ACTIVE CELLS: EXTENSIBLE HARDWARE

Problems with the first Approach

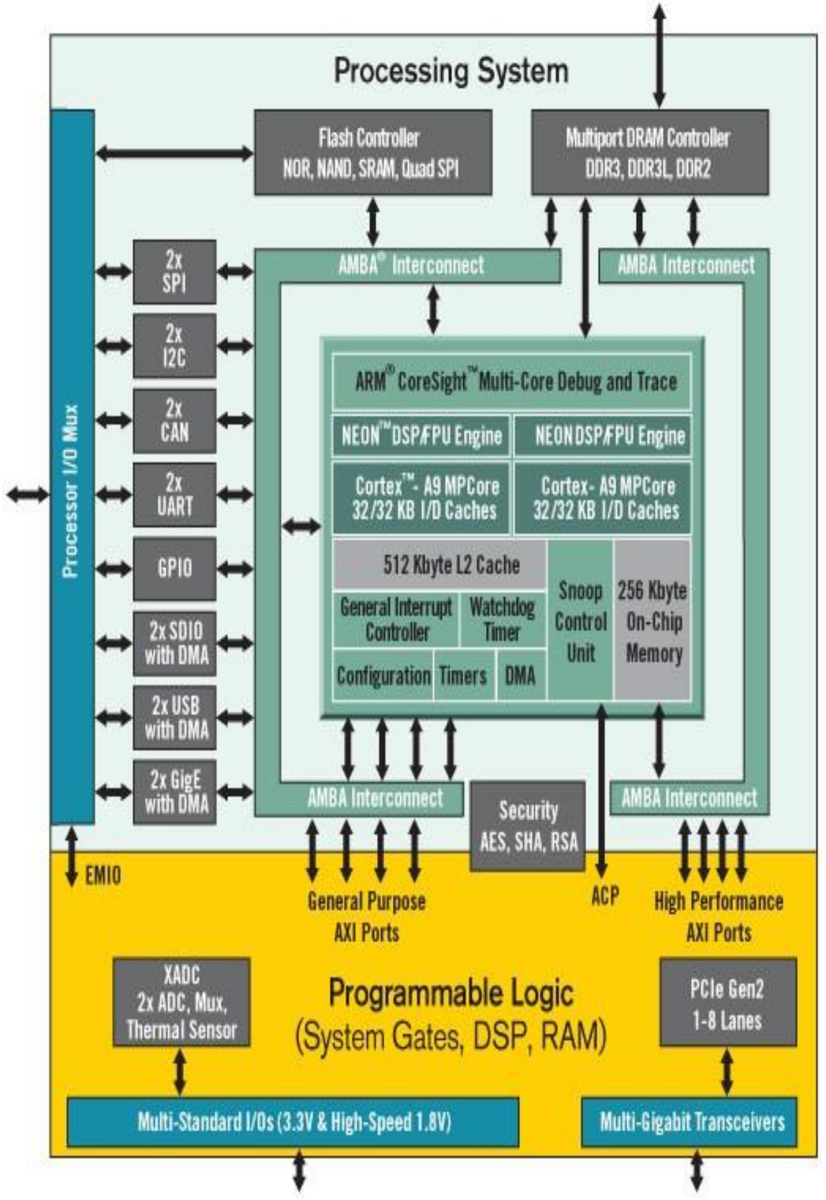
```
cellnet Example;
import RS232;
type
    UserInterface = cell {RS232} (out1, out2: port out; in:
    port in)
    (*...*) end UserInterface;

    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.
```

what, if several UART components shall be made available?

How to extend the hardware without rewriting (parts of) the compiler each time?

