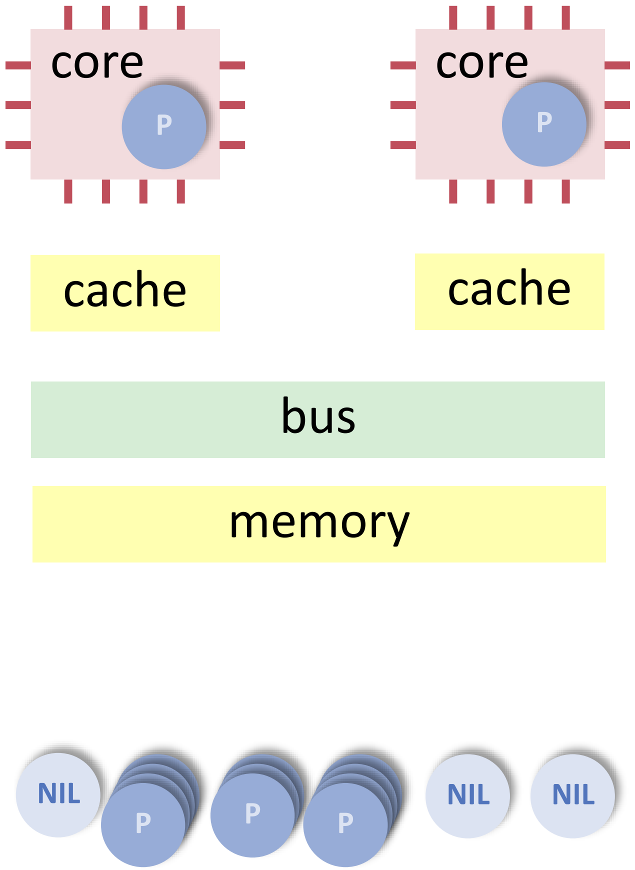
Active Cells:

A Programming Model for Configurable Multicore Systems

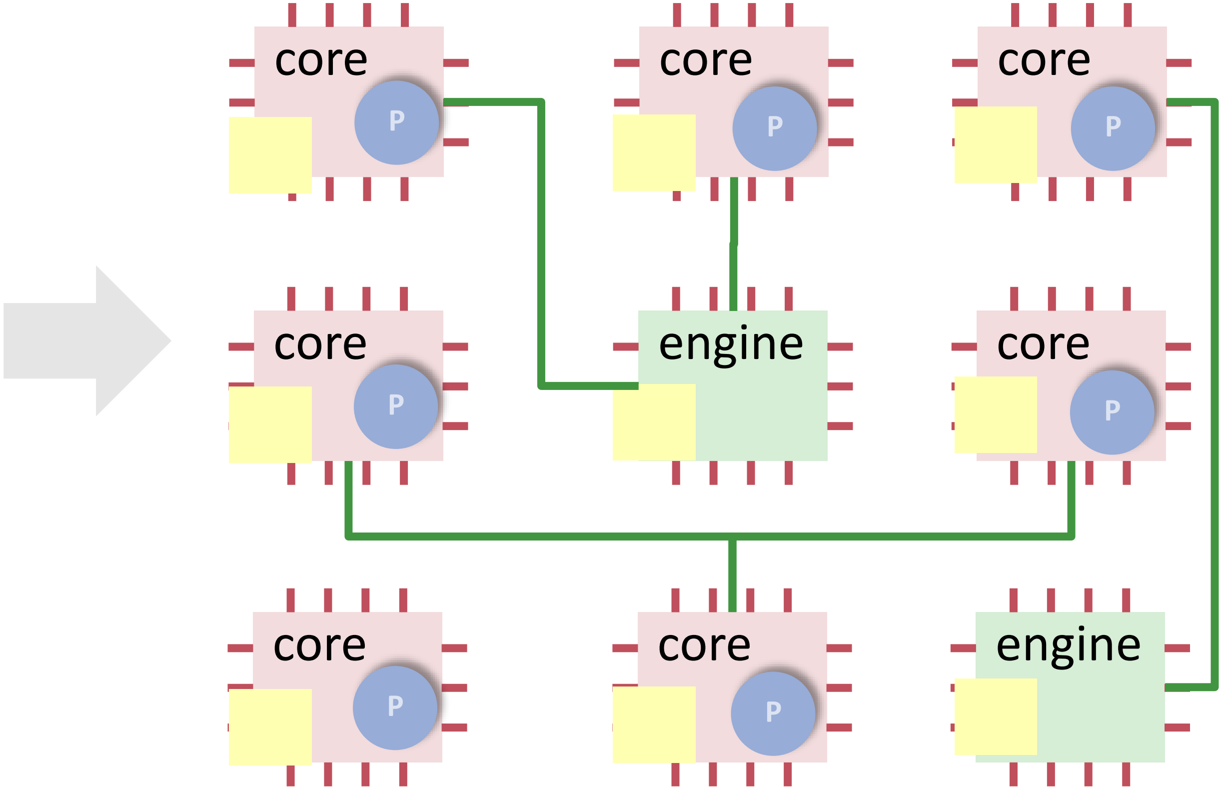
CASE STUDY 4: CUSTOM DESIGNED MULTI-PROCESSOR SYSTEM

Vision

General Purpose Shared Memory Computer



Application Specific Multicore Network On Chip



Objectives

- TRM Processor and Interconnects
- **Software Hardware Co-Design**
- The Active Cells Toolchain
- Case Studies and Examples

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

Project *Supercomputer in the Pocket*

Funded by Microsoft in ICES programme, 2009 - 2014

Manycore architecture for embedded systems on the basis of programmable hardware (FPGA)

- Emphasis on high-performance computing in the small in the field of sensor driven medical IT
- Enhance industrial applications and ease teaching of parallel computing

General purpose manycore for teaching

Processor Designs and Interconnects

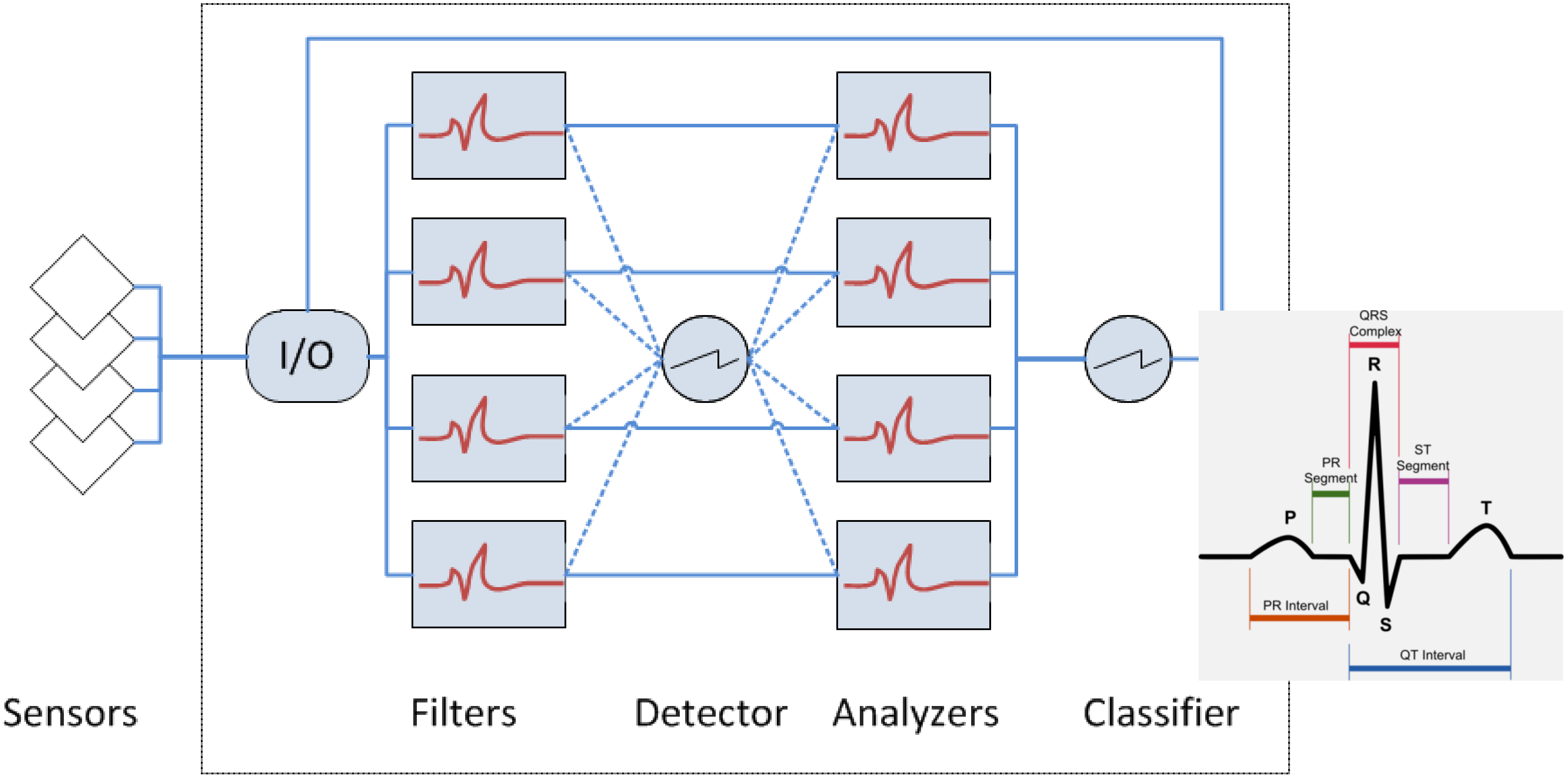
Idea
"Configurability over all levels"

Novel computing model and toolchain for constructing distributed system on chip.

