ReconOS: An RTOS Supporting Hardware and Software Threads

Enno Lübbers and Marco Platzner
Computer Engineering Group
University of Paderborn

{enno.luebbers, platzner}@upb.de

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Operating Systems for Reconfigurable Hardware

- traditional approaches integrate hardware accelerators as slave coprocessors

- Linux-based integration of reconfigurable logic
  - microblaze-ucLinux [Bergmann et. al. 2006]
    - preferred communication through FIFOs
  - BORPH [So et. al. 2006]
    - file-system based communication between hardware and software

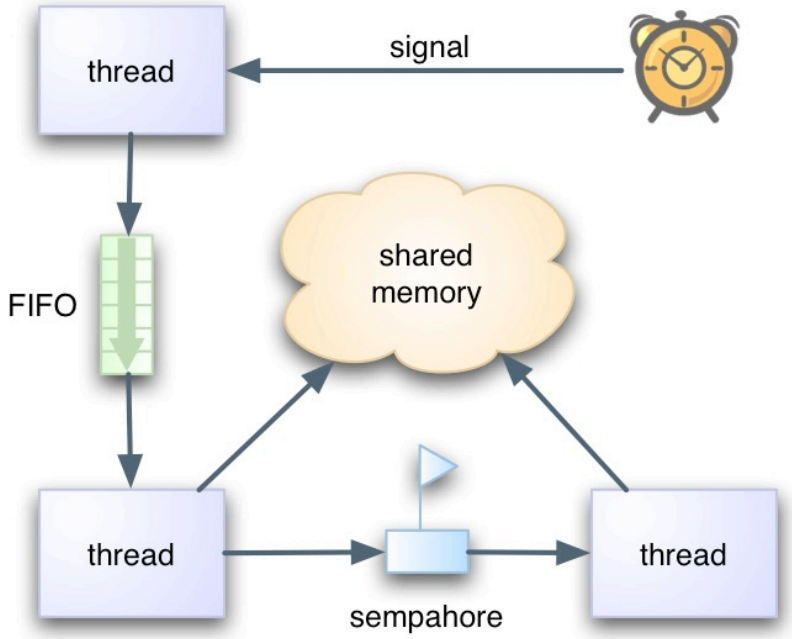
- unified programming model for software and hardware threads
  - hthreads [Peck et. al. 2006]
    - hardware threads generated from multithreaded C-source
    - OS functionality realized in hardware
  - **ReconOS**
    - based on software RTOS (eCos)
    - hardware threads are written in VHDL

Outline

- motivation
- programming model
  - operating system objects
  - hardware thread design
- execution model
  - system architecture
  - OS call delegation
  - toolchain
- experimental results
  - operating system overheads
  - case study
- conclusion & outlook

RTOS-like Programming Model

- applications are modelled with a set of objects
  - tasks/threads, semaphores, FIFOs, shared memory, timers, etc.



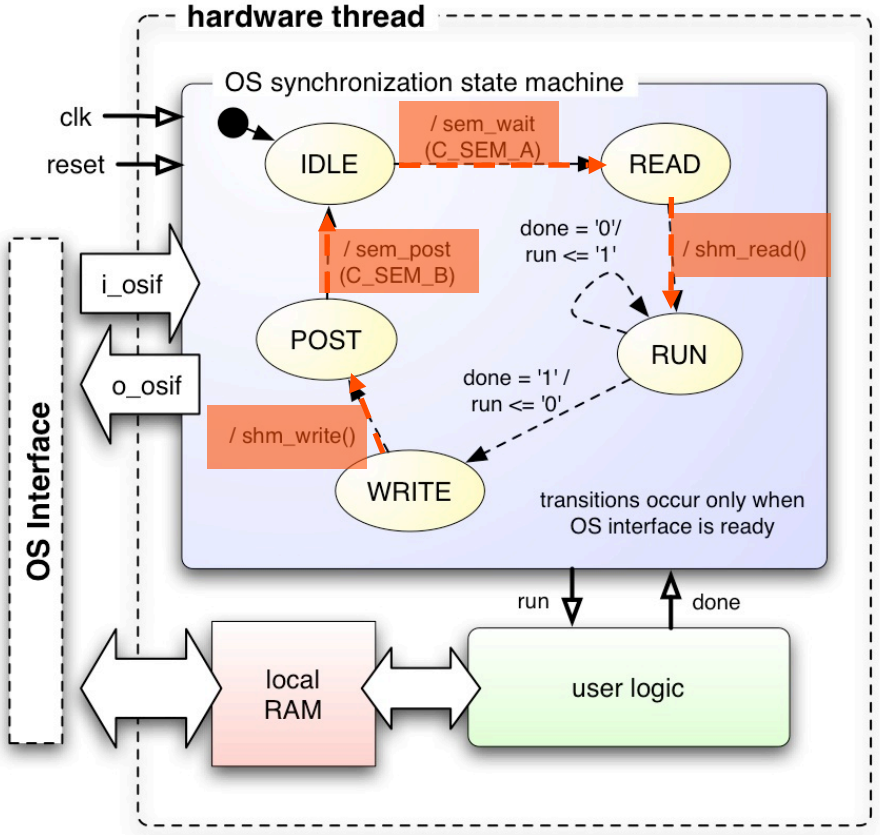
  - these objects, their semantics and possible relationships form the programming model

RTOS-like Programming Model

- classic (embedded) software implementation
  - threads interact with the OS through API functions
    - eg. semaphore_post(), thread_create(), malloc()
    - distinction between blocking and non-blocking calls
  - sequential execution of threads
- challenges in translating this model to hardware
  - hardware is inherently parallel
    - "hardware thread" is actually a misleading term
    - hardware has no notion of function calls or even blocking function calls
  - parallel execution of several hardware threads and one software thread
    - SW-HW and HW-HW synchronization and communication
    - scheduling
- ReconOS approach: extend software RTOS
  - hardware threads with OS synchronization state machine
  - delegate threads

