A Portable Abstraction Layer for Hardware Threads

Enno Lübbers and Marco Platzner
Computer Engineering Group
University of Paderborn

{enno.luebbers, platzner}@upb.de
Design of CPU/FPGA Systems

- hardware modules typically integrated as slave coprocessors
- hardware/software boundary explicit
- tedious and error-prone to program
- portability issues
Design of CPU/FPGA Systems

- hardware modules typically integrated as slave coprocessors
- hardware/software boundary explicit
- tedious and error-prone to program
- portability issues
Multithreaded Programming

application
Multithreaded Programming

- multithreaded programming model
Multithreaded Programming

- multithreaded programming model
Multithreaded Programming

- multithreaded programming model
**Multithreaded Programming**

- multithreaded programming model
  - extended to reconfigurable hardware (ReconOS)
multithreaded programming model
- extended to reconfigurable hardware (ReconOS)
Multithreaded Programming

- multithreaded programming model
  - extended to reconfigurable hardware (ReconOS)
  - provides transparent synchronization and communication b/w threads
multithreaded programming model
- extended to reconfigurable hardware (ReconOS)
- provides transparent synchronization and communication b/w threads
- operating system provides low-level synchronization and communication
Application Domains
Application Domains

- multithreaded programming model applicable to several application domains, e.g.
Application Domains

- multithreaded programming model applicable to several application domains, e.g.

- reconfigurable embedded computing
  - efficient exploitation of fine-grained parallelism with tight constraints (memory, area, power, processor performance)
  - demand for easy design space exploration regarding HW/SW partitioning
  - short reaction times, possibly real-time requirements
Application Domains

multithreaded programming model applicable to several application domains, e.g.

- reconfigurable embedded computing
  - efficient exploitation of fine-grained parallelism with tight constraints (memory, area, power, processor performance)
  - demand for easy design space exploration regarding HW/SW partitioning
  - short reaction times, possibly real-time requirements

- reconfigurable high-performance computing
  - transparent communication and synchronization in heterogeneous execution environments (e.g. CPU nodes + FPGA accelerators)
  - exploitation of both fine-grained and thread-level parallelism, possibly across multiple machines
Application Domains

- multithreaded programming model applicable to several application domains, e.g.
  
- reconfigurable embedded computing
  - efficient exploitation of fine-grained parallelism with tight constraints (memory, area, power, processor performance)
  - demand for easy design space exploration regarding HW/SW partitioning
  - short reaction times, possibly real-time requirements

- reconfigurable high-performance computing
  - transparent communication and synchronization in heterogeneous execution environments (e.g. CPU nodes + FPGA accelerators)
  - exploitation of both fine-grained and thread-level parallelism, possibly across multiple machines

- show applicability of ReconOS approach across different host operating systems and CPU/FPGA architectures
Overview

- motivation

- ReconOS abstraction layer
  - programming model
  - hardware architecture
  - hardware threads
  - OS interface & delegate threads

- host OS implementations
  - ReconOS/eCos
  - ReconOS/Linux

- experimental results

- conclusion
Programming Model

- applications are divided into threads
- threads communicate via operating system objects
  - semaphores
  - mailboxes
  - shared memory
  - ...

MBOX_IN1
MBOX_IN2
MBOX_DATA
MBOX_OUT

SEM_NEW
SEM_READY

THREAD_A
THREAD_B
THREAD_C

shared memory
Programming Model

- applications are divided into threads
- threads communicate via operating system objects
  - semaphores
  - mailboxes
  - shared memory
  - ...

examples for API functions used by threads

<table>
<thead>
<tr>
<th>software (POSIX, C)</th>
<th>hardware (ReconOS, VHDL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sem_post()</td>
<td>reconos_sem_post()</td>
</tr>
<tr>
<td>pthread_mutex_lock()</td>
<td>reconos_mutex_lock()</td>
</tr>
<tr>
<td>mq_send()</td>
<td>reconos_mbox_put()</td>
</tr>
<tr>
<td>value = *ptr</td>
<td>reconos_read()</td>
</tr>
<tr>
<td>pthread_exit()</td>
<td>reconos_thread_exit()</td>
</tr>
</tbody>
</table>
Hardware Architecture

- based on CoreConnect bus topology
- system CPU runs OS kernel and software threads
- hardware threads are synthesized to FPGA fabric
  - connected to OS kernel via OS interface modules and buses

Diagram:
- CPU
- OS kernel
- I/O controller
- memory controller
- external memory
- system buses (PLB, DCR)
- interrupt controller
Hardware Architecture

