UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Department of Computer Science
Computer Engineering Group
www.upb.de/cs/ag-platzner

Enno Lübbers, Marco Platzner

Cooperative Multithreading in Dynamically Reconfigurable Systems

FPL 2009, 01.09.2009

Motivation

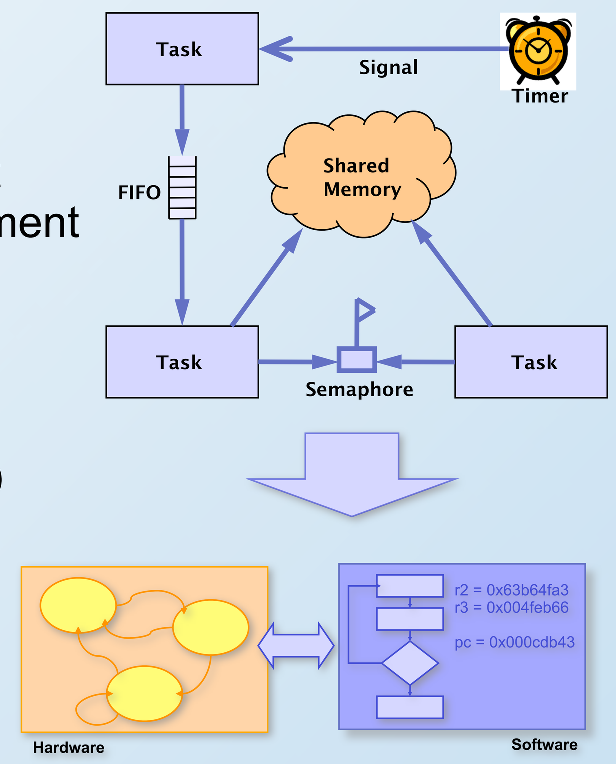
- partial reconfiguration enables time-sharing of reconfigurable hardware resources
- hardware threads, as implemented by ReconOS, provide partitioning of an application into suitable modules for hardware multitasking
- non-preemptive multitasking techniques are unsuitable for many applications
  - long-running threads may make system unresponsive
  - asynchronous (i.e. blocking) operations must be registered with an event loop via callback functions
- preemptive multitasking faces substantial challenges when applied to partially reconfigurable devices
  - determining and accessing the relevant context of a hardware module is a complex task
  - readback or scan chain techniques involve significant overheads and are often device-dependent

ReconOS Programming Model

similar to existing APIs
- eCos
- POSIX

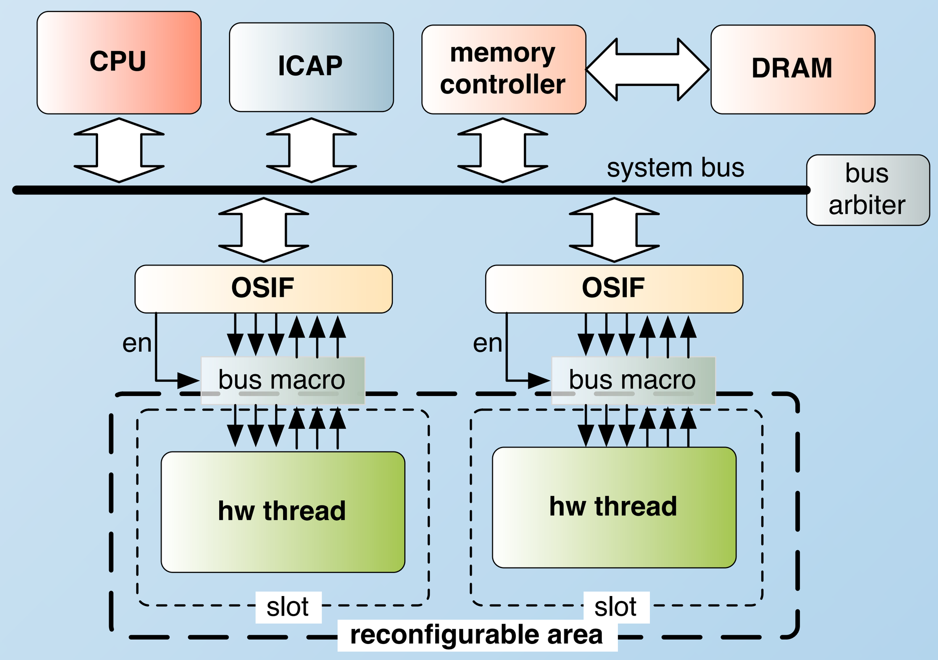
OS services
- task management
- memory management
- synchronization
- communication