Problems with the first Approach

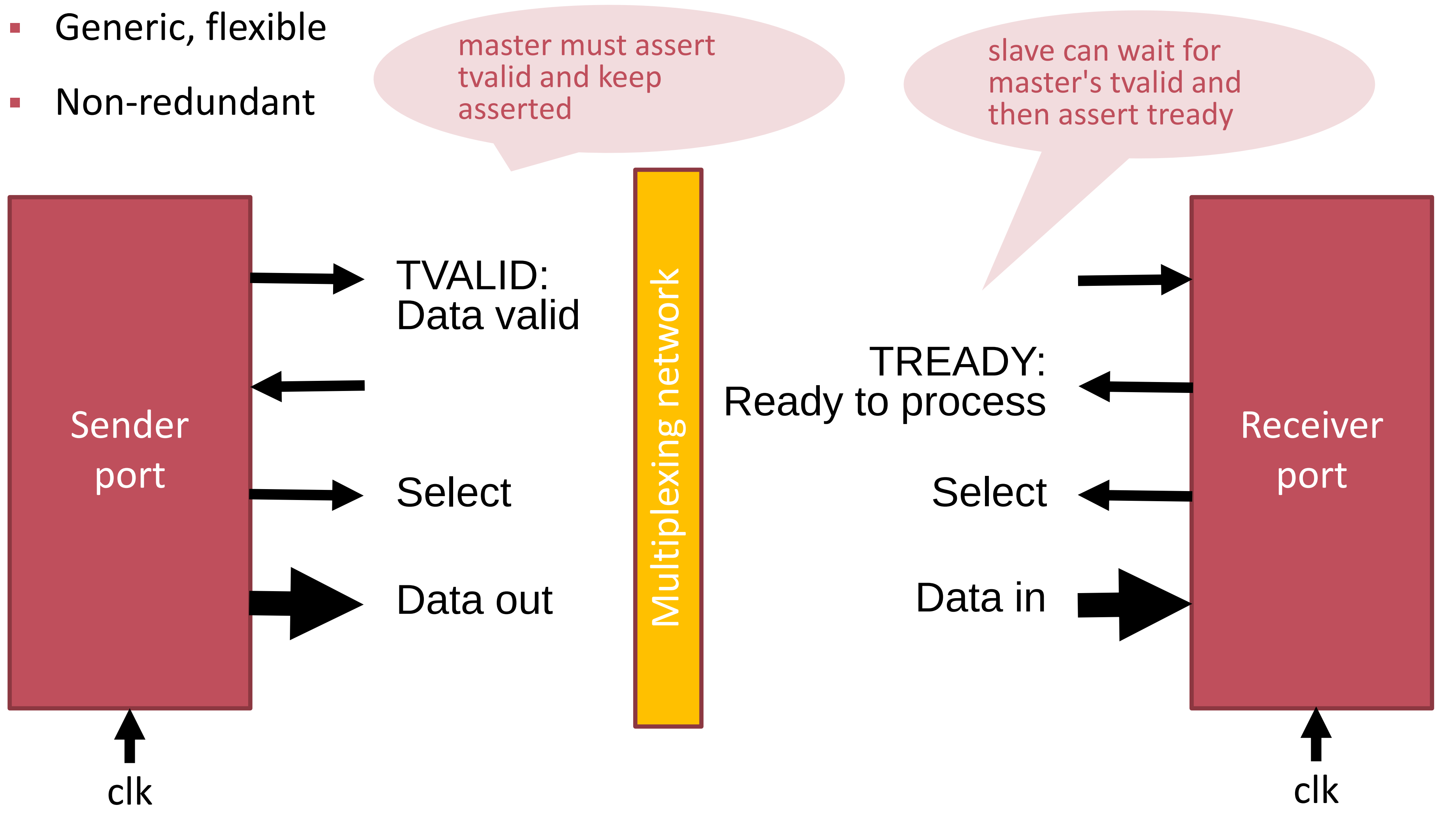
- How to support Xilinx Zynq SOC (Dual ARM + FPGA logic) platform?



