

Active Cells:

A Programming Model for Configurable Multicore Systems

CASE STUDY 4: CUSTOM DESIGNED MULTI- PROCESSOR SYSTEM

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).

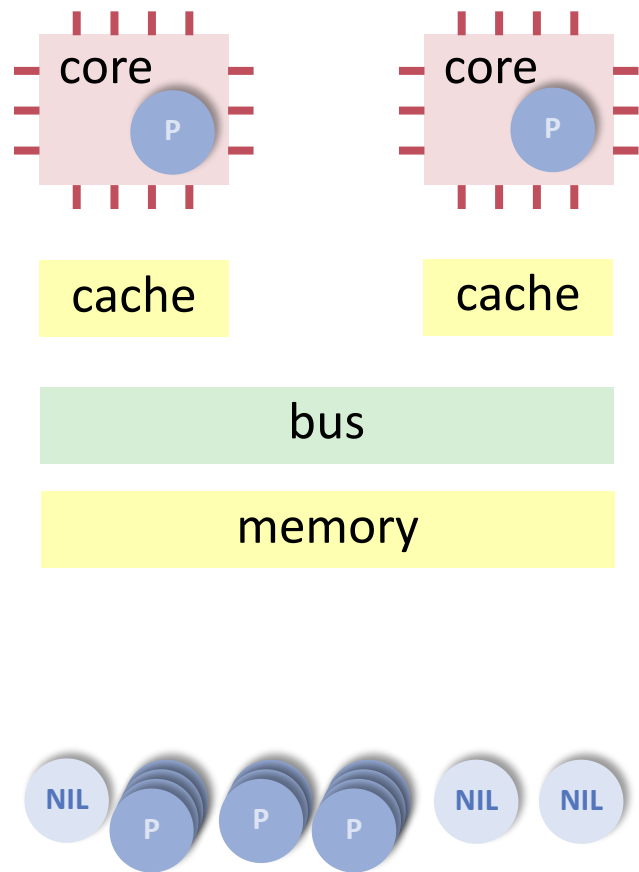
We are particularly indebted for consulting to Chuck Thacker, Niklaus Wirth, Timothée Martiel, Paul Reed and Florian Negele.

Thanks for Support from Xilinx Academic Program.

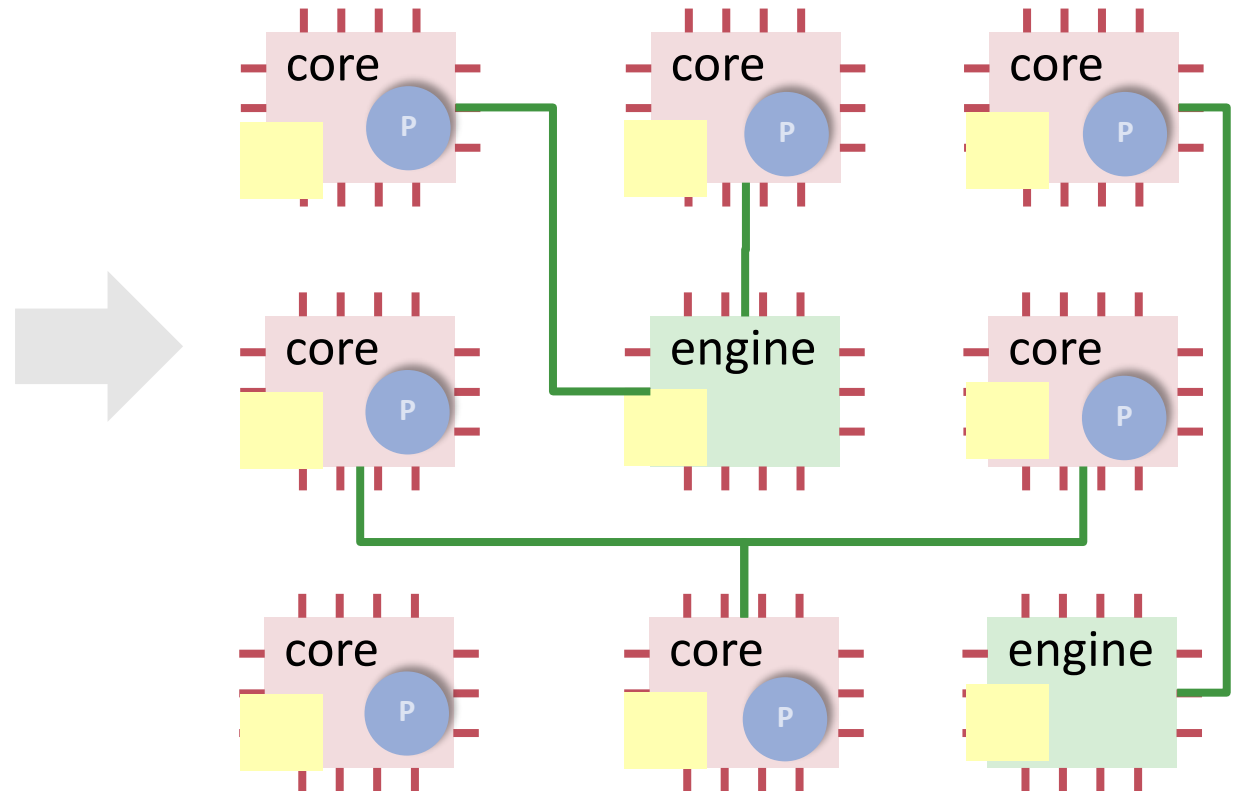
CRBM Implementation and exercise 12 by Stephan Koster.

Vision

General Purpose Shared Memory Computer



Application Specific Multicore Network On Chip



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**

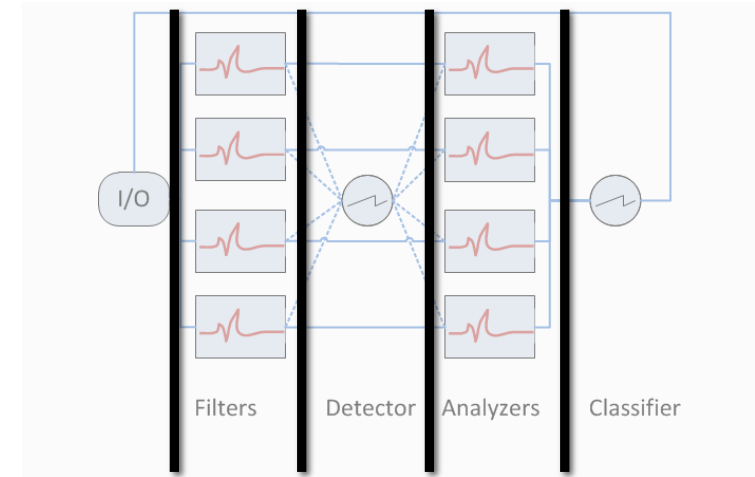
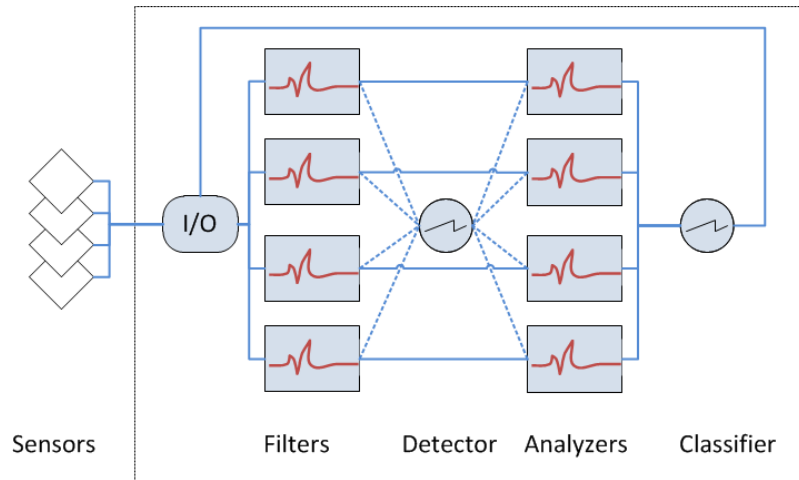


Industry: High Performance Sensor Driven Medical IT

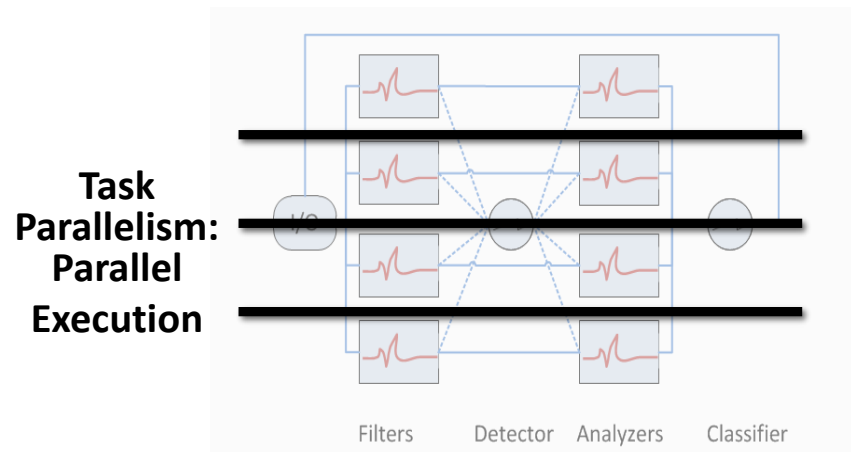
- **streaming** applications: ultrasound, tomography, hemodynamics, etc.



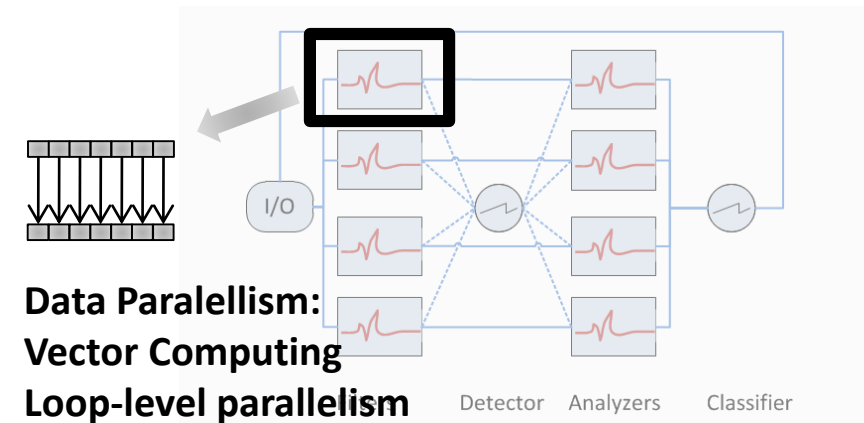
Focus: Streaming Applications



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: $d \times 512$ words of 32 bits.
 - local program memory: $i \times 1024$ 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

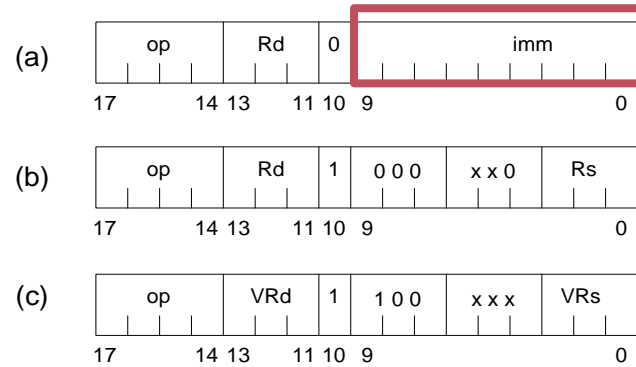
* Invented and implemented by [Dr. Ling Liu](#) and Prof. Niklaus Wirth

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)

