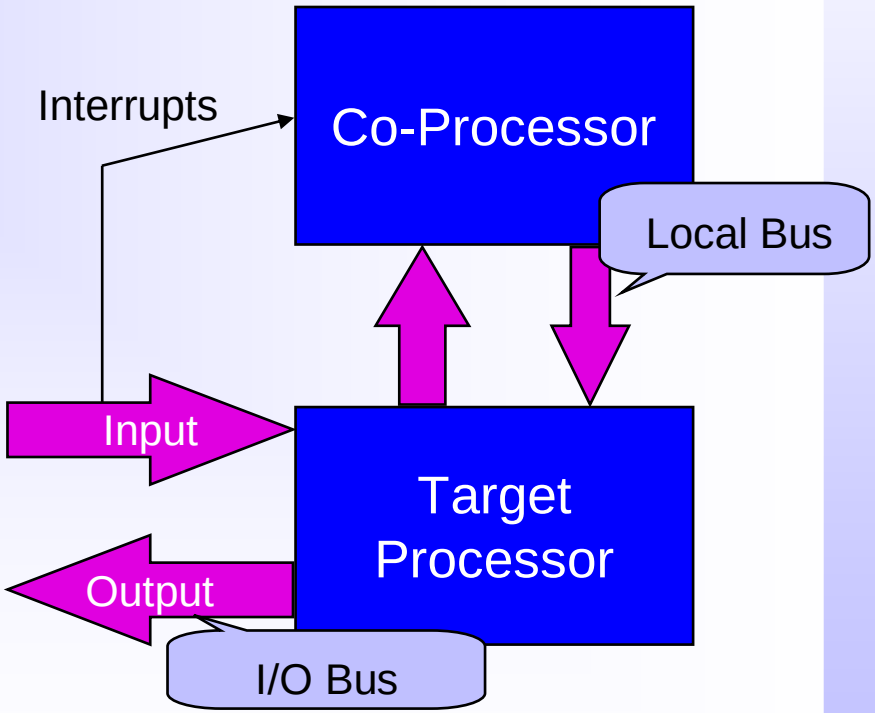
- Use a hardware co-processor to perform time-critical sections of RTEX.
- Time intensive operations necessary for real-time operation (e.g. scheduling) performed by co-processor.
- Increases time available on target processor for useful work.
- Potential for increased reliability.

Co-Processor Objectives

- Aim: Remove burden of supporting multi-tasking from target processor.
- Good performance.
- Simple interfacing.
- Language, compiler and processor independence.

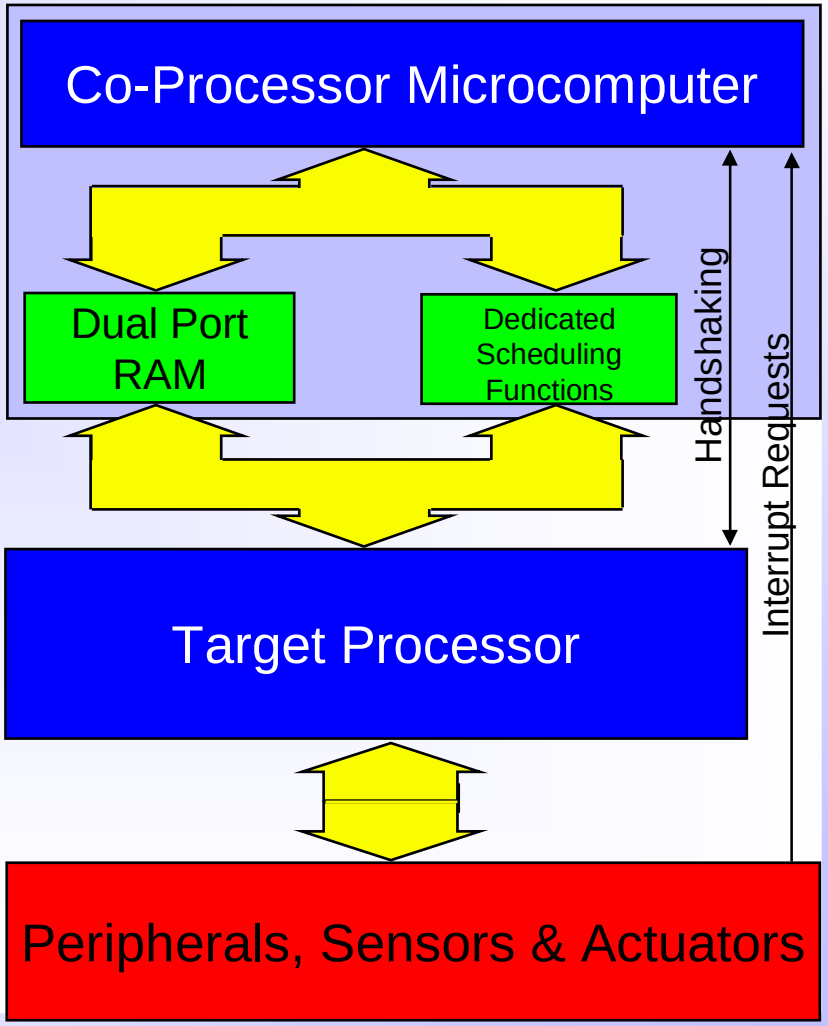
Co-Processor Structure

- System I/O through Target.
- System interrupt requests through co-processor.
- Co-processor can-not manipulate target peripherals.
- Communicate using interrupts and dual-port memory.



Co-Processor Implementation

- Combines software and hardware.
- Flexibility & speed.
- Communication via dual-port memory and interrupts.
- *Xilinx* XC4005 FPGA
- Intel 80C186-EB µP
- Serial comms
- Flash memory, and expansion bus



Mode of Operation

- Scheduling algorithm executed by microprocessor within co-processor.
- Dispatching & task management performed within hardware.
  - Migrate other functions (e.g. timing, interrupt management) to hardware.
- Processors interact through dual-port memory; message passing & interrupts.
- Leaves target free to perform target functions.

Comparisons

- Many similar projects:
- Specialised for a particular language:
  - Roos, Lundberg; targeted to Ada
- Specialised for a particular kernel:
  - Spring kernel accelerator (SSCoP), Mach
- This one is general purpose.
- Processor and language independent.
- Use of software provides flexibility.
- Hardware provides speed.

Figures & Future Work

- Full context switch (inc. reschedule) time less than 35µs.
- Reschedule time of 9.6µs.
- Use faster processors (e.g. *i*960) and increase clock speed.
- More functions in hardware (e.g. interrupt management & timing)
- Integrated real-time design & simulation environment.

Concluding Remarks

A scheduling co-processor:

- Reduces multi-tasking overhead on target processor.
- Can reduce target overhead with suitable processor support (e.g. register files).
- Can never remove all target overhead.
- May be used as test bed for embedded design & simulation work.