Processing System
Flash Controller NOR, NAND, SRAM, Quad SPI
Multiport DRAM Controller DDR3, DDR3L, DDR2
Processor I/O Mux
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
AMBA® Interconnect
AMBA Interconnect
ARM® CoreSight™ Multi-Core Debug and Trace
NEON™ DSP/FPU Engine
NEON DSP/FPU Engine
Cortex™- A9 MPCore 32/32 KB I/D Caches
Cortex- A9 MPCore 32/32 KB I/D Caches
512 Kbyte L2 Cache
General Interrupt Controller
Watchdog Timer
Configuration
Timers
DMA
Snoop Control Unit
256 Kbyte On-Chip Memory
AMBA Interconnect
Security AES, SHA, RSA
AMBA Interconnect
EMIO
General Purpose AXI Ports
ACP
High Performance AXI Ports
XADC 2x ADC, Mux, Thermal Sensor
Programmable Logic (System Gates, DSP, RAM)
PCIe Gen2 1-8 Lanes
Multi-Standard I/Os (3.3V & High-Speed 1.8V)
Multi-Gigabit Transceivers

Generic Peer-to-Peer Communication Interface

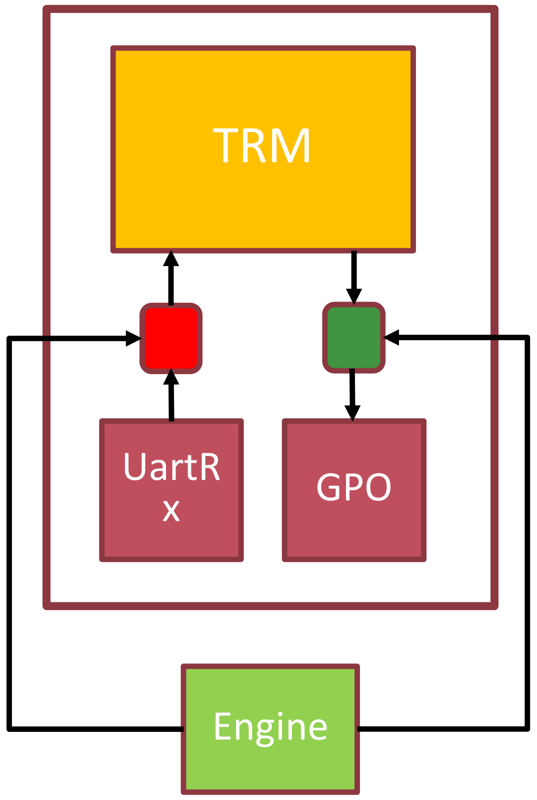
- Use of **AXI4 Stream** interconnect standard from ARM
  - Generic, flexible
  - Non-redundant



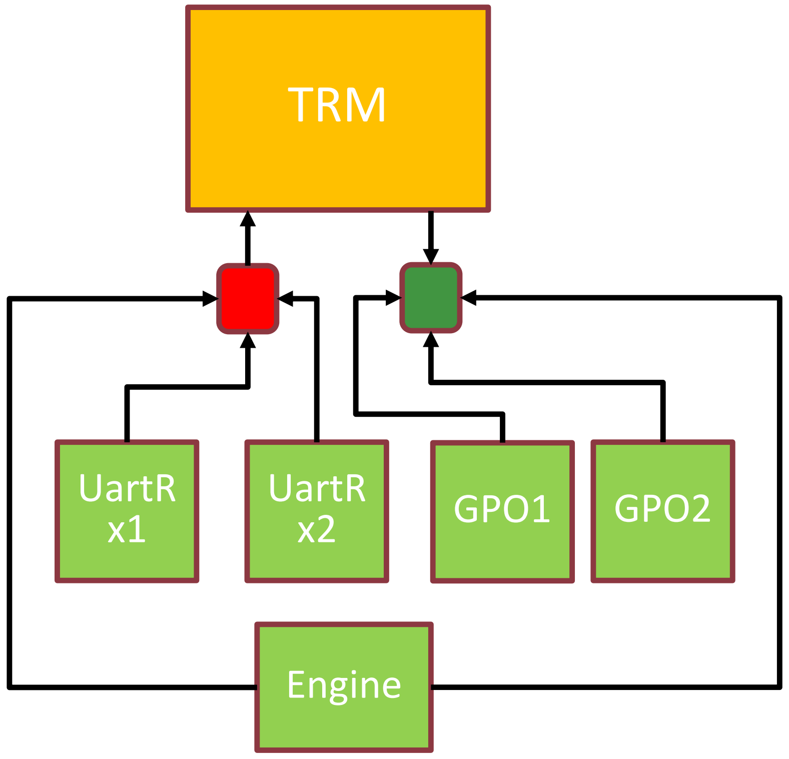
Peripherals

Peripherals generalized to components with connection to the network via AXI-4 Stream