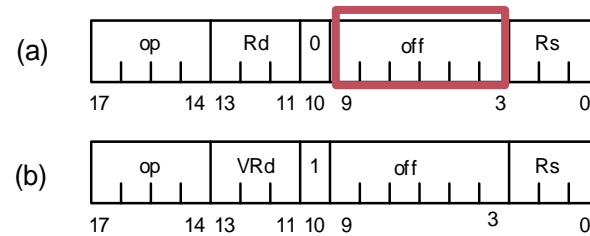
Encoding Overview

Register Operations



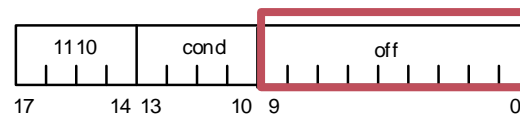
imm is zero extended to 32 bits

Load and Store



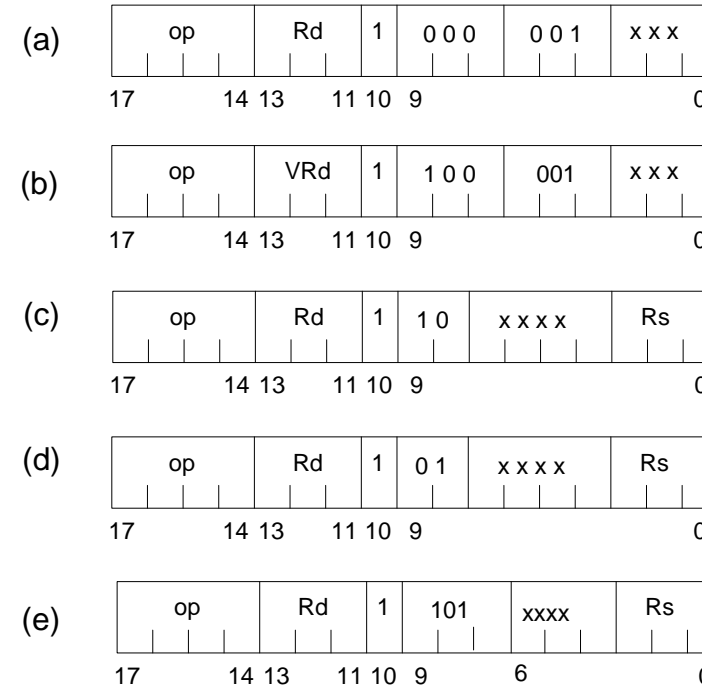
off is zero extended

Conditional Branches

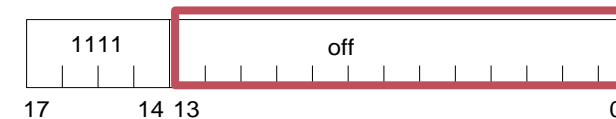


off is sign extended

Special Instructions



Branch and Link



off is 14-bit offset

TRM architecture

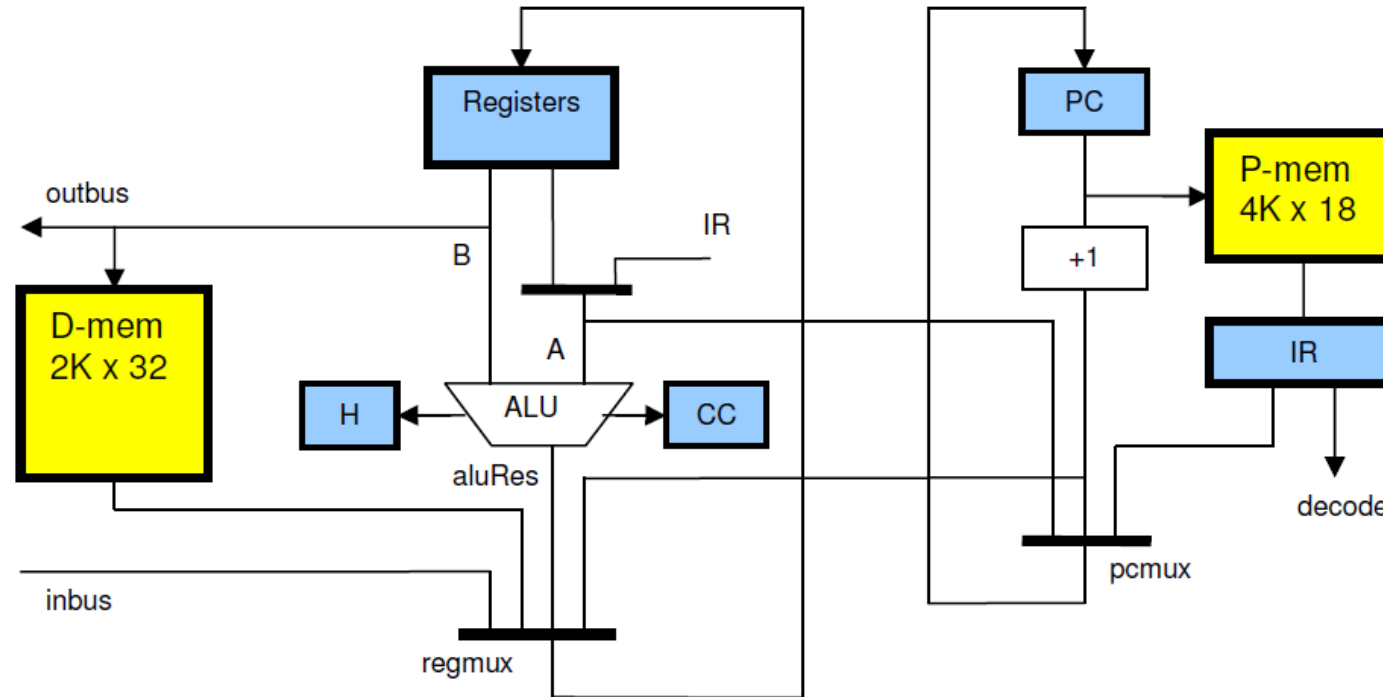


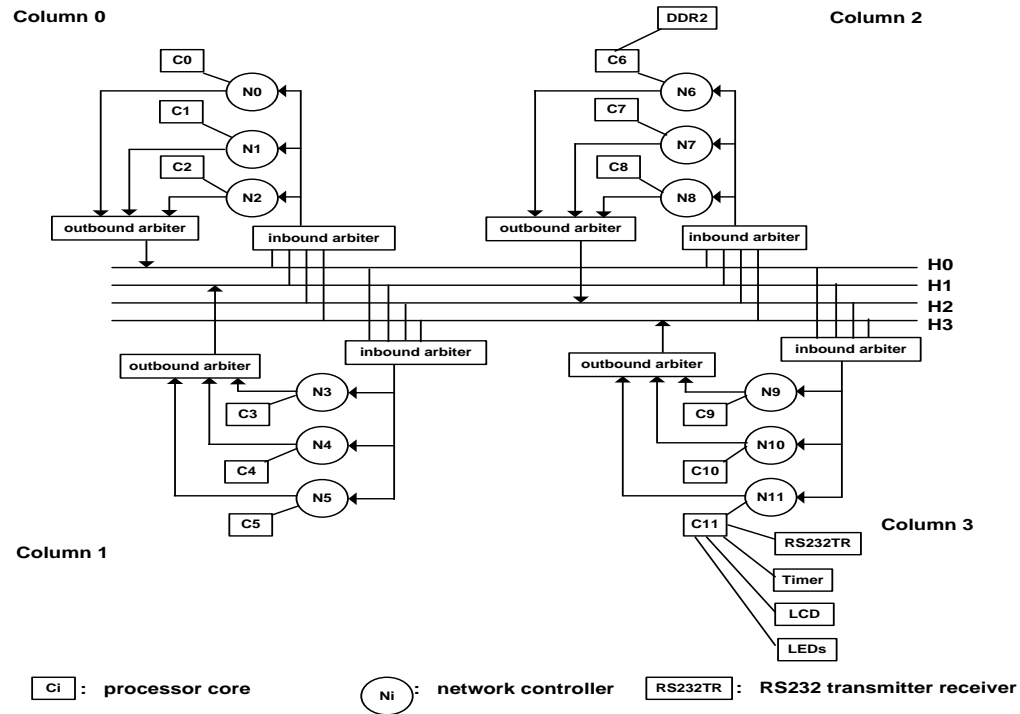
Figure from: Niklaus Wirth, *Experiments in Computer System Design*, Technical Report, August 2010
<http://www.inf.ethz.ch/personal/wirth/Articles/FPGA-relatedWork/ComputerSystemDesign.pdf>

Variants of TRM

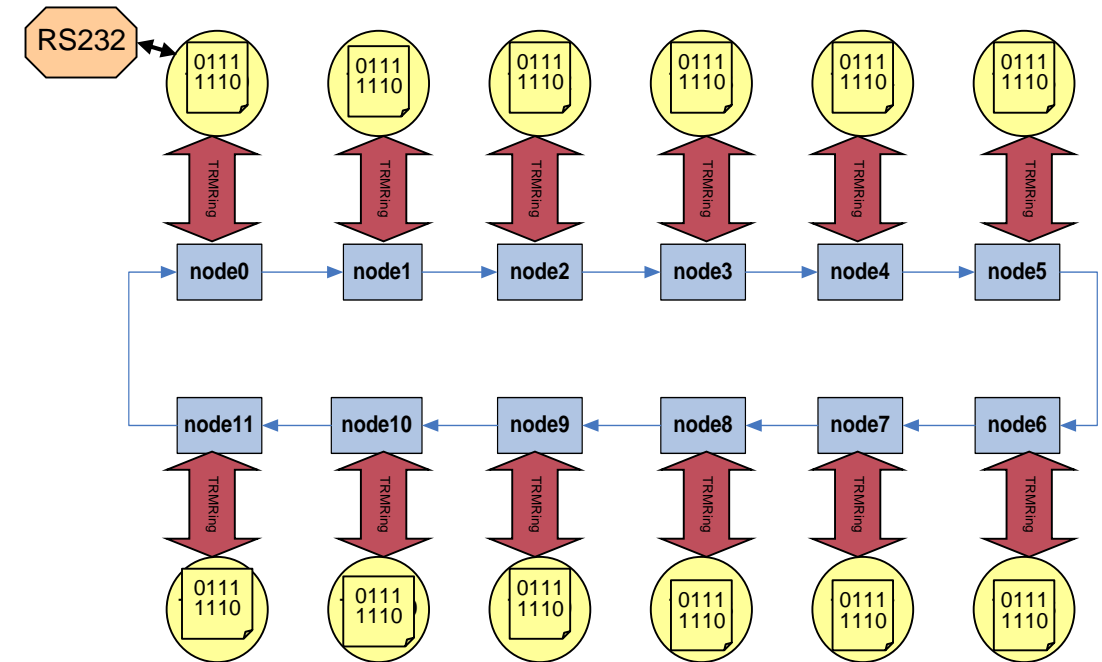
- FTRM
 - includes floating point unit
- VTRM (Master Thesis Dan Tecu)
 - includes a vector processing unit
 - supports 8 x 8-word registers
 - available with / without FP unit
- TRM with software-configurable instruction width (Master Thesis Stefan Koster, 2015)

Initial Experiments

TRM12 Bus



TRM12 Ring

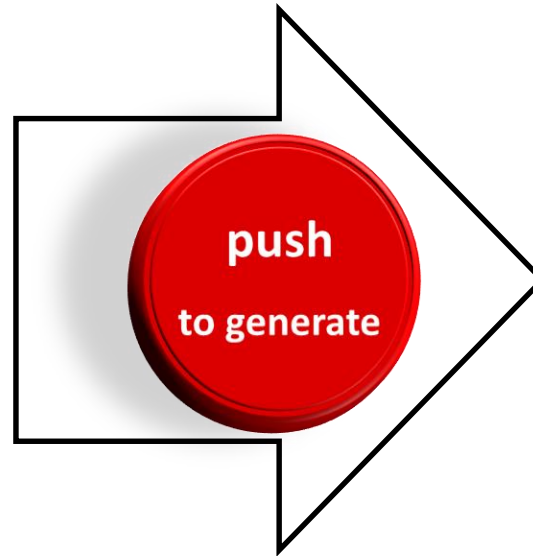


4.2. HARDWARE SOFTWARE CODESIGN

Software / Hardware Co-design

Vision: Custom System on Button Push

**System
design as
high-level
program
code**



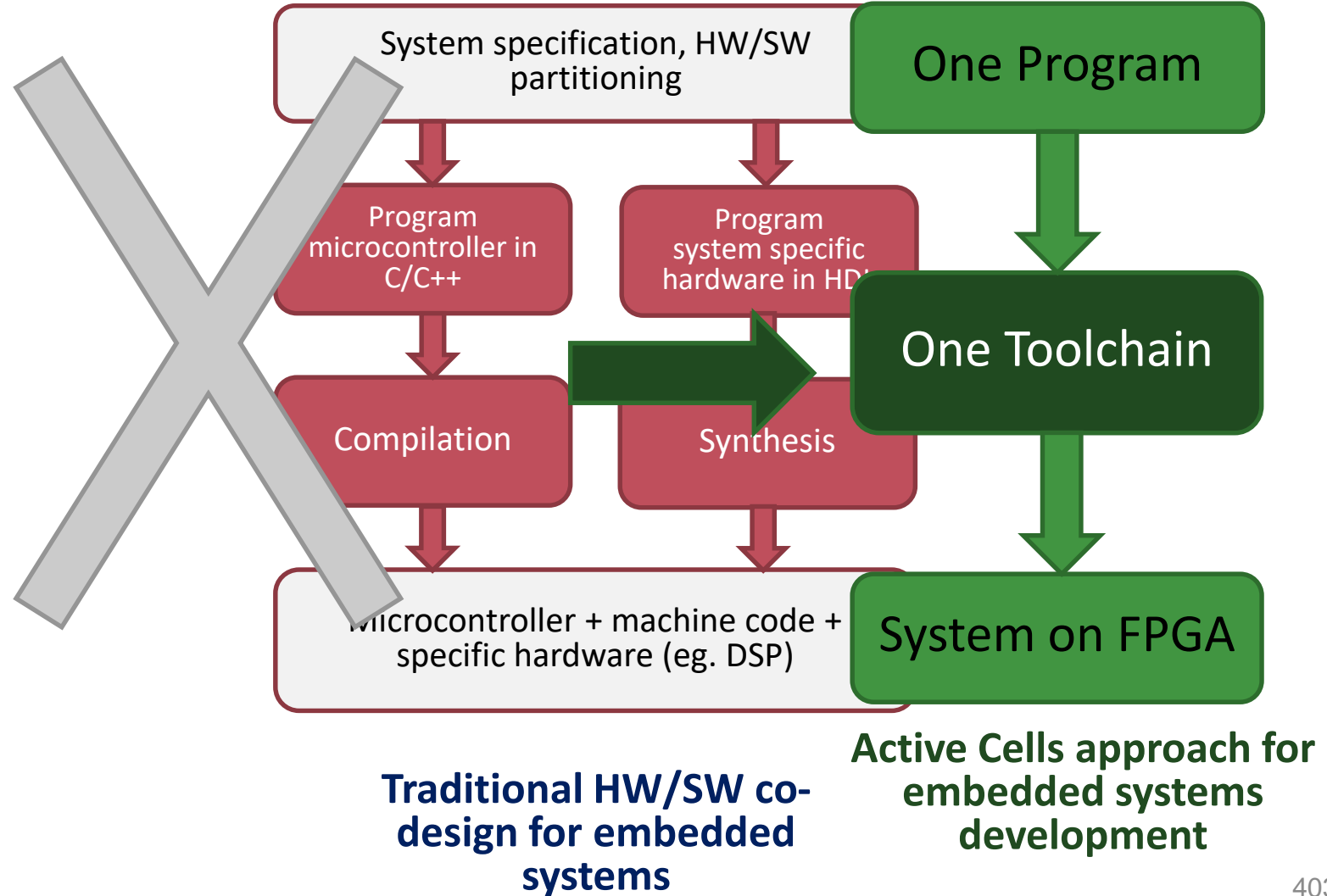
**Electronic
circuits**

Computing model
Programming
Language

Compiler,
Synthesizer,
Hardware Library,
Simulator

Programmable
Hardware
(FPGA)

