Silicon Hive

TIM guide

machine description & building blocks

Freedom To Create

Silicon Hive BV
High Tech Campus 43, 5656 AE Eindhoven, The Netherlands
info@siliconhive.com, http://www.siliconhive.com

Company Confidential

© 2008 Silicon Hive BV

Silicon Hive

Contents

- Processor design methodology: overview
- Machine description,
  TIM hierarchy & syntax
- Core IO description, overview
- processor building blocks
- operation name conventions

© 2008 Silicon Hive BV

Introduction Silicon Hive (I)

Corporate History

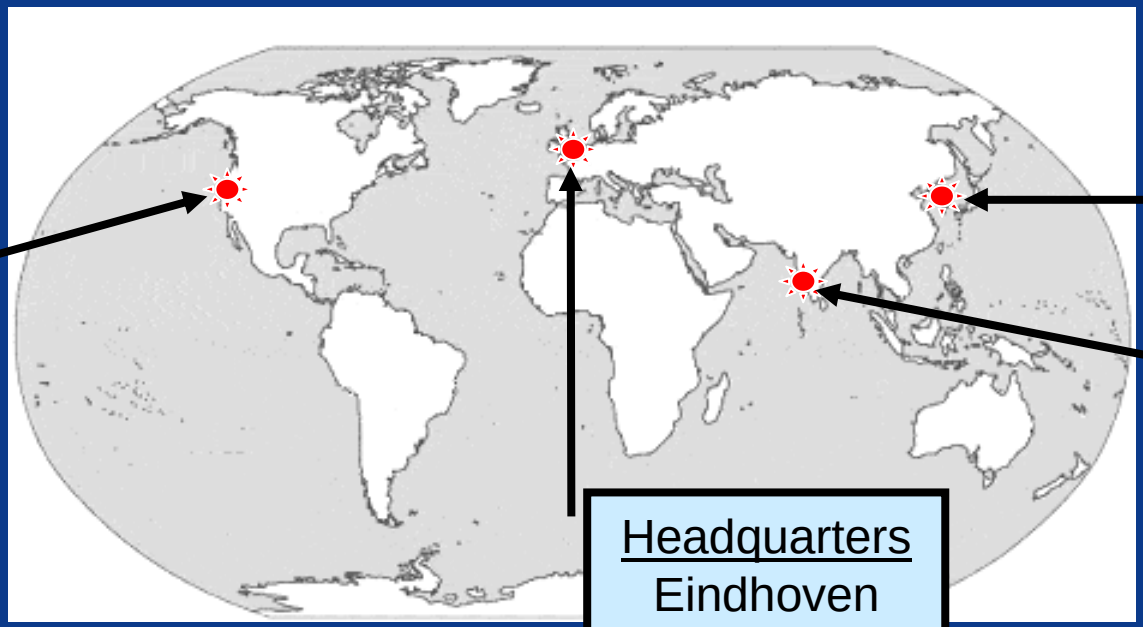
- **1985…** Philips Research work on a.o.
  - » Behavioural synthesis (Cathedral II, EU project)
  - » Datapath synthesis (AR|T, Phideo)
  - » TriMedia architecture
  - » Spatial compilation
  - » Software-defined Radio
- **2001** Seeded within Philips Technology Incubator
- **2002/03** Productized and extended patented Philips programmable processors, tools and SW
- **2004** Licensed first IP cores (to NXP and Agere Systems)
- **2004** Awarded at Microprocessor Forum (InStat, best new microprocessor)
- **2005** Closed first license in Far East with Posdata
- **2006** Opened foreign offices
- **2007** Silicon Hive B.V. spins out of Philips

© 2008 Silicon Hive BV

Introduction Silicon Hive (II)

