CASE STUDY 4: CUSTOM DESIGNED MULTI-PROCESSOR SYSTEM
Vision

General Purpose Shared Memory Computer

- core
- cache
- bus
- memory

Application Specific Multicore Network On Chip

- core
- engine

355
Motivation: Multicore Systems Challenges

- Cache Coherence
- Shared Memory Communication Bottleneck
- Thread Synchronization Overhead

→ Hard to predict performance of a program
→ Difficult to scale the design to massive multi-core architecture
Operating System Challenges

- Processor Time Sharing
  - Interrupts
  - Context Switches
  - Thread Synchronisation

- Memory Sharing
  - Inter-process: Paging
  - Intra-process, Inter-Thread: Monitors
Focus

Academia: Education

- **holistic** design of computing systems
- simplicity
- consistency

Industry: High Performance Sensor Driven Medical IT

- **streaming** applications: ultrasound, tomography, hemodynamics, etc.
Focus: Streaming Applications

Stream-Parallelism: Pipelining

Task Parallelism: Parallel Execution

Data Parallelism: Vector Computing
Loop-level parallelism
4.1. HARDWARE BUILDING BLOCKS
TRM AND INTERCONNECTS
TRM: Tiny Register Machine*

- Extremely simple processor on FPGA with Harvard architecture.
- Two-stage pipelined
- Each TRM contains
  - Arithmetic-logic unit (ALU) and a shifter.
  - 32-bit operands and results stored in a bank of 2*8 registers.
  - Local data memory: d*512 words of 32 bits.
  - Local program memory: i*1024 instructions with 18 bits.
  - 7 general purpose registers
  - Register H for storing the high 32 bits of a product, and 4 conditional registers C, N, V, Z.
- No caches

* Invented and implemented by Dr. Ling Liu and Prof. Niklaus Wirth
TRM Machine Language

- Machine language: binary representation of instructions
- 18-bit instructions
- Three instruction types:
  - Type a: arithmetical and logical operations
  - Type b: load and store instructions
  - Type c: branch instructions (for jumping)
Encoding Overview

- **Register Operations**
  - (a) \[ \begin{array}{ccc} \text{op} & \text{Rd} & 0 \\ \hline 17 & 14 & 13 \\ \end{array} \] \[ \begin{array}{c} \text{imm} \\ \hline 9 & 0 \\ \end{array} \]
  - (b) \[ \begin{array}{ccc} \text{op} & \text{Rd} & 0 \\ \hline 17 & 14 & 13 \\ \end{array} \] \[ \begin{array}{c} \text{000} & \text{x} & \text{x} \\ \hline 0 & 0 & 0 \\ \end{array} \]
  - (c) \[ \begin{array}{ccc} \text{op} & \text{VRd} & 0 \\ \hline 17 & 14 & 13 \\ \end{array} \] \[ \begin{array}{c} \text{100} & \text{x} & \text{x} \\ \hline 9 & 0 & 0 \\ \end{array} \]

- **Load and Store**
  - (a) \[ \begin{array}{ccc} \text{op} & \text{Rd} & 0 \\ \hline 17 & 14 & 13 \\ \end{array} \] \[ \begin{array}{c} \text{off} & \text{Rs} \\ \hline 9 & 0 \\ \end{array} \]
  - (b) \[ \begin{array}{ccc} \text{op} & \text{VRd} & 0 \\ \hline 17 & 14 & 13 \\ \end{array} \] \[ \begin{array}{c} \text{off} & \text{Rs} \\ \hline 9 & 0 \\ \end{array} \]

- **Conditional Branches**
  - \[ \begin{array}{ccc} \text{1110} & \text{cond} & \text{off} \\ \hline 17 & 14 & 13 \\ \end{array} \]

