ReconOS: Multithreaded Programming and Execution Models for Reconfigurable Hardware

Enno Lübbers and Marco Platzner
Computer Engineering Group
University of Paderborn

{enno.luebbers, platzner}@upb.de
Outline

- motivation & approach

- ReconOS
  - programming model
  - execution model
  - performance & overheads

- ongoing development & further research

- cooperations
Motivation

- rising complexity of reconfigurable devices
  - increasing logic density
  - increasing design complexity
  - addition of hard IP cores, e.g. processors, memories

- platform FPGAs / reconfigurable System-on-Chip (rSoC)
  - explicit hardware-software interface
  - mostly incompatible tools and IP cores from different vendors
  - limited re-use of hardware modules / portability

→ programming models for rSoCs cannot keep up with capabilities of modern FPGAs
Motivation

- traditional approaches for CPU/FPGA systems typically integrate hardware accelerators as slave coprocessors
  - software needs to explicitly communicate with accelerators (e.g. using device registers)
  - difficult and tedious to program (too close to hardware)
  - portability and scalability issues

```c
int main() {
    ...
    data = do_something();
    io_out32(HW_ACCEL_DATA_REG, data);
    io_out32(HW_ACCEL_CMD_REG, CMD_GO);
    while (io_in32(HW_ACCEL_STATUS_REG) != DONE)
        sleep();
    result = io_in32(HW_ACCEL_RESULT_REG);
    ...
}
```
Approach: Multithreaded Programming

- unified programming model for hard- and software
  - designer partitions application into threads (hard- and software)
  - threads communicate using abstractions (e.g. message boxes) provided by the operating system
  - simpler to program (hardware interface is hidden)
  - portable and scalable
  - enables partial reconfiguration

```c
int main() {
  ...
  data = do_something();
  msg_send(DATA_MBOX, data);
  result = msg_recv(RESULT_MBOX);  // blocking
  ...
}
```
Outline

- motivation & approach

- ReconOS
  - programming model
  - execution model
  - performance & overheads

- ongoing development & further research

- cooperations
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

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

```vhdl
object_file: process (clk, reset)
begin
  if (reset = '1') then
    state <= IDLE;
    run <= '0';
  elsif rising_edge (clk) then
    reconos_reset (o_osif, i_osif);
    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 if;
end process;
```
### Supported OS Calls

- **Semaphores (counting and binary)**
  - `reconos_semaphore_post()`
  - `reconos_semaphore_wait()`
  - **basic synchronization primitives**

- **Mutexes**
  - `reconos_mutex_lock()`
  - `reconos_mutex_trylock()`
  - `reconos_mutex_unlock()`
  - `reconos_mutex_release()`
  - **synchronize access to mutual exclusive operations (critical sections)**

- **Condition Variables**
  - `reconos_cond_wait()`
  - `reconos_cond_signal()`
  - `reconos_cond_broadcast()`
  - **allow waiting until arbitrary conditions are satisfied**

- **Mailboxes**
  - `reconos_mbox_get()`
  - `reconos_mbox_tryget()`
  - `reconos_mbox_put()`
  - `reconos_mbox_tryput()`
  - **message passing primitives (blocking and not blocking)**

- **Memory access**
  - `reconos_read()`
  - `reconos_write()`
  - `reconos_read_burst()`
  - `reconos_write_burst()`
  - **CPU-independent access to the entire system address space (memory and peripherals)**
System Architecture

- Development platforms
  - Xilinx ML403 (XC4VFX12)
  - Avnet Virtex-4 PCIe Kit (XC4VFX100)
  - Xilinx XUPV2P (XC2VP30)
  - Erlangen Slot Machine (XC2V6000)

- Operating system
  - eCos for PowerPC ported to development platforms
  - eCos is a widely-used open source RTOS
  - supplemented with OS interface for hardware threads
  - also ported to Linux 2.6 (µBlaze + PPC)
OS Integration

- 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
  - a hardware scheduler software thread manages the reconfigurable fabric and reconfigures hardware threads on demand

- advantages
  - no modification of the operating system kernel required
  - extremely flexible and portable (most functionality in user space)
  - transparent to kernel and other threads

- drawbacks
  - increased overhead due to interrupt processing, context switches, and user-mode / kernel mode switches
- hardware thread initiates request; OS interface raises interrupt
- delegate is synchronized to interrupts through semaphores (eCos) or blocking filesystem accesses (Linux)
- delegate thread is woken up and retrieves OS call and parameters
- in eCos, all code executes in kernel mode; simple hardware access possible
- no direct hardware access possible from Linux user space; needs driver
Partial Reconfiguration

- HW threads can be partially reconfigured
  - partial bitstreams are pre-generated at design time and placed in DRAM