ReconOS Hardware Threads

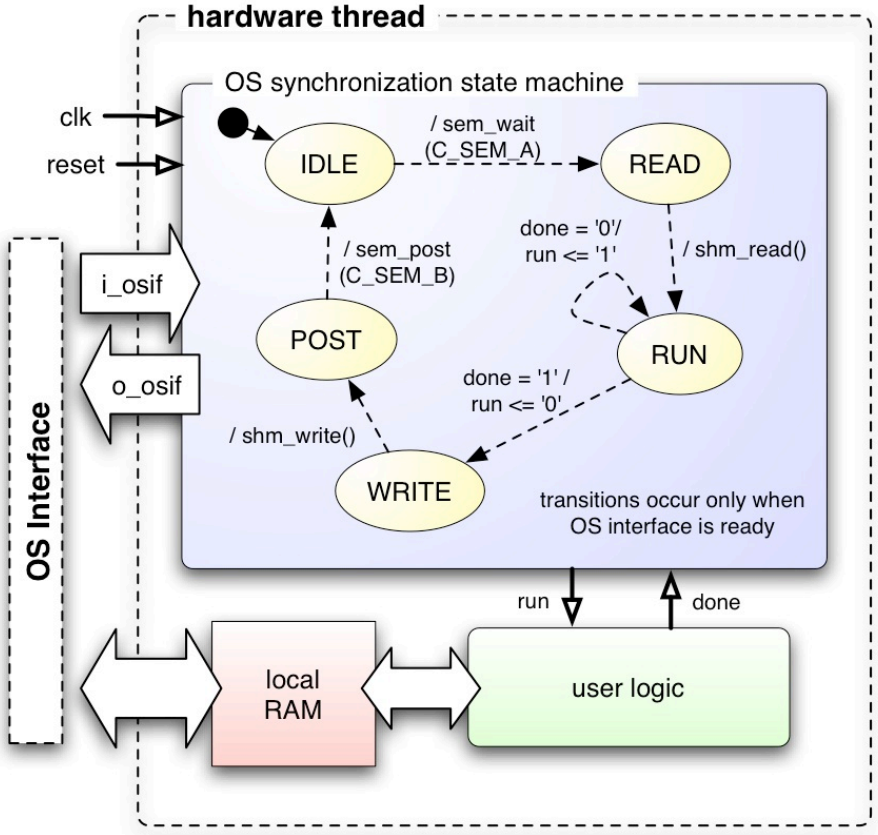
- a hardware thread consists of two parts
  - an OS synchronization state machine
    - synchronizes thread with operating system calls
    - serializes access to OS objects via the OS interface
    - can be blocked by the OS interface
  - parallel “user processes”
    - communicate with OS synchronization state machine
    - can directly access local memory blocks
    - are not necessarily blocked



ReconOS API for Hardware Threads

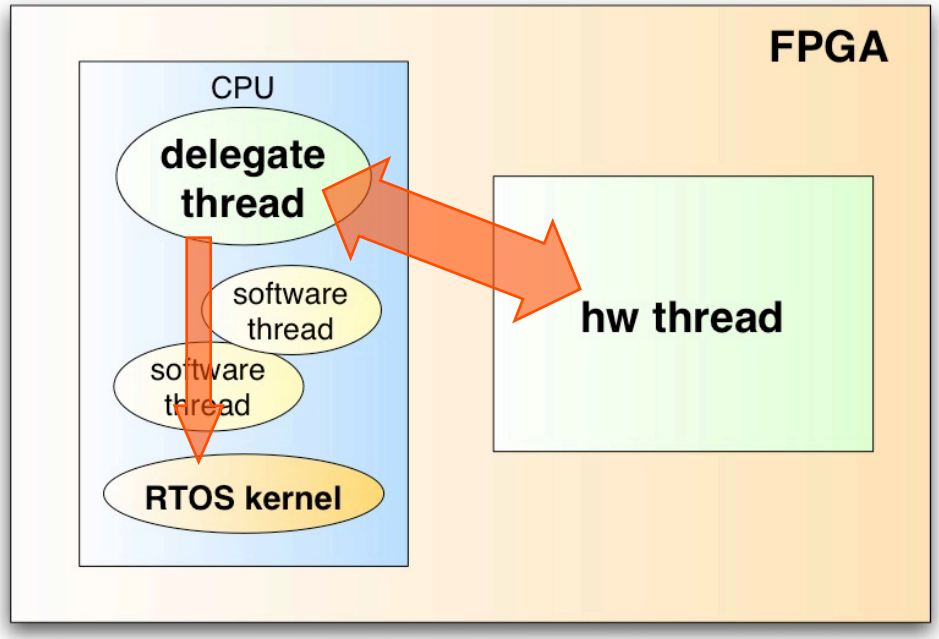
```
1  osif_fsm: process(clk, reset)
2  begin
3    if (reset = '1') then
4      state <= IDLE;
5      run <= '0';
6      reconos_reset(o_osif, i_osif);
7    elsif rising_edge(clk) then
8      reconos_begin(o_osif, i_osif);
9      if reconos_ready(i_osif) then
10       case state is
11         when IDLE =>
12           reconos_sem_wait(o_osif, i_osif, C_SEM_A);
13           state <= READ;
14
15         when READ =>
16           reconos_shm_read_burst(o_osif, i_osif,
17                                  local_address,
18                                  global_address);
19           state <= RUN;
20
21         when RUN =>
22           run <= '1';
23           if done = '1' then
24             run <= '0';
25             state <= WRITE;
26           end if;
27
28         when WRITE =>
29           reconos_shm_write_burst(o_osif, i_osif,
30                                   local_address,
31                                   global_address);
32           state <= POST;
33
34         when POST =>
35           reconos_sem_post(o_osif, i_osif, C_SEM_B);
36           state <= IDLE;
37
38         when others => null;
39       end case;
40     end if;
41   end if;
42 end process;
```

- VHDL function library
- may only be used in the OS synchronization state machine