World Wide Operations

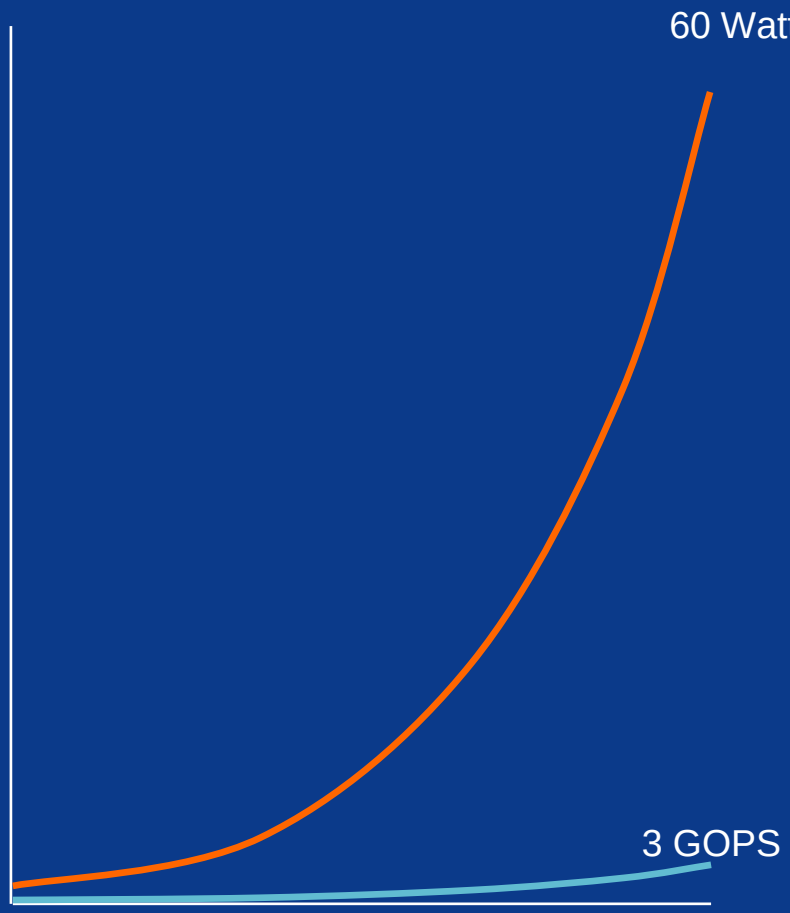
- 45 staff, split between 4 offices



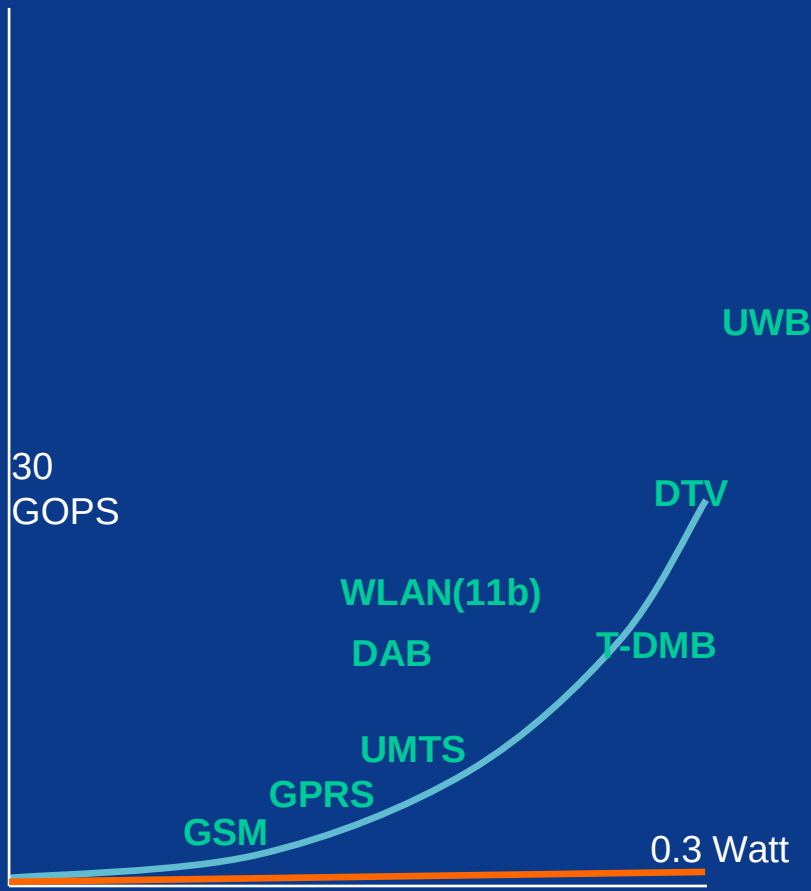
© 2008 Silicon Hive BV

Trends in SoC design (I)