Traditional HW/SW co-design



Active Cells Computing Model

On-chip distributed system

Cell

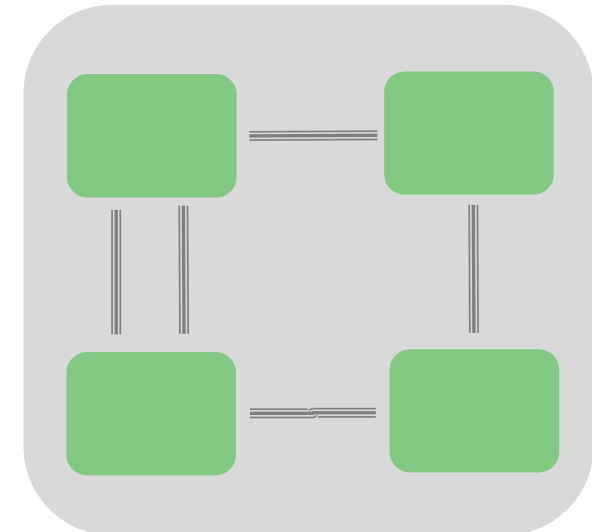
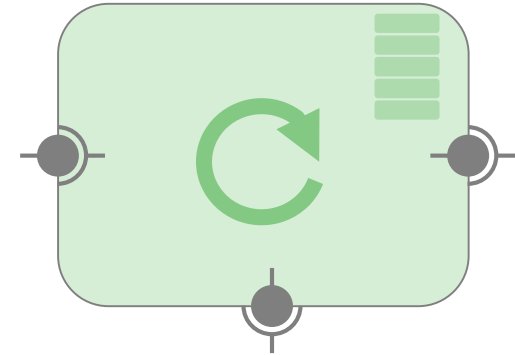
- Scope and environment for a running **isolated process**.
- Integrated **control thread(s)**
- Provides **communication ports**

Net

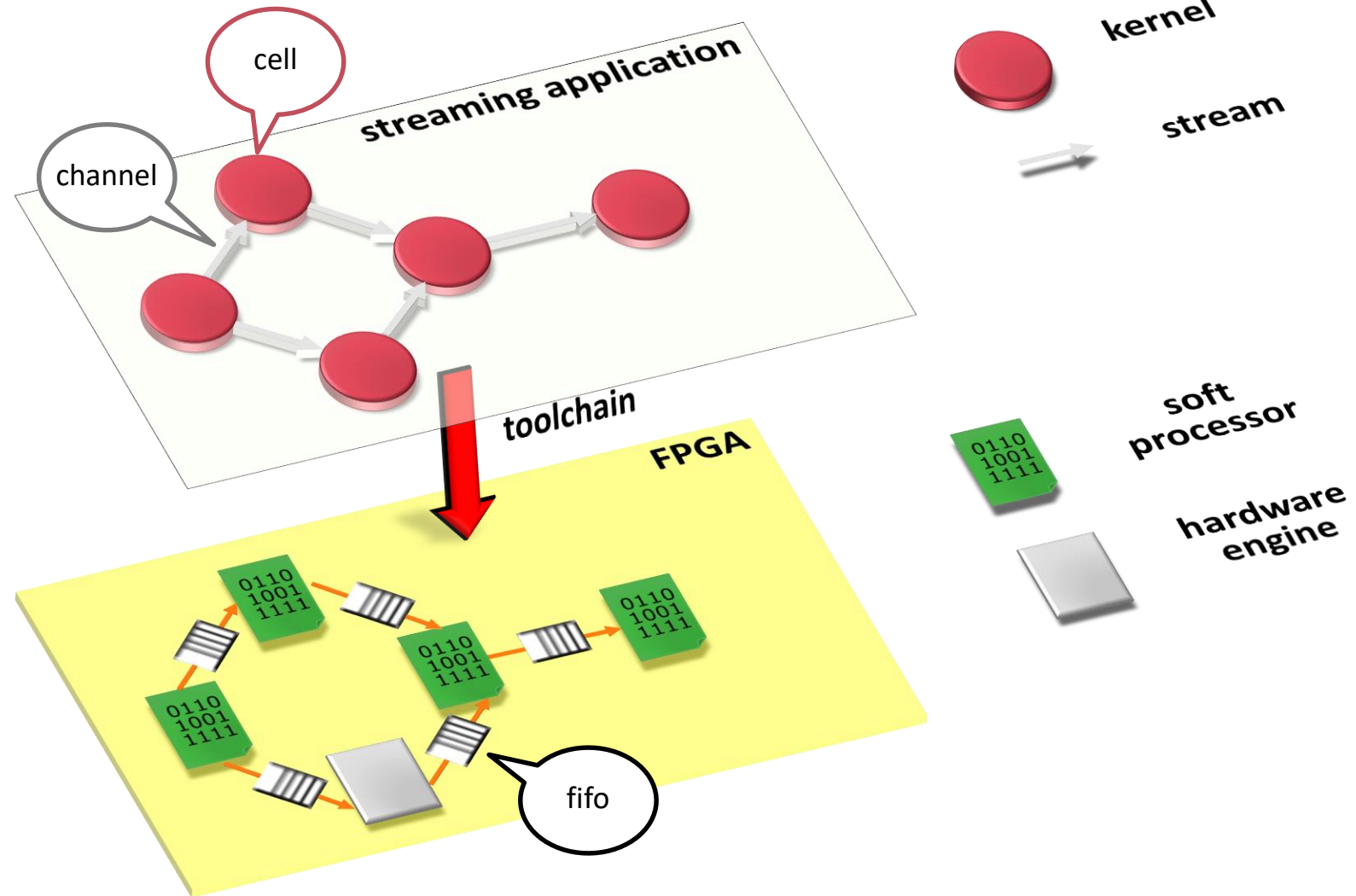
- **Network** of communication cells
- Cells connected via **channels** (FIFOs)

Inspired by

- Kahn Process Networks
- Dataflow Programming
- CSP (e.g. Google's Go)
- Actor Model (e.g. 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

Cell

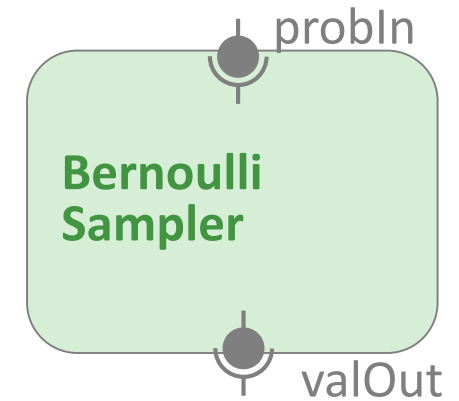
non-typed communication ports

```
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;
```

blocking receive

asynchronous send

Cell Activity



Properties

Properties can influence both, generation of hardware and the generation of software code.

type

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

...

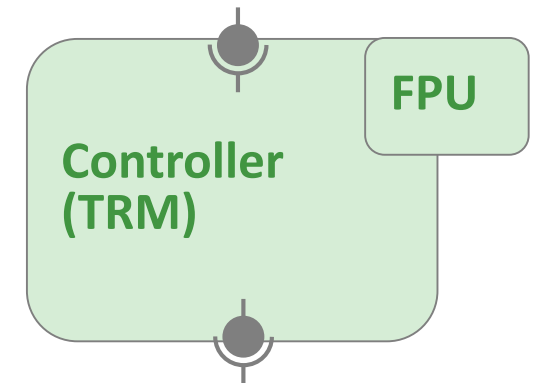
begin

```
(* ... controller action ... *)
```

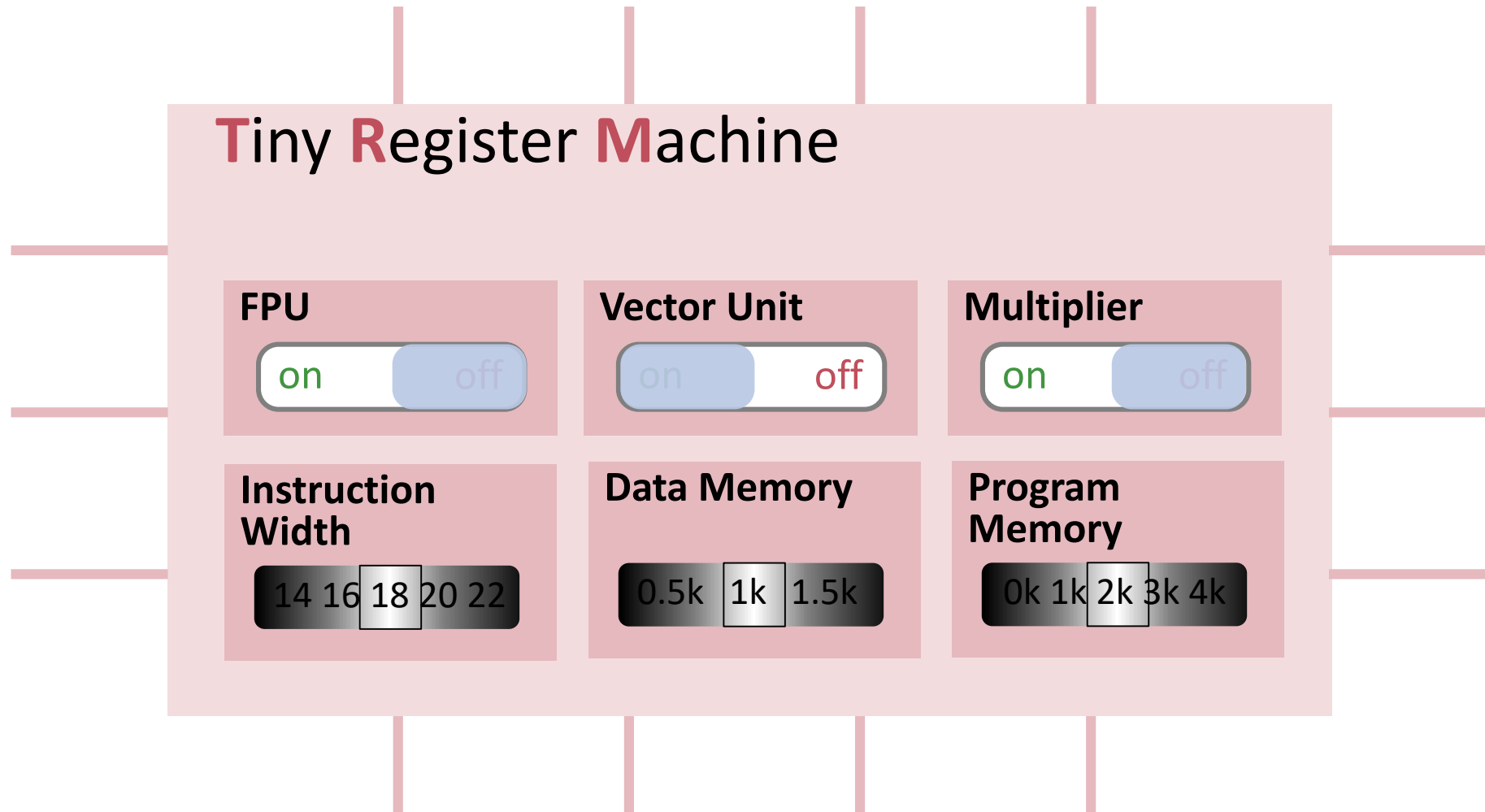
```
end Controller;
```

....