Project Time

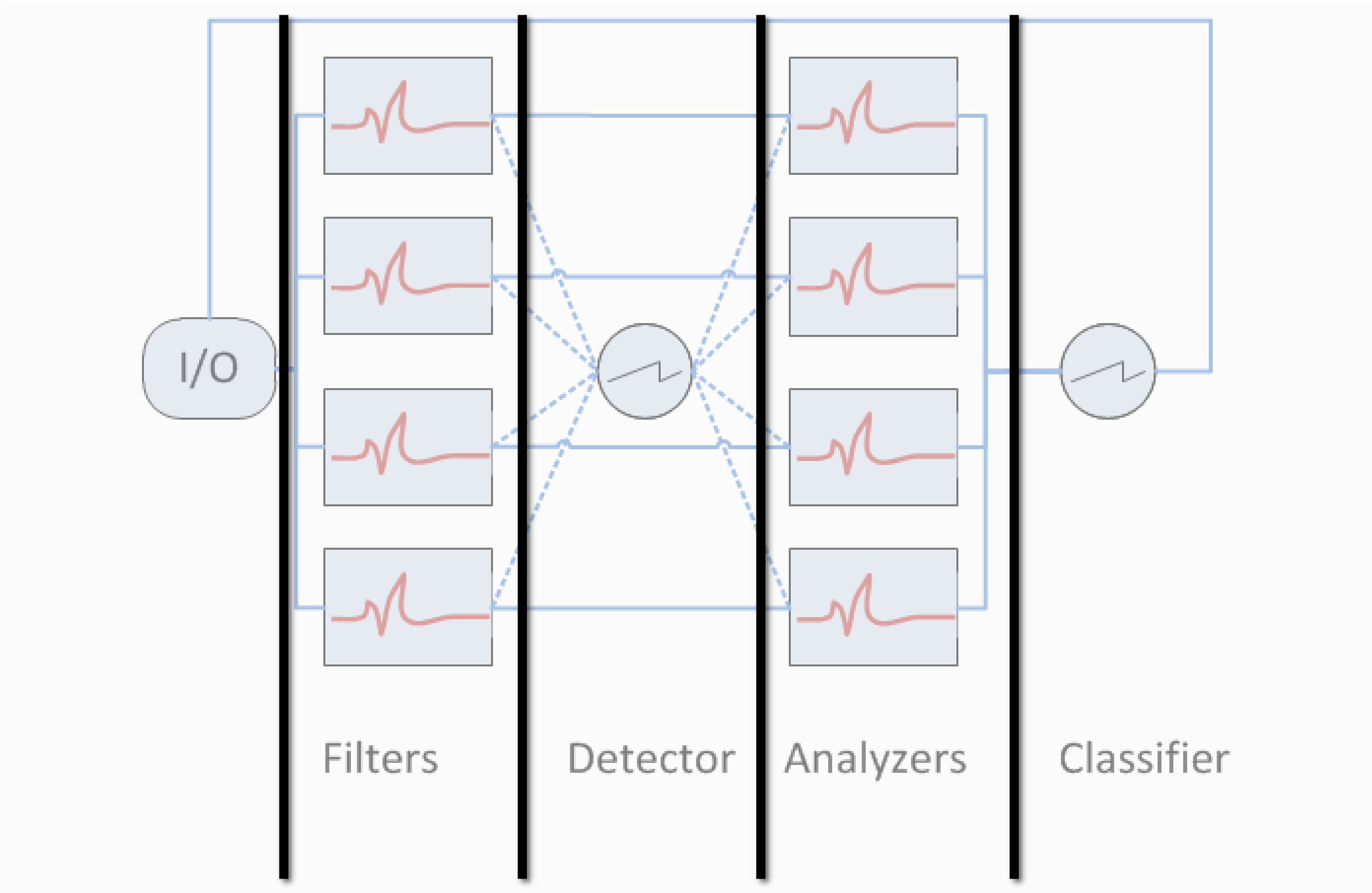
ICES II

Focus: Streaming Applications



Structural Example:
ECG for realtime desease detection

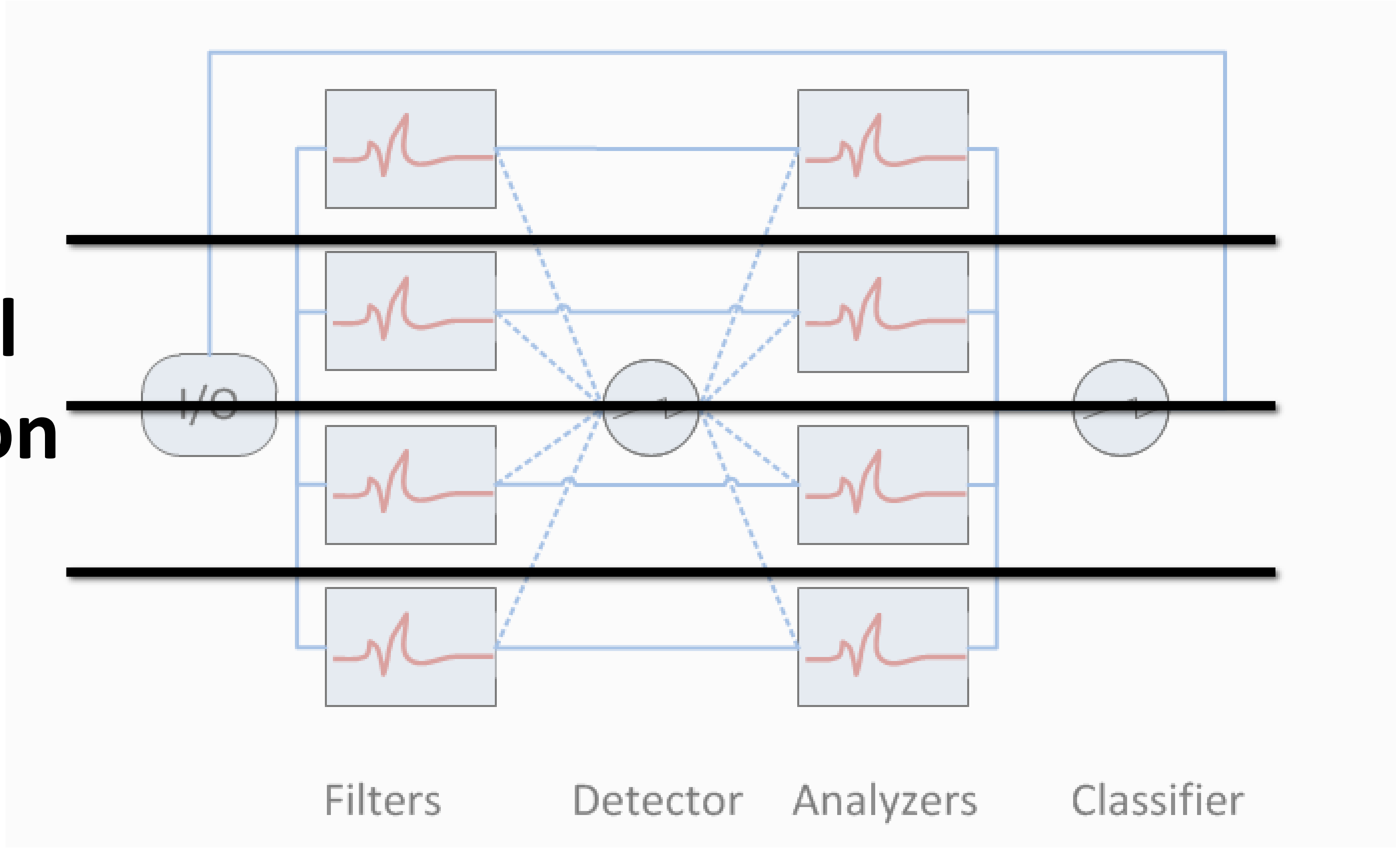
Stream Parallelism



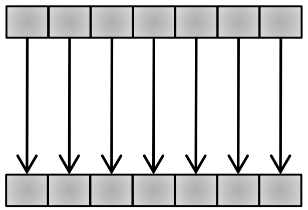
Pipelining

Task Parallelism

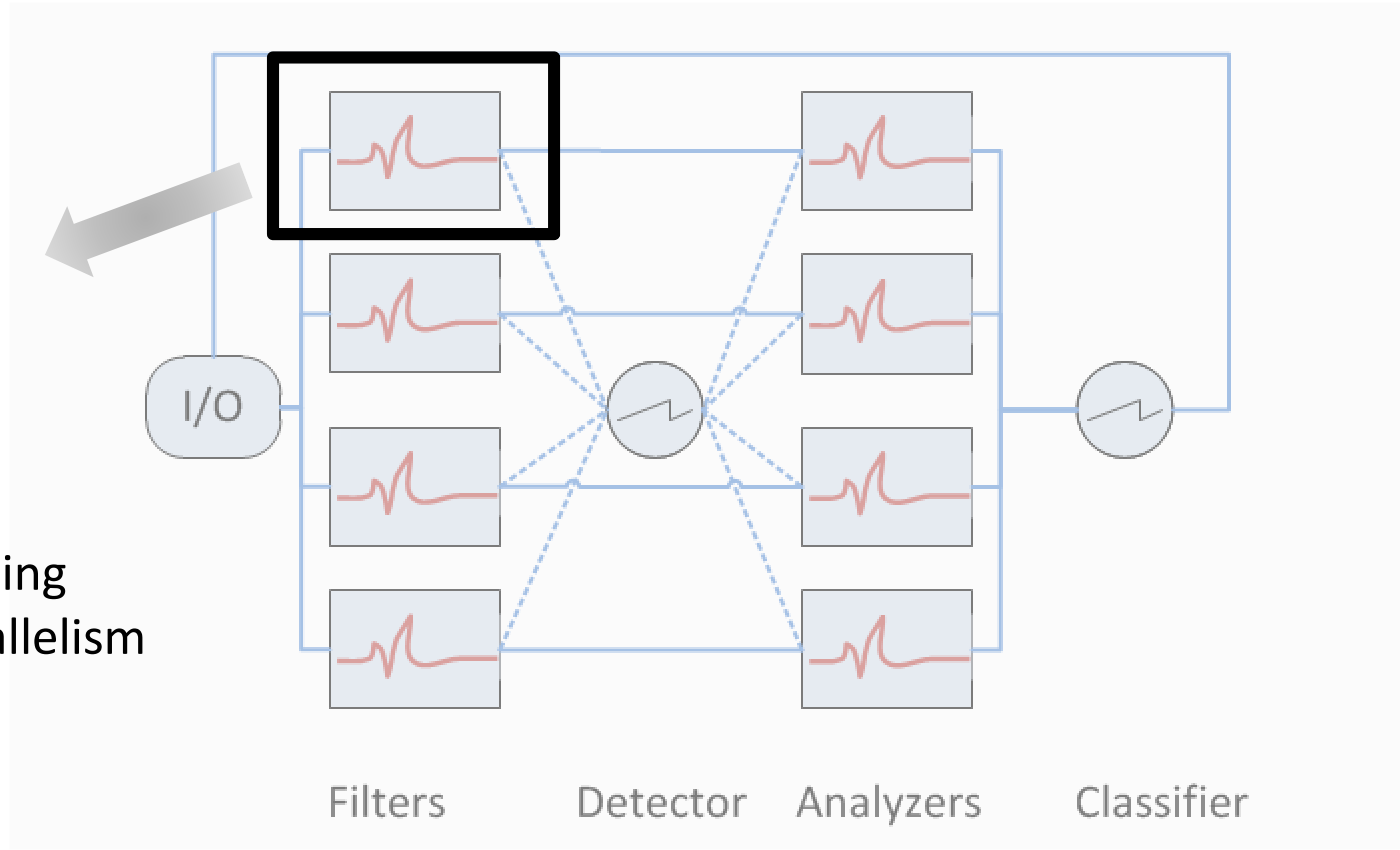
Parallel Execution



Data Parallelism



Vector Computing
Loop-level parallelism



Key Idea: On-chip distributed system

- Replace shared memory by local memory
  - Message passing for interaction between processes
- Separate processor for each process
  - Very simple processors
  - No scheduling, no interrupts,
  - Application-aware processors