- **Special Instructions**
  - (a) \[ \begin{array}{ccc} \text{op} & \text{Rd} & 1 \\ \hline 17 & 14 & 13 \\ \end{array} \] \[ \begin{array}{c} \text{000} & \text{0} & \text{x} \\ \hline 9 & 0 & 0 \\ \end{array} \]
  - (b) \[ \begin{array}{ccc} \text{op} & \text{VRd} & 1 \\ \hline 17 & 14 & 13 \\ \end{array} \] \[ \begin{array}{c} \text{000} & \text{x} & \text{x} \\ \hline 9 & 0 & 0 \\ \end{array} \]
  - (c) \[ \begin{array}{ccc} \text{op} & \text{Rd} & 1 \\ \hline 17 & 14 & 13 \\ \end{array} \] \[ \begin{array}{c} \text{x} & \text{x} & \text{x} \\ \hline 9 & 0 & 0 \\ \end{array} \]
  - (d) \[ \begin{array}{ccc} \text{op} & \text{Rd} & 1 \\ \hline 17 & 14 & 13 \\ \end{array} \] \[ \begin{array}{c} \text{x} & \text{x} & \text{x} \\ \hline 9 & 0 & 0 \\ \end{array} \]
  - (e) \[ \begin{array}{ccc} \text{op} & \text{Rd} & 1 \\ \hline 17 & 14 & 13 \\ \end{array} \] \[ \begin{array}{c} \text{101} & \text{xxxx} & \text{Rs} \\ \hline 6 & 0 & 0 \\ \end{array} \]

- **Branch and Link**
  - \[ \begin{array}{ccc} \text{1111} & \text{off} \\ \hline 17 & 14 & 13 \\ \end{array} \]

Notes:
- imm is zero extended to 32 bits
- off is zero extended to 13 bits
- off is sign extended to 12 bits
- off is 14-bit offset
TRM architecture

Figure from: Niklaus Wirth, *Experiments in Computer System Design*, Technical Report, August 2010
http://www.inf.ethz.ch/personal/wirth/Articles/FPGA-relatedWork/ComputerSystemDesign.pdf
Variants of TRM

- **FTRM**
  - includes floating point unit

- **VTRM** (Master Thesis Dan Tecu)
  - includes a vector processing unit
  - supports 8 x 8-word registers
  - available with / without FP unit

- **TRM with software-configurable instruction width**
  (Master Thesis Stefan Koster, 2015)
1. Bus Interconnect TRM12

- 12 RISC Cores (two stage pipelined at 116MHz)
- Message passing architecture
- Bus based on-chip interconnect
- On-chip Memory controller

- Not scalable
- Huge resource consumption

Ling Liu, A 12-Core-Processor Implementation on FPGA, ETH Technical Report, October 2009
2. Ring Interconnect

RS232

Slide from Dr. Ling Liu's Lecture Series on Reconfigurable Computing
Connection TRM / Ring

TRM

Adapter

Ring

- Ring interconnect very simple
- Small router
- Predictable latency

- Large delays
- Limited Scalability

Slide from Dr. Ling Liu's Lecture Series on Reconfigurable Computing
4.2. HARDWARE SOFTWARE CODESIGN
Software / Hardware Co-design

Vision: Custom System on Button Push

System design as high-level program code

Electronic circuits

- Computing model Programming Language
- Compiler, Synthesizer, Hardware Library, Simulator
- Programmable Hardware (FPGA)
Traditional HW/SW co-design

System specification, HW/SW partitioning

Program microcontroller in C/C++

Compilation

System on FPGA

Program system specific hardware in HDL

Synthesis

One Program

One Toolchain

Microcontroller + machine code + specific hardware (eg. DSP)

Active Cells approach for embedded systems development

Traditional HW/SW co-design for embedded systems
Active Cells Computing Model

On-chip distributed system

Cell

- Scope and environment for a running isolated process.
- Integrated control thread(s)
- Provides communication ports

Net

- **Network** of communication cells
- Cells connected via channels (FIFOs)

Inspired by
- Kahn Process Networks
- Dataflow Programming
- CSP (i.e. Google’s Go)
- Actor Model (i.e. Erlang)
Software → Hardware Map
Consequences of the approach

- No global memory
- No processor sharing
- No peculiarities of specific processor
- No predefined topology (NoC)
- No interrupts

→ No operating system
type
BernoulliSampler* = cell (probIn: port in; valOut: port out);
var
  r: Random.Generator;
  p: real;
begin
  new(r);
  loop
    p << probIn;
    valOut << r.Bernoulli(p);
  end
end BernoulliSampler;
Properties

type
Controller = cell {Processor=TRM, FPU, DataMemory=2048, BitWidth=18}
 (in: port in (64); result: port out);
...

begin
 (* ... controller action ... *)
 end Controller;
....