OS objects
- tasks (HW or SW)
- shared memory
- semaphores
- queues/FIFOs
- timers
- signals
- ...



ReconOS Execution Model

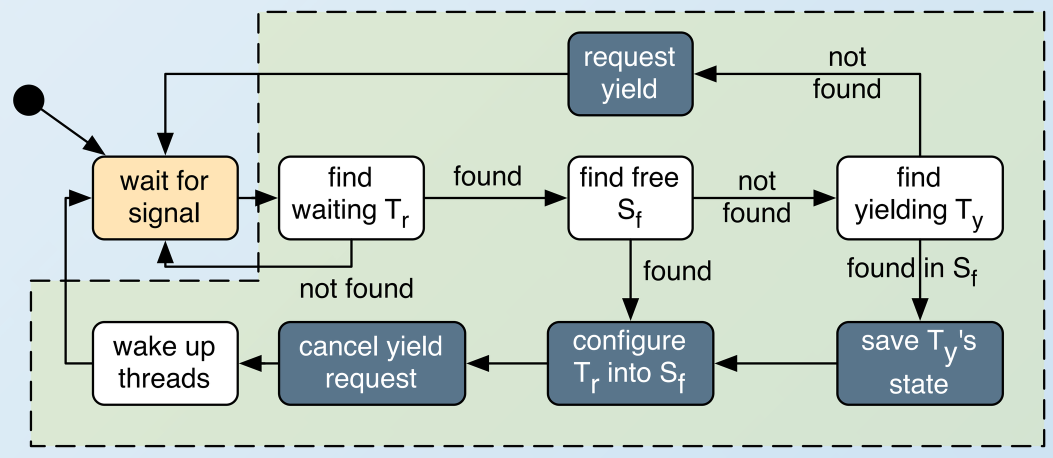
- OS interface module (*OSIF*) enables transparent communication and synchronization between hardware and software
- OS calls from hardware are relayed to *delegate threads* running on the system's CPU
- HW multitasking through partial reconfiguration