→ Minimal operating system

→ Conceptually no memory bottleneck

→ Higher reliability and predictability by design

338

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) imm is zero extended to 32 bits

op	Rd	0	imm
17 – 14	13 – 11	10	9 – 0

(b)

op	Rd	1	0 0 0	x x 0	Rs
17 – 14	13 – 11	10	9		0

(c)

op	VRd	1	1 0 0	x x x	VRs
17 – 14	13 – 11	10	9		0

Load and Store

(a) off is zero extended to 13 bits

op	Rd	0	off	Rs
17 – 14	13 – 11	10	9 – 3	0

(b)

op	VRd	1	off	Rs
17 – 14	13 – 11	10	9 – 3	0

Conditional Branches

off is sign extended to 12 bits

1110	cond	off
17 – 14	13 – 10	9 – 0

Special Instructions

(a)

op	Rd	1	0 0 0	0 0 1	x x x
17 – 14	13 – 11	10	9		0

(b)

op	VRd	1	1 0 0	001	x x x
17 – 14	13 – 11	10	9		0

(c)

op	Rd	1	1 0	x x x x	Rs
17 – 14	13 – 11	10	9		0

(d)

op	Rd	1	0 1	x x x x	Rs
17 – 14	13 – 11	10	9		0

(e)

op	Rd	1	101	xxxx	Rs
17 – 14	13 – 11	10	9	6	0

Branch and Link

off is 14-bit offset

1111	off
17 – 14	13 – 0

TRM architecture

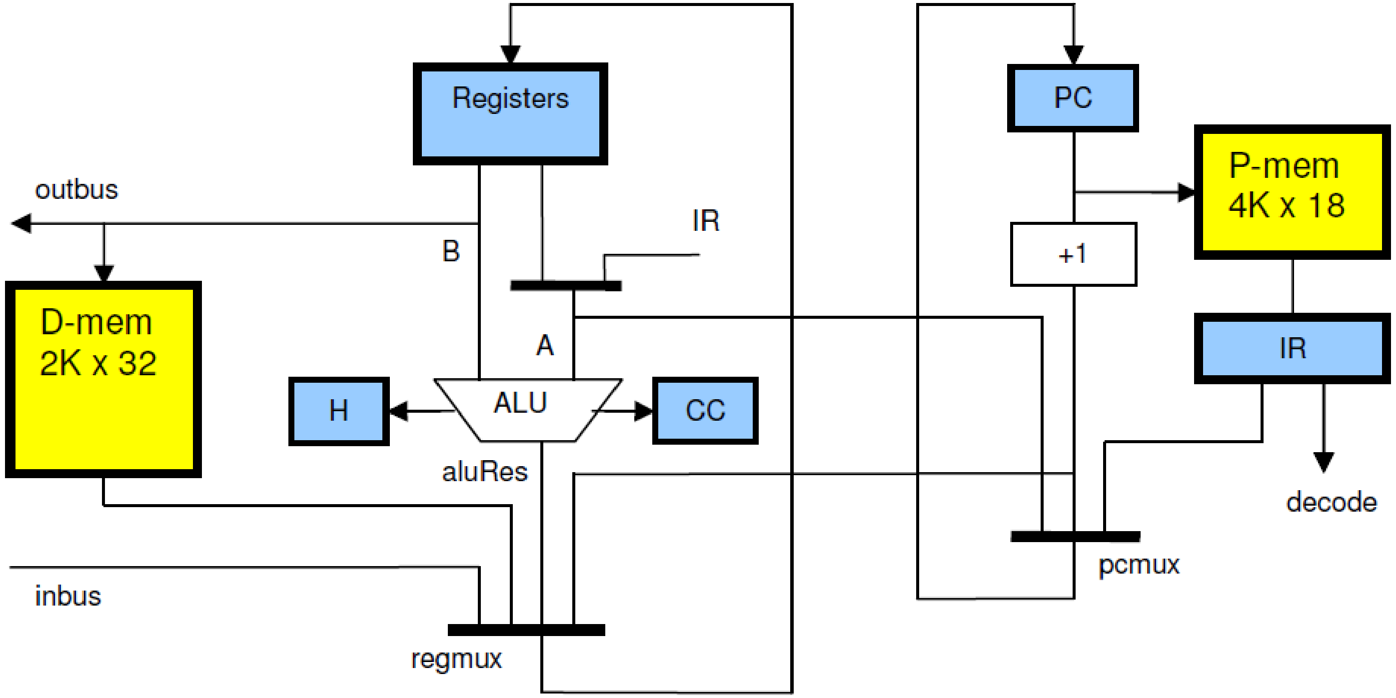


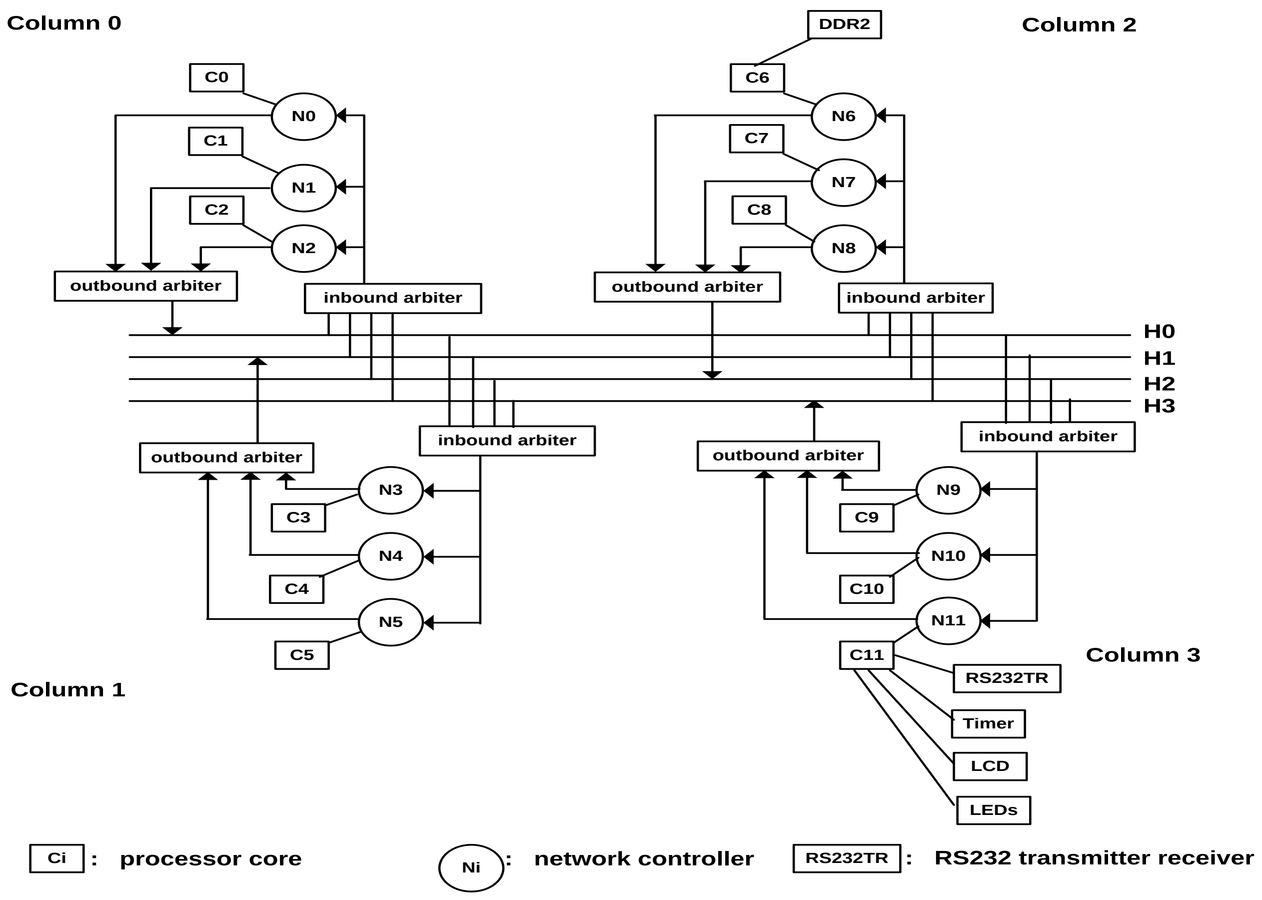
Figure from: Niklaus Wirth, *Experiments in Computer System Design*, Technical Report, August 2010
http://www.inf.ethz.ch/personal/wirth/Articles/FPGA-relatedWork/ComputerSystemDesign.pdf

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)

First Experiment: TRM12

A Multicore Processor Architecture on FPGA



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

Ling Liu, A 12-Core-Processor Implementation on FPGA, ETH Technical Report, October 2009
ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/6xx/646.pdf

Interface to network and I/O

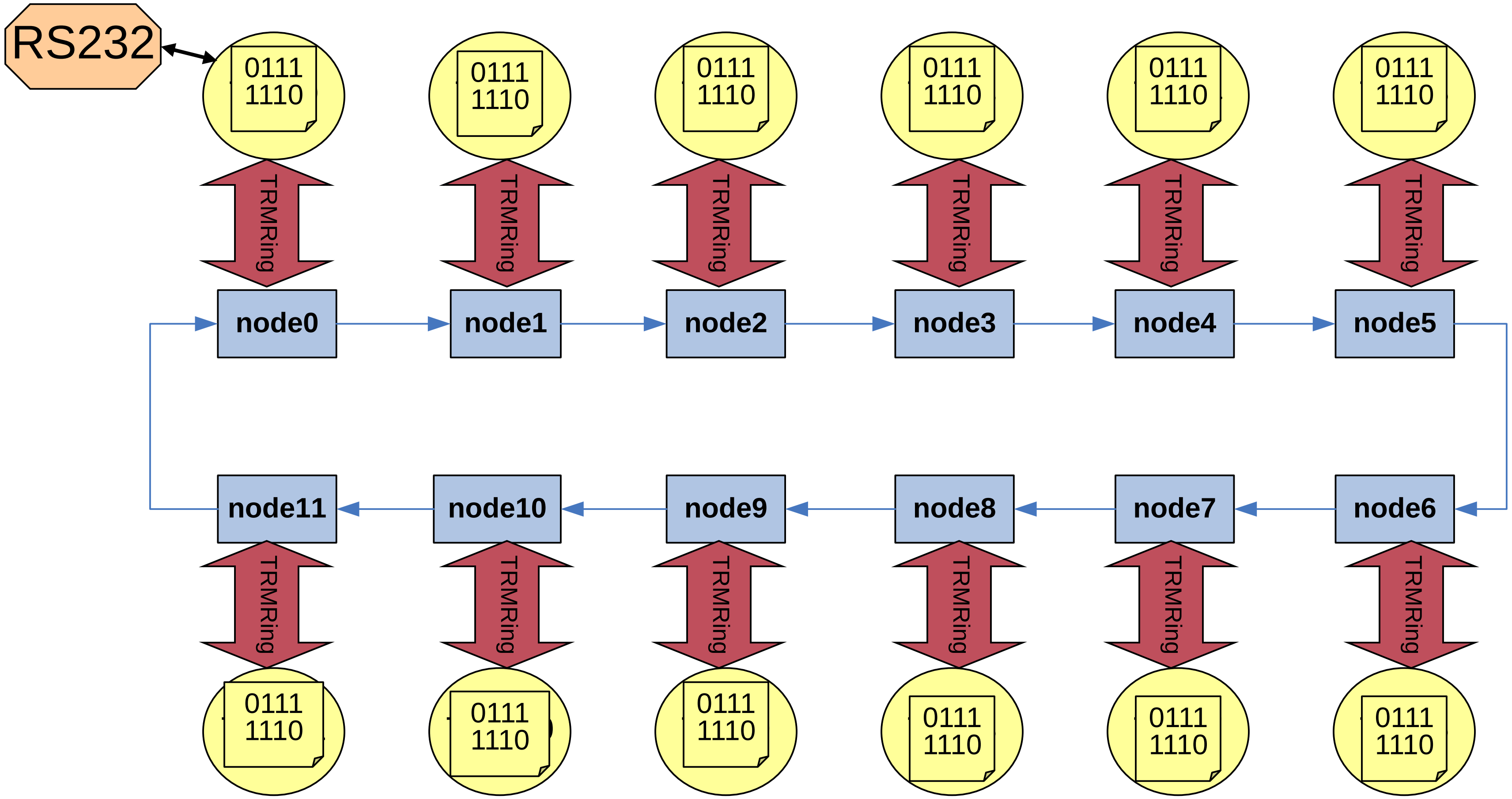
- TRM processor connected to a network controller („NetNode“)
- TRM core 11 connected to RS232 controller, a 2-line LCD controller, a timer and 8 LEDs
- TRM processor core 6 connected to 512 MB DDR2 controller
- Netnodes and RS232 controller treated as I/O port to the TRM processor, communication with TRM core through 32-bit I/O bus
- I/O accessed via memory mapped I/O at fixed addresses