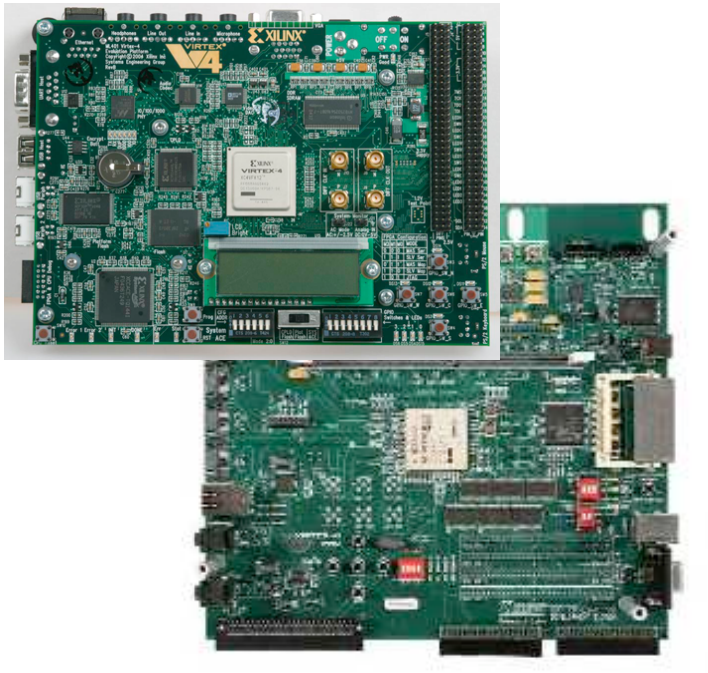


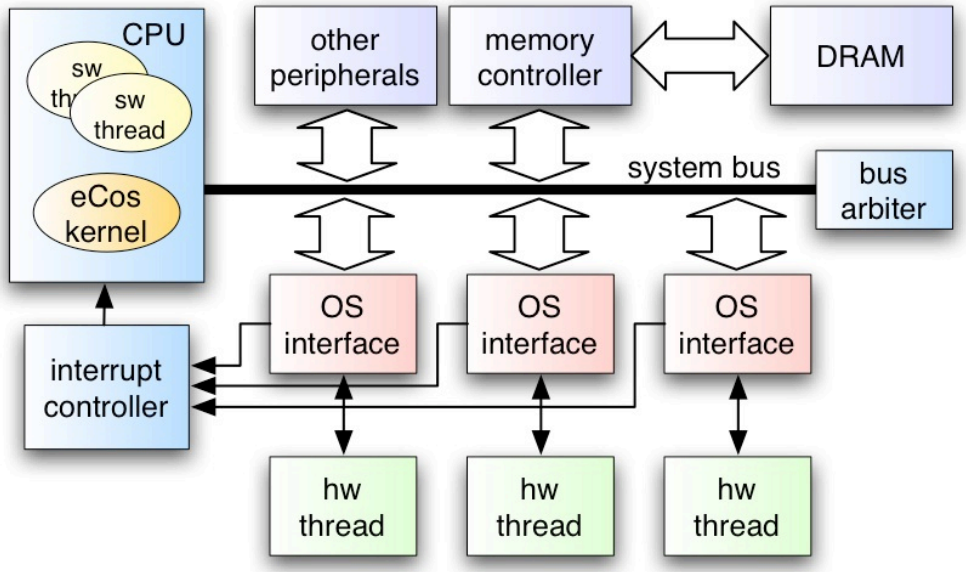
Delegate Threads

- basic mechanism
  - a delegate thread in software is associated with every hardware thread
  - the delegate thread calls the OS kernel on behalf of the hardware thread
  - all kernel responses are relayed back to the hardware thread
- advantages
  - no modification of the kernel required
  - extremely flexible
  - transparent to kernel and other threads
- drawbacks
  - increased overhead due to interrupt processing and context switch