Performance vs. Low-power Envelope



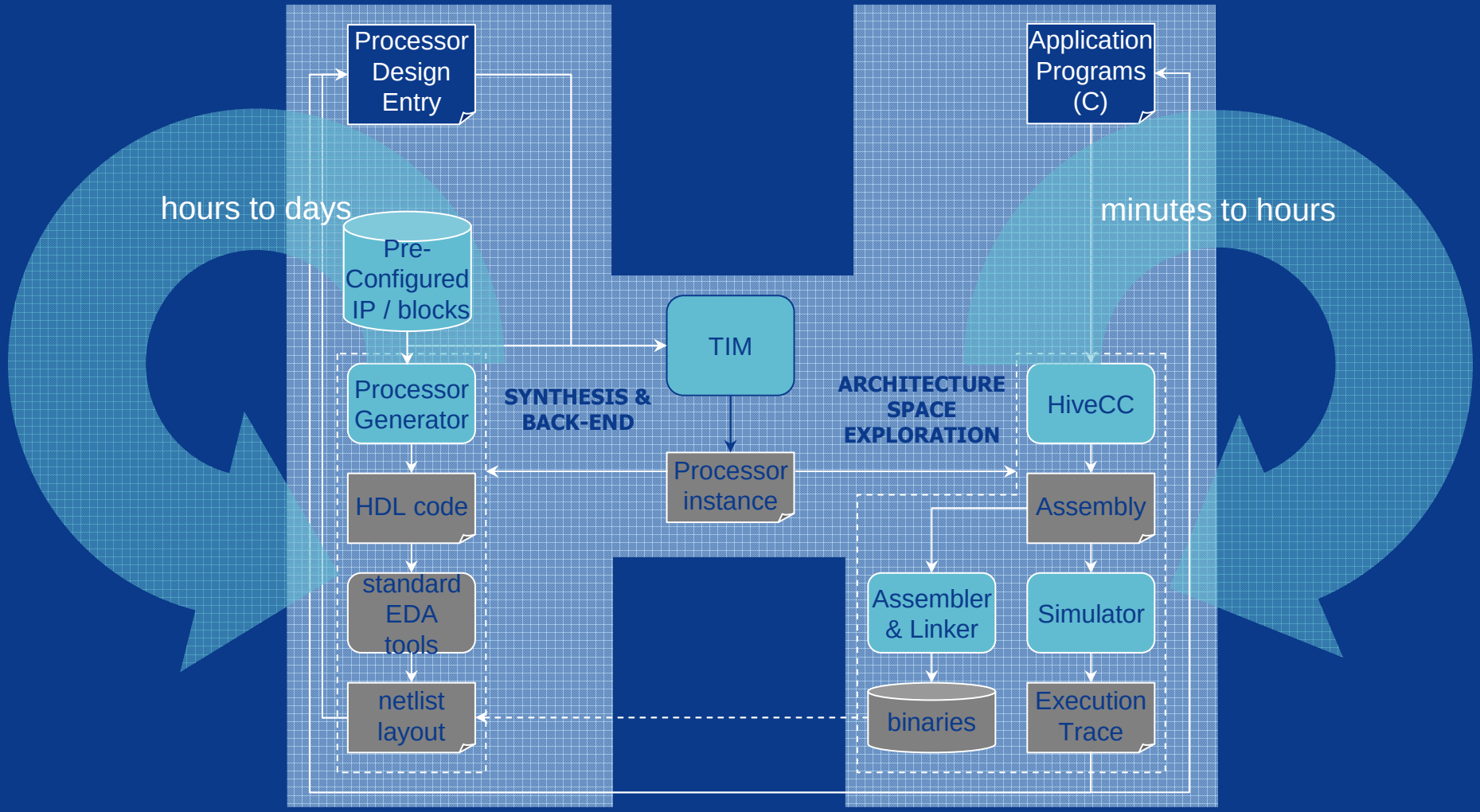
Current trend for (Desk-top) CPUs



Required trend for digital transmission

© 2008 Silicon Hive BV

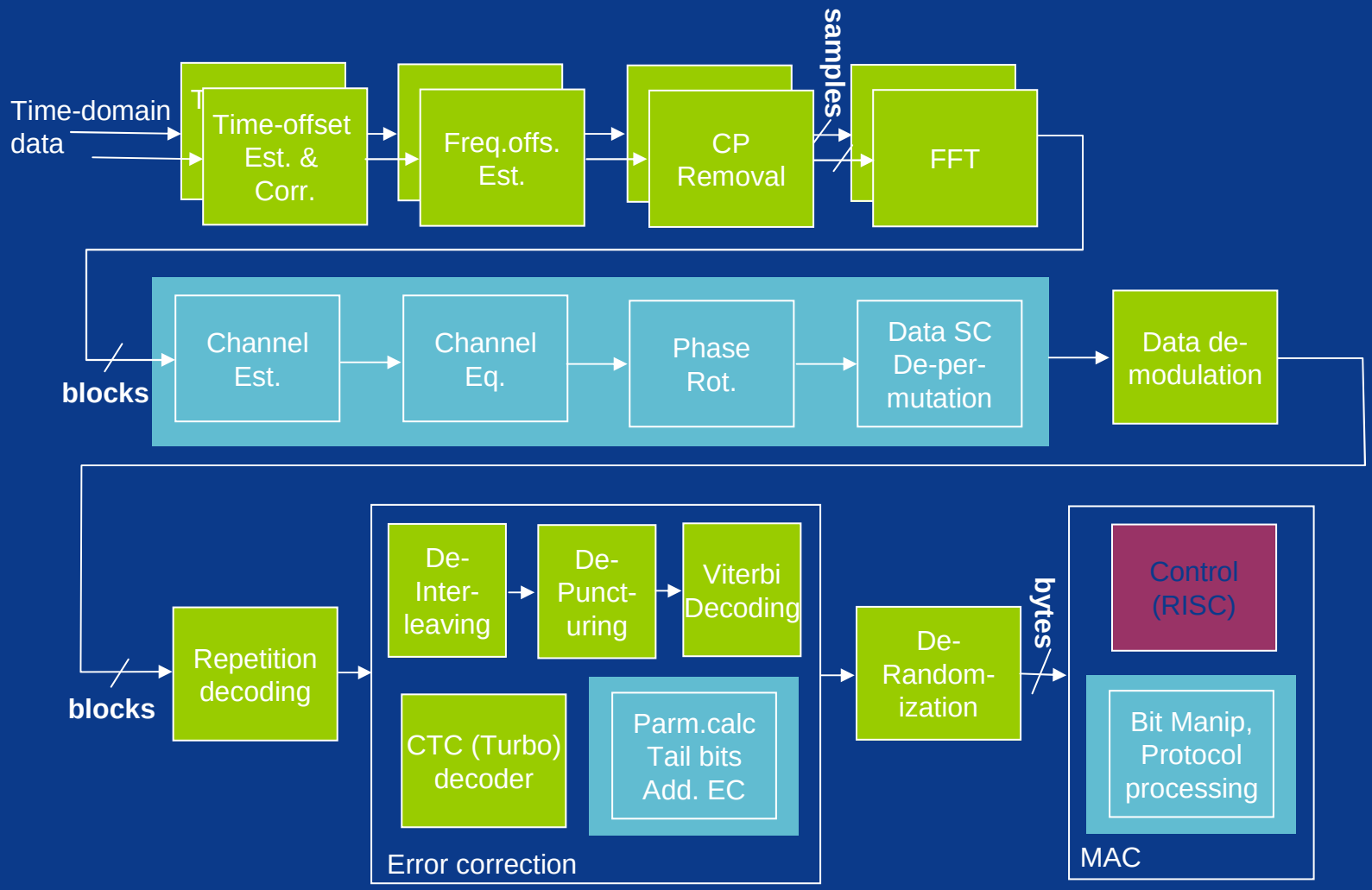
Processor Design Methodology



© 2008 Silicon Hive BV

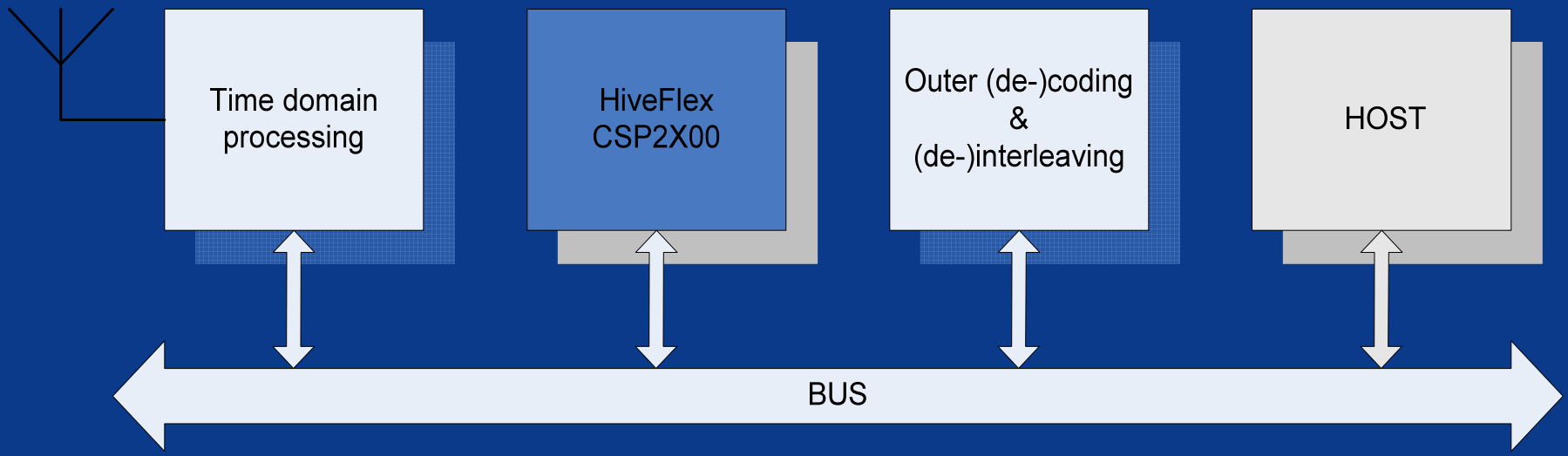
Example System

802.16e-OFDMA Filter Example