346

Problems with this approach

- Not scalable
- Huge resource consumption
- Little but existing contention

Second Experiment: Ring of 12 TRMs



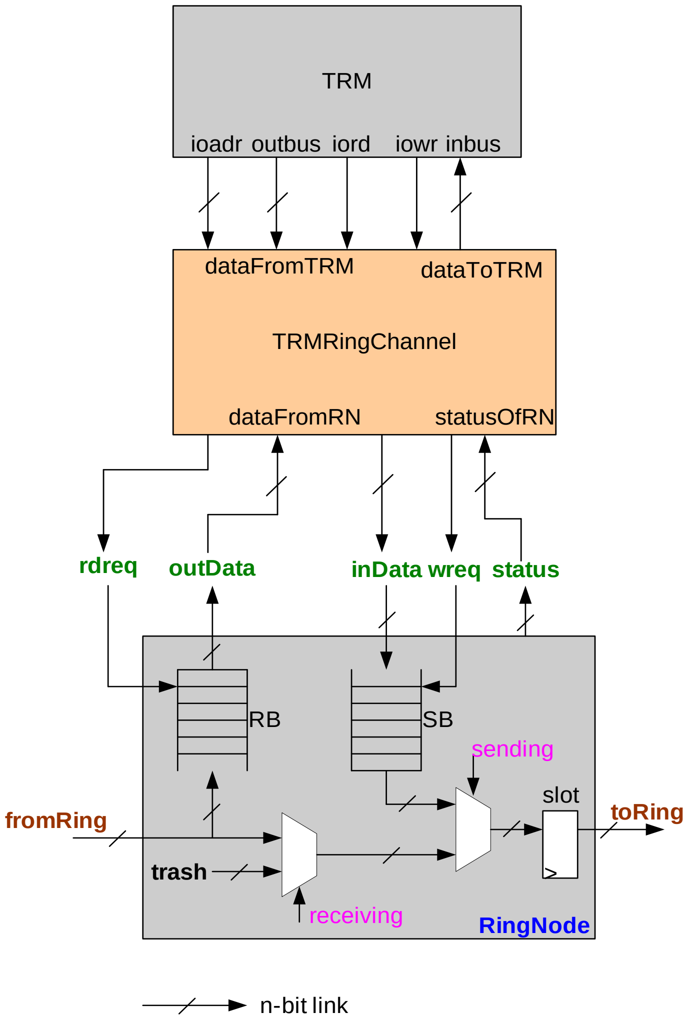
Slide from Dr. Ling Liu's Lecture Series on Reconfigurable Computing

Connection TRM / Ring

TRM

Adapter

Ring



- Ring interconnect very simple
- Small router
- Predictable latency

Problems with this approach

- Not scalable without huge loss of performance
- Large delays

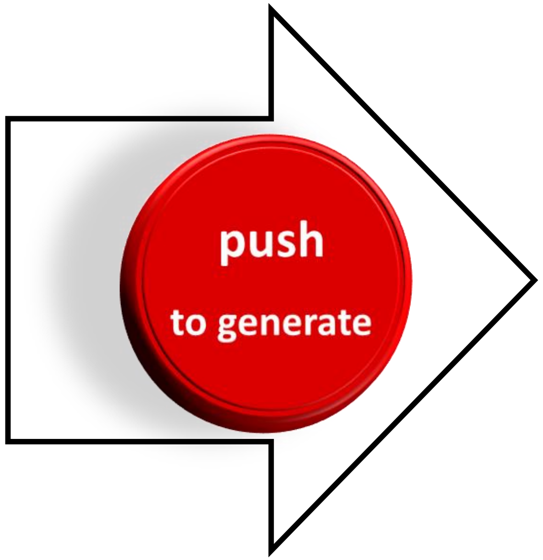
Programming Model

Case Studies

4.2 ACTIVE CELLS

Vision: Custom System on Button Push

System design as high-level program code



Electronic circuits

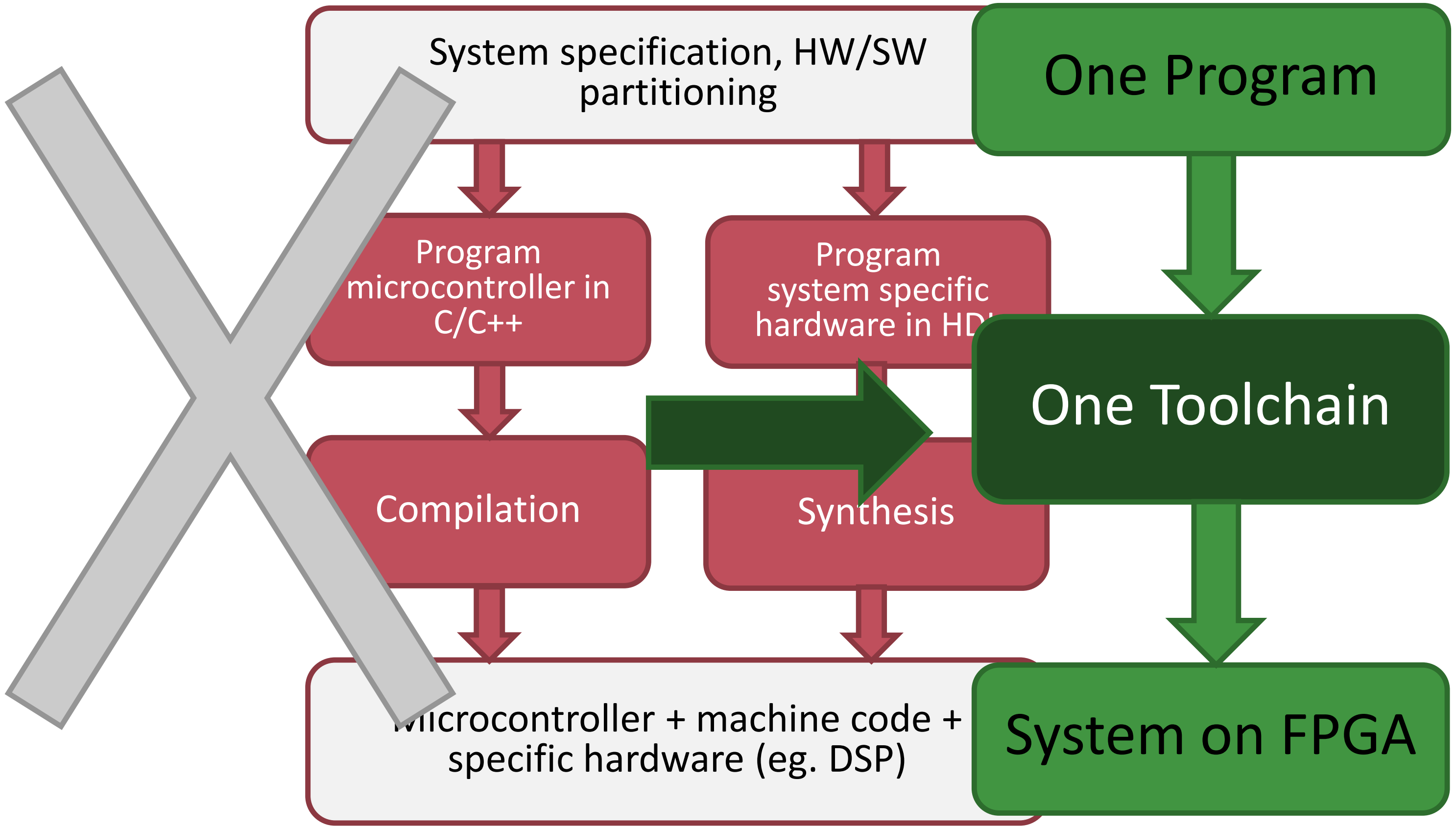
Computing model
Programming Language

Compiler, Synthesizer,
Hardware Library,
Simulator

Programmable Hardware
(FPGA)