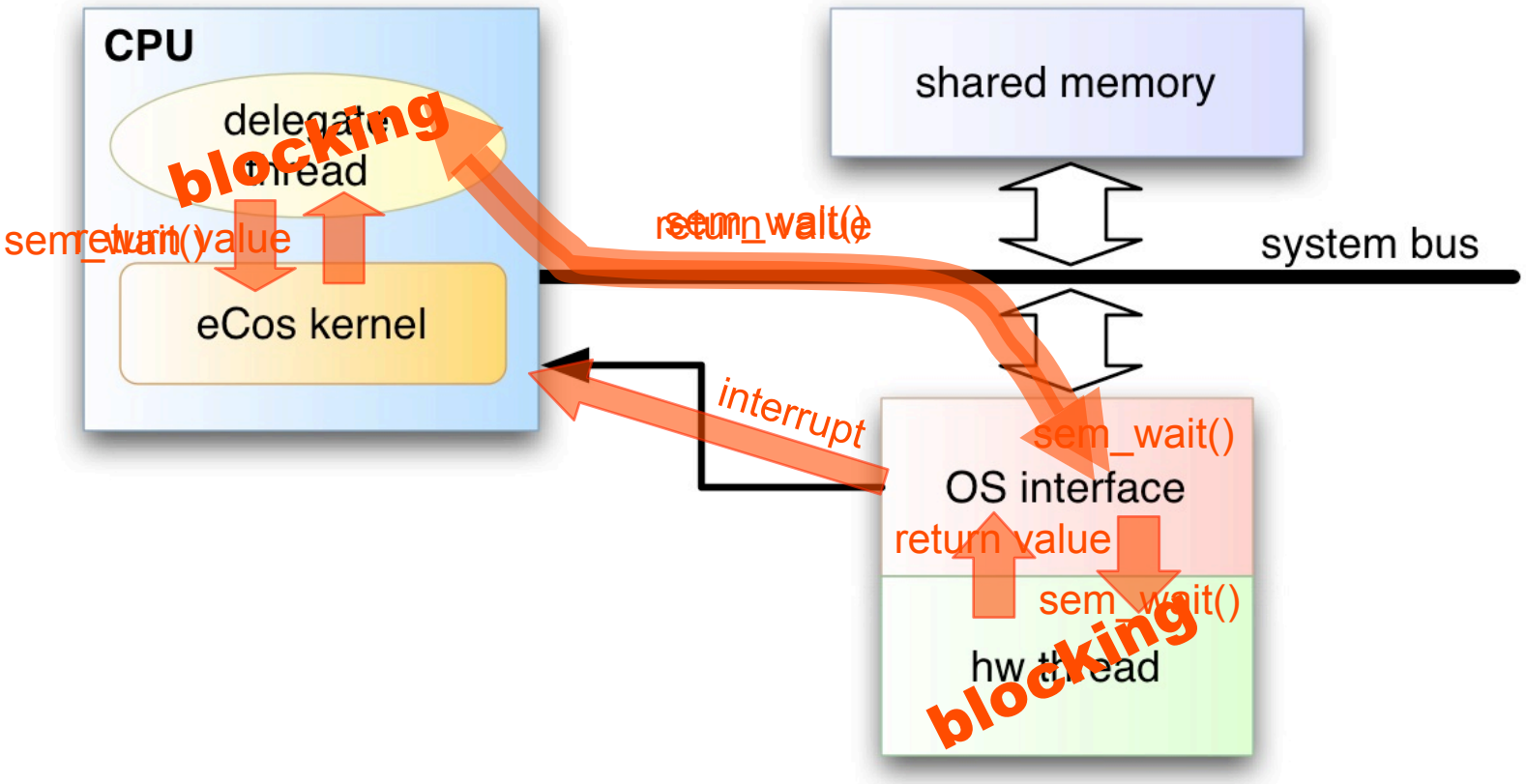
System Architecture





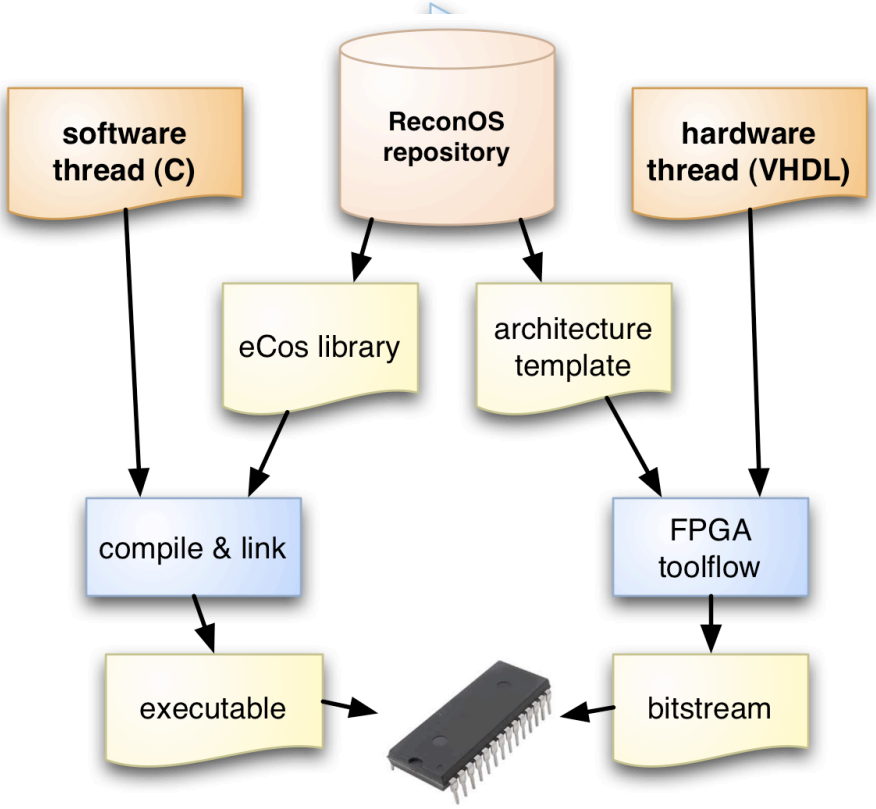
- development platforms
  - Xilinx ML403 (Virtex-4FX)
  - Xilinx XUPV2P (Virtex-II Pro)
  - embedded PowerPC 405 CPU(s)
  - CoreConnect bus architecture
  - FPGAs support partial reconfiguration
- real-time operating system
  - eCos for PowerPC ported to development platforms
  - eCos is a widely-used open source RTOS
  - modular, extensible design
  - supplemented with OS interface for hardware threads

OS Call Implementation



Toolchain

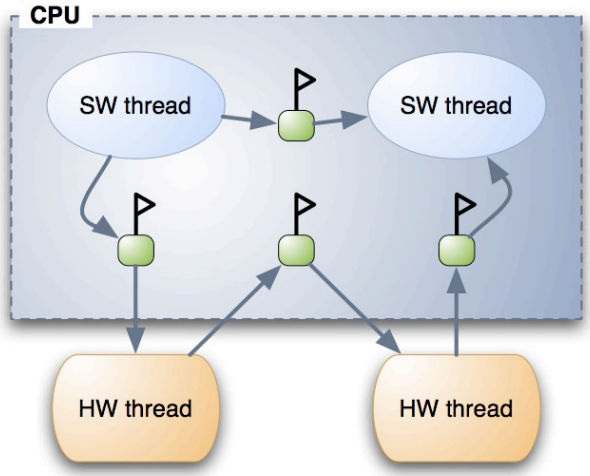
- software threads are written in C
  - using the eCos software API
- hardware threads are written in VHDL
  - using the ReconOS VHDL API
- architecture generation
  - automatically inserts OS interfaces and hardware threads into Xilinx EDK platform templates
  - configures and builds static eCos library



- eCos extensions
  - hardware thread object encapsulating delegate thread and OS interface “driver”
  - profiling support to track the state of the hardware threads' OS synchronization state machines

OS Overheads

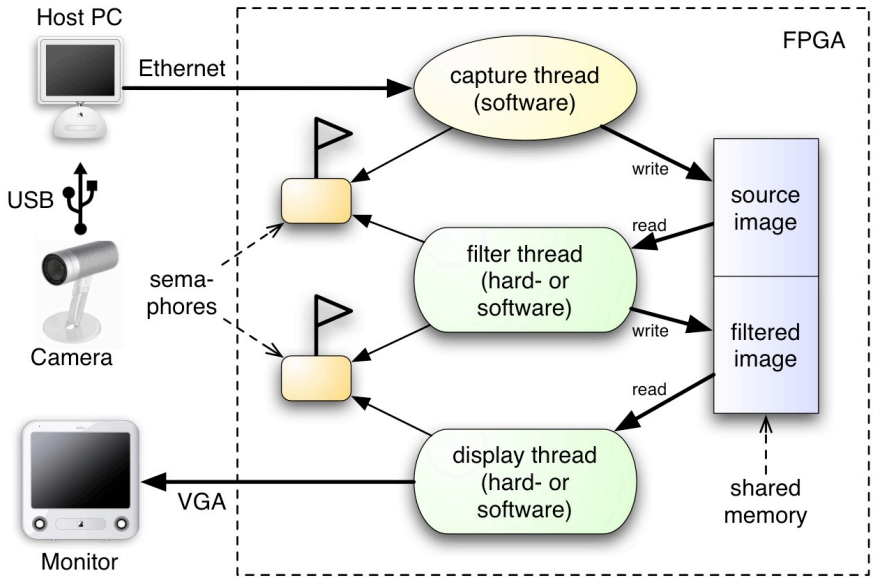
- synthetic hardware and software threads
  - semaphore processing time (post → wait)
  - time for non-blocking OS calls (i.e. `reconos_sem_post()`)
  - OS interface takes 1051 slices (7% of XC2VP30)
- OS calls involving hardware exhibit higher latencies
  - additional context switch to delegate
  - interrupt processing
  - bus access vs. cache access
- limited impact on system performance
  - logic resources mainly used for heavy data-parallel processing
  - less synchronization-intensive control dominated code



Semaphores (post → wait)	
SW-to-SW	7.69 µs
SW-to-HW	13.84 µs
HW-to-SW	27.13 µs
HW-to-HW	34.19 µs
non-blocking OS call	
SW	1.59 µs
HW	16.51 µs

Case Study - Image Processing Filter

- three threads
  - capture image from Ethernet
  - apply LaPlacian filter
  - display image on VGA monitor