- based on CoreConnect bus topology
- system CPU runs OS kernel and software threads
- hardware threads are synthesized to FPGA fabric
  - connected to OS kernel via OS interface modules and buses
Hardware Architecture

- based on CoreConnect bus topology
- system CPU runs OS kernel and software threads
- hardware threads are synthesized to FPGA fabric
  - connected to OS kernel via OS interface modules and buses
Hardware Architecture

- based on CoreConnect bus topology
- system CPU runs OS kernel and software threads
- hardware threads are synthesized to FPGA fabric
  - connected to OS kernel via OS interface modules and buses
ReconOS API for Hardware Threads

- VHDL function library
- used similar to software API
- may only be used inside OS synchronization state machine

```vhdl
osif fsm: process(clk, reset)
begin
  if (reset = '1') then
    state <= IDLE;
    run <= '0';
    reconos_reset(o_osif, i_osif);
  elsif rising_edge(clk) then
    reconos_begin(o_osif, i_osif);
  if reconos_ready(i_osif) then
    case state is
    when IDLE =>
      reconos_sem_wait(o_osif, i_osif, C_SEM_A);
      state <= READ;
    when READ =>
      reconos_shm_read_burst(o_osif, i_osif,
                             local_address,
                             global_address);
      state <= RUN;
    when RUN =>
      run <= '1';
      if done = '1' then
        run <= '0';
        state <= WRITE;
      end if;
    when WRITE =>
      reconos_shm_write_burst(o_osif, i_osif,
                              local_address,
                              global_address);
      state <= POST;
    when POST =>
      reconos_sem_post(o_osif, i_osif, C_SEM_B);
      state <= IDLE;
    when others => null;
    end case;
  end if;
end process;
```

OS Interface

- clk
- reset

Hardware Thread

- IDLE
  - /sem_wait(C_SEM_A)
  - done = '0'/run <= '1'

- READ
  - /sem_post(C_SEM_B)
  - done = '1'/run <= '0'

- POST
  - /shm_write()

- WRITE
  - /shm_read()

- RUN
  - transitions occur only when OS interface is ready

User Logic

- local RAM
- done

E.Lübbers & M.Platzner, University of Paderborn
OS Interface and Delegate Threads

- **OS interface**
  - processes requests from HW thread
  - relays OS object interactions to CPU
  - executes memory accesses
  - provides dedicated FIFO channels
**OS Interface and Delegate Threads**

- **OS interface**
  - processes requests from HW thread
  - relays OS object interactions to CPU
  - executes memory accesses
  - provides dedicated FIFO channels

- **delegate thread**
  - associated with every hardware thread
  - calls kernel functions on behalf of hardware thread
OS Interface and Delegate Threads

- **OS interface**
  - processes requests from HW thread
  - relays OS object interactions to CPU
  - executes memory accesses
  - provides dedicated FIFO channels

- **delegate thread**
  - associated with every hardware thread
  - calls kernel functions on behalf of hardware thread

- provide stable API on different OS’s and platforms
  - OS interface manages low-level communication to CPU and memory
  - delegate translates HW thread requests to OS kernel API
Overview

- motivation

- ReconOS abstraction layer
  - programming model
  - hardware architecture
  - hardware threads
  - OS interface & delegate threads

- host OS implementations
  - ReconOS/eCos
  - ReconOS/Linux

- experimental results

- conclusion
eCos
- configurable, small-footprint operating system for embedded domain
- all code executes in kernel mode; simple hardware access possible

OS call sequence
- hardware thread initiates request; OS interface raises interrupt
- delegate is synchronized to interrupts through semaphores
- delegate thread is woken up and retrieves OS call and parameters
**OS Call Sequence (Linux)**

- **Linux**
  - flexible and widely used OS for embedded and HPC domain
  - no direct hardware access possible from Linux user space; needs driver

- **OS call sequence**
  - hardware thread initiates request; OS interface raises interrupt
  - delegate is synchronized to interrupts through blocking filesystem accesses
  - delegate thread is woken up and retrieves OS call and parameters
Tool Flow

- .c (software threads)
- .vhd (hardware threads)
- FPGA
- CPU
- buses
- peripherals
- reference design
Tool Flow