Traditional HW/SW co-design

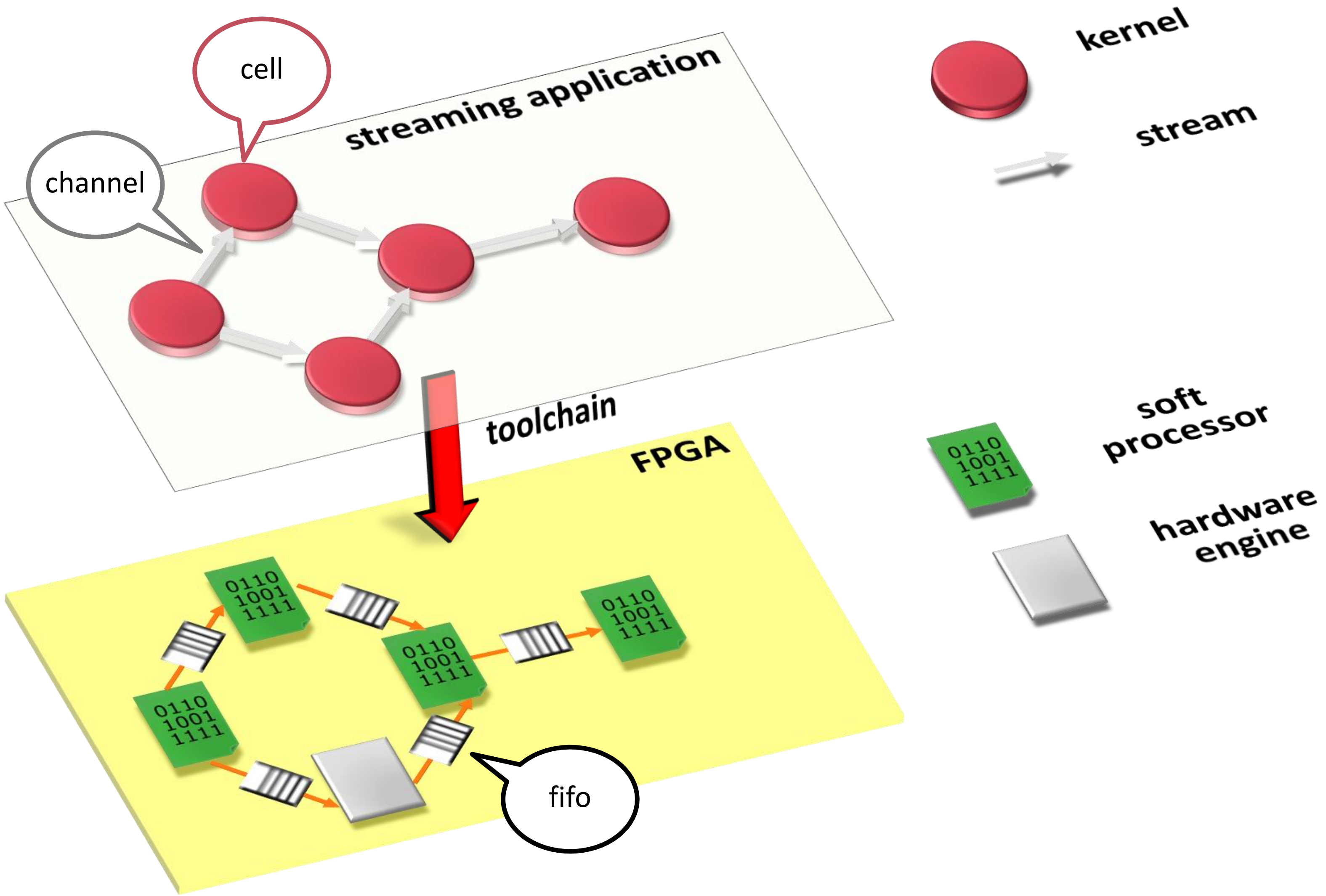
Goal



Traditional HW/SW co-design for embedded systems

Active Cells approach for embedded systems development

Software → Hardware Map



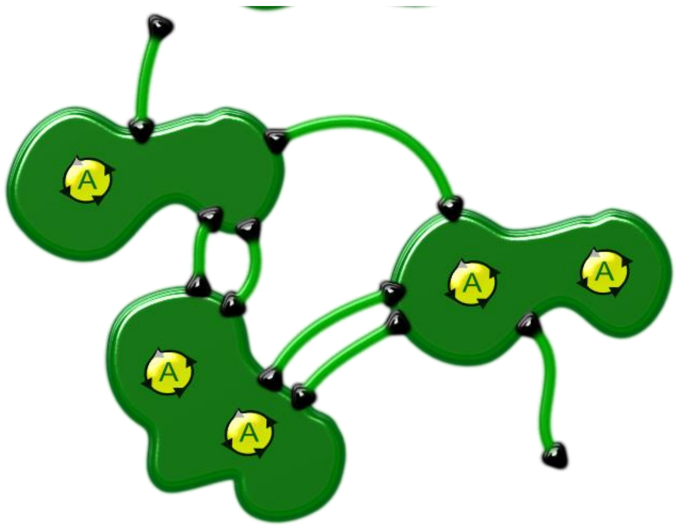
Consequences of the approach

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

➔ **No operating system**

Active Cells Computing Model

- Distributed system in the small
- Computation units: "Cells"
- Different parallelism levels addressed by
  - Communication Structure (Pipelining, Parallel Execution)
  - Cell Capabilities (Vector Computing, Simultaneous Execution)
- Inspired by
  - Kahn Process Networks
  - Dataflow Programming
  - CSP
  - ..



Active Cell Components

- Active Cell
  - Object with private state space
  - Integrated control thread(s)
  - Connected via channels
- Cell Net
  - Network of communication cells

Active Cells

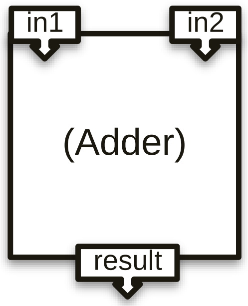
- Scope and environment for a running isolated process.
- Cells do not immediately share memory
- Defined as types with port parameters

communication ports

```
type
    Adder = cell (in1, in2: port in; result: port out);
    var summand1, summand2: integer;
    begin
        in1 ? summand1;
        in2 ? summand2;
        result ! summand1 + summand2
    end Adder;
```

blocking receive

non-blocking send



Cell Constructors

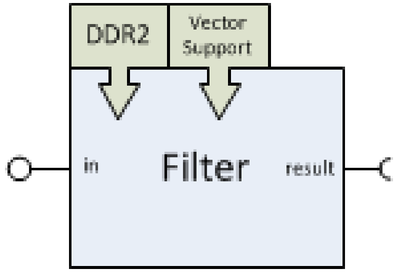
- Constructors to parameterize cells during allocation time

constructor

```
type
    Filter = cell (in: port in; result: port out);
    var ...; filterLength: integer;

        procedure & Init(filterLength: integer)
        begin self.filterLength := filterLength
        end Init;

    begin
        (* ... filter action ... *)
    end Filter;

var filter: Filter;
begin

.... new(filter, 32); (* initialization parameter filterlength = 32 *)
```

Further Configurations:Cell Capabilities

- Cells can be parametrized further, being provided with further capabilities or non-default values.




```
type
    Filter = cell {Vector, DataMemory(2048), DDR2}
                (in: port in (64); result: port out);
    var ...

    begin
        (* ... filter action ... *)
    end Filter;
....
```

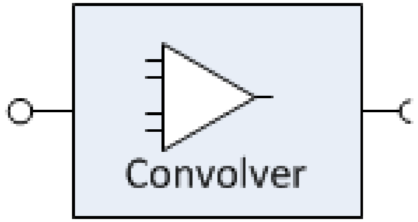
Cell is a VectorTRM with 2k of Data Memory and has access to DDR2 memory

This port is implemented with a (bit-)width of 64

Engine Cell Made From Hardware

- Special cells are provided as prefabricated hardware components (***Engines***).

```
type
    Convolver2d= cell {Engine}
                    (in: port in (64); result: port out);

    end Convolver2d;
```



Hierarchic Composition: Cell Nets

- Cellnets consist of a set of cells that can be connected over their ports.
  - Allocation of cells: **new** statement
  - Connection of cells: **connect** statement

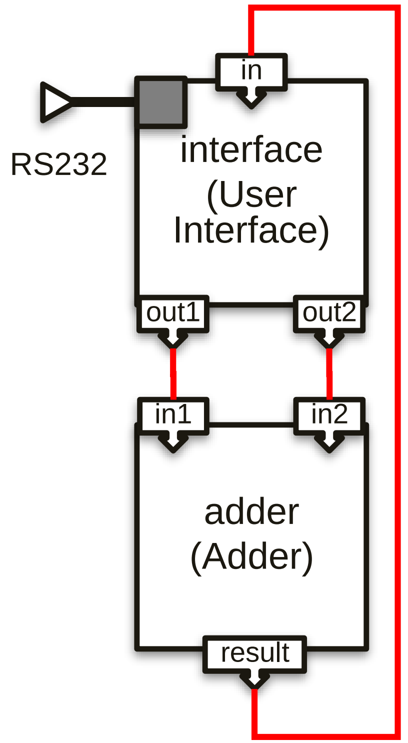
- Cellnets can **provide ports**, ports of cells can be **delegated** to the ports of the net
  - Delegation of cells: **delegate** statement

- *Terminal* (or *closed*) Cellnets* can be deployed to hardware

*i.e. Cellnets without ports

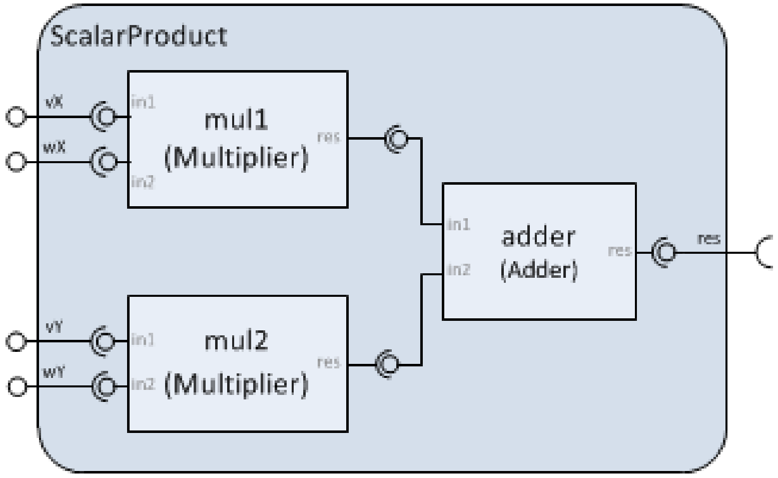
Terminal Cellnet Example

```
cellnet Example;
import RS232;
type
    UserInterface = cell {RS232}(out1, out2: port out; in:
    port in)
    (*...*) end UserInterface;

    Adder = cell(in1, in2: port in; out: port out)
    (* ... *) end Adder;

var interface: UserInterface; adder: Adder
begin
    new(interface);
    new(adder);
    connect(interface.out1, adder.in1);
    connect(interface.out2, adder.in2);
    connect(adder.result, interface.in);
end Example.
```



Hierarchic Composition Example

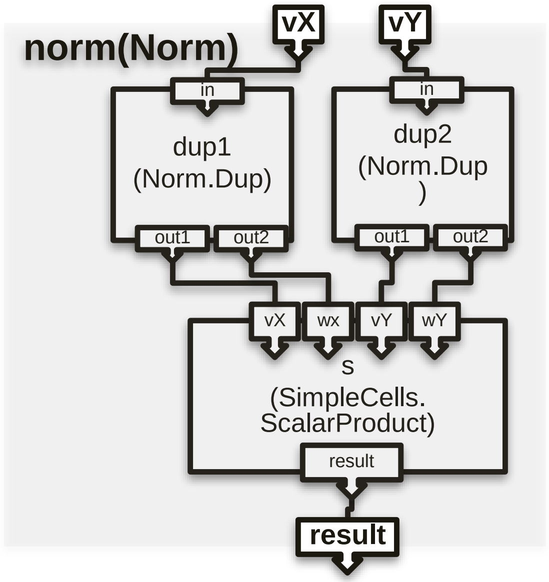
```
module SimpleCells
import RS232;
type
    Adder = cell (in1, in2: port in; result: port out)
    (* ... *) end Adder;

    Multiplier = cell (in1, in2: port in; result: port out)
    (* ... *) end Adder;


    ScalarProduct*= cellnet (vx,vy,xw,xy: port in; result: port out)
    var adder: Adder; multiplier1, multiplier2: Multiplier;
    begin
        new(mul1); new(mul2); new(adder);
        delegate(vx, mul.in1); delegate(wx, mul1.in2);
        delegate(vy, mul2.in1); delegate(wy, mul2.in2);
        connect(mul1.result, adder.in1); connect(mul2.result, adder.in2);
        delegate(result, adder.result)
    end ScalarProduct;

end SimpleCells
```