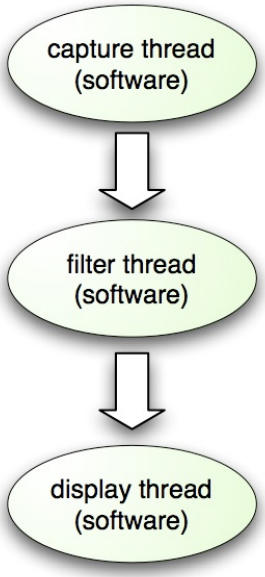


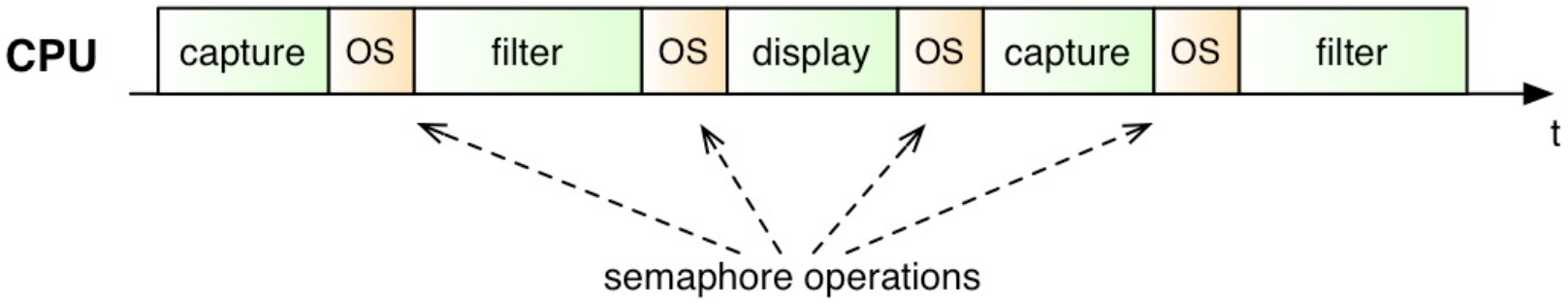
- platform
  - Xilinx XUPV2P (Virtex-II Pro)
  - PPC @ 300MHz, rest @ 100MHz

- threads communicate through shared memory
  - image resolution: 320x240 pixels, 8 bit greyscale
  - image data organized into blocks (e.g. 40 lines = 1 block)
  - a block is protected by two semaphores
    - “ready” semaphore: data can be safely written into this block
    - “new” semaphore: new data is available in this block

Case Study - Implementation #1

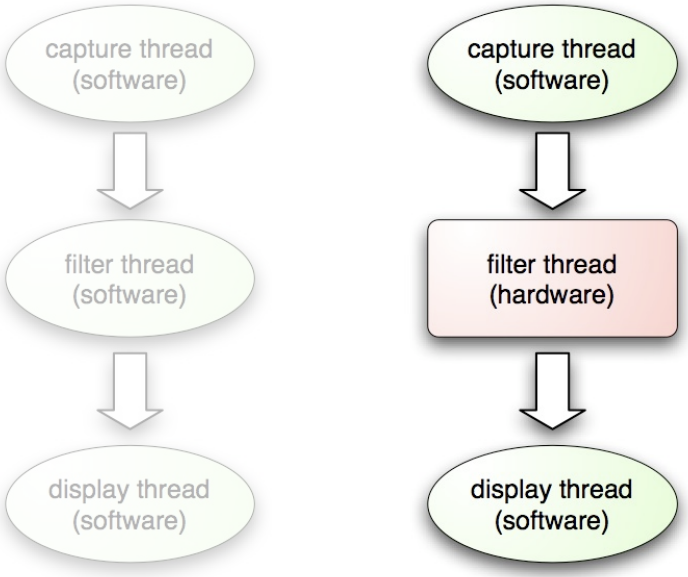
- all threads in software
  - all computations occur sequentially, with low OS overhead

