Developing hard real-time systems using FPGAs and soft CPU cores
Outline

- Hard Real Time control systems: background
- The Mock Turtle Core
- The μRV – a soft processor core designed at CERN
- Application examples
- Summary and outlook
Hard Real Time Controls

What do we mean by hard RT?

**Input:** receive an event or trigger, take a measurement of a control input

**Processing:**
- decode a timing event
- calculate action execution time
- update a feedback loop

**Execute** the result of the Processing step: produce a pulse, update a DAC, etc.

-worst case latency

-idle time

-hard deadline

-time
## Hard RT: technologies

### FPGAs
- ✔ Very low and deterministic latency
- ✔ Easy to customize
- ✗ Long development cycle
- ✗ Difficult to program by the end user

### Embedded processors
- ✔ Deterministic latency
- ✔ Easy to program
- ✗ Difficult to customize (often need a companion FPGA)
- ✗ Portability issues

### PLCs
- ✔ Standardized environment
- ✔ Easy to program by the end users
- ✗ Too slow for many accelerator applications

### Linux PCs
- ✔ Very short development cycle
- ✔ Easy to program by the end users
- ✗ Can't guarantee determinism (at least on x86 platform)
Example: CERN timing receiver

- Custom VHDL design, done from scratch
- Only a small part (counters) really needs dedicated VHDL
- Limited flexibility
Example: LHC Instability Trigger
a.k.a. The LIST

- Instruments in Point 4 detect an onset of beam instability
- Generate a trigger
- Distribute the trigger to other instruments and acquire a massive amount of data for offline study.
LIST: The Challenge

Distributed hard real time system

- Exchange triggers between any pair of devices in the network
- Never exceed the design latency (max 270 µs)
- Never miss a trigger
- Deliver a feature-rich system in a reasonable time...
Hard RT: technologies

- **FPGAs**
  - ✔ Very low and deterministic latency
  - ✔ Easy to customize
  - ✗ Long development cycle
  - ✗ Difficult to program by the end user

- **Embedded processors**
  - ✔ Deterministic latency
  - ✔ Easy to program
  - ✗ Difficult to customize (often need a companion FPGA)
  - ✗ Portability issues

- **PLCs**
  - ✔ Standardized environment
  - ✔ Easy to program by the end users
  - ✗ Too slow for many accelerator applications

- **Linux PCs**
  - ✔ Very short development cycle
  - ✔ Easy to program by the end users
  - ✗ Can't guarantee determinism (at least on x86 platform)
Outline

- Hard Real Time control systems: background
- The Mock Turtle Core
- The μRV – a soft processor core designed at CERN
- Application examples
- Summary and outlook
Mock Turtle: the idea

- Take an FPGA chip
- Take a CPU core that is deterministic
- Put as many CPUs as needed
- Let them communicate with each other...
- ... and with the external world
- Connect user cores and hardware
Mock Turtle: the idea

- Take an FPGA chip
- Take a CPU core that is deterministic
- Put as many CPUs as needed
- Let them communicate with each other...
- ... and with the external world
- Connect user cores and hardware
Mock Turtle: the idea

- Take an FPGA chip
- Take a CPU core that is deterministic
- Put as many CPUs as needed
- Let them communicate with each other...
- ... and with the external world
- Connect user cores and hardware
Mock Turtle: the idea

- Take an FPGA chip
- Take a CPU core that is deterministic
- Put as many CPUs as needed
- Let them communicate with each other...
- ... and with the external world
- Connect user cores and hardware
Mock Turtle: the idea

- Take an FPGA chip
- Take a CPU core that is deterministic
- Put as many CPUs as needed
- Let them communicate with each other...
- ... and with the external world
- Connect user cores and hardware
Mock Turtle: the idea

- Take an FPGA chip
- Take a CPU core that is deterministic
- Put as many CPUs as needed
- Let them communicate with each other...
- ... and with the external world
- Connect user cores and hardware
Mock Turtle: the idea

Make it a service!

- Standard.
- Portable.
- Reusable.

With or without White Rabbit as the means of communication and synchronization.
Inside Mock Turtle

The CPU Cores

- Up to 8 LM32 cores, with private code/data memory
- Programed in bare metal C, using standard GCC tool chain
- Each core runs a single task in a tight loop.
- No caches, no interrupts
- Program loading and flow control from the host system
- Run with same time base if used with WR
Inside Mock Turtle

Shared Memory

- Foreseen for inter-core communication and task synchronization
- Small, but atomically accessed
  - Add/subtract
  - Bit operations
  - Test-and-set
- Task synchronization primitives
  - Mutexes
  - Semaphores
  - Queues
  - Flags
  - Events...
Inside Mock Turtle Communication System