Port Width



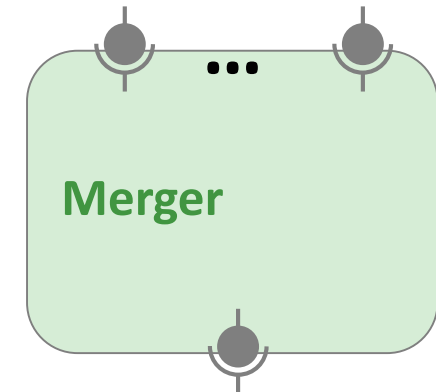
Configurable Processor on PL



Engine

```
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
            data << ind[i];
            outd << data;
        end
    end
end Merger;
```

Engines are prebuilt components instantiated as electronic circuits on a target hardware

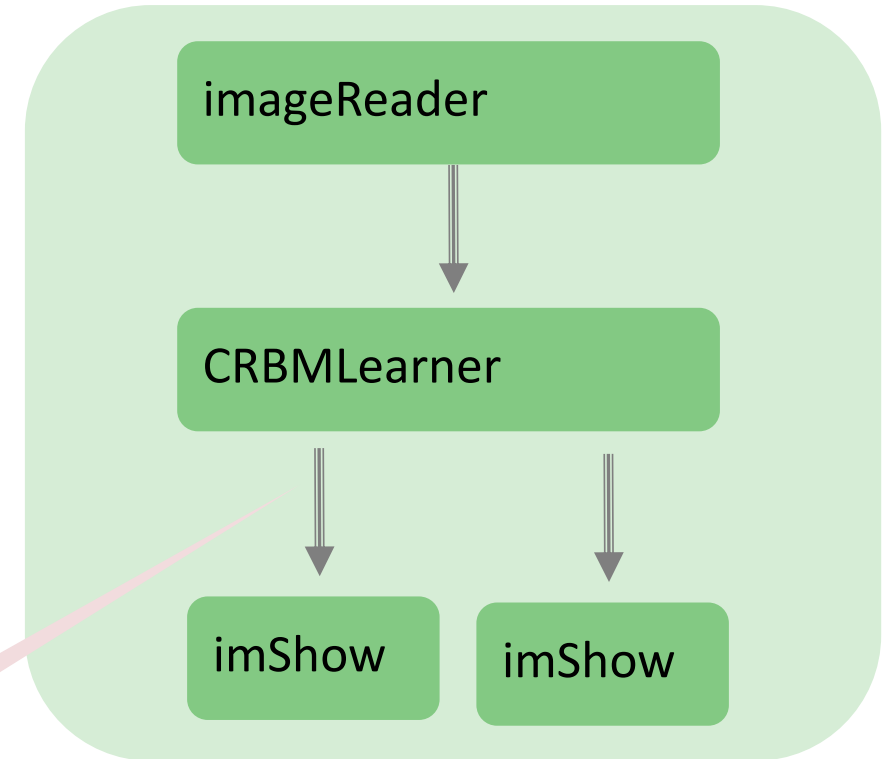


Unit of Deployment: (Terminal) Cellnet

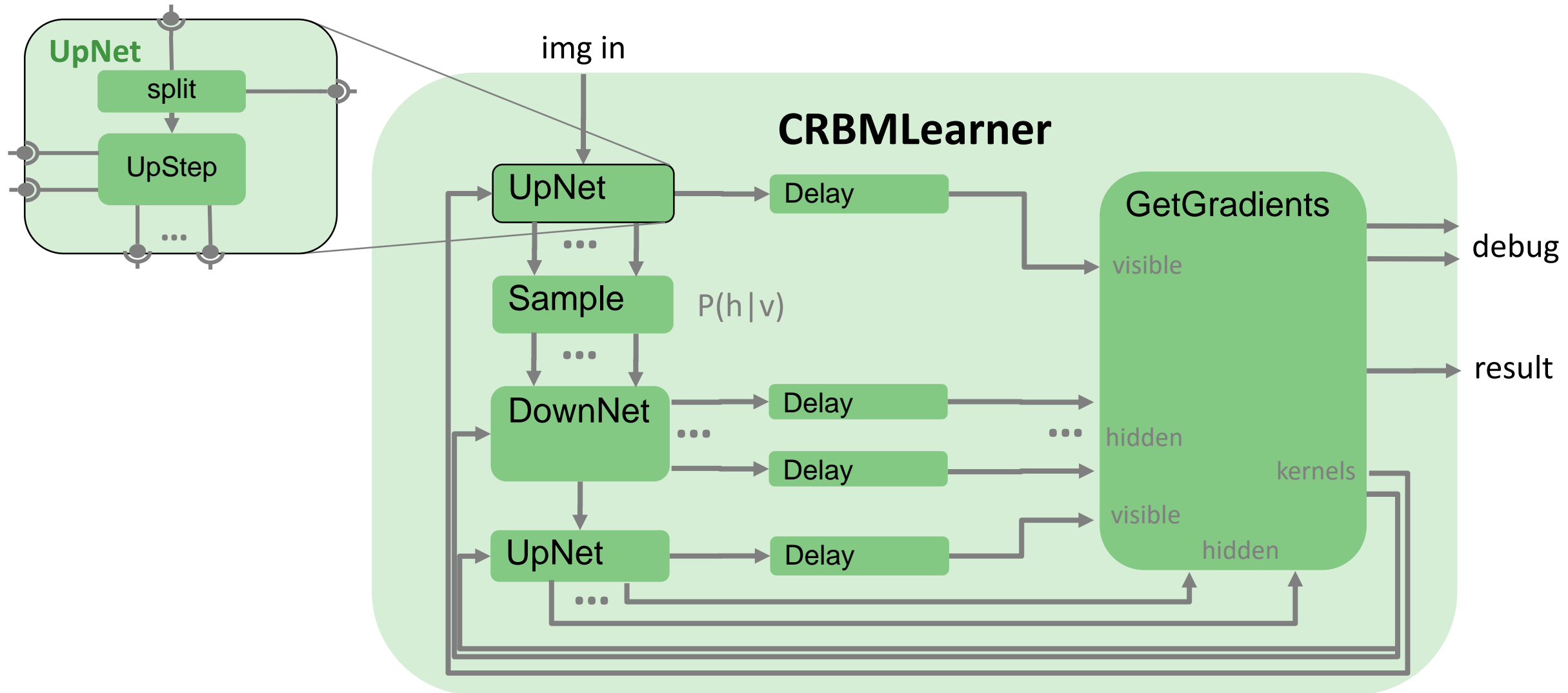
```
LearnTest = cellnet;  
var  
  learner: CRBMNet.CRBMLearner;  
  reader: MLUtil.imageReader;  
  ims0,ims1: MLUtil.imshow;  
  ...  
begin  
  new(learner)  
  new(reader);  
  new(ims0{name='v0debug',posx=0,posy=100});  
  new(ims1{name='v1debug',posx=300,posy=100});  
  ...  
  reader.imageOUT >> learner.imgIN;  
  learner.v0DebugOUT >> ims0.imageIN;  
  learner.v1DebugOUT >> ims1.imageIN;  
  ...  
end LearnTest;
```

dynamic
construction

connection



Hierarchical Composition

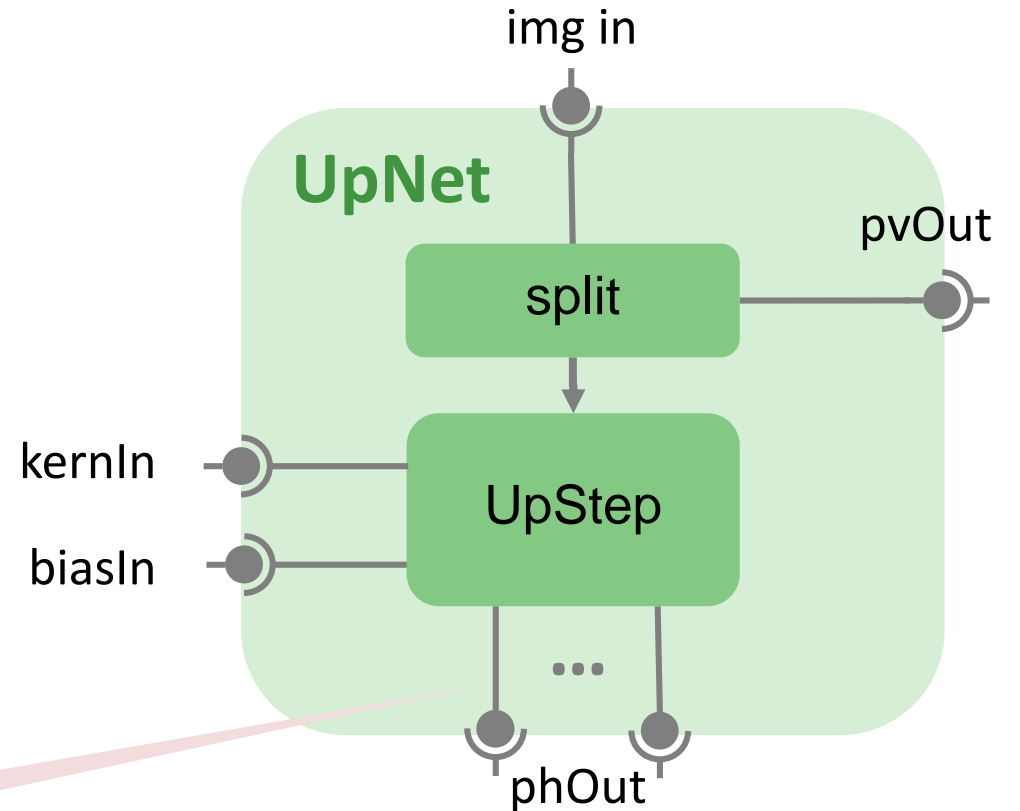


Hierarchic Composition: non-terminal Cellnet

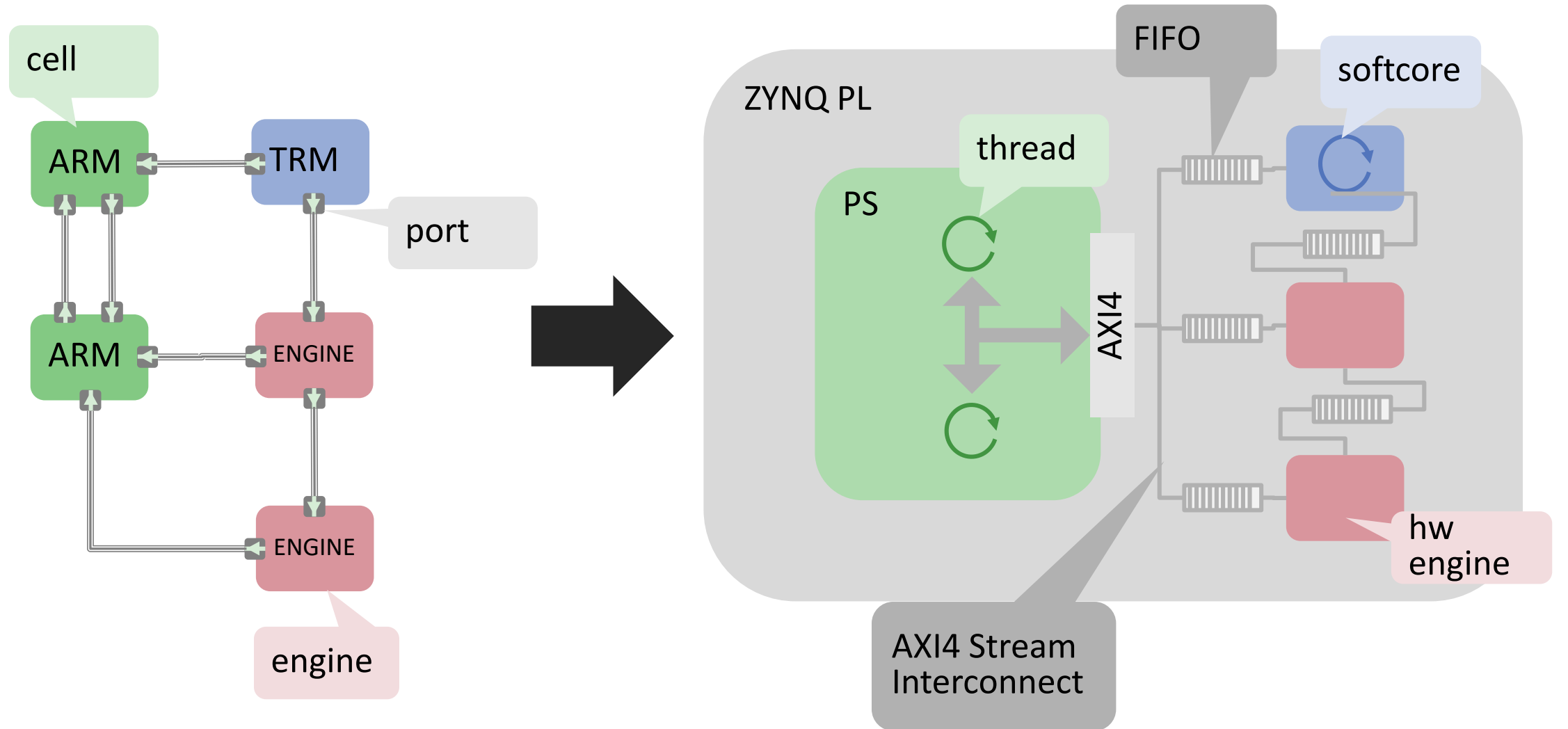
ports and properties

```
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;
  kerIN >> UpstepCell.kerIN;
  bIN >> UpstepCell.bIN;
  for i:=0 to k-1 do
    upstep.phOUT[i] >> phOUT[i];
  end;
end CRBMUpNet;
```

port delegation



Software → Hardware Map

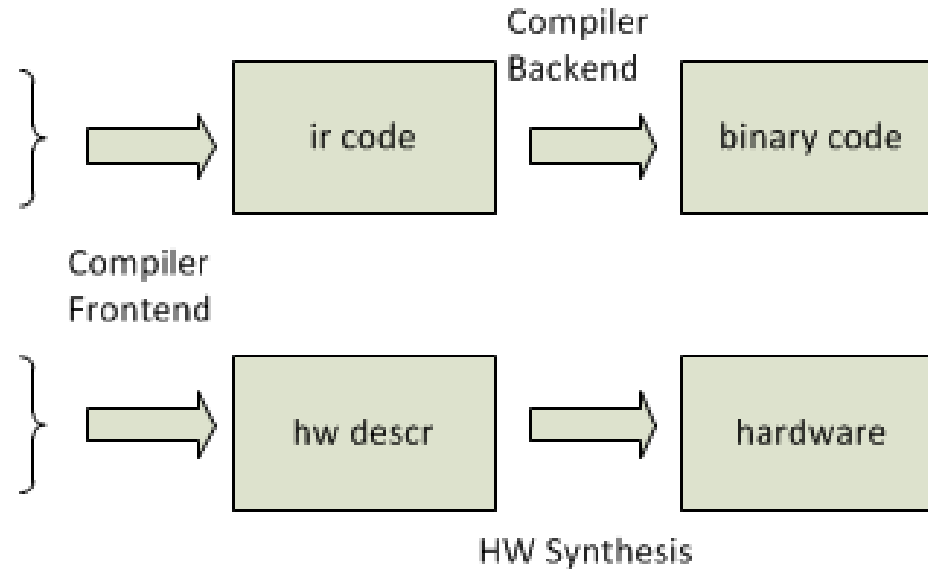


Hybrid Compilation

cellnet N;