© 2008 Silicon Hive BV

Typical Filter SoC Architecture



© 2008 Silicon Hive BV

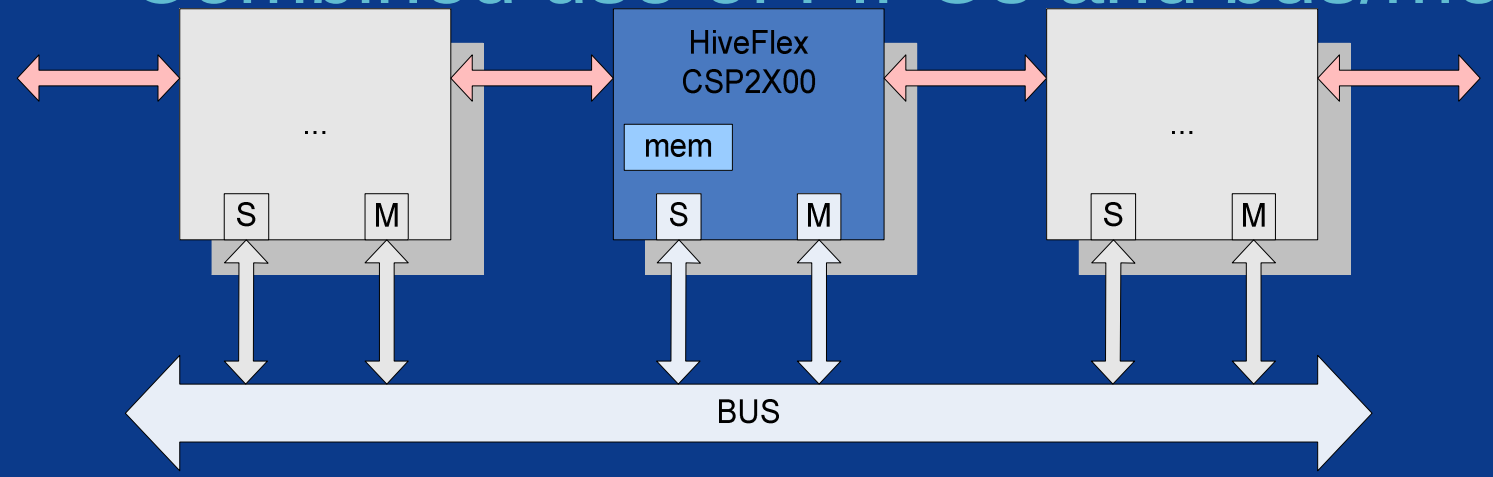
How to exchange data between cores?

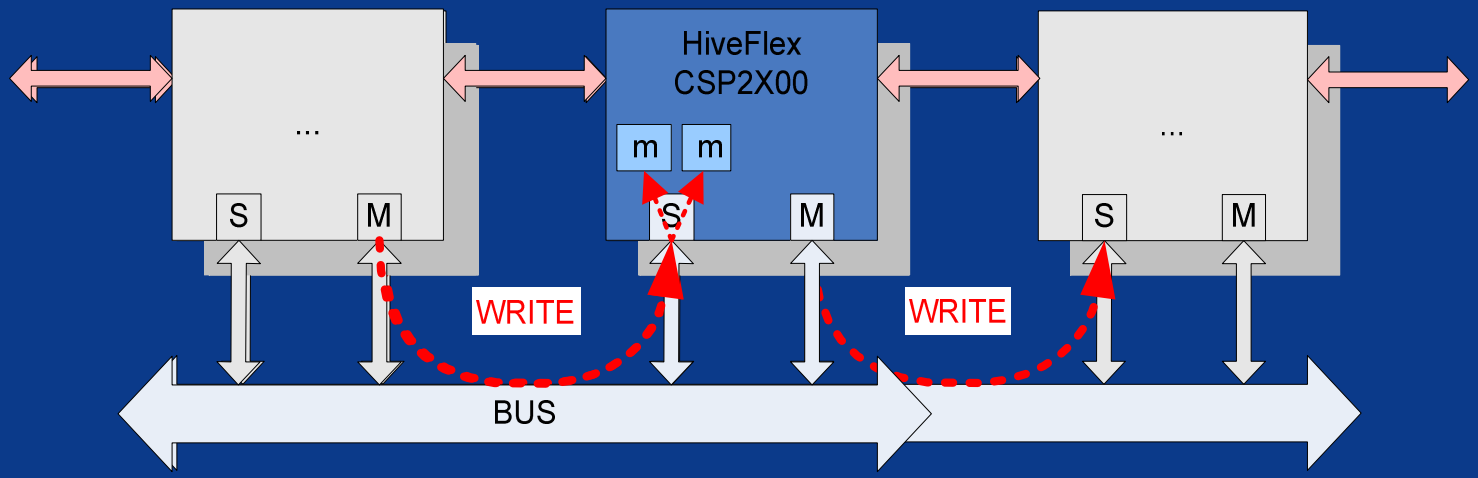
- Memory-mapped I/O (i.e. via bus)
- Through DMA units
- Streaming I/O (point-to-point, no addresses)
- Using Hive Communication API (HCIO), which unites all three above