Properties can influence both, generation of hardware and the generation of software code.
Configurable Processor on PL

Tiny Register Machine

- **FPU**: on/off
- **Vector Unit**: on/off
- **Multiplier**: on/off
- **Instruction Width**: 14, 16, 18, 20, 22
- **Data Memory**: 0.5k, 1k, 1.5k
- **Program Memory**: 0k, 1k, 2k, 3k, 4k
type
Merger = cell {Engine, inputs=1}
    (ind: array inputs of port in; outd: port out);
var data: longint;
begin
    loop
        for i := 0 to len(in)-1 do
            data << ind[i]
            outd << data;
    end
end Merger;

Engines are prebuilt components instantiated as electronic circuits on a target hardware.
LearnTest = cellnet;
var
    learner: CRBMNet.CRBMLearner;
    reader: MLUtil.imageReader;
    ims0,ims1: MLUtil.imshow;
...
begin
    new(learner)
    new(reader);
    new(ims0{name='v0debug',posx=0,posy=100});
    new(ims1{name='v1debug',posx=300,posy=100});
    ...
    reader.imageOUT >> learner.imgIN;
    learner.v0DebugOUT >> ims0.imageIN;
    learner.v1DebugOUT >> ims1.imageIN;
    ...
end LearnTest;
Hierarchic Composition: non-terminal Cellnet

UpNet = cellnet
{vr=28,vc=28,kr=5,kc=5,k=9,c=2,name='upstep'}
(pvIN, kerIN, bIN : port in; phOUT: array k of port out; pvSideOUT: port out);

var
i,hr,hc: longint;
upstep: CRBMUpstepCell;
split: MLFunctions.SplitterCell;
begin
hr:=vr-kr+1; hc:=vc-kc+1;
new(vSplit {dataSize=vr*vc,numOut=2});
new(upstep {vr=vr,vc=vc,kr=kr,kc=kc,k=k,c=c});
pvIN >> vSplit.dataIN;
vSplit.dataOUT[0] >> pvSideOUT;
vSplit.dataOUT[1] >> UpstepCell.vIN;
kernIN >> UpstepCell.kerIN;
biasIN >> UpstepCell.bIN;
for i:=0 to k-1 do
  upstep.phOUT[i] >> phOUT[i];
end;
end CRBMUpNet;
Software ➔ Hardware Map

- ARM
- TRM
- ENGINE
- CELL
- port

ZYNQ PL
- AXI4 Stream Interconnect
- AXI4
- PS
- thread
- FIFO
- softcore
- hw engine

ARM ENGINE

engine
Hybrid Compilation

cellnet N;

type A = cell(pi: port in; po: port out);
var x: integer;
begin
... pi ? x; ... po ! x; ...
end A;

var a,b: A;
begin
... connect(a.po, b.pi)
end N.

<table>
<thead>
<tr>
<th>Code body</th>
<th>Role</th>
<th>Compilation method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cell (Softcore)</td>
<td>Program logic</td>
<td>Software Compilation</td>
</tr>
<tr>
<td>Cell (Engine)</td>
<td>Computation unit</td>
<td>Hardware Generation</td>
</tr>
<tr>
<td>Cell Net</td>
<td>Architecture</td>
<td>Hardware Compilation</td>
</tr>
</tbody>
</table>
Implementation Alternatives (1)

- simple
- redundant
- not flexible
- hard to extend
Implementation Alternatives (2)

- + extensible
- + not redundant
- - not simple
- - static configuration limits flexibility
Implementation Alternatives (3)

+ extensible
+ not redundant
+ simpler
+ simulation becomes execution
Extensibility: Defining components and platforms

**Software HLL Code**
```
type Gpo = cell{Engine,DataWidth=32,InitState="0H"} (input: port in);
```

**Build Command**
```
AcHdlBackend.Build
   --target="ZyboBoard"
   -p="Vivado"
   CRBM.TestCellnet ~
```

**Component Specification**

**Hardware HDL Code**
```
module Gpo
#( parameter integer DW = 8 ... )
( input aclk, input aresetn , input [DW-1:0] ... );
```