port delegation

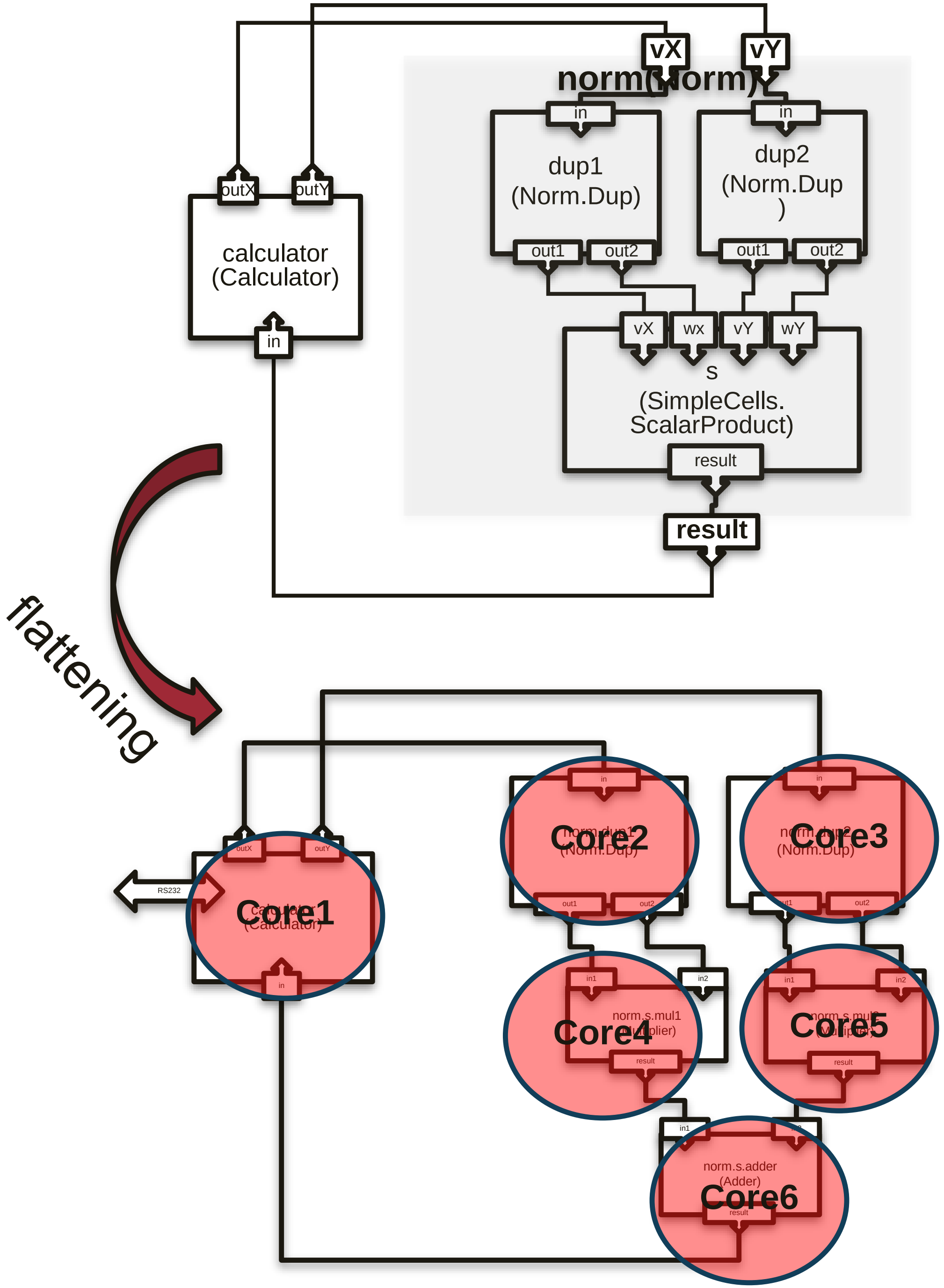
Example of a wired Cellnet

```
cellnet Test;
import SimpleCells, RS232;
type
    Norm*=cellnet (vX,vY: port in; result: port out)
    type
        Dup*=cell(in: port in; out1,out2: port out)
        var val: LONGINT;
        begin
            loop in ? val; out1 ! val; out2 ! val end
        end Dup;

    var s: SimpleCells.ScalarProduct2d; dup1, dup2: Dup;
    begin
        new(s); new(dup1); new (dup2);
        connect (dup1.out1,s.vX); connect(dup1.out2,s.wX);
        connect(dup2.out1,s.vY); connect(dup2.out2,s.wY);
        delegate(vX,dup1.in);delegate(vY,dup2.in);
        delegate(result,s.result);
    end Norm;
```



Flattening

```
Calculator*=cell {RS232} (in: port in;
                          outX,outY: port out)
var result: longint; vX,vY,wX,wY: longint;
begin
    loop
        RS232.ReceiveInteger(vX);
        RS232.ReceiveInteger(vY);
        send (outX,vX); send(outY,vY);
        receive (in,result);
        RS232.SendInteger(result);
    end;
 end Calculator;

var calculator: Calculator; norm:Norm;
begin
    new(calculator); new(norm);
    connect(calculator.outX,norm.vX);
    connect(calculator.outY,norm.vY);
    connect(norm.result,calculator.in);
end Test.
```

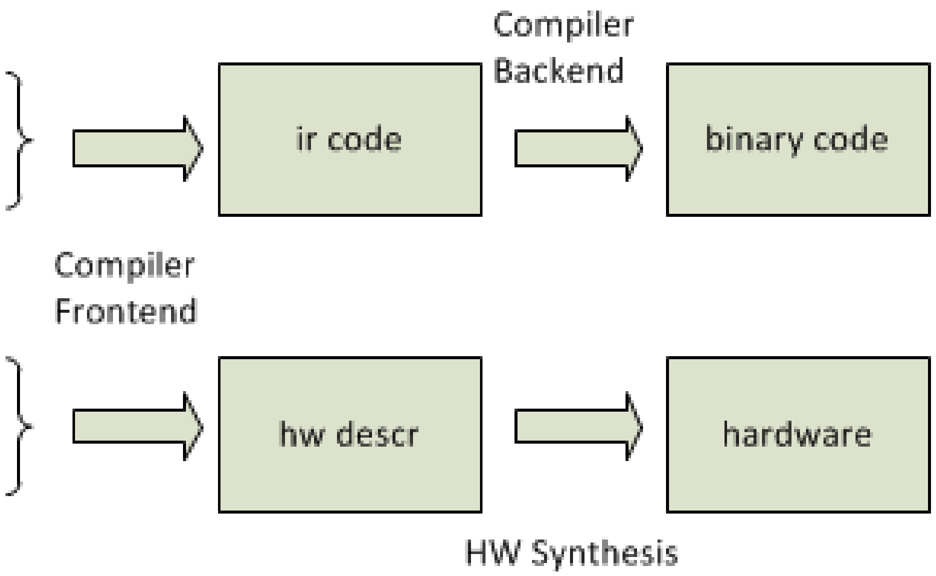


Hybrid Compilation

cellnet N;

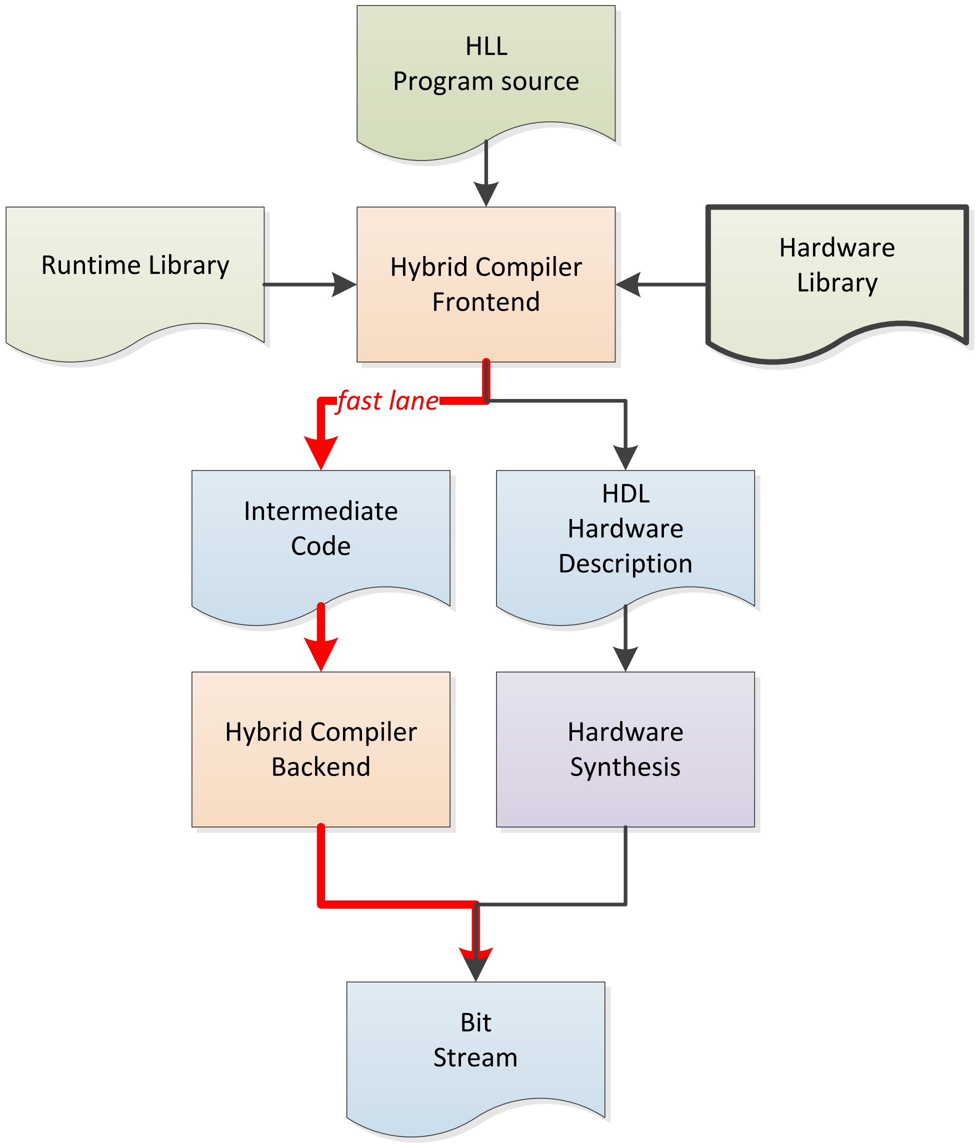
type A=**cell**(pi: port in; po: port out);
var x: **integer**;
begin
… pi ? x; … po ! x; …
end A;

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



Code body	Role	Compilation method
Cell (Softcore)	Program logic	Software Compilation
Cell (Engine)	Computation unit	Hardware Generation
Cell Net	Architecture	Hardware Compilation

Automated Mapping to FPGA



Hardware Library

Computation Components

- General purpose minimal machine: TRM, FTRM
- Vector machine: VTRM
- MAC, Filters etc.