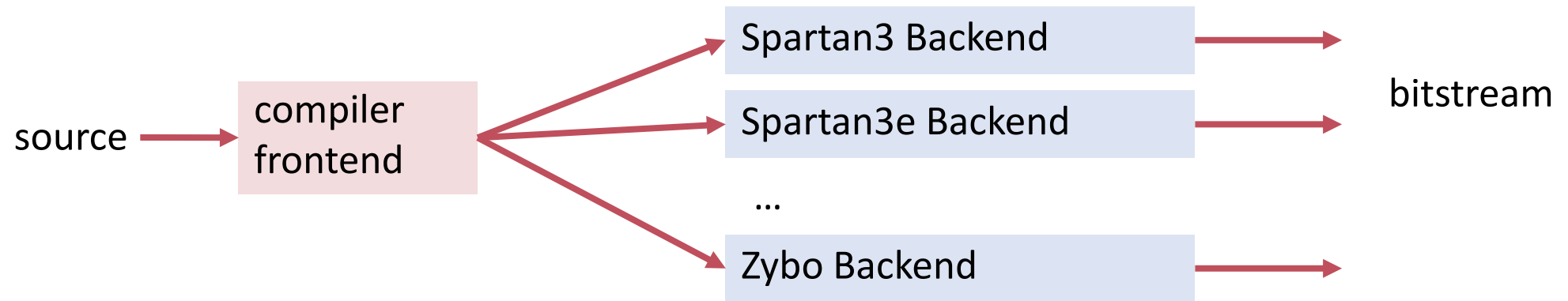
```
type A=cell(pi: port in; po: port out);
var x: integer;
begin
... pi ? x; ... po ! x; ...
end A;
```

```
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

Implementation Alternatives (1)



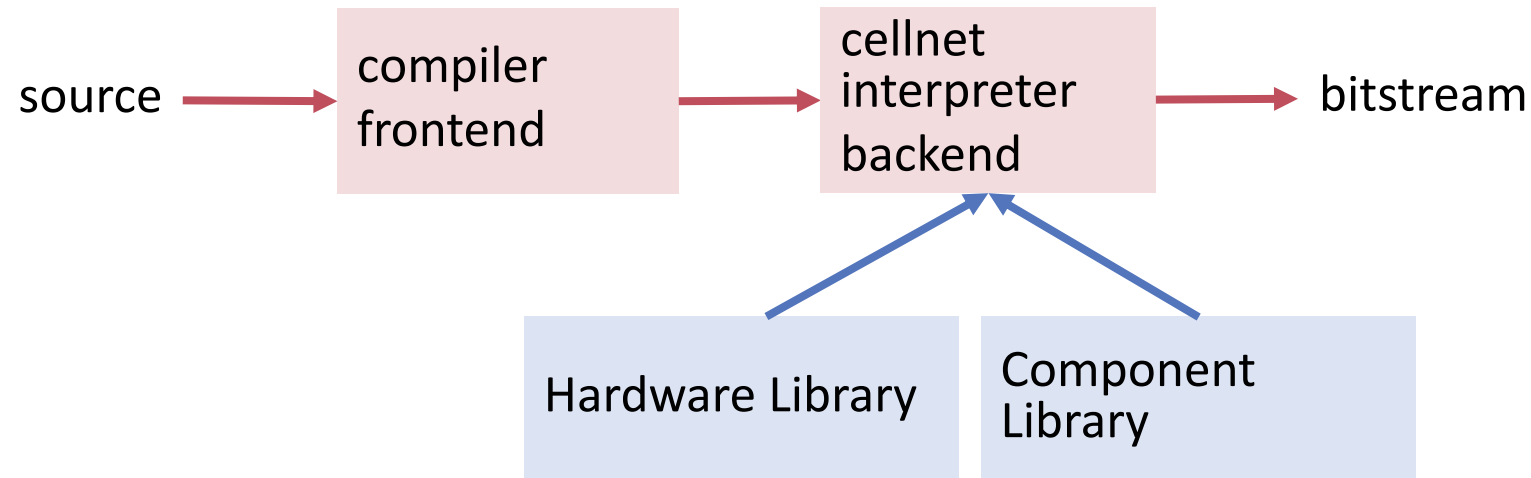
+ simple

- redundant

- not flexible

- hard to extend

Implementation Alternatives (2)



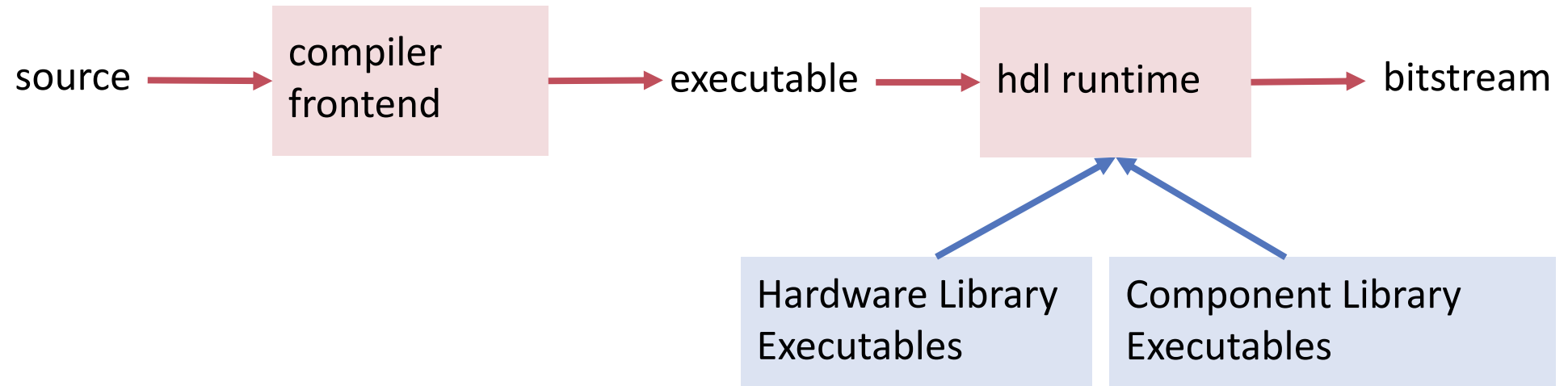
+ extensible

+ not redundant

- not simple

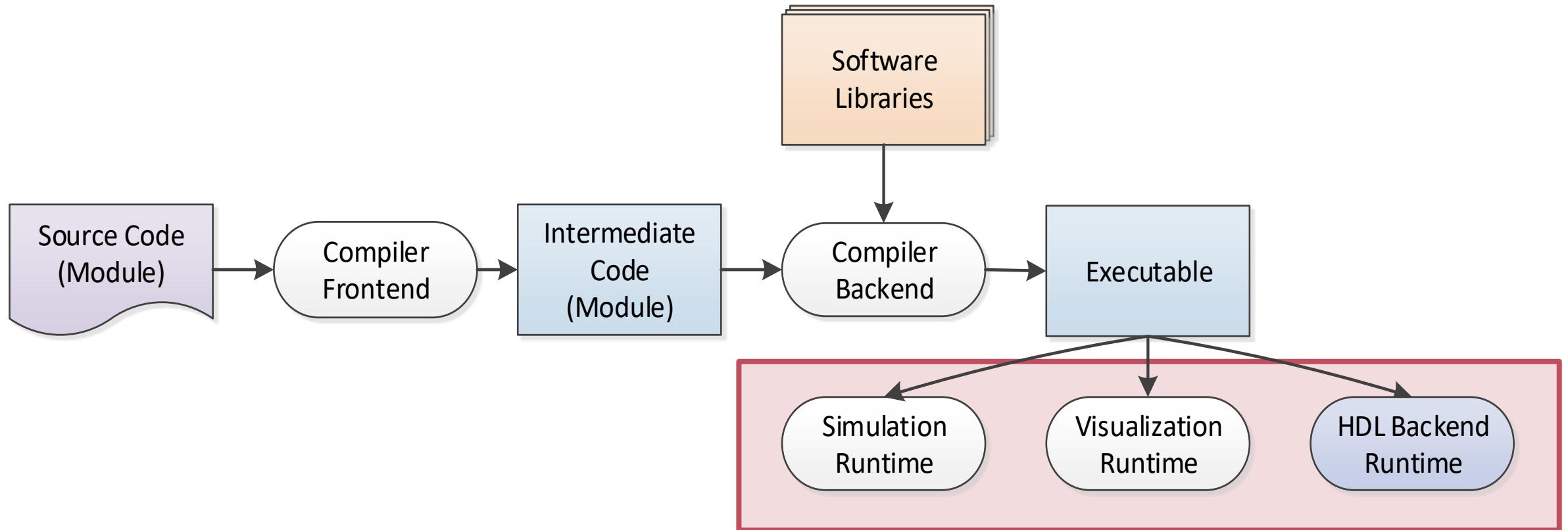
- static configuration limits flexibility

Implementation Alternatives (3)

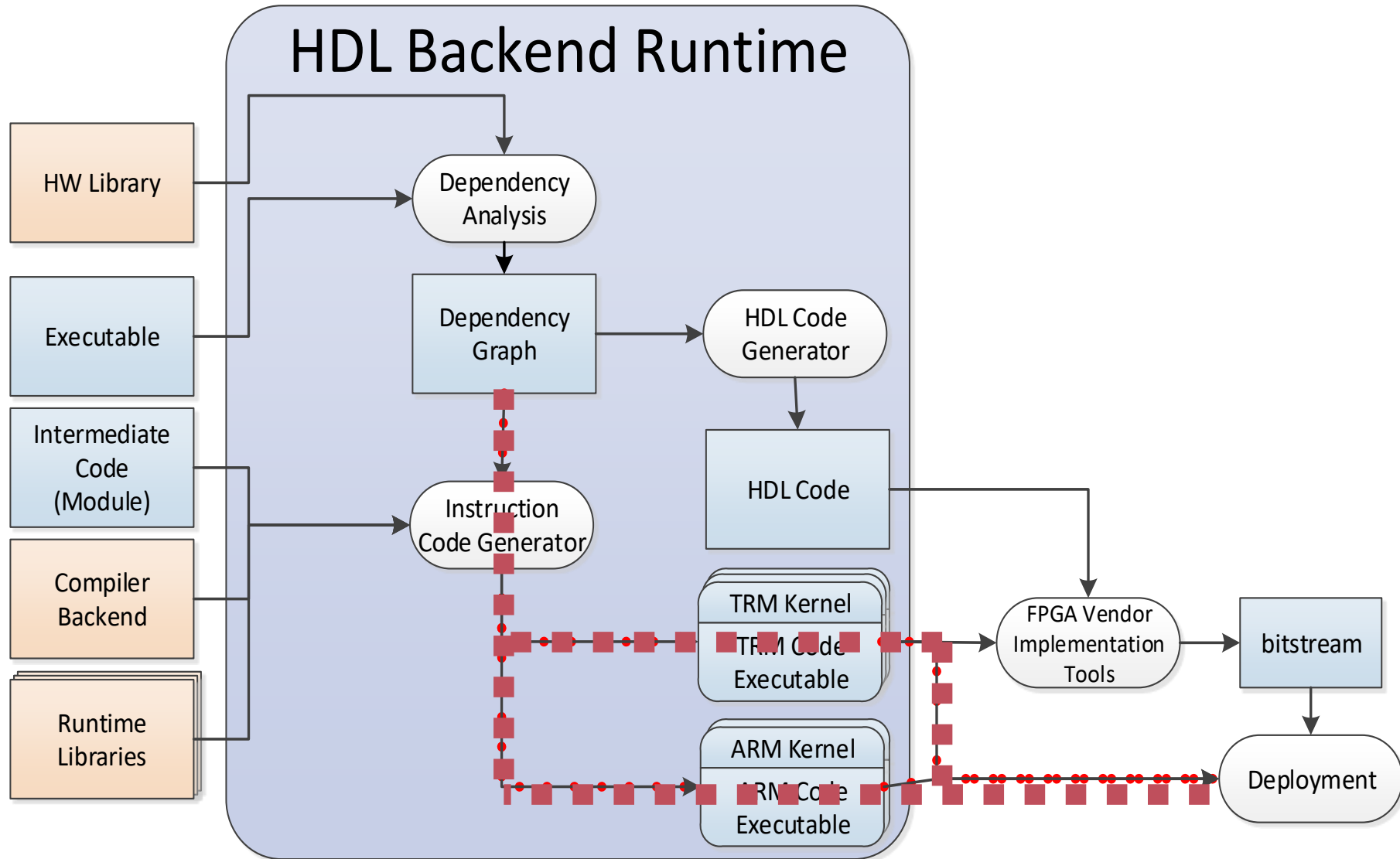


- + extensible
- + not redundant
- + simpler
- + simulation becomes execution

Frontend



Backend



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

Platform Specification

Hardware HDL Code

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

Hardware Platform and Tools

Hardware Types
Platform Instances
Vendor specific tools

Component Specification