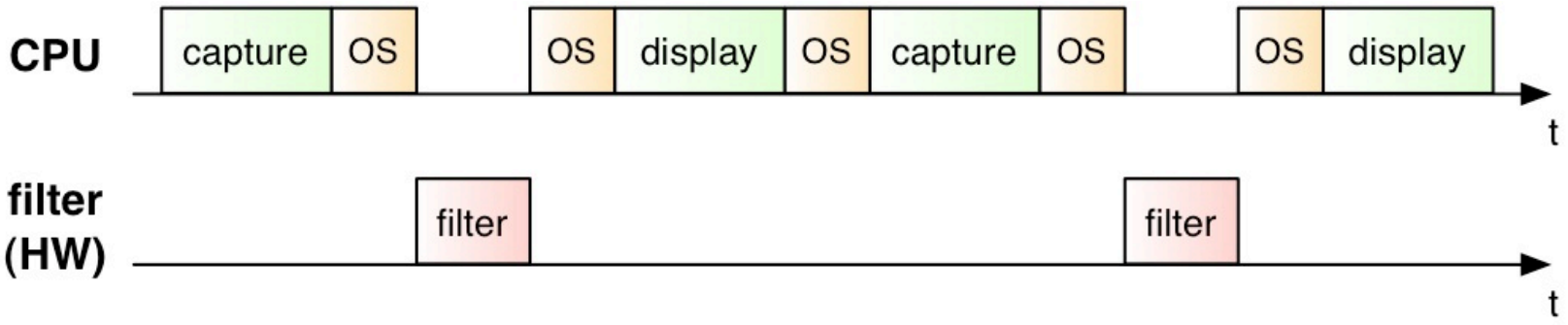




Case Study - Implementation #2

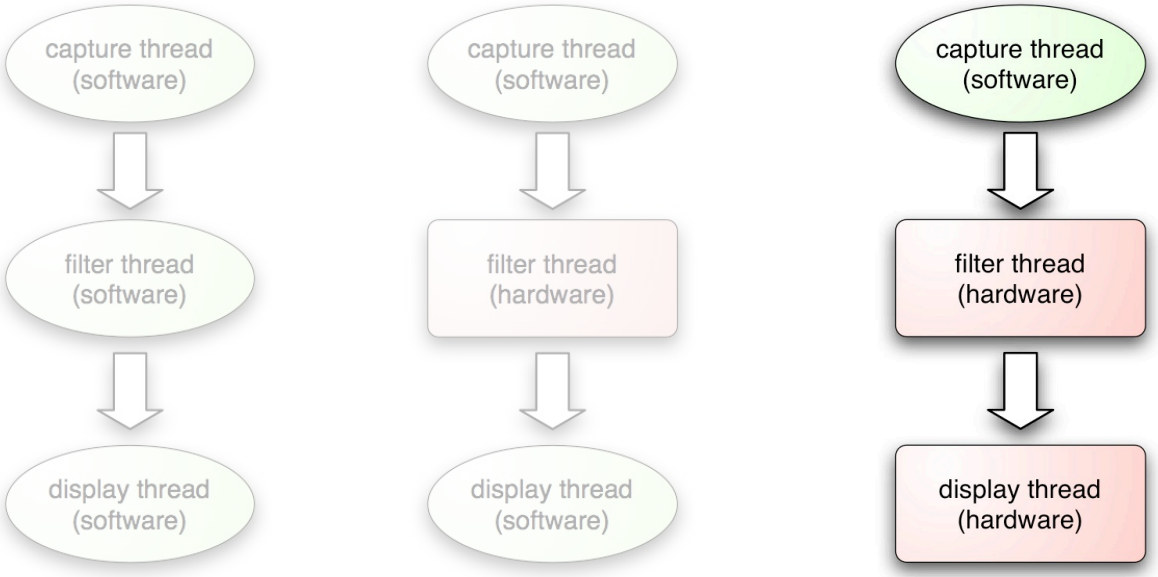
- move filter thread to hardware
  - convolution filters allow for efficient parallelization

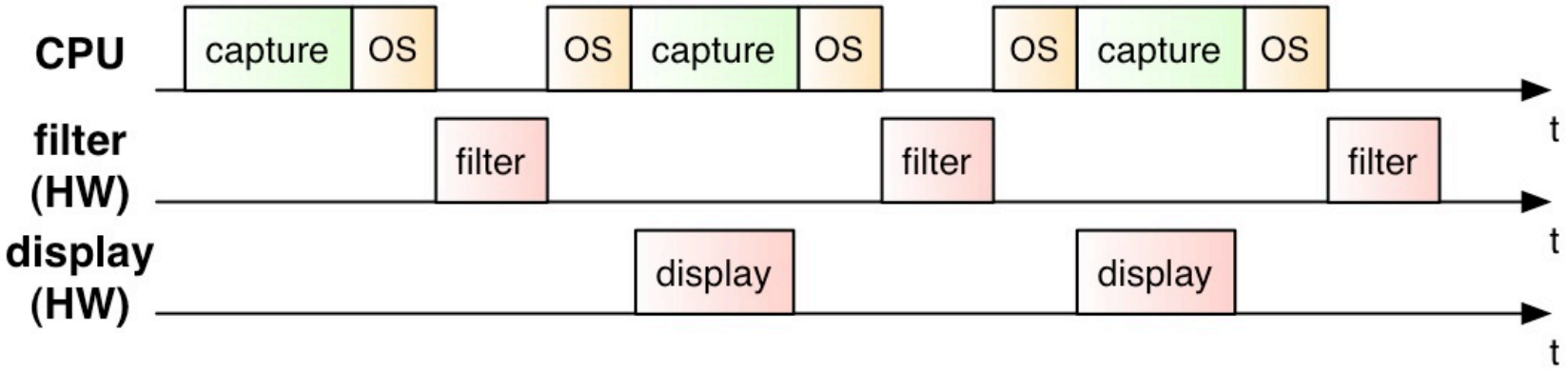




Case Study - Implementation #3

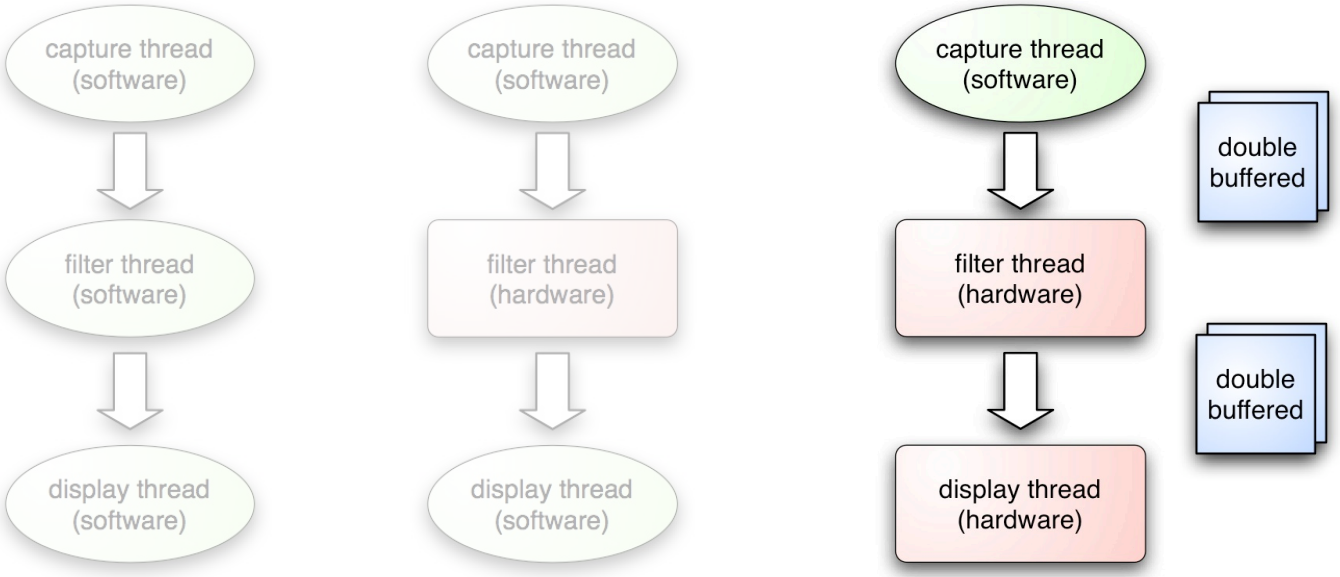
- move also display thread to hardware
  - u display thread can output data concurrently with capture thread