© 2008 Silicon Hive BV

Possible Filter Subsystem Architectures

Combined use of FIFOs and bus/memory

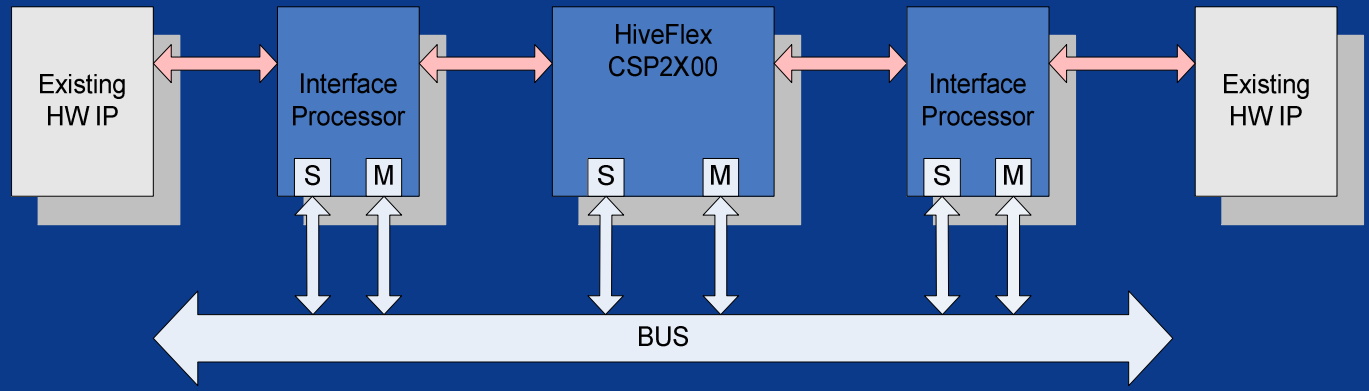


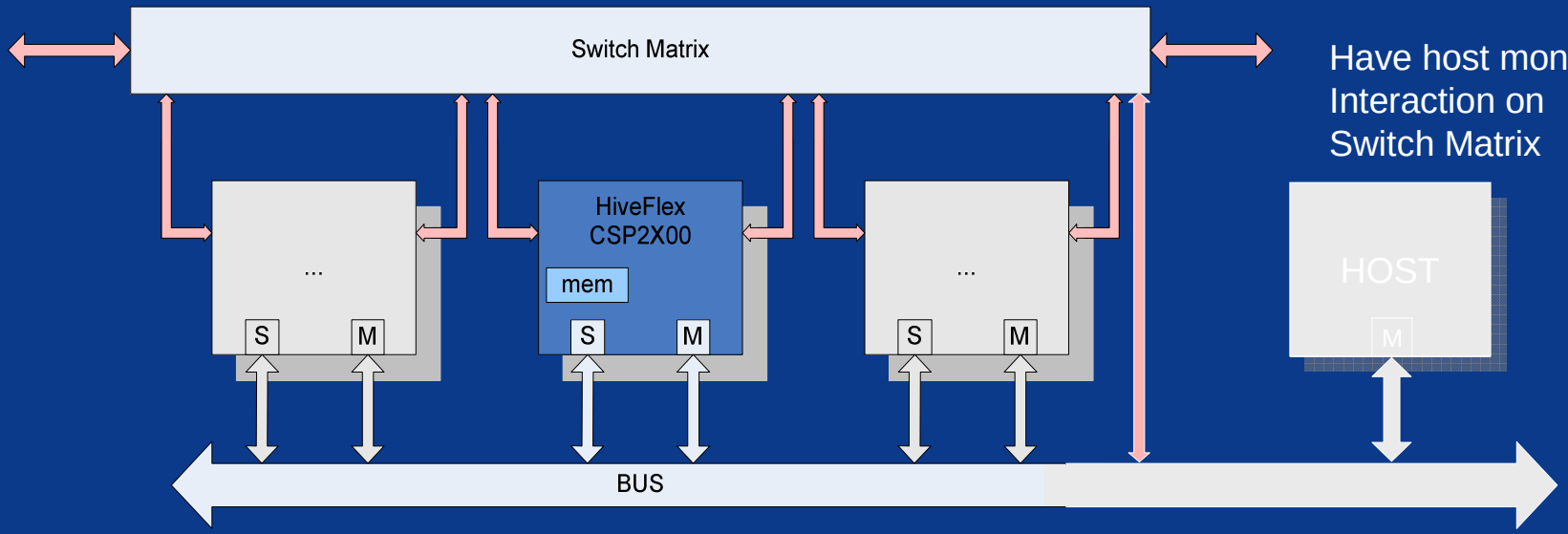


© 2008 Silicon Hive BV

Possible Filter Subsystem Architectures

Switch Matrix on FIFOs for increased flexibility

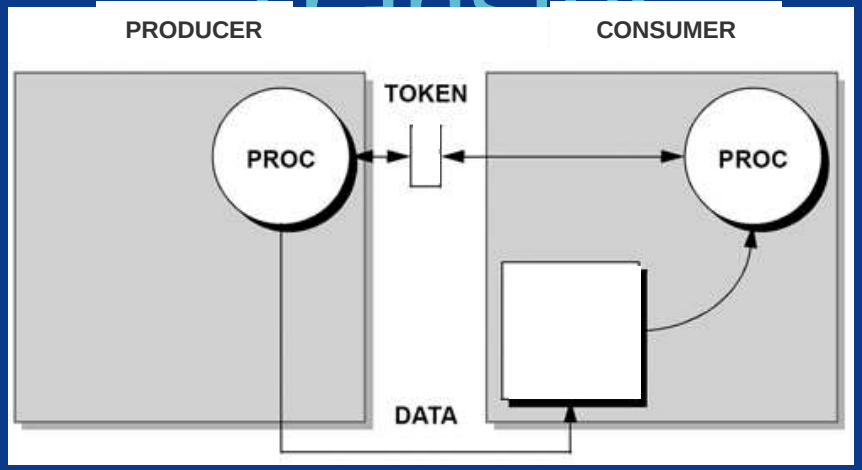