```
module Gpo;  
import Hd1 := AcHdlBackend;  
var c: Hd1.Engine;  
begin
```

```
  new(c, "Gpo", "Gpo");  
  c.SetDescription("General Purpose Output ... ");
```

Identification and description

```
  c.SupportedOn("*"); (* portable *)
```

Supported devices

```
  c.NewProperty("DataWidth", "DW", Hd1.NewInteger(32), Hd1.IntegerPropertyRangeCheck(1, Hd1.MaxInteger));  
  c.NewProperty("InitState", "InitState", Hd1.NewBinaryValue("0H"), nil);
```

Parameters

```
  c.SetMainClockInput("aclk"); (* main component's clock *)  
  c.SetMainResetInput("aresetn", Hd1.ActiveLow); (* active-low reset *)  
  c.NewAxisPort("input", "inp", Hd1.In, 8);  
  c.NewExternalHdlPort("gpo", "gpo", Hd1.Out, 8);
```

Ports

```
  c.NewDependency("aclk", true, false);
```

Dependencies

```
  c.AddPostParamSetter("DataWidth", c.NewIntegerFromProperty("inp", "DW"));  
  c.AddPostParamSetter("InitState", c.NewIntegerFromProperty("gpo", "DW"));
```

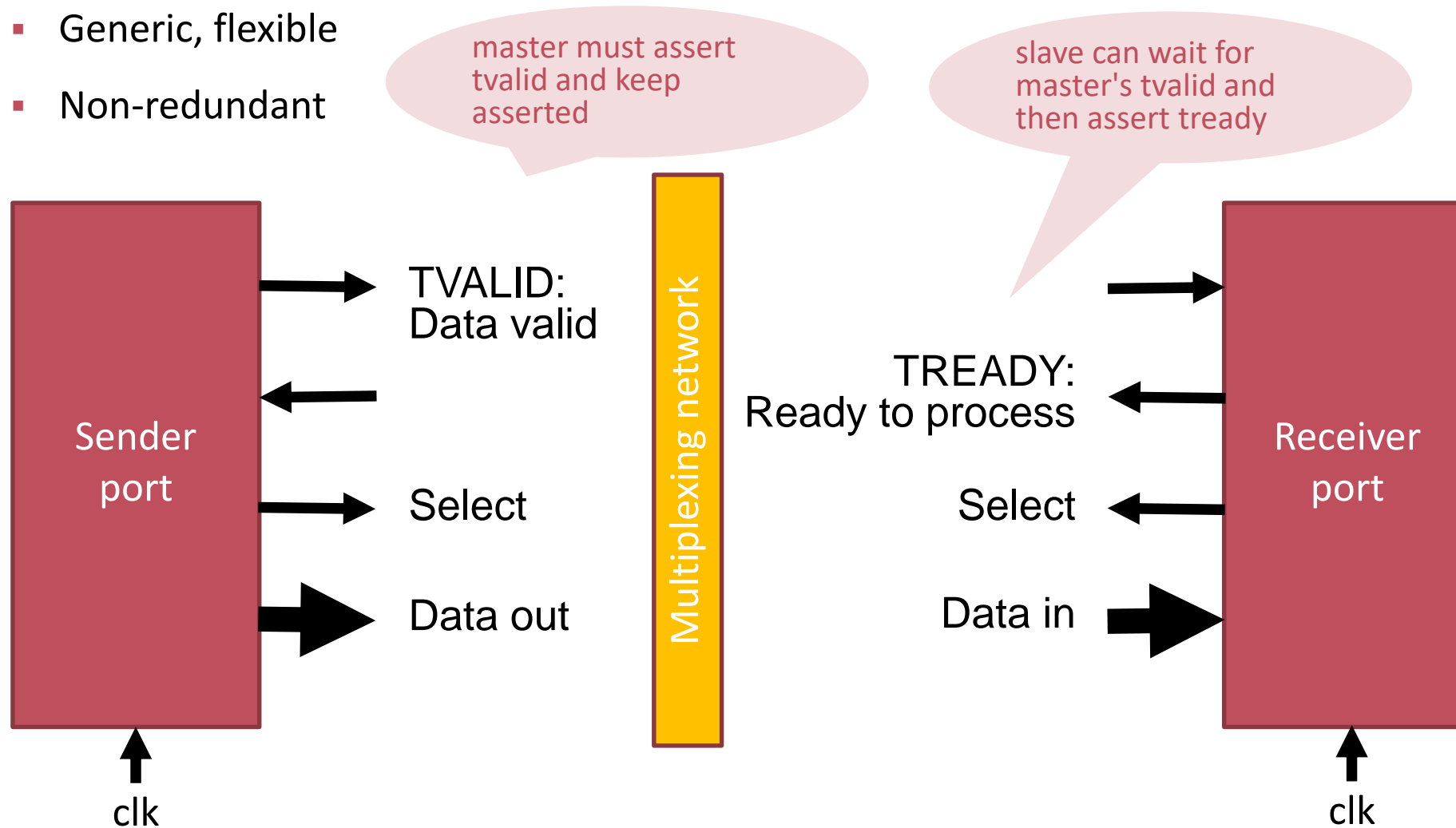
Configuration Actions

```
  Hd1.hwLibrary.AddComponent(c);  
end Gpo.
```

```
c.NewExternalHdlPort("gpo", "gpo", Hd1.Out, 8);
```

Generic Peer-to-Peer Communication Interface

- Use of **AXI4 Stream** interconnect standard from ARM
 - Generic, flexible
 - Non-redundant



Target Platform Specification

```
module Basys2Board;  
import Hd1 := AcHdlBackend, AcXilinx;  
var t: Hd1.TargetDevice;  
    pldPart: AcXilinx.PldPart;  
    ioSetup: Hd1.IoSetup;  
    pin: Hd1.IoPin;  
begin
```

```
new(pldPart,"XC3S100E-4CP132");  
pldPart.SetJtagChainIndex(0);  
new(t,"Basys2Board",pldPart);
```

FPGA Part

```
new(pin,Hd1.In,"B8","LVCMOS33");  
t.NewExternalClock(pin,50000000,50,0); (* ExternalClock0 *)  
t.SetSystemClock(t.clocks.GetClockByName("ExternalClock0"),1,1);  
new(pin,Hd1.In,"G12","LVCMOS33");  
t.SetSystemReset(pin,true);
```

System Signals

```
new(ioSetup,"Gpo_0");  
ioSetup.NewIoPort("gpo",Hd1.Out,"U16,E19,U19,V19","LVCMOS33");  
t.AddIoSetup(ioSetup);
```

Mapping of external Ports

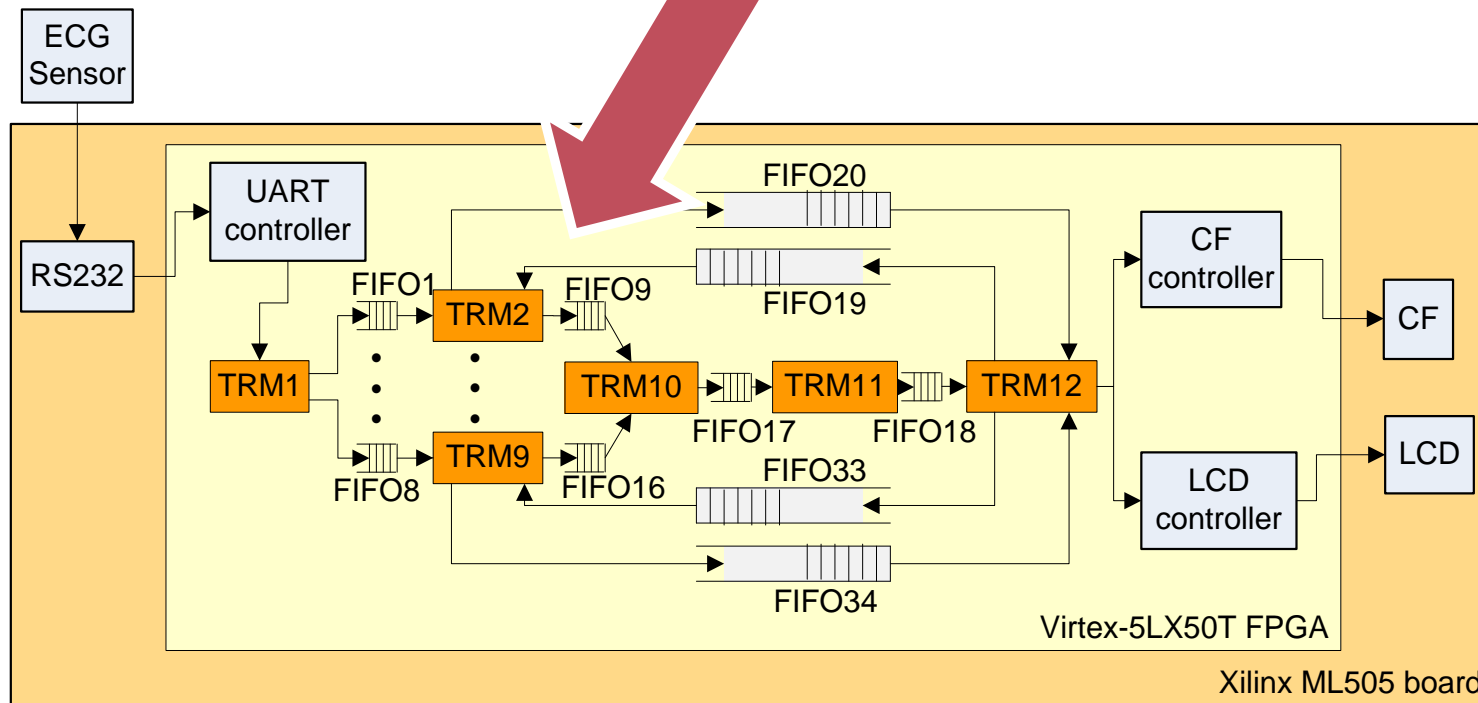
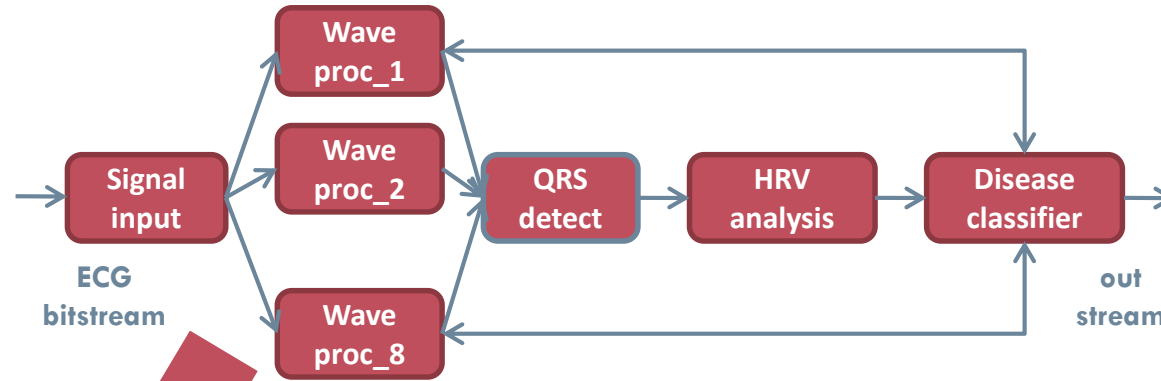
```
Hd1.hwLibrary.Add  
end Basys2Board.
```