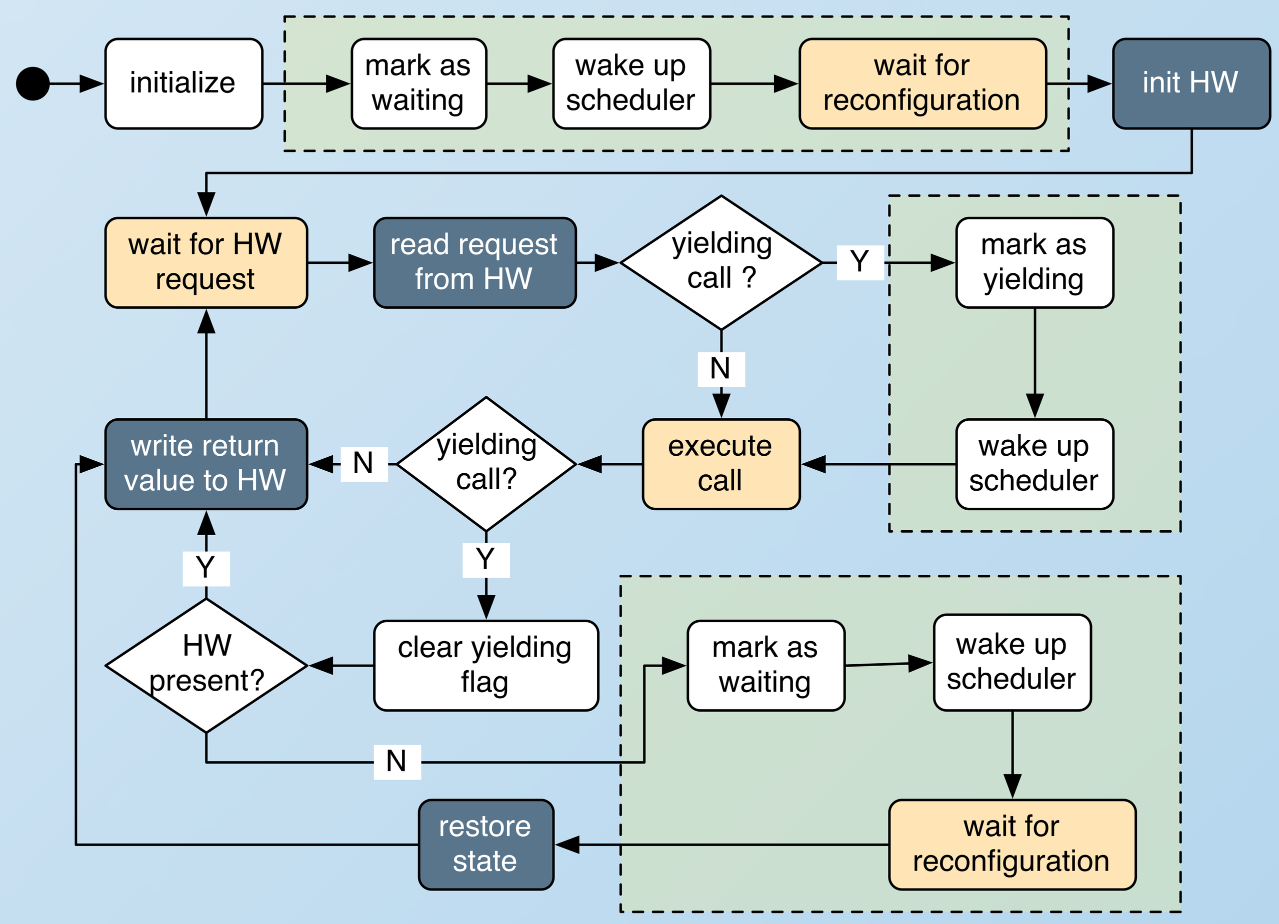
Cooperative Multitasking

Approach

- threads can voluntarily relinquish (`yield()`) their execution slot
- threads are responsible for saving and restoring their state on yield or resume
- ideally, threads yield on blocking OS calls, during which they would not perform any computations



hardware scheduler



delegate thread

hardware access | synchronized | blocking

Implementation in ReconOS

- in ReconOS, cooperative multitasking is only employed for HW threads; SW threads are scheduled preemptively
- the task of managing the reconfigurable resources is shared between two software threads
  - a hardware thread's *delegate* thread and a high-priority *hardware scheduler* thread
  - no changes to the OS kernel are necessary
- a synthesized hardware circuit representing a thread's functionality is called a *core*
- for every *slot* in the system, a core is placed and routed, resulting in $n_{slots} \times n_{cores}$ partial *bitstreams*
- data structures modeling the relationships between slots, hardware threads, cores, and bitstreams are shared between the delegates and the hardware scheduler