Have host monitor Interaction on Switch Matrix

© 2008 Silicon Hive BV

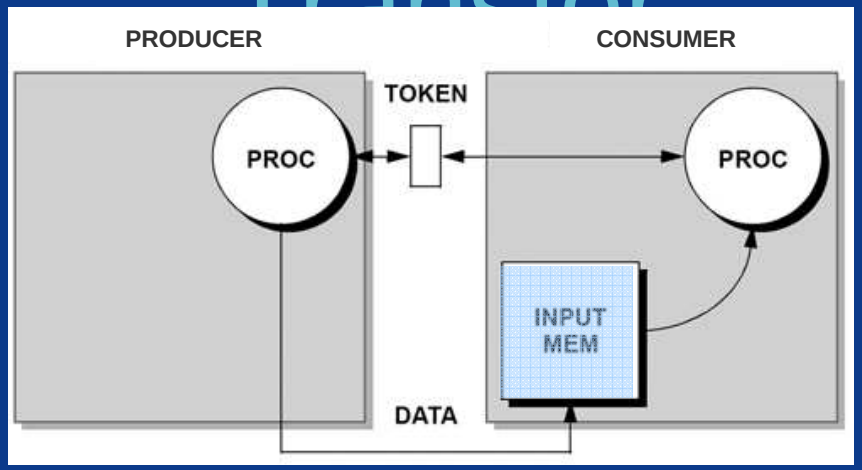
Synchronized Block Data Transfer

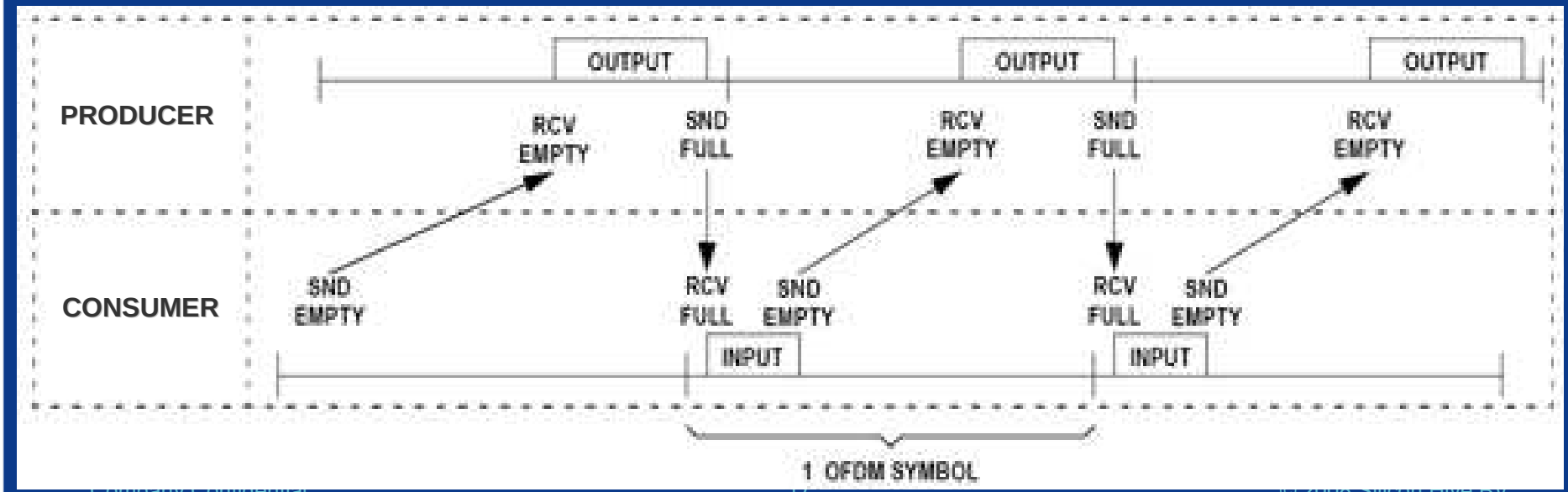


PRODUCER	
CONSUMER	

© 2008 Silicon Hive BV

Synchronized Block Data Transfer

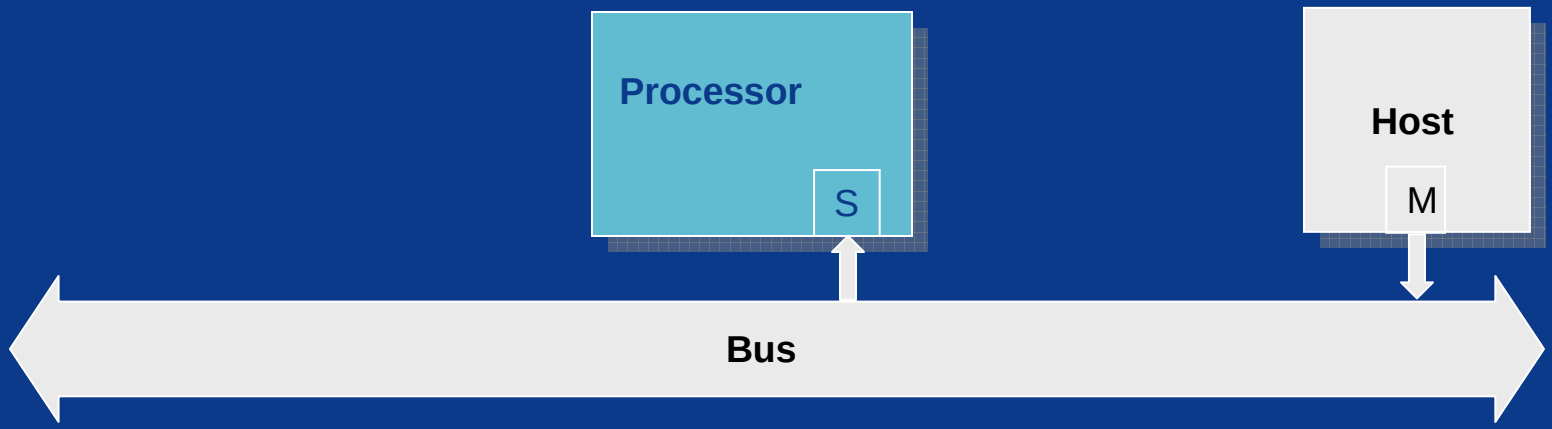




© 2008 Silicon Hive BV

Multi-core Simulations (I)