```
ioSetup.NewIoPort("gpo",Hd1.Out,"U16,E19,U19,V19","LVCMOS33");
```

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

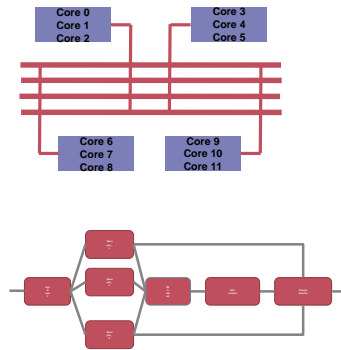
- Maximum number of TRMs in communication chain

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

Comparative Power Usage

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

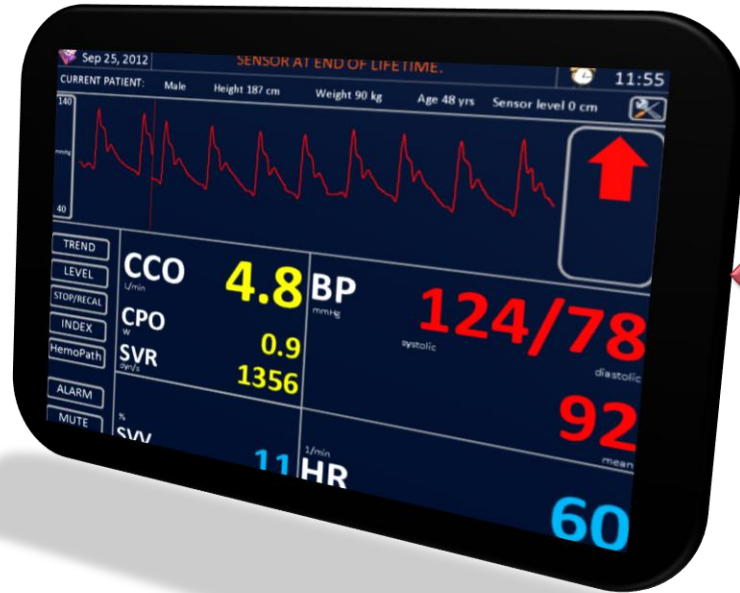


System	Static Power (W)	Dynamic Power (W)
Preconfigured ("TRM12")	3.44	0.59
Dynamically configured	0.5	0.58

86% saving!

Case Study: Non-Invasive Continuous Blood Pressure Monitor

Focus: Development Cycle Time



A2 Host OS with GUI on ARM

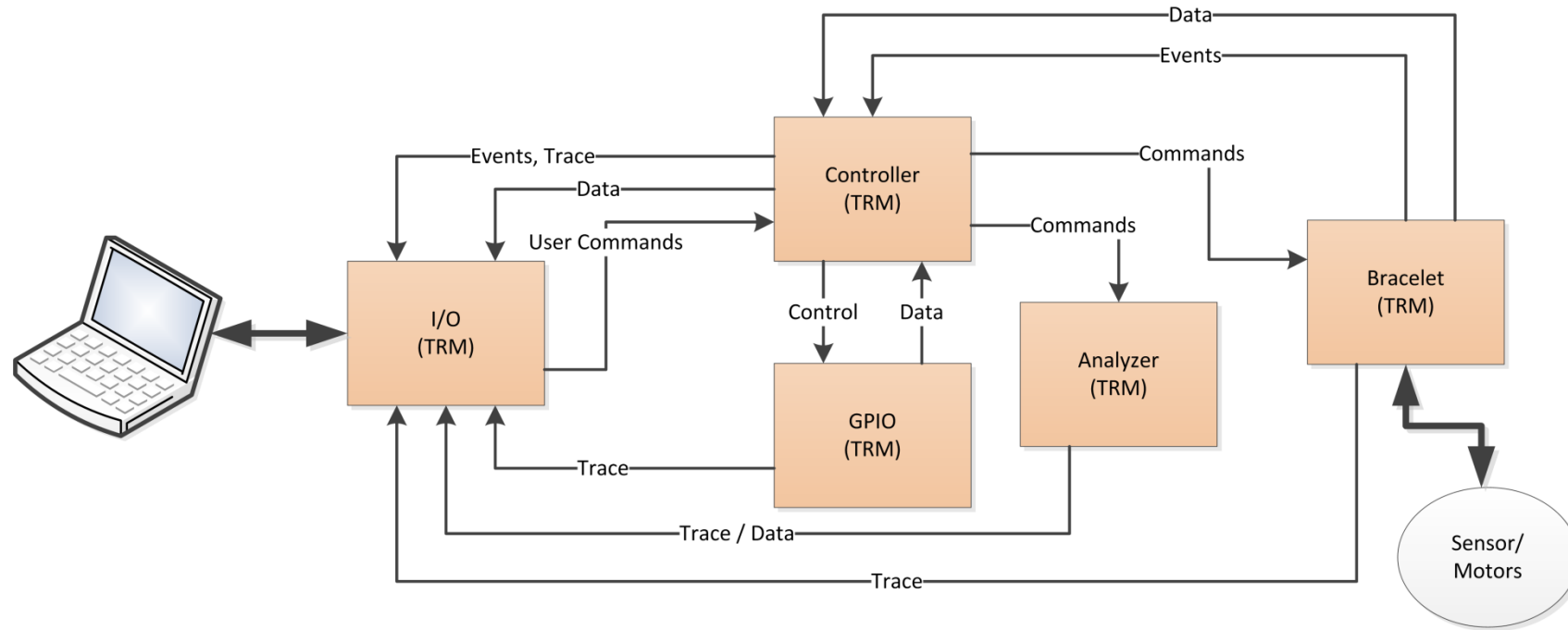


Sensor control and
medical algorithms on
Zynq PL



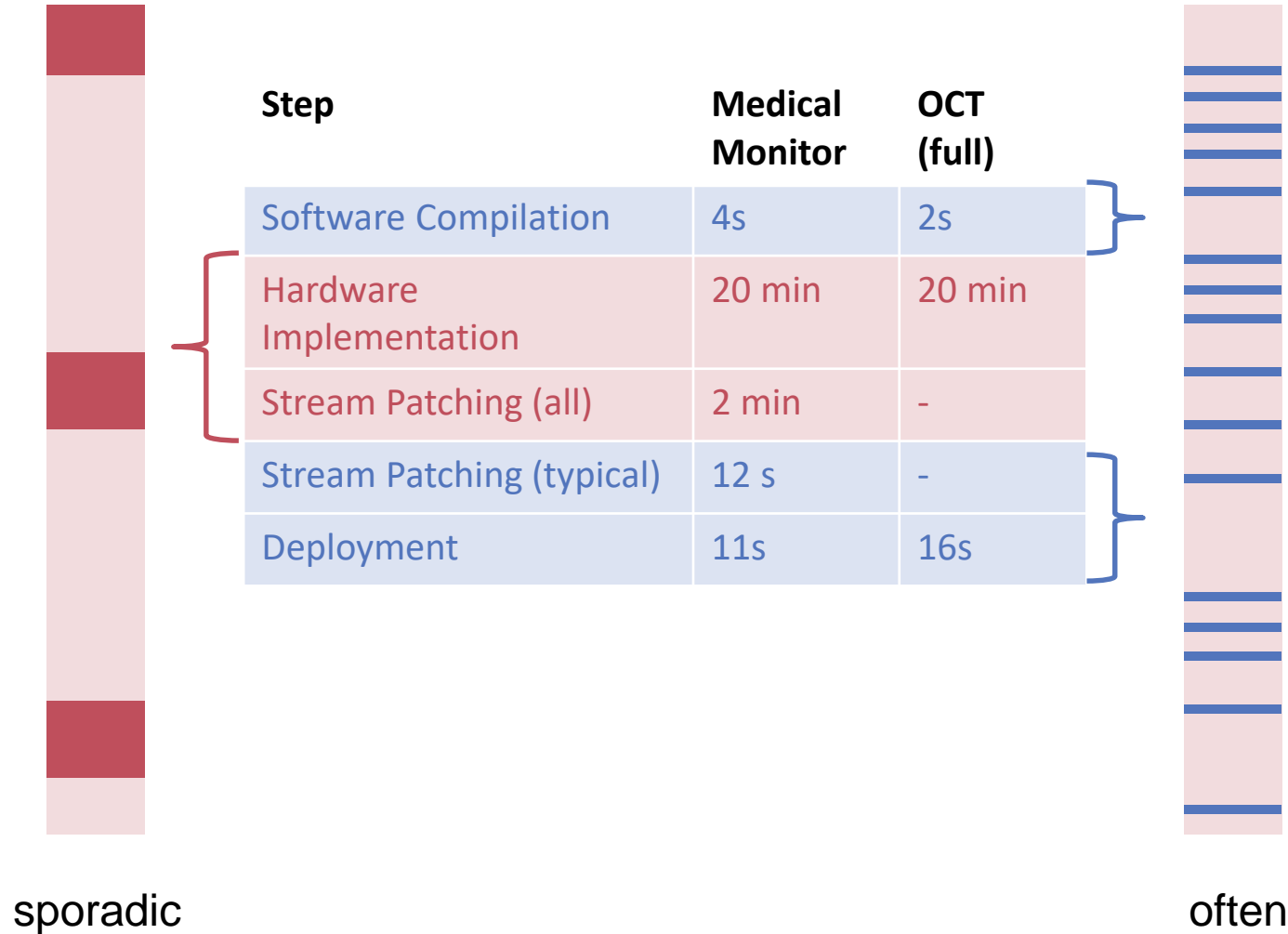
Sensors and Motors
on Bracelet

Medical Monitor Network On Chip



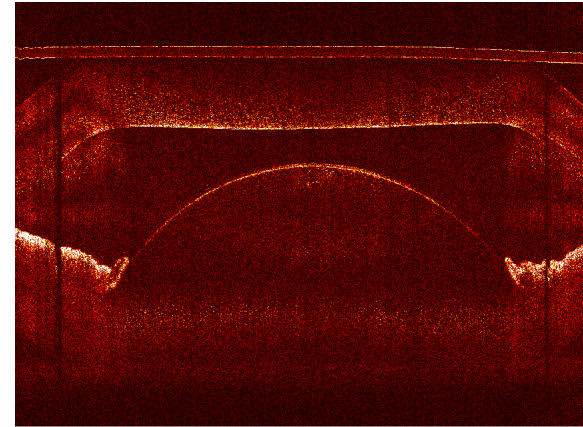
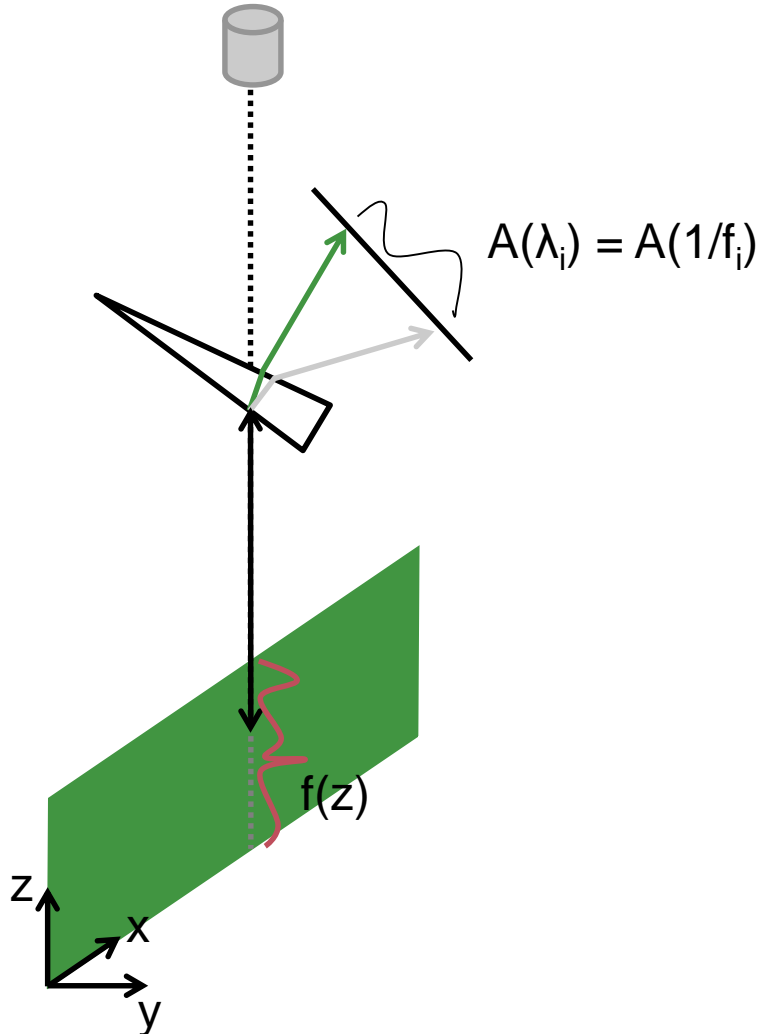
Dominated by TRM processors. Feedback driven. Not performance critical.

Development Cycle Times



Case Study 3: Optical Coherence Tomography

Focus: Performance



z-Axis Processing

1. Non uniform sampling

$$A(\lambda_i) \rightarrow \tilde{A}(f_i)$$

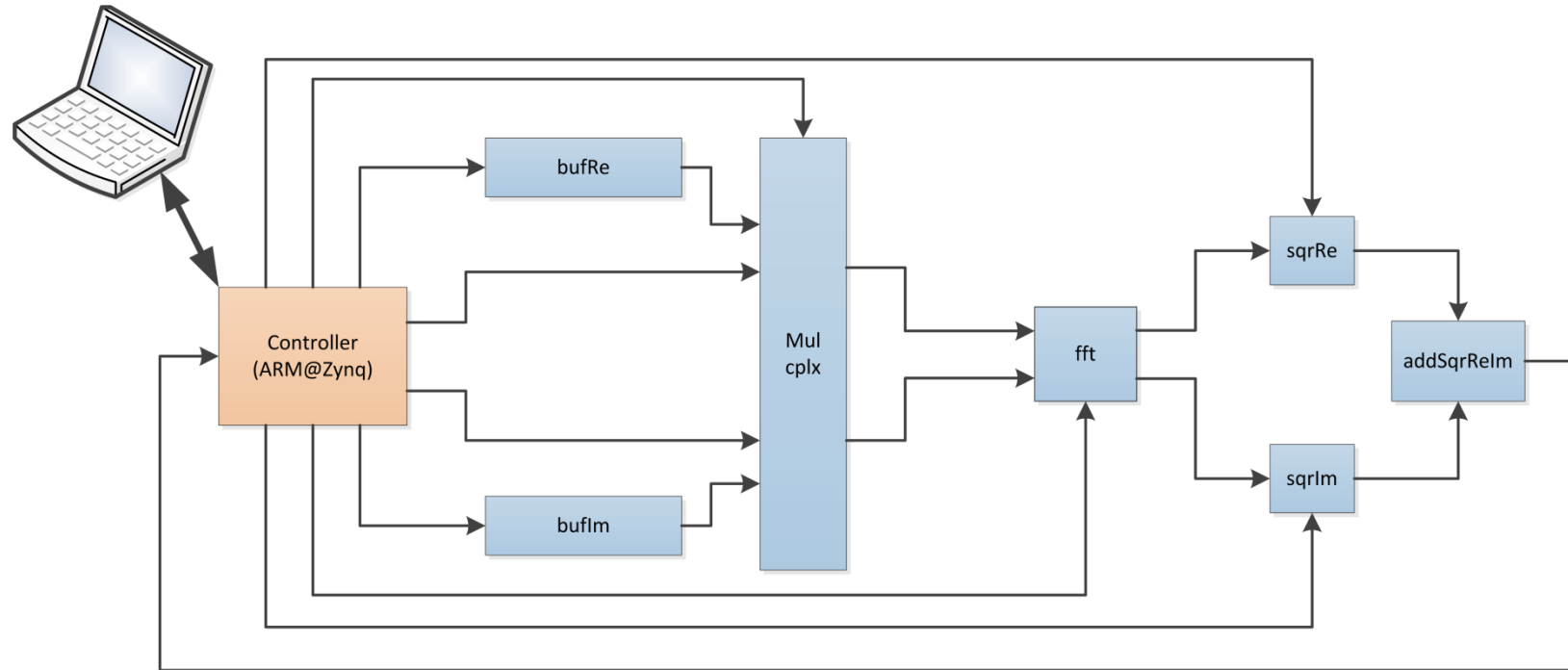
2. Dispersion compensation
3. (Inverse) FFT

... for many lines x in a row (2d)

... and many rows y in a column (3d)

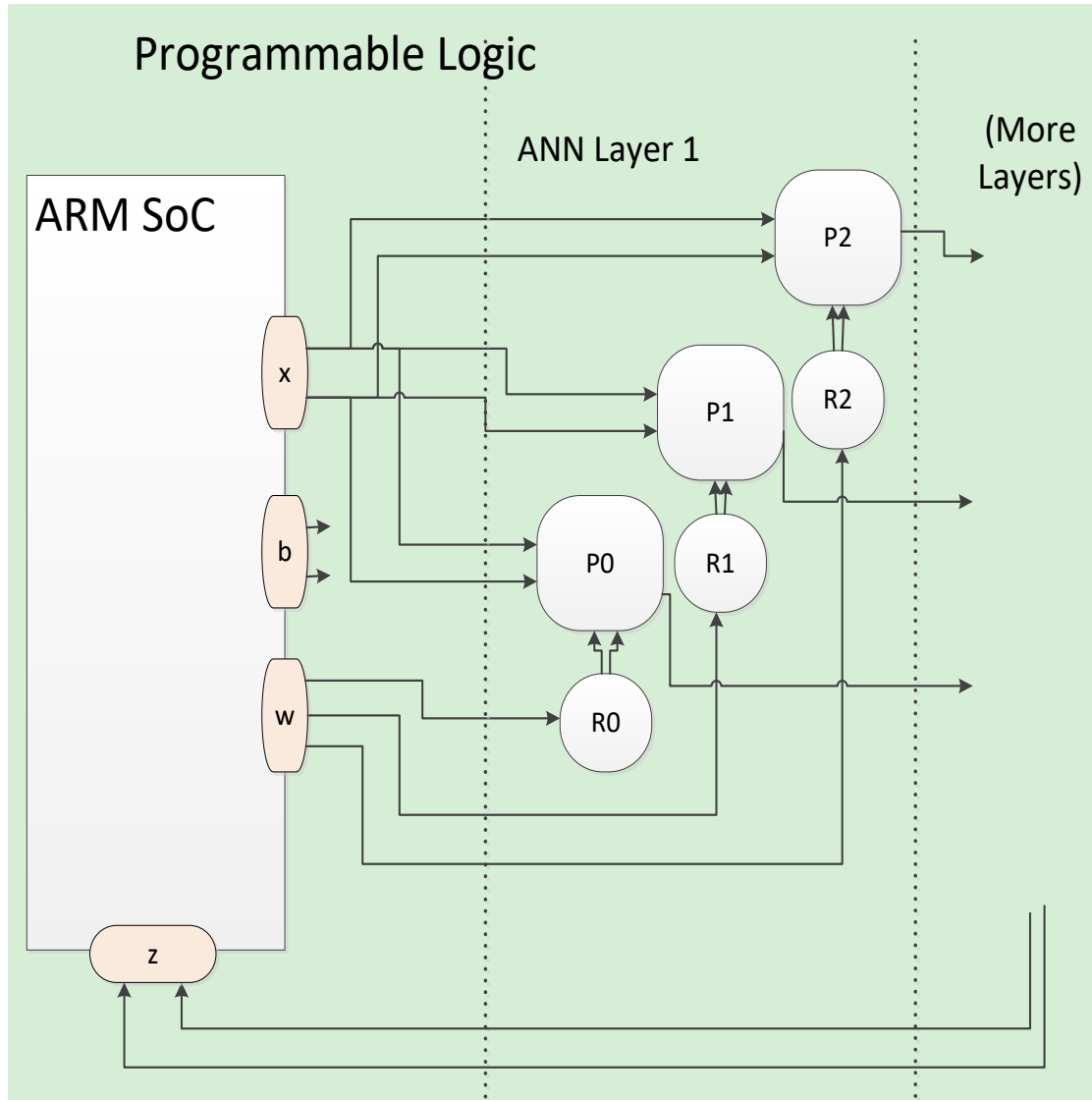
A component of OCT image processing

Dispersion Compensation

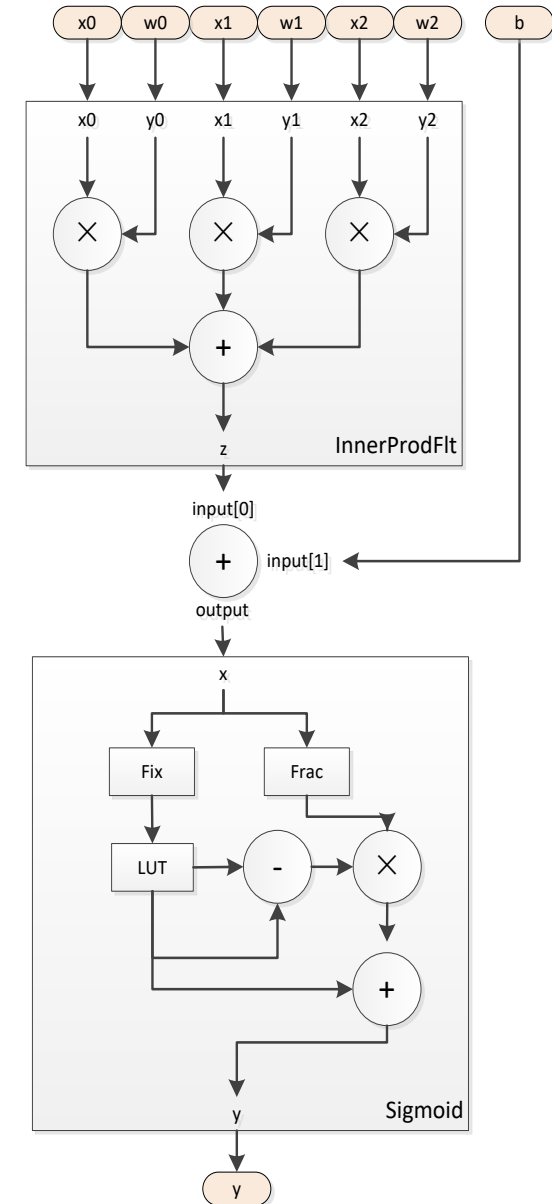


Dominated by Engines. Dataflow driven.

Case Study: ANN



Perceptron



Performance and Resource Usage

	Medical Monitor	Simple OCT	OCT	Perceptron
Architecture	Spartan 6 XC6SLX75	Zynq 7000 XC7Z020	Zynq 7000 XC7Z020	Zynq 7000 XC7Z010
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	83% Slice LUTs, 67% Slice Registers 13.33% BRAMs, 21% DSP 1 ARM Cortex A9
Clock Rate	58 MHz	118 MHz	50 MHz	147 MHz
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)	9.6 GBits/s in 9.6 GBits/s out
Performance	--	8.3 GFPOps* up to 32 GFPops**	4.3 GFPOps*	4.9 GFlops
Power	~2W	~5W	~5W	2.1 W

** 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

Mapping to Hardware – Simple Example