- cooperative scheduling
  - hardware threads can cooperatively share computing resources by "yielding" their slots to other waiting threads
  - scheduling decisions are handled globally by hardware scheduler
Partial Reconfiguration

- HW threads can be partially reconfigured
  - partial bitstreams are pre-generated at design time and placed in DRAM

- cooperative scheduling
  - hardware threads can cooperatively share computing resources by "yielding" their slots to other waiting threads
  - scheduling decisions are handled globally by hardware scheduler
communication primitives
- shared memory and software mailboxes
- dedicated FIFOs for HW thread to HW thread communications
- message queues with direct access to HW thread memory

Communication Throughput

<table>
<thead>
<tr>
<th>Operation</th>
<th>$[\mu s]$</th>
<th>[MB/s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEM→HW (burst read)</td>
<td>45.74</td>
<td>170.80</td>
</tr>
<tr>
<td>HW→MEM (burst write)</td>
<td>40.54</td>
<td>192.71</td>
</tr>
<tr>
<td>MEM→SW→MEM (memcpy)</td>
<td>132.51</td>
<td>58.96</td>
</tr>
<tr>
<td>HW→HW (mailbox)</td>
<td>61.42</td>
<td>127.20</td>
</tr>
<tr>
<td>SW→HW (mailbox)</td>
<td>58,500</td>
<td>0.13</td>
</tr>
<tr>
<td>HW→SW (mailbox)</td>
<td>58,510</td>
<td>0.13</td>
</tr>
<tr>
<td>SW→HW (message queue)</td>
<td>472.00</td>
<td>16.55</td>
</tr>
<tr>
<td>HW→SW (message queue)</td>
<td>482.31</td>
<td>16.20</td>
</tr>
</tbody>
</table>
Overheads

- **synchronization overheads**
  - processing time (post → wait, unlock → lock)
  - time for non-blocking OS calls (i.e. `reconos_sem_post()``)

- **OS calls involving hardware exhibit higher latencies**
  - limited impact on system performance
    - logic resources mainly used for heavy data-parallel processing
    - less synchronization-intensive control dominated code

- **scheduling overheads**
  - `thread_create()`
  - `yield() / resume`
  - state save/restore
  - reconfiguration

<table>
<thead>
<tr>
<th>OS operation</th>
<th>eCos/PPC</th>
<th>Linux/PPC</th>
<th>Linux/MicroBlaze</th>
</tr>
</thead>
<tbody>
<tr>
<td>thread initialization</td>
<td>1.76 ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thread suspend</td>
<td>93.12 µs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>thread resume</td>
<td>192.32 µs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>state save (4096 bytes)</td>
<td>37.51 µs (104.1 MB/s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>state restore (4096 bytes)</td>
<td>45.19 µs (86.4 MB/s)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>reconfiguration time (233 kBytes)</td>
<td></td>
<td></td>
<td>99.96 ms ⚫</td>
</tr>
</tbody>
</table>

1 Xilinx opb_hwicap

All values given in bus cycles (1 cycle = 10 ns).
Outline

- motivation & approach

- ReconOS
  - programming model
  - execution model
  - performance & overheads

- ongoing development & further research

- cooperations
Ongoing ReconOS Development

- asymmetric multi-processor support
  - connect additional processors (PPC/µBlaze) via OSIF
  - support OS calls while avoiding cache coherency issues

- virtual memory for hardware threads
  - add memory management unit to OSIF
  - provide shared memory support for ReconOS/Linux

- token-ring network
  - reduce bus-based thread-to-thread communication
  - add additional network interface for OSIF
  - integration in ReconOS hardware API
Further Research based on ReconOS

- adaptive HW/SW systems
  - system reacts to changing performance requirements and workloads through partial reconfiguration
  - based on ReconOS programming and execution model
  - example: framework for sequential Monte Carlo methods

- runtime verification for reconfigurable computers
  - verify properties of dynamically loaded models, e.g. occupied area or combinational equivalence
  - ReconOS as infrastructure for „proof-carrying hardware“
Cooperations

- **SPP 1148**
  - University of Erlangen-Nuremberg, TU Dresden, TU Munich
  - ReconOS/ESM demonstrator at FPL 2008
  - ReconOS port to ESM
  - high-level synthesis for reconfigurable hardware threads
  - fast partial reconfiguration

- **ReconOS/hthreads**
  - Kansas University and University of Arkansas (USA)
  - mutual research visits
  - comparison between hthreads and ReconOS
  - programming model support for different communication channels (FIFOs, ring networks)
  - application of programming model to multicores

E. Lübbers & M. Platzner, University of Paderborn
Cooperations

- adaptive networking
  - based on the Autonomic Networking Architecture project (ANA, EU IST FP6)
  - dynamic adaptation and reorganization of network nodes and entire networks
  - cooperation with ETH Zurich, Computer Engineering and Networks Lab (TIK)
Publications


Thank you

www.reconos.de