Simple System

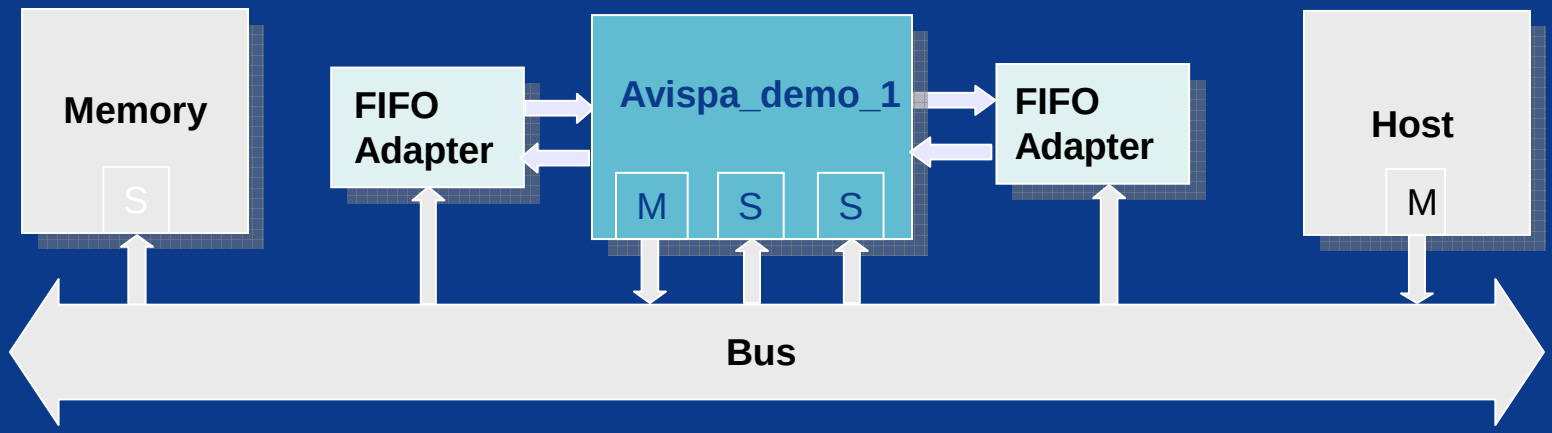



```
#include “processor.hsd”
#include “host.hsd”
#include “bus.hsd”

System SimpleSystem
{
    Processor   proc(bus.op0);
    Host        host;
    SystemBus   bus(host.op0);
}
```


© 2008 Silicon Hive BV

Multi-core Simulations (II)

Default System

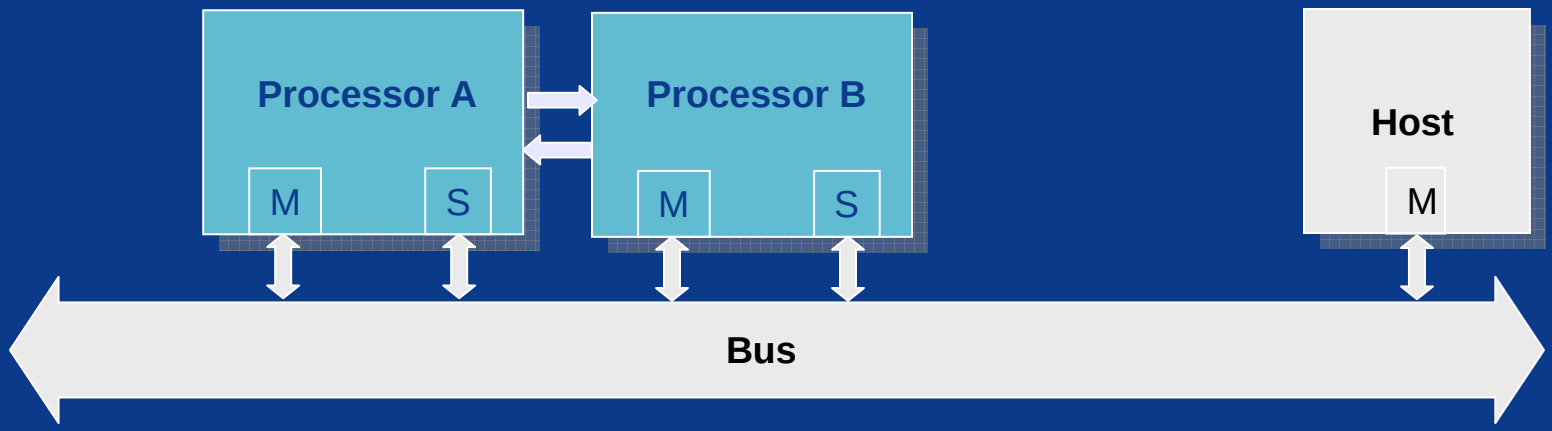



```
System avispa_demo_1_system
{
  avispa_demo_1 avispa_demo_1( bus.op0, bus.op1, fa0.op, fa1.op );
  FifoAdapter   fa0( avispa_demo_1.st_op0, bus.op2 );
  FifoAdapter   fa1( avispa_demo_1.st_op1, bus.op3 );
  Memory        external_memory( bus.op4 );
  SystemBus     bus( host.op0, avispa_demo_1.mt_op );
  HostProcessor host;
};
```


© 2008 Silicon Hive BV

Multi-core Simulations (III)