**Platform Specification**

**Hardware Platform and Tools**
- Hardware Types
- Platform Instances
- Vendor specific tools
module Gpo;
import Hdl := AcHdlBackend;
var c: Hdl.Engine;
begin
new(c,"Gpo","Gpo");
c.SetDescription("General Purpose Output … ");
c.SupportedOn("*"); (* portable *)
c.NewProperty("DataWidth","DW",Hdl.NewInteger(32),Hdl.IntegerPropertyRangeCheck(1,Hdl.MaxInteger));
c.NewProperty("InitState","InitState",Hdl.NewBinaryValue("0H"),nil);
c.SetMainClockInput("aclk"); (* main component's clock *)
c.SetMainResetInput("aresetn",Hdl.ActiveLow); (* active-low reset *)
c.NewAxisPort("input","inp",Hdl.In,8);
c.NewExternalHdlPort("gpo","gpo",Hdl.Out,8);
c.NewDependency("Gpo.v",true,false);
c.AddPostParamSetter(Hdl.SetPortWidthFromProperty("inp","DW"));
c.AddPostParamSetter(Hdl.SetPortWidthFromProperty("gpo","DW"));
Hdl.hwLibrary.AddComponent(c);
c.NewExternalHdlPort("gpo","gpo",Hdl.Out,8);
end Gpo.
Use of AXI4 Stream interconnect standard from ARM

- Generic, flexible
- Non-redundant

TVALID: Data valid

Select

Data out

TREADY: Ready to process

Select

Data in

Master must assert TVALID and keep asserted

Slave can wait for master's TVALID and then assert TREADY
module Basys2Board;
import Hdl := AcHdlBackend, AcXilinx;
var t: Hdl.TargetDevice;
  pldPart: AcXilinx.PldPart;
  ioSetup: Hdl.IoSetup;
  pin: Hdl.IoPin;
begin
new(pldPart,"XC3S100E-4CP132");
pldPart.SetJtagChainIndex(0);
new(t,"Basys2Board",pldPart);

new(pin,Hdl.In,"B8","LVCMOS33");
t.NewExternalClock(pin,50000000,50,0); (* ExternalClock0 *)
t.SetSystemClock(t.clocks.GetClockByName("ExternalClock0"),1,1);
new(pin,Hdl.In,"G12","LVCMOS33");
t.SetSystemReset(pin,true);

new(ioSetup,"Gpo_0");
t.AddIoSetup(ioSetup);
end Basys2Board.

Case Study 1: ECG
Focus: Resources and Power

Real-time ECG Monitor
Resources

- ECG Monitor*

<table>
<thead>
<tr>
<th>#TRMs</th>
<th>#LUTs</th>
<th>#BRAMs</th>
<th>#DSPs</th>
<th>TRM load</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>13859 (48%)</td>
<td>52 (86%)</td>
<td>12 (25%)</td>
<td>&lt;5% @116 MHz</td>
</tr>
</tbody>
</table>

- Maximum number of TRMs in communication chain

<table>
<thead>
<tr>
<th>FPGA</th>
<th>#TRMs</th>
<th>#LUTs</th>
<th>#BRAMs</th>
<th>#DSPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtex-5</td>
<td>30</td>
<td>27692 (96%)</td>
<td>60 (100%)</td>
<td>30 (62%)</td>
</tr>
<tr>
<td>Virtex 6</td>
<td>500</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*8 physical channels @ 500 Hz sampling frequency implemented on Virtex 5
Comparative Power Usage

- Preconfigured FPGA (#TRMs, IM/DM, I/O, Interconnect fixed) versus fully configurable FPGA (Active Cells)

<table>
<thead>
<tr>
<th>System</th>
<th>Static Power (W)</th>
<th>Dynamic Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preconfigured (&quot;TRM12&quot;)</td>
<td>3.44</td>
<td>0.59</td>
</tr>
<tr>
<td>Dynamically configured</td>
<td>0.5</td>
<td>0.58</td>
</tr>
</tbody>
</table>

86% saving!
Case Study: Non-Invasive Continuous Blood Pressure Monitor
Focus: Development Cycle Time