- Simple FIFO queues holding multiple messages
  - Polling in a tight loop on the RT side
  - Interrupt-driven on the host side
- Each queue provides a configurable number of channels (slots)
- Host Message Queue for communication with the host software
- Optional Remote Message Queue for communication with other nodes over WR network
  - Uses Etherbone protocol
Inside Mock Turtle
Software Architecture

- Unified kernel driver + user space C library
- Host code written in user space → easier development
- Message Queue and Shared Memory access
- Loading CPU applications and controlling each core's execution flow
- Real-time library for the CPU cores.
- Python bindings available.
Based on the Wishbone bus.
Each CPU can have its exclusive Dedicated Peripheral.
All CPUs can access Shared Peripheral(s).
Etherbone master/slave port for communication with other nodes in a WR network (optional).
Time interface from the WR PTP Core.
Control port for the host system.
Outline

• Hard Real Time control systems: background
• The Mock Turtle Core
• The $\mu$RV – a soft processor core designed at CERN
• Application examples
• Summary and outlook
What is $\mu$RV?

- **Micro RISC-V:** a compact soft processor (CPU) core for use in FPGAs.
  - $\rightarrow$ Written in Verilog, no external dependencies.
- **Reduced Instruction Set (RISC) processor.**
  - $\rightarrow$ Small number of instructions, each doing one function at a time.
- **Based on the RISC-V architecture developed at the University of Berkeley.**
  - $\rightarrow$ Well thought and promising instruction set.
- **Small, portable and deterministic.**
  - $\rightarrow$ Determinism of execution over performance.
Why \( \muRV \)?

- A number of our projects depend on soft CPUs
- Popular architectures have either license limitations (LM32) or are patented (ARM, MIPS)
- FPGA vendor-provided cores (Microblaze, Nios) are not portable (no sources available either)
- RISC-V: a very nice architecture, but with no small Verilog/VHDL implementation
The $\mu$RV design

- Simple, 4-stage pipeline
- All instructions except jumps (branches) in one clock cycle
- Code/data in a private memory
- Wishbone bus for peripheral access
- 100 MHz and 20% of a small FPGA (Spartan-6 SLX9)
Outline

- Hard Real Time control systems: background
- The Mock Turtle Core
- The $\mu$RV – a soft processor core designed at CERN
- Application examples
- Summary and outlook
Trigger Distribution Node

- **Dual-core system:**
  - 32 kB private RAM per CPU

- **CPU 1 responsible for the inputs:**
  - Poll TDC for incoming pulses
  - Apply dead time
  - Assign trigger ID
  - Send to the WR network
  - Log the trigger

- **CPU 0 takes care of the outputs:**
  - Poll Remote MQ for trigger messages
  - Search a hash table for matching triggers
  - Apply delay & dead time
  - Program the pulse generator
  - Log the trigger
Trigger Distribution Node

Example of a real time task

```c
main()
{
    init();

    // our tight real-time loop
    for(;;)
    {
        do_input(); // handle incoming TDC pulses & send triggers
        do_control(); // handle commands coming from the host
        wr_update_link(); // update the status of the WR link
    }

    return 0;
}
```
Outline

• Hard Real Time control systems: background
• The Mock Turtle Core
• The μRV – a soft processor core designed at CERN
• Application examples
• Summary and outlook
Summary & outlook

- Proven real-time performance
  - Trigger distribution system working in the LHC
  - Other projects (FIP master & RF distribution) ongoing

- Savings in development time
  - Minimal amount of dedicated VHDL
  - Standardized software stack and interfaces

- A service that anyone can use

- The μRV soft processor
  - Passes CoreMark & official RISC-V test suite
  - Migration of MT Cores from LM32 in progress

Sources available at the Open Hardware Repository: ohwr.org
Questions?
MasterFIP project

WorldFIP master

- Prog Data Mem [cycle config]
- Host Message Queue Control Registers
- Host Access [PCIe]
- Shared Mem [real time data]
- CPU core 1 [host requests for real time data]
- CPU core 0 [WorldFIP cycle]
- Mock Turtle core
- WorldFIP core
- TX/RX core
- Serializer
- Deserializer

[2.5Mbps] [TXD] [RXD]

WorldFIP cycle

WorldFIP node

Question Frame [cycle config]
Response Frame [real time data]
Question Frame [cycle config]
Response Frame [real time data]
Question Frame [cycle config]
Response Frame [real time data]
MasterFIP project

- **Clean implementation of WorldFIP bus master**
  - Critical technology for LHC controls in radiation areas

- **Two-CPU design**
  - CPU0 plays the current FIP macrocycle
  - CPU1 exchanges the cycle configuration and variables with the host
  - Dedicated VHDL only needed to serialize/deserialize FIP frames

- **Shared Memory holds the FIP variables and macrocycle setup**

- **No White Rabbit**