Scheduling Example

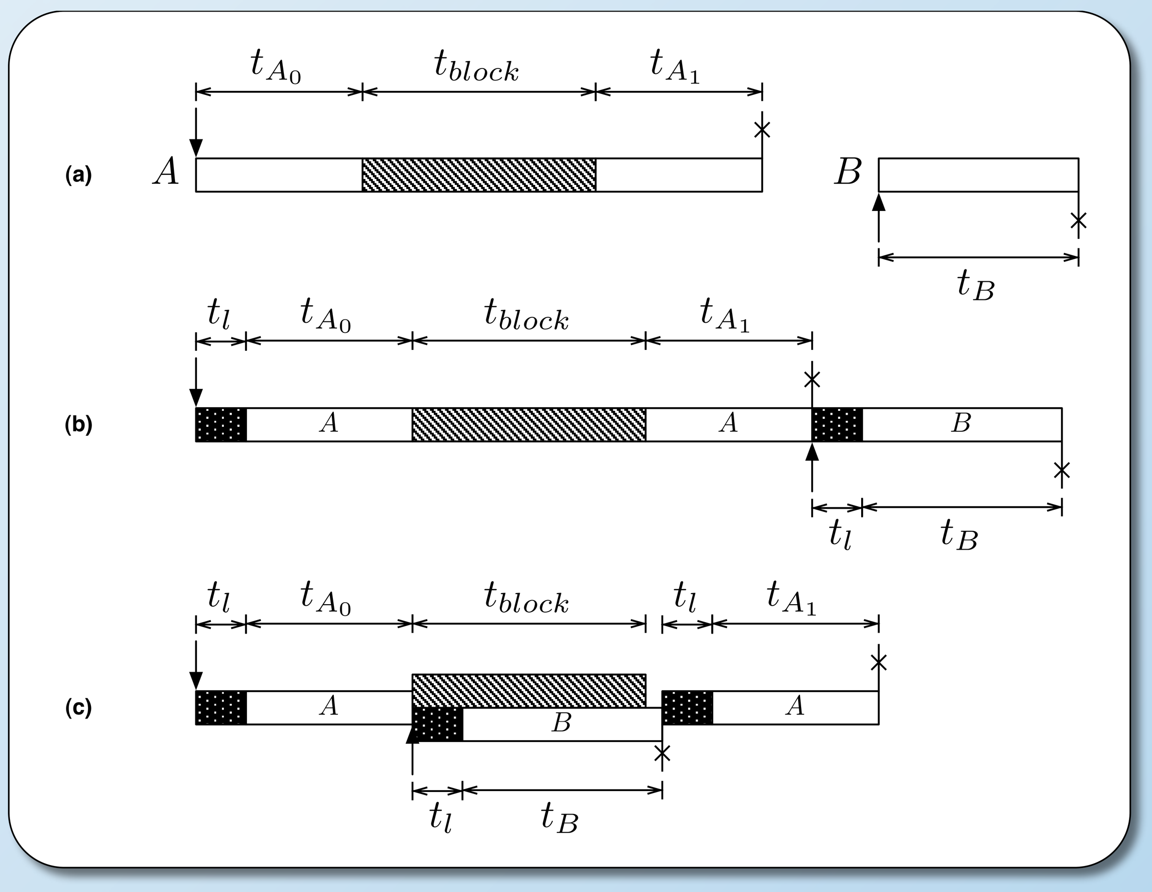
- (a) consider two threads, A and B
  - thread A runs for t_{A0}, blocks for t_{block}, and then runs again for t_{A1}
  - thread B simply runs for t_B
  - loading a thread onto the FPGA takes t_l
- (b) with non-preemptive multitasking, threads A and B are executed consecutively, with a total run time

$$T_n(A, B) = 2t_l + t_{A0} + t_{A1} + t_{block} + t_B$$

- (c) with cooperative multitasking, thread A can *yield* its execution slot to thread B while blocking (i.e. on an OS call), resulting in an execution time of

$$T_c(A, B) = 2t_l + t_{A0} + t_{A1} + t_{As} + t_{Ar} + max(t_{block}, t_l + t_B)$$

t_{As} and t_{Ar} are the times to save and restore A's state, respectively.