A2 Host OS with GUI on ARM

Sensor control and medical algorithms on Zynq PL

Sensors and Motors on Bracelet
Medical Monitor Network On Chip

Development Cycle Times

<table>
<thead>
<tr>
<th>Step</th>
<th>Medical Monitor</th>
<th>OCT (full)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Compilation</td>
<td>4s</td>
<td>2s</td>
</tr>
<tr>
<td>Hardware Implementation</td>
<td>20 min</td>
<td>20 min</td>
</tr>
<tr>
<td>Stream Patching (all)</td>
<td>2 min</td>
<td>-</td>
</tr>
<tr>
<td>Stream Patching (typical)</td>
<td>12 s</td>
<td>-</td>
</tr>
<tr>
<td>Deployment</td>
<td>11s</td>
<td>16s</td>
</tr>
</tbody>
</table>
Case Study 3: Optical Coherence Tomography
Focus: Performance

z-Axis Processing

1. Non uniform sampling
   \[ A(\lambda_i) \rightarrow \tilde{A}(f_i) \]
2. Dispersion compensation
3. (Inverse) FFT

… for many lines x in a row (2d)
… and many rows y in a column (3d)
A component of OCT image processing

Dispersion Compensation

Dominated by Engines. Dataflow driven.
Case Study: ANN

Programmable Logic

ARM SoC

ANN Layer 1

P0

P1

P2

R0

R1

R2

b

w

x

z

(More Layers)

Perceptron

\[
\begin{align*}
\text{InnerProdFlt} & : x_0 w_0 + x_1 w_1 + x_2 w_2 \\
\text{Sigmoid} & : \frac{1}{1 + e^{-x}}
\end{align*}
\]

Fix

Frac

LUT

\[ y = z \]

\[ b \]

Input 0

Input 1

Output

Fix

Frac

LUT

- 

\[ z = \text{InnerProdFlt} \]

\[ y = \text{Sigmoid}(z) \]
# Performance and Resource Usage

<table>
<thead>
<tr>
<th></th>
<th>Medical Monitor</th>
<th>Simple OCT</th>
<th>OCT</th>
<th>Perceptron</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architecture</strong></td>
<td>Spartan 6 XC6SLX75</td>
<td>Zynq 7000 XC7Z020</td>
<td>Zynq 7000 XC7Z020</td>
<td>Zynq 7000 XC7Z010</td>
</tr>
<tr>
<td><strong>Resources</strong></td>
<td>28% Slice LUTs, 4% Slice Registers, 80% BRAMs, 24% DSPs</td>
<td>11% Slice LUTs, 6% Slice Registers, 7% BRAMs, 15% DSPs, 1 ARM Cortex A9</td>
<td>17% Slice LUTs, 8% Slice Registers, 22% BRAMs, 31% DSPs, 1 ARM Cortex A9</td>
<td>83% Slice LUTs, 67% Slice Registers, 13.33% BRAMs, 21% DSP, 1 ARM Cortex A9</td>
</tr>
<tr>
<td><strong>Clock Rate</strong></td>
<td>58 MHz</td>
<td>118 MHz</td>
<td>50 MHz</td>
<td>147 MHz</td>
</tr>
<tr>
<td><strong>Data Bandwidth</strong></td>
<td>1.25 Mbit/s (in), 23 kBit/s (out)</td>
<td>236 MWords/s (in), 118 MWords/s (out)</td>
<td>50 MWords/s (in), 50 MWords/s (out)</td>
<td>9.6 GBits/s in, 9.6 GBits/s out</td>
</tr>
<tr>
<td><strong>Performance</strong></td>
<td>--</td>
<td>8.3 GFPOps* up to 32 GFPOps**</td>
<td>4.3 GFPOps*</td>
<td>4.9 GFlops</td>
</tr>
<tr>
<td><strong>Power</strong></td>
<td>~2W</td>
<td>~5W</td>
<td>~5W</td>
<td>2.1 W</td>
</tr>
</tbody>
</table>

** Fixed point operations, 32bit
* when instantiated 4 times
Conclusion

ActiveCells: Computing model and tool-chain for configurable computing

- Configurable interconnect → Simple Computing, Power Saving
- Embedding of task engines → High Performance
- Hybrid compilation → Quick Development
- Backend execution → Eased Flexibility and Extensibility
Acknowledgements

Results originate in project "Supercomputer in the Pocket" (J. Gutknecht, L. Liu, A. Morzov, P. Hunziker, A. Gokhberg, FF) in the Microsoft Innovation Cluster for Embedded Systems funded by Microsoft Research (2009-2014).

We are particularly indebted for consulting to Chuck Thacker, Niklaus Wirth, Timothée Martiel, Paul Reed and Florian Negele.

Thanks for Support from Xilinx Academic Program.

CRBM Implementation by Stephan Koster.