Storage Components

- DDR2 controller
- configurable BRAMs
- CF controller

Communication Components

- FIFOs
  - 32 * 128
  - 512 * 128
  - 32, 64, 128, 1k * 32

I/O Components

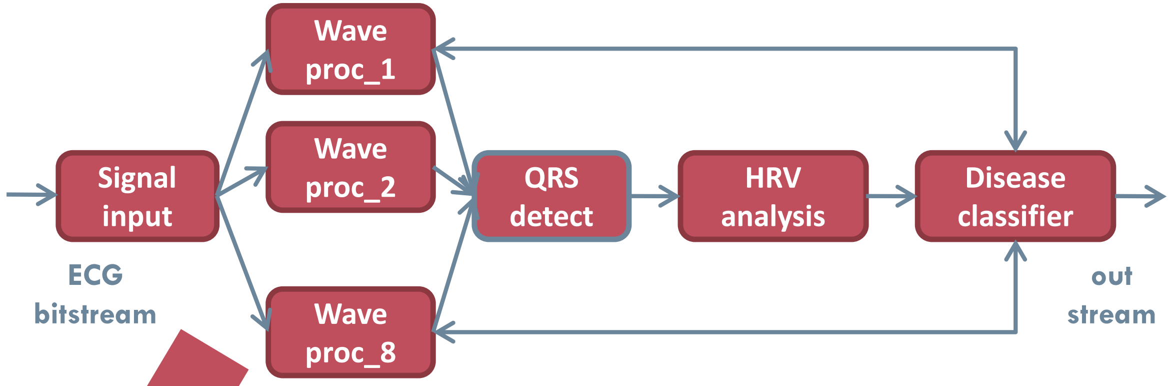
- UART controller
- LCD, LED controller
- SPI, I2C controller
- VGA, DVI controller

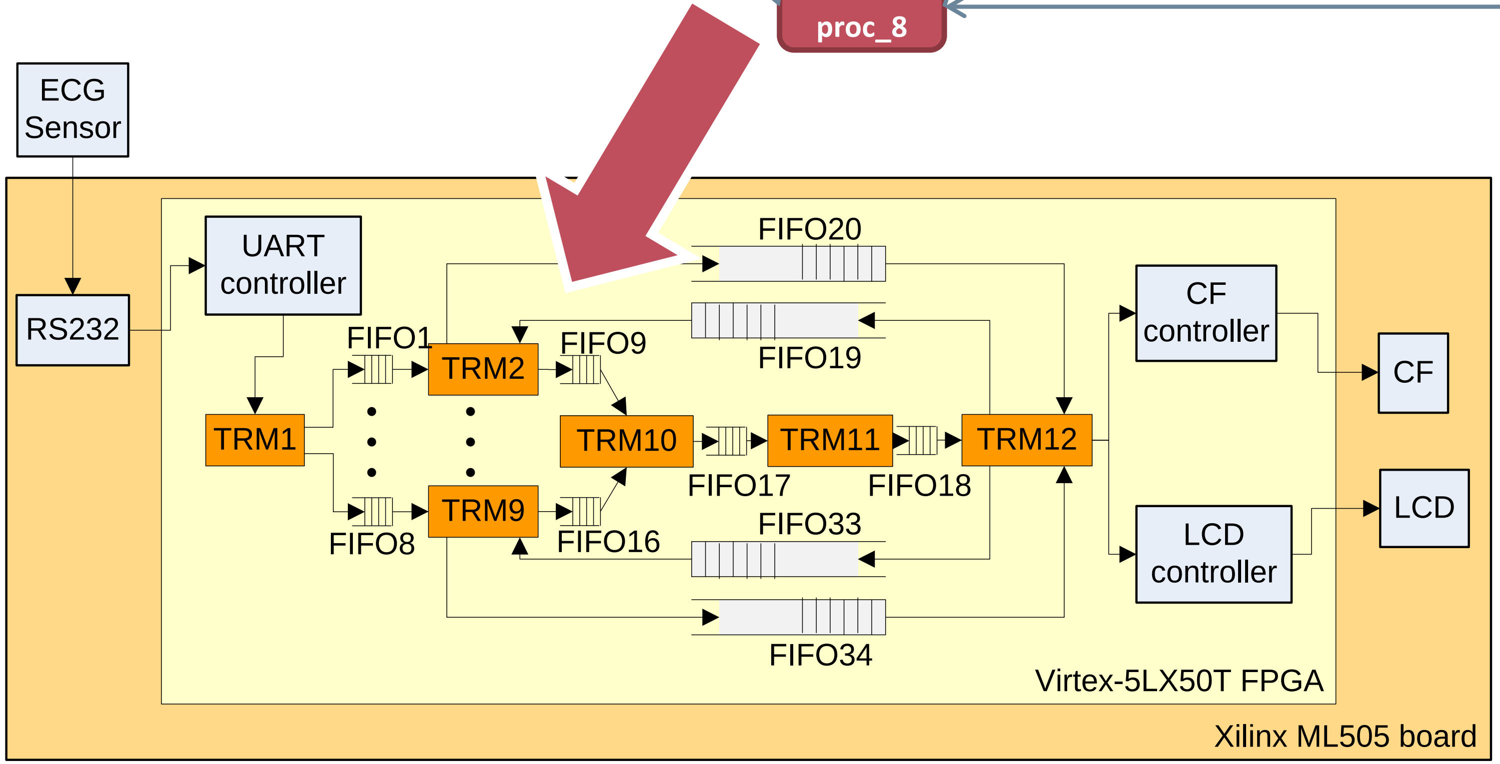
Case Study 1: ECG

Focus: Resources and Power

Real-time

ECG Monitor





Resources

- ECG Monitor*

#TRMs	#LUTs	#BRAMs	#DSPs	TRM load
12	13859 (48%)	52 (86%)	12 (25%)	**<5%** @116 MHz

- Maximum number of TRMs in communication chain

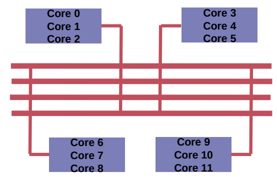
FPGA	#TRMs	#LUTs	#BRAMs	#DSPs
Virtex-5	30	27692 (96%)	60 (100%)	30 (62%)
Virtex 6	500			

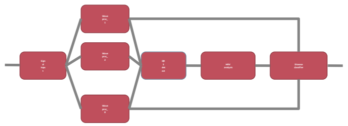
*8 physical channels @ 500 Hz sampling frequency implemented on Virtex 5

371

Comparative Power Usage

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



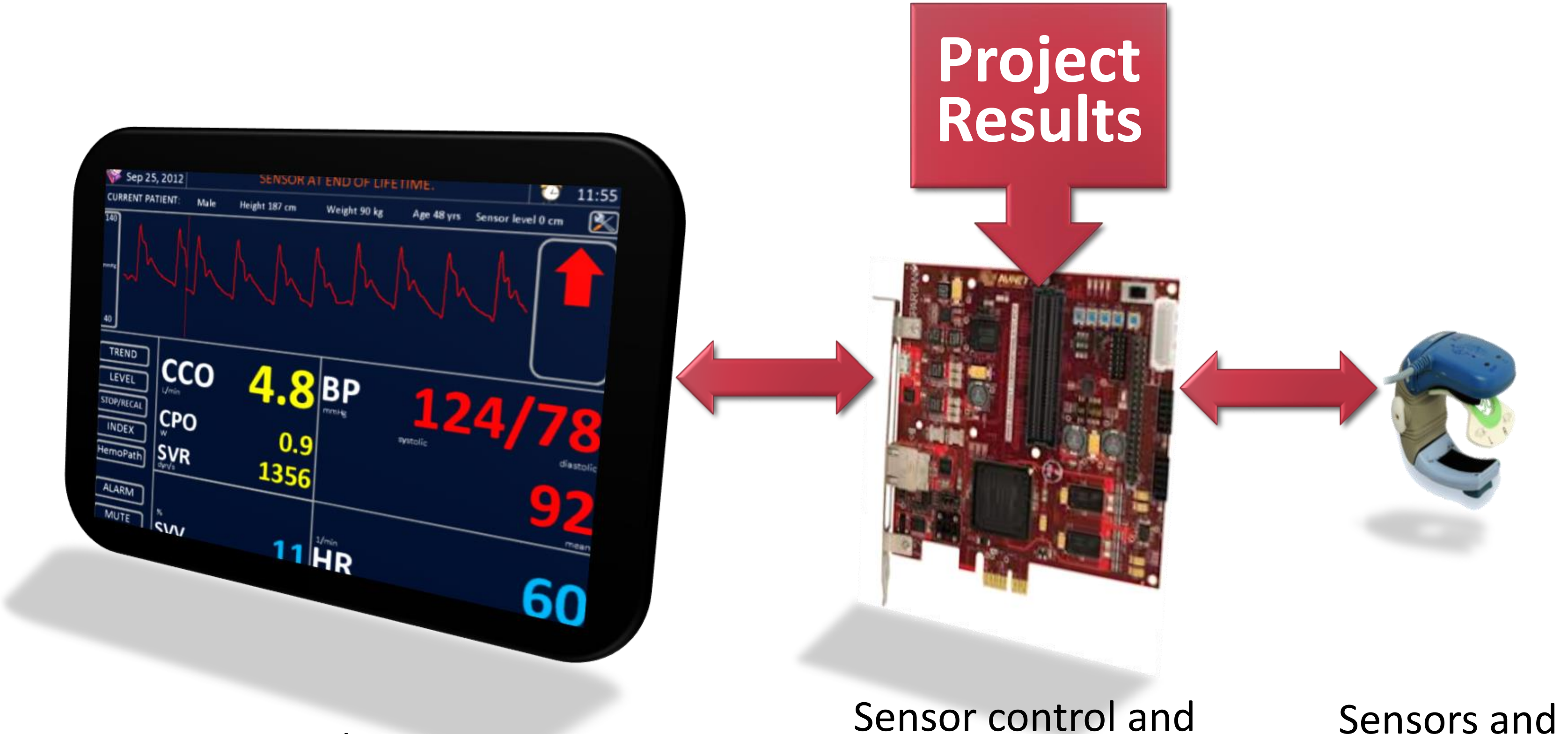


System	Static Power (W)	Dynamic Power (W)
Preconfigured ("TRM12")	**3.44**	**0.59**
Dynamically configured	**0.5**	**0.58**

86% saving!