- .c: software threads
- .vhd: hardware threads

Reference Design

Platform Generation

Synthesis / Place & Route

CPU

buses

peripherals

FPGA

.bit
Tool Flow

- eCos
  - delegate threads
  - ReconOS/eCos (static library)
  - OSIF driver

- software threads
- compile & link

- .C

- hardware threads
- .vhd

- platform generation

- .vhd

- synthesis / place & route

- .bit

- CPU
- buses
- peripherals

- reference design

- ReconOS/eCos (static library)
- OSIF driver

- .elf

- .elf

- FPGA

- compile & link
Tool Flow

Linux

- libreconos (static library)
- delegate threads
- compile & link
- .elf
- OSIF driver
- linux.elf

eCos

- ReconOS/ eCos (static library)
- delegate threads
- compile & link
- .elf
- .vhd
- hardware threads
- platform generation
- synthesis / place & route

FPGA

- CPU
- buses
- peripherals
- reference design
- .bit
Overview

- motivation

- ReconOS abstraction layer
  - programming model
  - hardware architecture
  - hardware threads
  - OS interface & delegate threads

- host OS implementations
  - ReconOS/eCos
  - ReconOS/Linux

- experimental results

- conclusion
OS Call Overheads

- synthetic hardware and software threads
  - semaphore and mutex processing time (post → wait / unlock → lock)

- executed on three prototypes
  - eCos/PPC
    - XC2VP30
    - PowerPC405 @300MHz
    - HW threads & bus @100MHz
  
  - Linux/PPC
    - XC2VP30
    - PowerPC405 @300MHz
    - HW threads & bus @100MHz
  
  - Linux/MicroBlaze
    - XC4VSX35
    - MicroBlaze 4.0 @100Mhz
    - HW threads & bus @100MHz

Semaphore Operations

- SW post
- HW post
- SW → HW
- SW ← HW
- HW → SW
- HW ← SW

µs

- eCos/PPC
- Linux/PPC
- Linux/MicroBlaze
Application Case Study

- sort application
  - sorts an array of integers (1MB) using a combination of bubble sort and merge sort
  - sort thread can be executed either in hardware or software

→ OS call overhead not a major factor in overall performance
Conclusion & Outlook

- we extended the established multithreaded programming model to reconfigurable hardware
- unified set of abstractions for hard- and software threads provides portability across different host OS‘s and CPU/FPGA architectures
- the additional abstraction layer shows acceptable performance in benchmarks and larger case studies

future work
- implementation on FPGA accelerators for high-performance computing
- extension of OS scheduler to allow hardware thread scheduling using partial reconfiguration
Overview

- motivation

- ReconOS abstraction layer
  - programming model
  - hardware architecture
  - hardware threads
  - OS interface & delegate threads

- host OS implementations
  - ReconOS/eCos
  - ReconOS/Linux

- experimental results

- conclusion

Thank you

www.reconos.de
Thank you

www.reconos.de
command decoder 23%
bus slave registers 12%
fifo manager 5%

Total OSIF slice count: 1213 slices
- most of this taken up by PLB IPIF logic
Supported OS Calls

- **Semaphores (counting and binary)**
  - `reconos_semaphore_post()`
  - `reconos_semaphore_wait()`

- **Mutexes**
  - `reconos_mutex_lock()`
  - `reconos_mutex_trylock()`
  - `reconos_mutex_unlock()`
  - `reconos_mutex_release()`

- **Condition Variables**
  - `reconos_cond_wait()`
  - `reconos_cond_signal()`
  - `reconos_cond_broadcast()`

- **Mailboxes**
  - `reconos_mbox_get()`
  - `reconos_mbox_tryget()`
  - `reconos_mbox_put()`
  - `reconos_mbox_tryput()`

- **Memory access**
  - `reconos_read()`
  - `reconos_write()`
  - `reconos_read_burst()`
  - `reconos_write_burst()`

- **Basic synchronization primitives**
  - Synchronize access to mutual exclusive operations (critical sections)

- **Message passing primitives**
  - Allow waiting until arbitrary conditions are satisfied

- **CPU-independent access**
  - To the entire system address space (memory and peripherals)

- **handled in software (via delegate thread)**

- **handled in hardware (via system bus / point-to-point links)**
ReconOS Software API (POSIX)