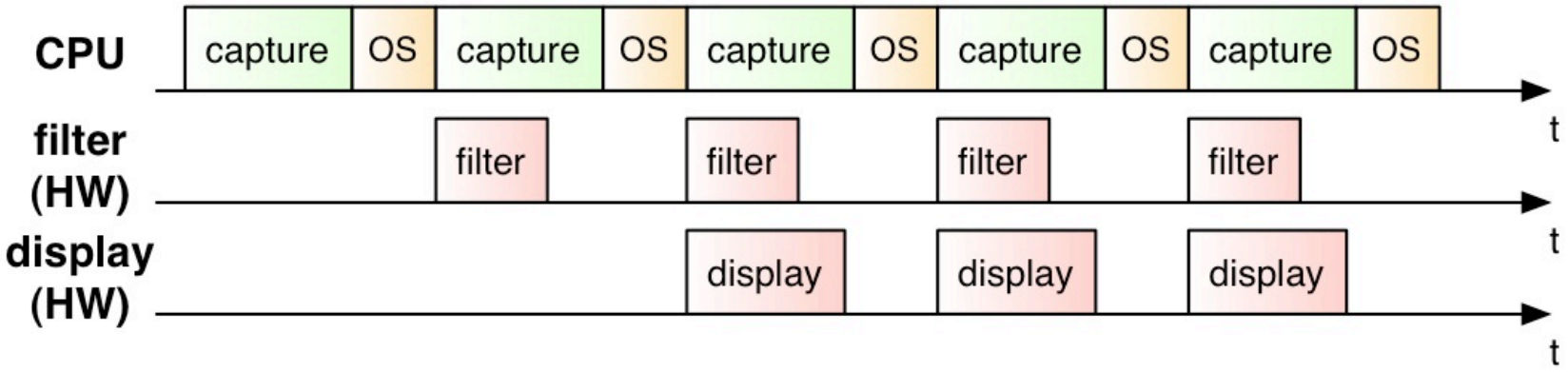




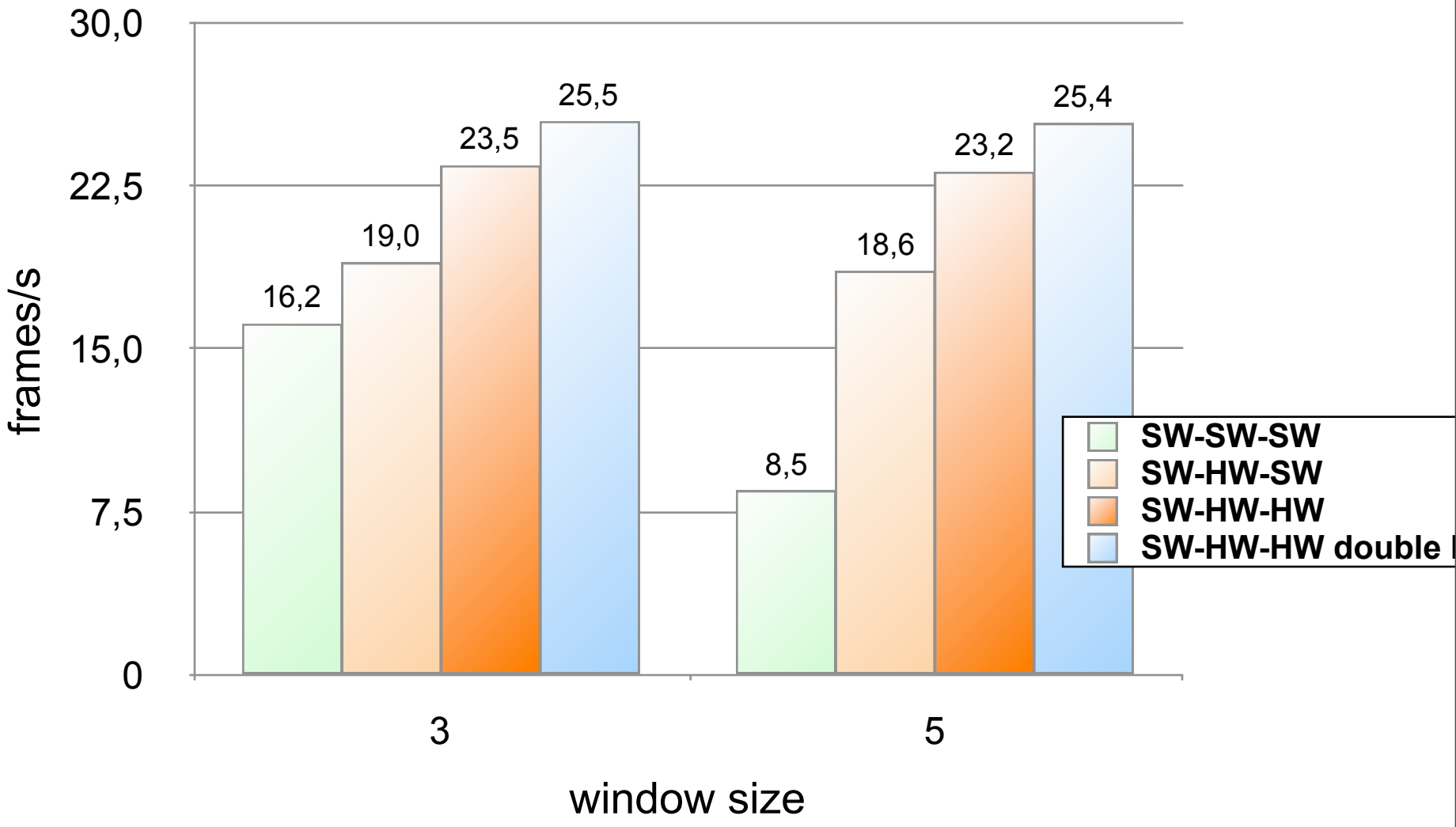
Case Study - Implementation #4

- parallel hardware threads
  - u double-buffer image data

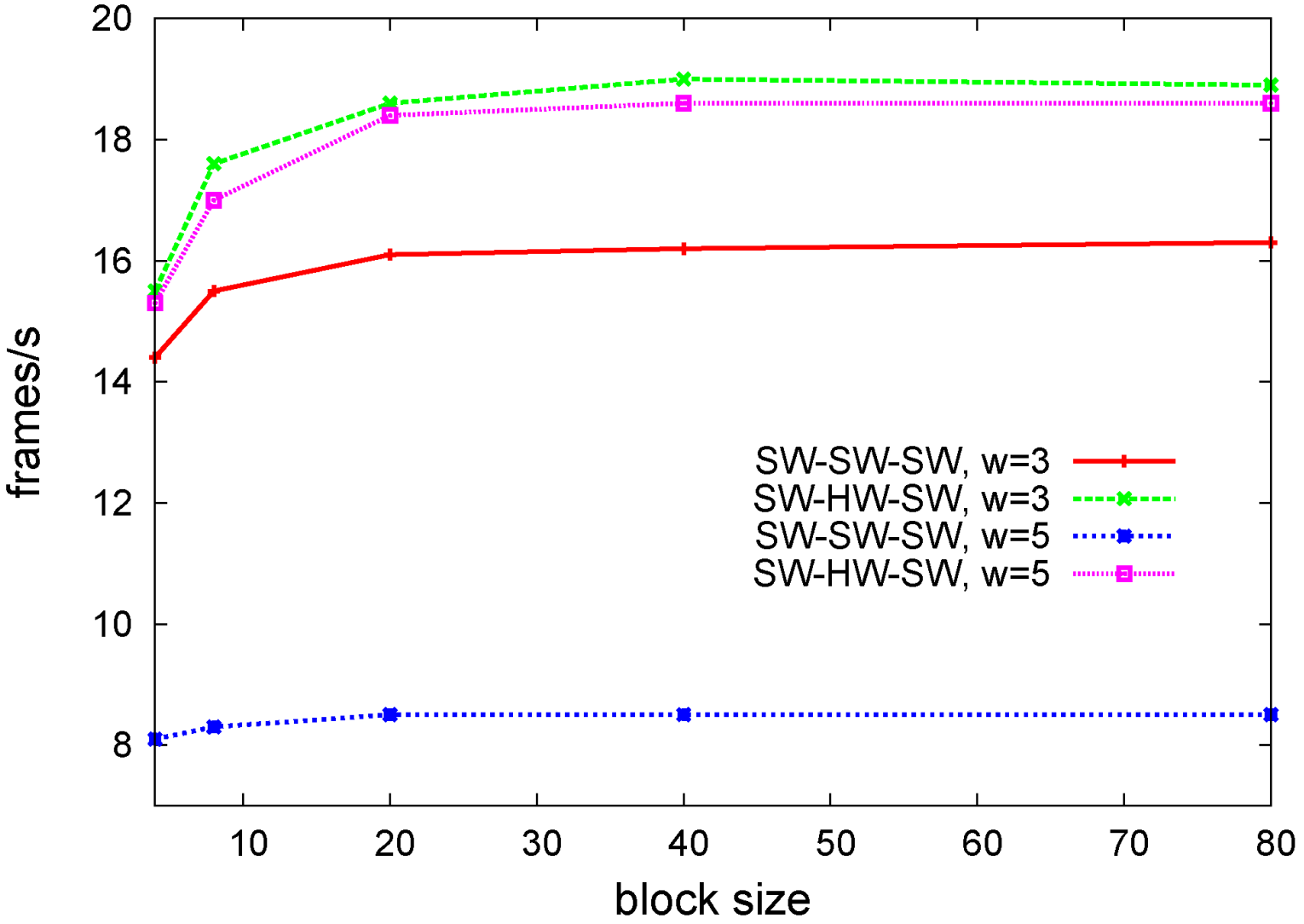




Case Study - Results



Case Study - Results



Conclusion & Outlook

- RTOS for hardware and software threads
  - unified programming model
    - transparent synchronization and communication between hardware and software threads
  - RTOS-centric execution model
    - extended eCos with support for hardware threads
  - case study

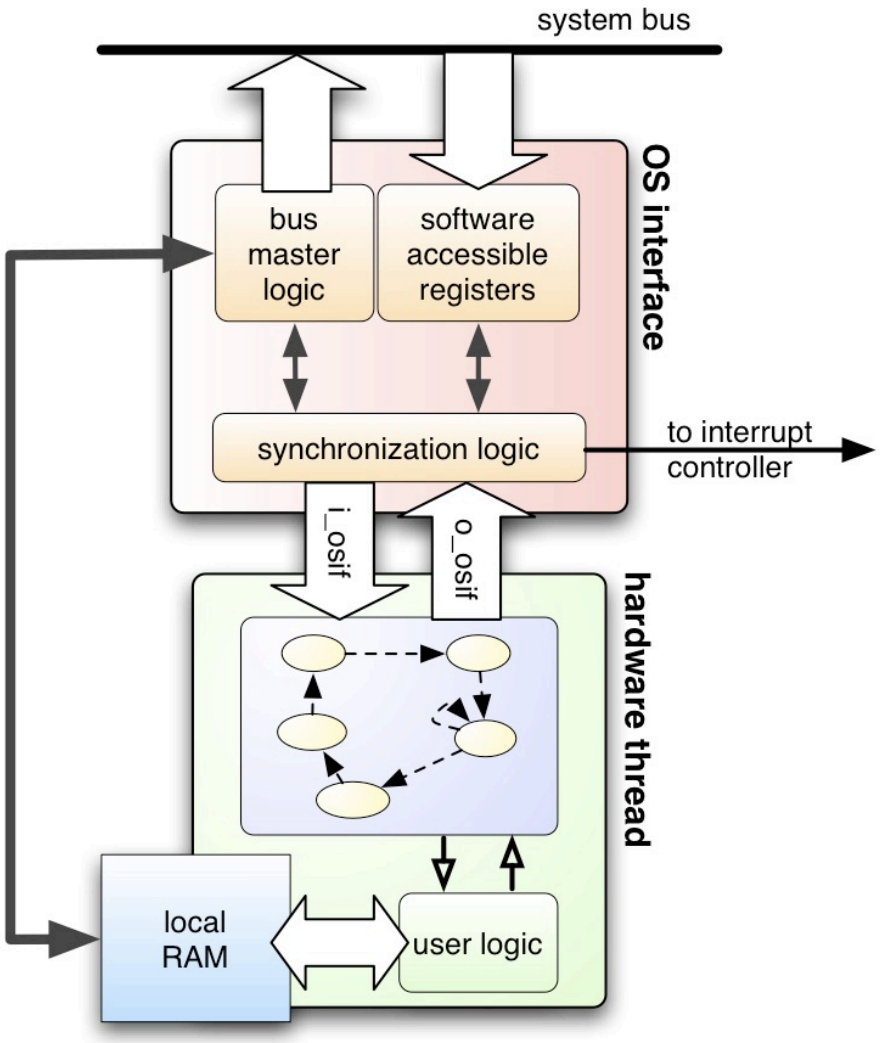
- ongoing work
  - include partial reconfiguration
    - extend eCos scheduler
    - preemption, task migration
  - additional platforms
    - Erlangen Slot Machine (ESM)

Thank you

www.reconos.de

OS Interface Implementation

- processes requests from hardware thread
  - handles blocking and resuming of hardware thread
- master interface
  - memory accesses are handled directly
  - single word and burst transfers
  - direct access to entire system's address space (memory and peripherals)
- slave interface
  - OS object interactions are relayed to delegate thread
  - dedicated CPU interrupt
  - CPU addressable registers
  - used for OS communication



OS Overheads

- OS call delays exhibit sporadic glitches
  - due to unpredictable bus arbitration
  - fix: use separate communication channel for OS calls and memory access