Case Study 2: Non-Invasive Continuous Blood Pressure Monitor

Focus: Development Cycle Time

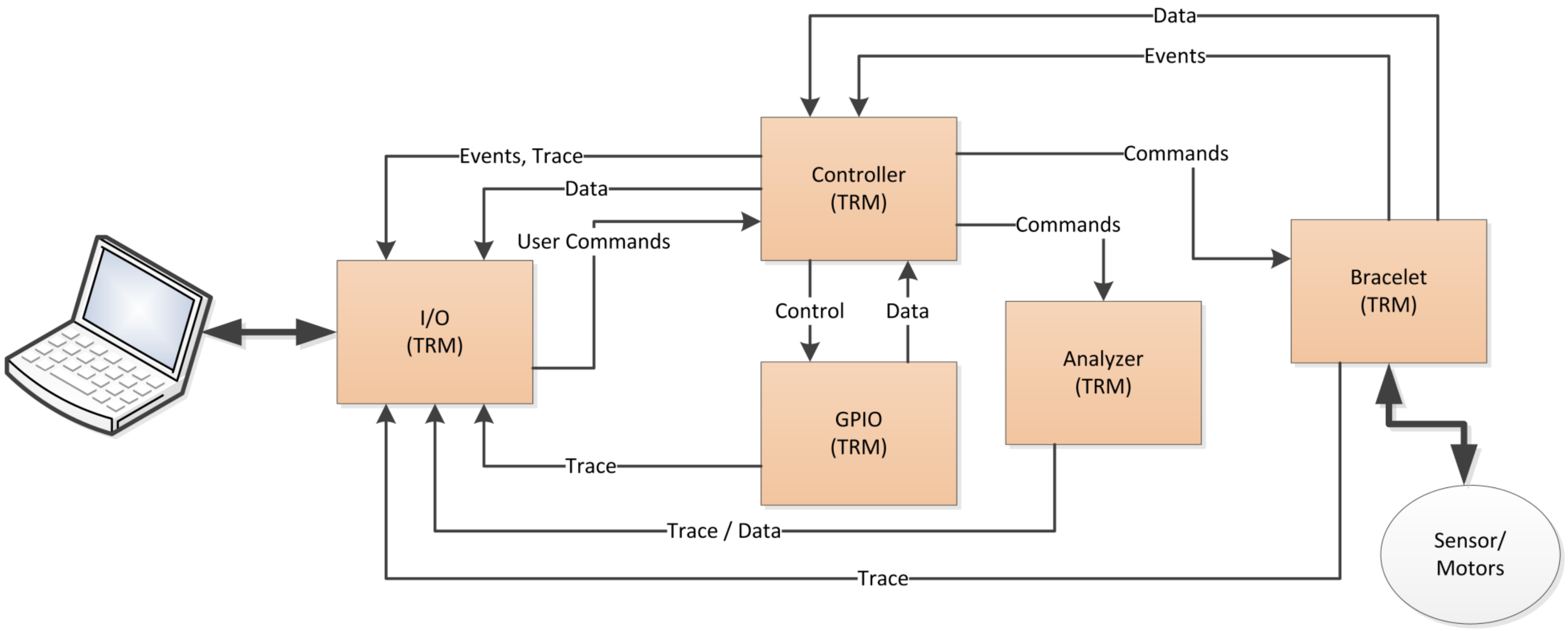


A2 Host OS with GUI on PC

Sensor control and medical algorithms on Spartan 6 FPGA

Sensors and Motors on Bracelet

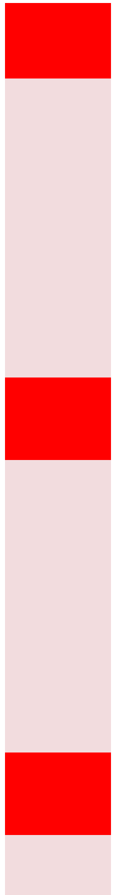
Medical Monitor Network On Chip



Dominated by TRM processors. Feedback driven.

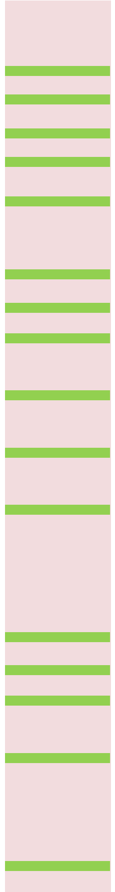
374

Development Cycle Times



sporadic

Step	Medical Monitor	OCT (full)
Software Compilation	4s	2s
Hardware Implementation	20 min	20 min
Stream Patching (all)	2 min	-
Stream Patching (typical)	12 s	-
Deployment	11s	16s

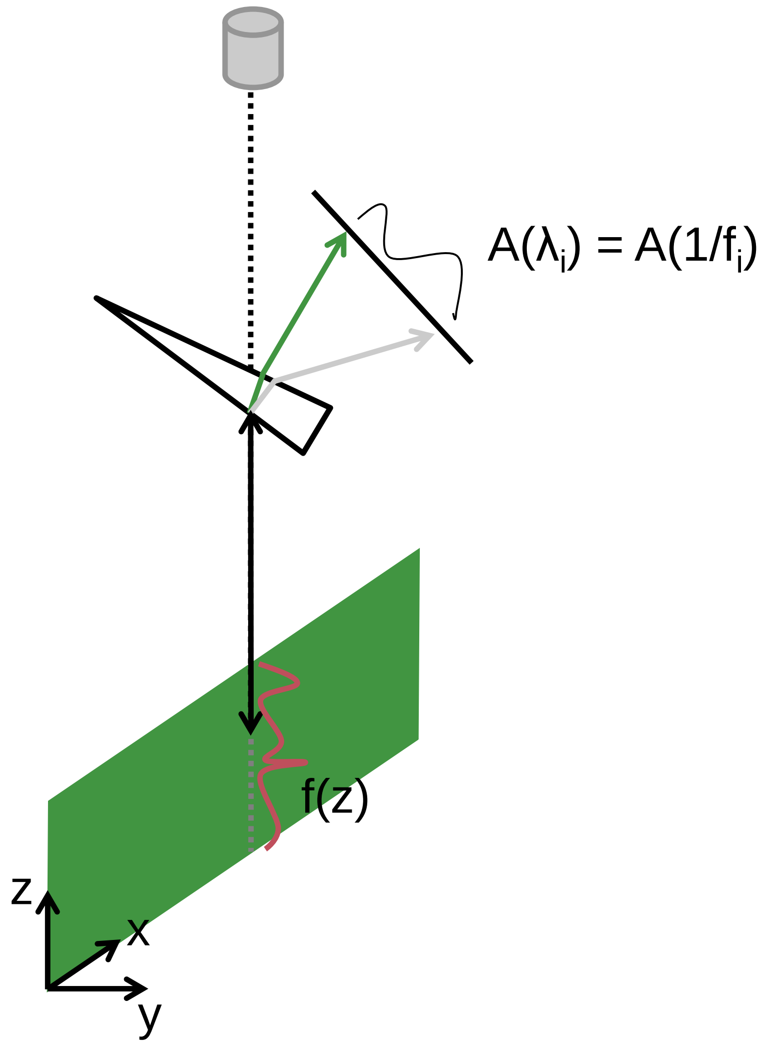


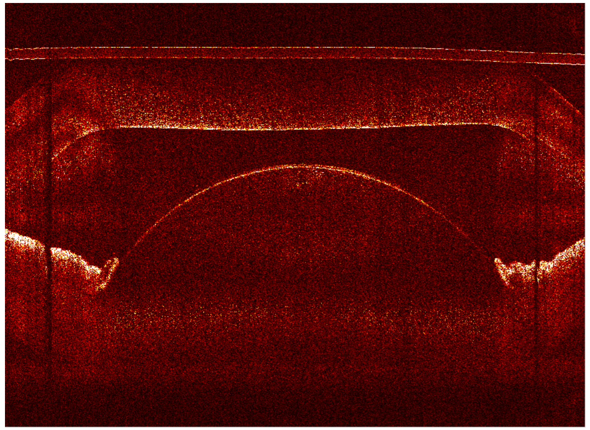
often

375

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

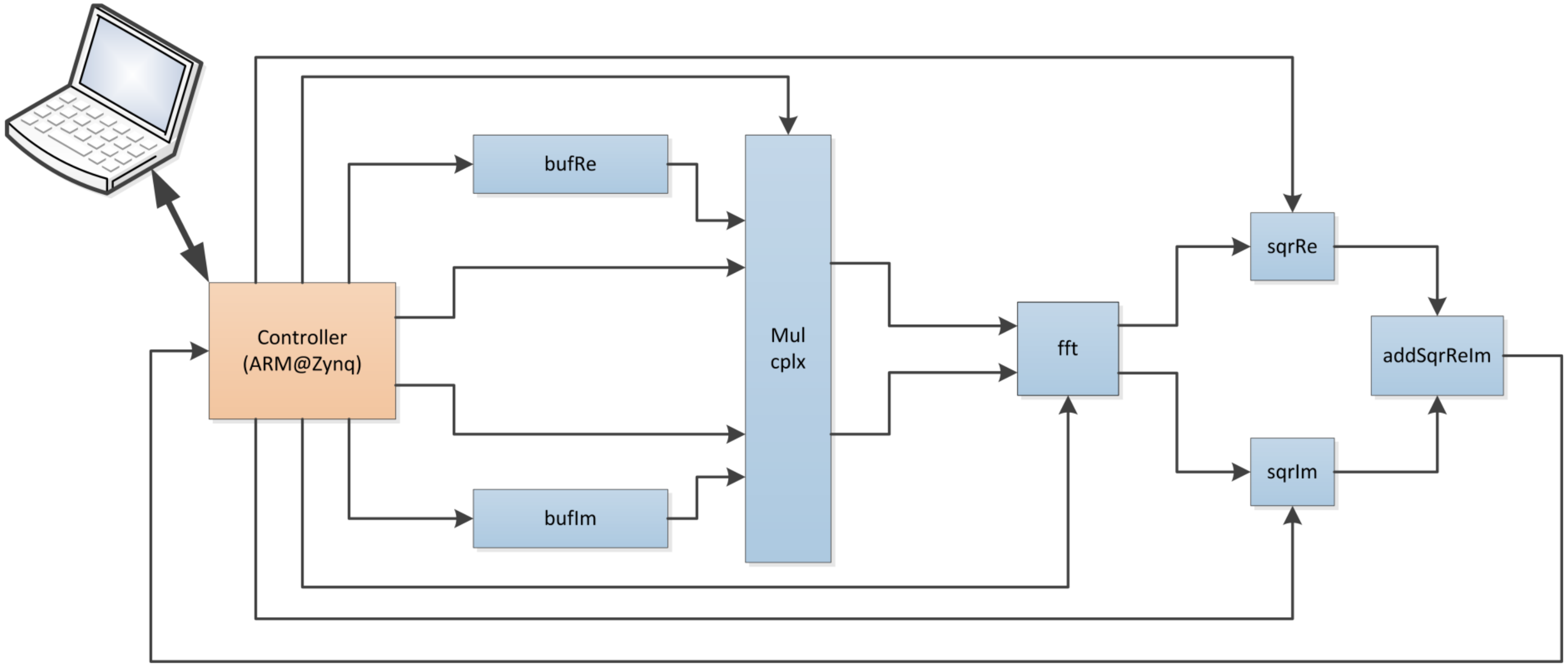
3. (Inverse) FFT

… for many lines x in a row (2d)
… and many rows y in a column (3d)

376

A component of OCT image processing

Dispersion Compensation



Dominated by Engines. Dataflow driven.

Performance and Resource Usage

	Medical Monitor	Dispersion Compensation	OCT
Architecture	Spartan 6 XC6SLX75	Zynq 7000 XC7Z020	Zynq 7000 XC7Z020
Resources	28% Slice LUTs, 4% Slice Registers 80% BRAMs 24% DSPs	11% Slice LUTs, 6% Slice Registers 7% BRAMs 15% DSPs 1 ARM Cortex A9	17% Slice LUTS 8% Slice Registers 22% BRAMs 31% DSPs 1 ARM Cortex A9
Clock Rate	58 MHz	118 MHz	50 MHz
Performance	--	**8.3 GFPOps*** *up to 32 GFPops*****	**4.3 GFPOps***
Data Bandwidth	1.25 Mbit /s (in) 23 kB/s (out)	236 MWords/s (in) 118 MWords/s (out)	50 MWords/s (in) 50 MWords/s (out)
Power	~2W	~5W	~5W

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

Conclusion

ActiveCells: Computing model and tool-chain for emerging configurable computing

- Configurable interconnect → Simple Computing, Power Saving
- Hybrid compilation→ Decreased Time to Market
- Embedding of task engines → High Performance