- standard POSIX thread creation
- ReconOS hardware thread creation

```c
mqd_t my_mbox;
sem_t my_sem;

pthread_t thread;
pthread_attr_t thread_attr;
...
pthread_attr_init(&thread_attr);

pthread_create(
    &thread,             // thread object
    &thread_attr,        // attributes
    thread_entry,        // entry point
    ( void * ) data      // entry data
);

mqd_t my_mbox;
sem_t my_sem;
reconos_res_t thread_resources[2] = {
    { &my_mbox, POSIX_MQD_T },
    { &my_sem,  POSIX_SEM_T }
};
rthread thread;
pthread_attr_t thread_swattr;
rthread_attr_t thread_hwattr;
...
pthread_attr_init(&thread_swattr);
prthread_attr_init(&thread_hwattr);
rthread_attr_setslotnum(&thread_hwattr, 0);
rthread_attr_setresources(&thread_hwattr,
        thread_resources, 2);
rthread_create(
    &thread,             // thread object
    &thread_swattr,       // software attributes
    &thread_hwattr,       // hardware attributes
    ( void * ) data       // entry data
);
```
Multi-Cycle Commands

- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles
Multi-Cycle Commands

- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles

![Diagram showing the flow of commands and states between HW thread, OSIF, and CPU.](image)

state = A

state = B

cmd + data(0)

reconos call

step = 0

command
data

busy/block

step
data

HW thread

OSIF

CPU
Multi-Cycle Commands

- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles

![Diagram showing multi-cycle commands](image)
Multi-Cycle Commands

- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles
Multi-Cycle Commands

- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles

Diagram:

- **State A**: Initial state
- **HW thread**: Receives command and data
- **OSIF**: Recons call and sends cmd + data(0)
- **CPU**: Receives cmd + data(0) and blocks
- **State B**: Transition to next state
- **HW thread**: Receives cmd + data(1)
- **OSIF**: Busy/block and sends function call
- **CPU**: Receives function call and unblocks
- **State B**: Transition to next state
- **HW thread**: Receives return value and unblocks
- **OSIF**: Receives return value
- **CPU**: Receives return value and unblocks
- **State B**: Transition to next state
- **HW thread**: Receives call finished

E. Lübbers & M. Platzner, University of Paderborn
Multi-Cycle Commands

- transfer of multiple parameters and return values with a single VHDL call
- distributes execution of an FSM state across multiple clock cycles
Toolchain

- software thread (C)
- ReconOS repository
- hardware thread (VHDL)
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
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
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
Case Study - Image Processing Filter

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

- 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

![Diagram of image processing system]

![Bar chart showing performance results with different window sizes]
Case Study - Image Processing Filter

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

- **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 - Results

frames/s

window size

SW-SW-SW
SW-HW-SW
SW-HW-HW
SW-HW-HW double buffered
Case Study - Results

- SW-SW-SW
- SW-HW-SW
- SW-HW-HW
- SW-HW-HW double buffered
OS Interface

64 bit PLB (memory bus)

PLB slave attachment
bus master controller

command decoder

DCR slave attachment
FIFO manager

OS interface

to interrupt controller

local RAM
user logic

hardware thread
to/from other threads

32 Bit DCR (control bus)

FIFO

FIFO

E.Lübbers & M.Platzner, University of Paderborn

FPL ‘08

27
OS Interface

- processes requests from hardware thread
  - handles blocking and resuming of hardware thread
OS Interface

- processes requests from hardware thread
  - handles blocking and resuming of hardware thread
- relays OS object interactions to CPU
  - DCR interface with bus-addressable registers
  - dedicated interrupt

Diagram:
- 64 bit PLB (memory bus)
- 32 Bit DCR (control bus)
- PLB slave attachment
- bus master controller
- DCR slave attachment
- command decoder
- FIFO manager
- local RAM
- user logic
- hardware thread
- to/from other threads
- FIFO
- OS interface

OS Interface

- processes requests from hardware thread
  - handles blocking and resuming of hardware thread
- relays OS object interactions to CPU
  - DCR interface with bus-addressable registers
  - dedicated interrupt
OS Interface

- processes requests from hardware thread
  - handles blocking and resuming of hardware thread

- relays OS object interactions to CPU
  - DCR interface with bus-addressable registers
  - dedicated interrupt

- executes memory accesses
  - PLB master interface
  - direct access to entire system’s address space (memory and peripherals)
OS Interface

- processes requests from hardware thread
  - handles blocking and resuming of hardware thread

- relays OS object interactions to CPU
  - DCR interface with bus-addressable registers
  - dedicated interrupt

- executes memory accesses
  - PLB master interface
  - direct access to entire system’s address space (memory and peripherals)

- dedicated FIFO channels
  - provide high-throughput hardware support for message passing
Delegate Threads
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
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
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

- **portability**
  - delegate acts as *protocol converter* between HW thread and OS kernel
  - only the delegate thread code needs to be changed to support a new OS API