```
IO*=CELL (input: PORT IN; output: PORT OUT;  
          buttons: PORT IN; leds: PORT OUT);  
BEGIN ...  
END IO;
```

```
Controller*=CELL (input: PORT IN; output: PORT OUT)  
BEGIN ...  
END Controller;
```

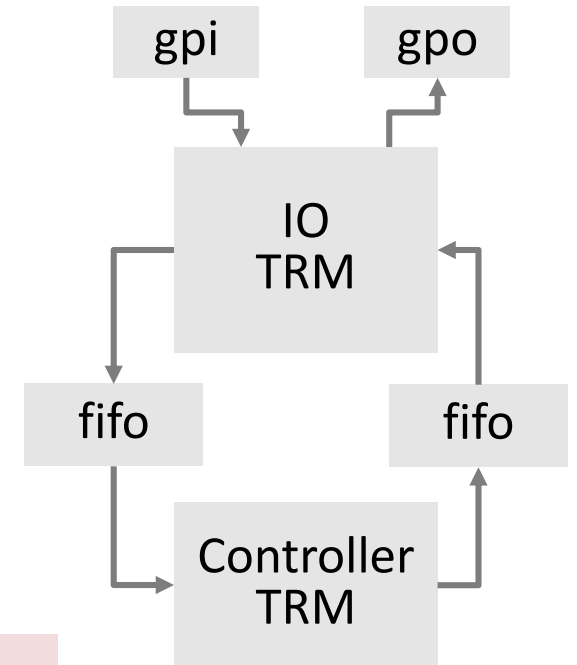
```
VAR controller: Controller; io: IO;  
    gpo{DataWidth=8}: Engines.Gpo;  
    gpi{DataWidth=11}: Engines.Gpi;
```

BEGIN

```
NEW(controller); NEW(io);  
NEW(gpi); NEW(gpo);
```

```
CONNECT (controller.output, io.input, 32);  
CONNECT (io.output, controller.input, 32);  
CONNECT (gpi.output, io.buttons);  
CONNECT (io.leds, gpo.input);
```

END Simple.



→ Blackboard / Code inspection

Spartan3 Board



Resources

4-input LUTs 3840

Flip Flops 3840

BRAMs 12

DSPs -

Resource Usage Scenarios

1 TRM

```
CELLNET Game;

IMPORT LED;

TYPE

IO*=CELL {DataMemorySize(512),
CodeMemorySize(1024), LED} (in: PORT
IN);
BEGIN
END IO;

VAR io: IO;
BEGIN
  NEW(io);
END Game.
```

TRM → TRM

```
VAR io: IO; trm: TRM;
BEGIN
NEW(io); NEW(trm);
  CONNECT(trm.out, io.in);
END Game.
```

TRM ← → TRM → TRM

(cf. next page)

Resources

4-input LUTs	27%
Flip Flops	5%
BRAMs	16%

Resources

4-input LUTs	58%
Flip Flops	9%
BRAMs	33%

Resources

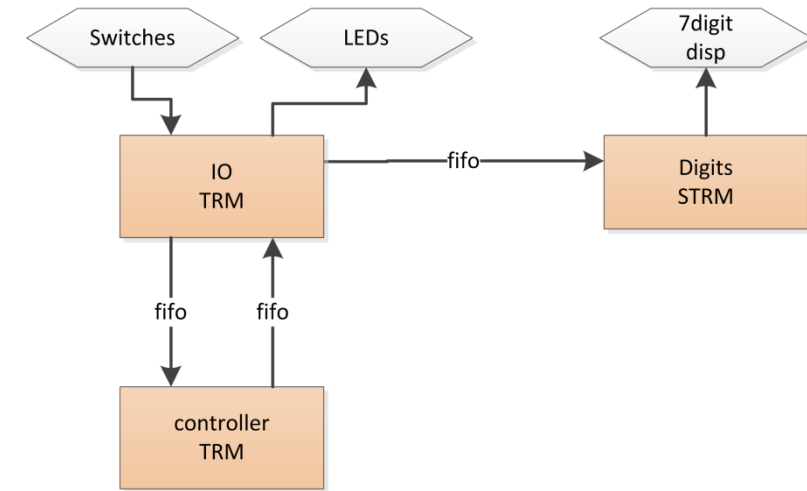
4-input LUTs	86%
Flip Flops	18%
BRAMs	75%

➔ TRM ≈ 21-27% (LUTs), 16% (BRAMs); Fifo ≈ 6% (LUTs)

Resource Usage (Lab)

```
controller: Controller;
ioTRM: IO;
digitsDriver: DigitsDriver;
digits: Engines.LEDDigits;
gpo{DataWidth=8}: Engines.Gpo;
gpi{DataWidth=11}: Engines.Gpi;

BEGIN
NEW(controller);
NEW(ioTRM);
NEW(digitsDriver);
NEW(digits);NEW(gpi);NEW(gpo);
CONNECT( controller.cmdOUT,ioTRM.cmdIN,10);
CONNECT( ioTRM.cmdOUT,controller.cmdIN,10);
CONNECT( gpi.output, ioTRM.ButtonsIN);
CONNECT( ioTRM.LEDOUT, gpo.input);
CONNECT(ioTRM.digitsOUT, digitsDriver.cmdIN,10);
CONNECT(digitsDriver.ledOUT, digits.input);
```



Resources

4-input LUTS	86%
Flip Flops	18%
BRAMs	75%

- 3 TRMs
- 3 FIFOs