ActiveCells 1



ActiveCells 2



Example: Programming Model

```
IMPORT Engines;
TYPE
    TrmCell = CELL {Processor="TRM"} (gpoOut, uartOut: PORT OUT; uartInp: PORT IN)
    VAR x: LONGINT;
    BEGIN
        LOOP
            uartInp ? x;
            gpoOut ! x; (* light led *)
            uartOut ! x; (* echo on uart *)
        END;
    END TrmCell;

    TestSpartan3Board = CELLNET
    VAR
        trm: TrmCell; gpo: Engines.Gpo; uartTx: Engines.UartTx; uartRx: Engines.UartRx;
    BEGIN
        NEW(trm);
        NEW(gpo {DataWidth=8} );
        NEW(uartTx {ClkDivisorWidth=16,InitClkDivisor=434,CtsPortUnused=1,InitEnableRtsCts=0} );
        NEW(uartRx {ClkDivisorWidth=16,InitClkDivisor=434,RtsPortUnused=1});
        CONNECT(trm.gpoOut, gpo.input);
        CONNECT(trm.uartOut, uartTx.input,128);
        CONNECT(uartRx.output, trm.uartInp,128);
    END TestSpartan3Board;
```

ActiveCells 2

- Flexible parameterization of the components using interpreted code in the component specification

XML-based specification (→ object persistency)

```
<element type="HdlParameter"  name="KernelLength" description="filter kernel length">
   <value type="IntegerValue" value="?{instance.capabilityParameters['KernelLength'].integer:7}?"/>
        <constraint>
       <element type="IntegerRangeValue" value="1:"/>
        </constraints>
</element>
```

Instance-wise parameterization is possible

```
var
controller{Arch="TRM"}: Controller;

new(fir[k]{CoeffWidth=16,InpWidth=16,OutWidth=16,KernelLength=12,InitShift=15});
```

Active Cells 3

Parameterization and Description of Hardware completely in one programming language

```
module Spartan3StarterBoard;
	t: AcHdlBackend.TargetDevice;
	pldPart: AcXilinx.PldPart;
	ioSetup: AcHdlBackend.IoSetup;
begin
	new(pldPart,"XC3S200FT256-4");
	new(t,"Spartan3StarterBoard",pldPart);

	new(ioSetup,"UartTx_0");
	ioSetup.NewPin("txd",Out,"R13","LVCMOS25");
	ioSetup.NewPin("cts",In,"T12","LVCMOS25");
	t.AddIoSetup(ioSetup);

	(* ... *)

	AcHdlBackend.hwLibrary.AddTarget(t);
end Spartan3StarterBoard.
```