- Thus, the cooperative multitasking approach reduces the total run-time, provided that both

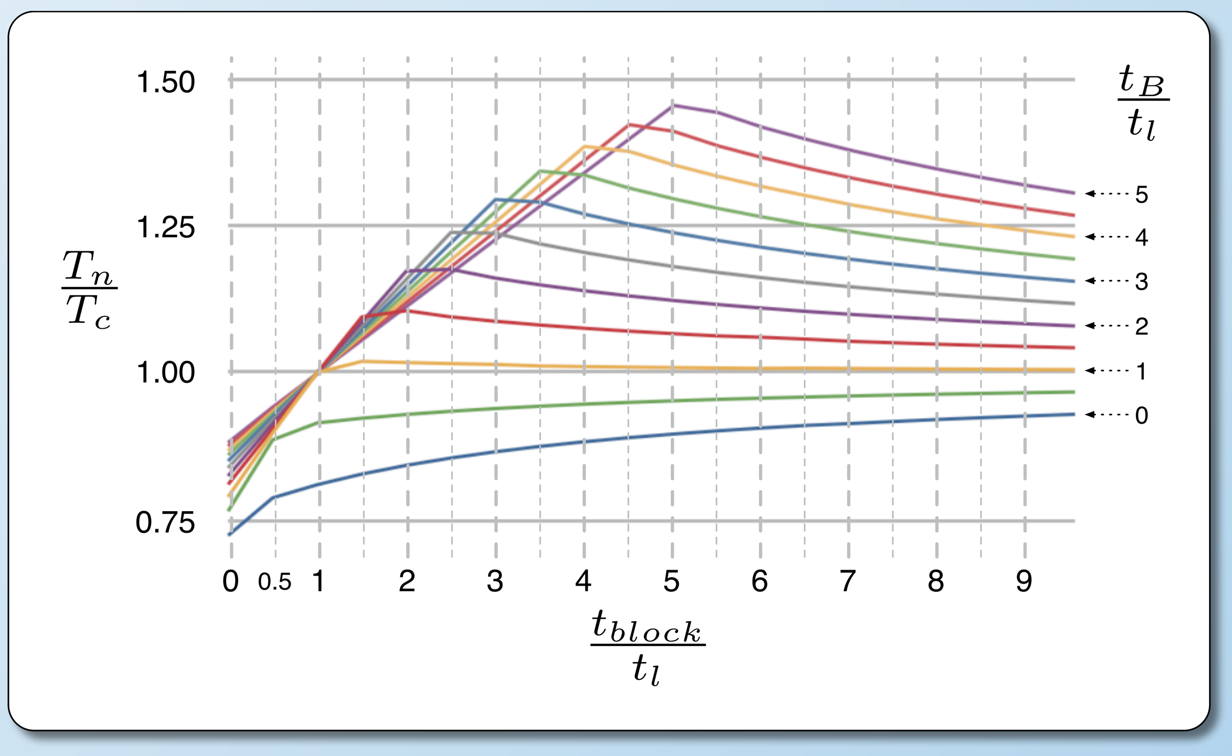
$$t_B > t_{As} + t_{Ar} \quad \text{and} \quad t_{block} > t_{As} + t_{Ar} + t_l$$

Experimental Results

- timing overheads of individual OS operations

thread initialization	1.76 ms
thread suspend	93.12 μs
thread resume	192.32 μs
state save (4096 bytes)	37.51 μs (104.1 MB/s)
state restore (4096 bytes)	45.19 μs (86.4 MB/s)
reconfiguration time (233 kBytes)	99.96 ms

- application execution time of a prototype implementation of the scheduling example



Outlook / Future Work

- efficient scheduling algorithms for a cooperatively multitasking subset of hardware threads in a preemptively scheduled multithreaded software system
- improved reconfiguration infrastructure to decrease reconfiguration overhead

References

- E. Lübbers and M. Platzner, „Multithreaded Programming for Reconfigurable Computers," *ACM Transactions on Embedded Computing Systems (TECS)*, 2009, to appear
- E. Lübbers and M. Platzner, „ReconOS: An RTOS supporting Hard- and Software Threads," in *17th IEEE International Conference on Field Programmable Logic and Applications (FPL)*, 2007

Contact

Enno Lübbers
Computer Engineering Group
University of Paderborn

enno.luebbers@uni-paderborn.de
+49 5251 605397
http://tr.im/luebbers