1. Platform specific settings
clock sources, pin locations etc

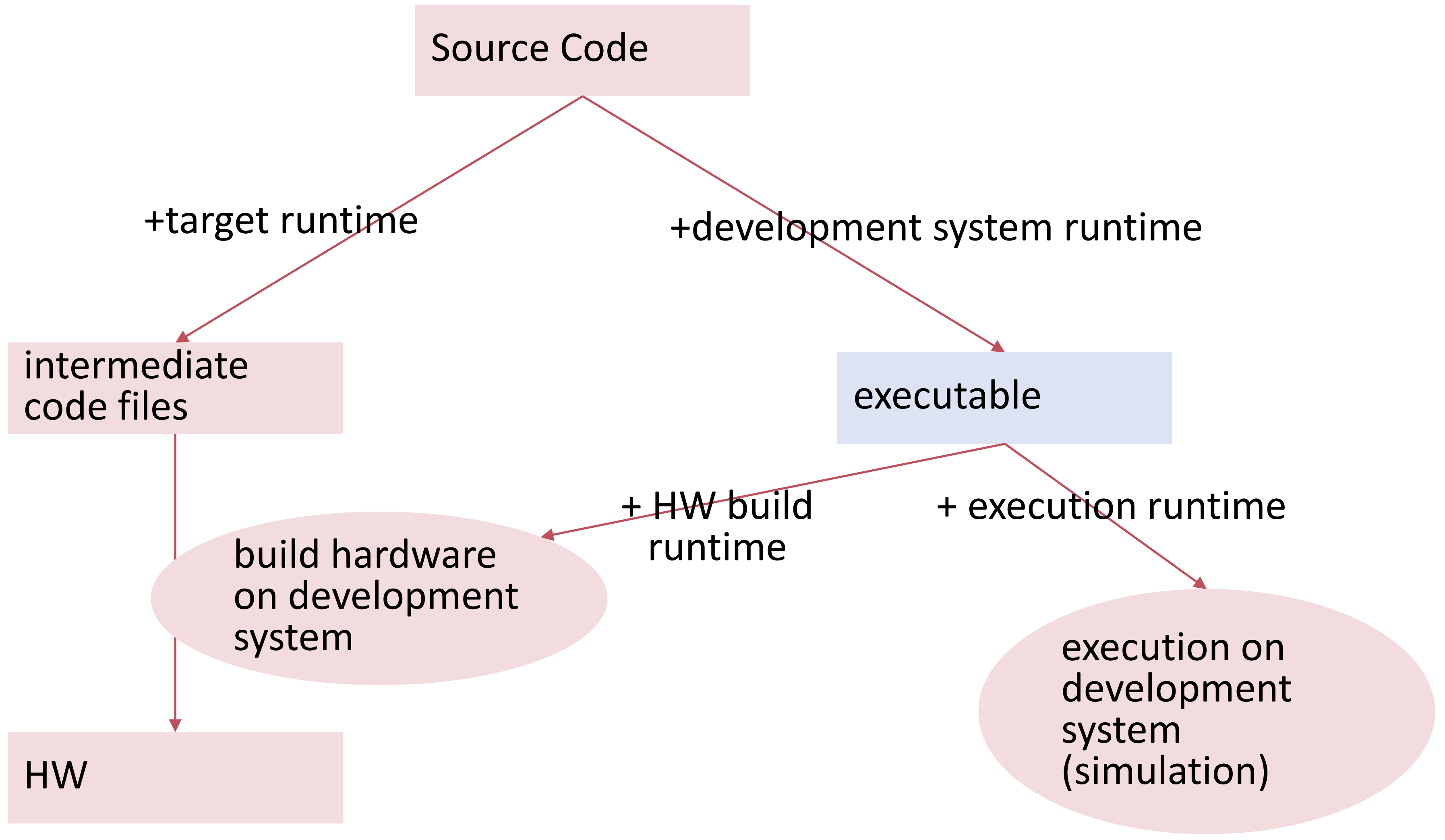
Active Cells 3

Parameterization and Description of Hardware completely in one programming language

```
module Gpo;
var
	c: HdlBackend.Engine;
begin
	new(c,"Gpo","Gpo");
	c.SetDescription("General Purpose Output (GPO) with parameterizable data width and stream control interface");
	c.NewProperty("DataWidth","DW",HdlBackend.NewInteger(32),HdlBackend.IntegerPropertyRangeCheck(1,HdlBackend.MaxInteger));

	c.SetMainClockInput("aclk"); (* main component's clock *)
	c.SetMainResetInput("aresetn",false); (* active-low reset *)
	c.NewAxisPort("input","inp",HdlBackend.In,8);

	c.NewDependency("Gpo.v",true,false);

	c.AddPostParamSetter(HdlBackend.SetPortWidthFromProperty("inp","DW"));
	HdlBackend.hwLibrary.AddComponent(c);
end Gpo.
```

2. Component specific settings:
dependencies (Verilog-Files)
parameters
port names
Can be made very generic with plugins.

Active Cells 3 Toolchain


Source Code
+target runtime
+development system runtime
intermediate code files
executable
+ HW build runtime
+ execution runtime
build hardware on development system
HW
execution on development system (simulation)