Simple Multi-core System

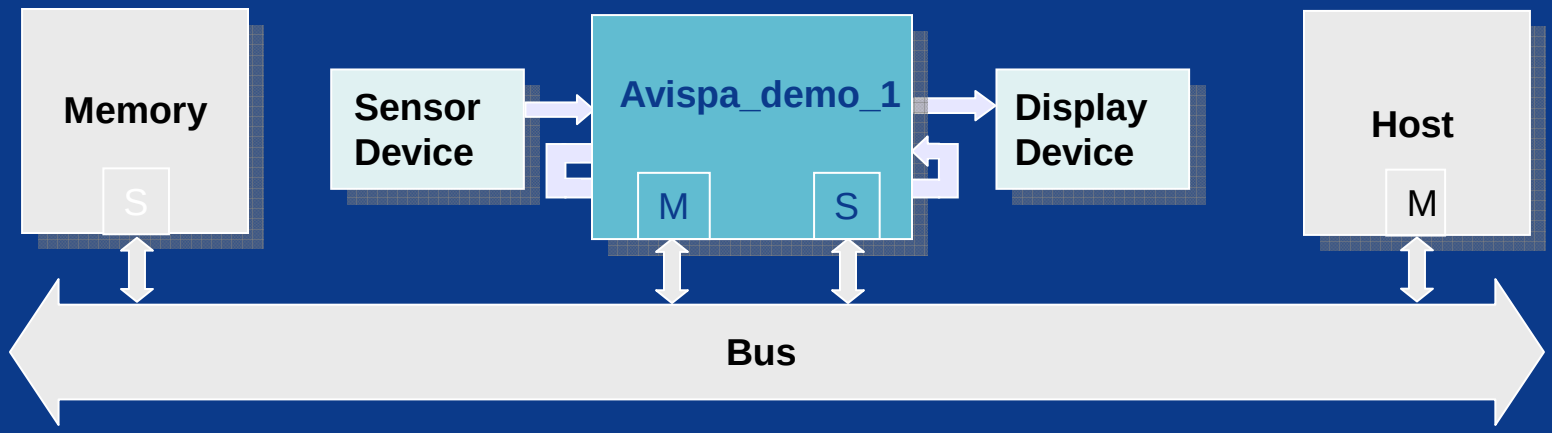



```
#include <hss/std_devices.hsd>
#include “processorA.hsd”
#include “processorB.hsd”

System MultiProcessorSystem
{
    ProcessorA   procA(bus.op0, procB.st_out);
    ProcessorB   procB(bus.op1, procA.st_out);
    Host         host;
    Memory       memory(bus.op2);
    SystemBus    bus(host.op0, procA.mt, procB.mt);
}
```


© 2008 Silicon Hive BV

Multi-core Simulations (IV)

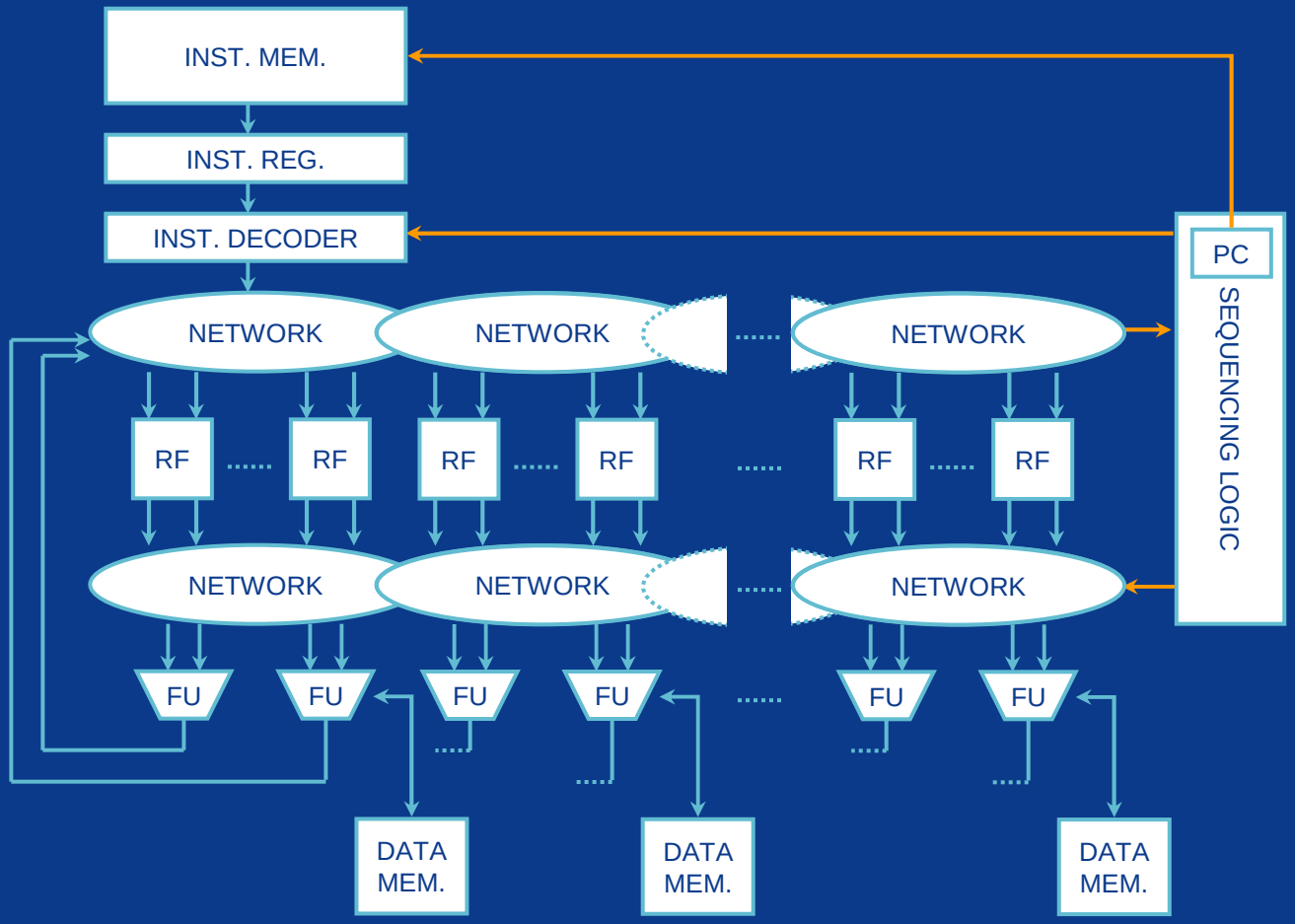
Camera System with Custom Devices



© 2008 Silicon Hive BV

Silicon Hive

Processor Template



© 2008 Silicon Hive BV

Silicon Hive

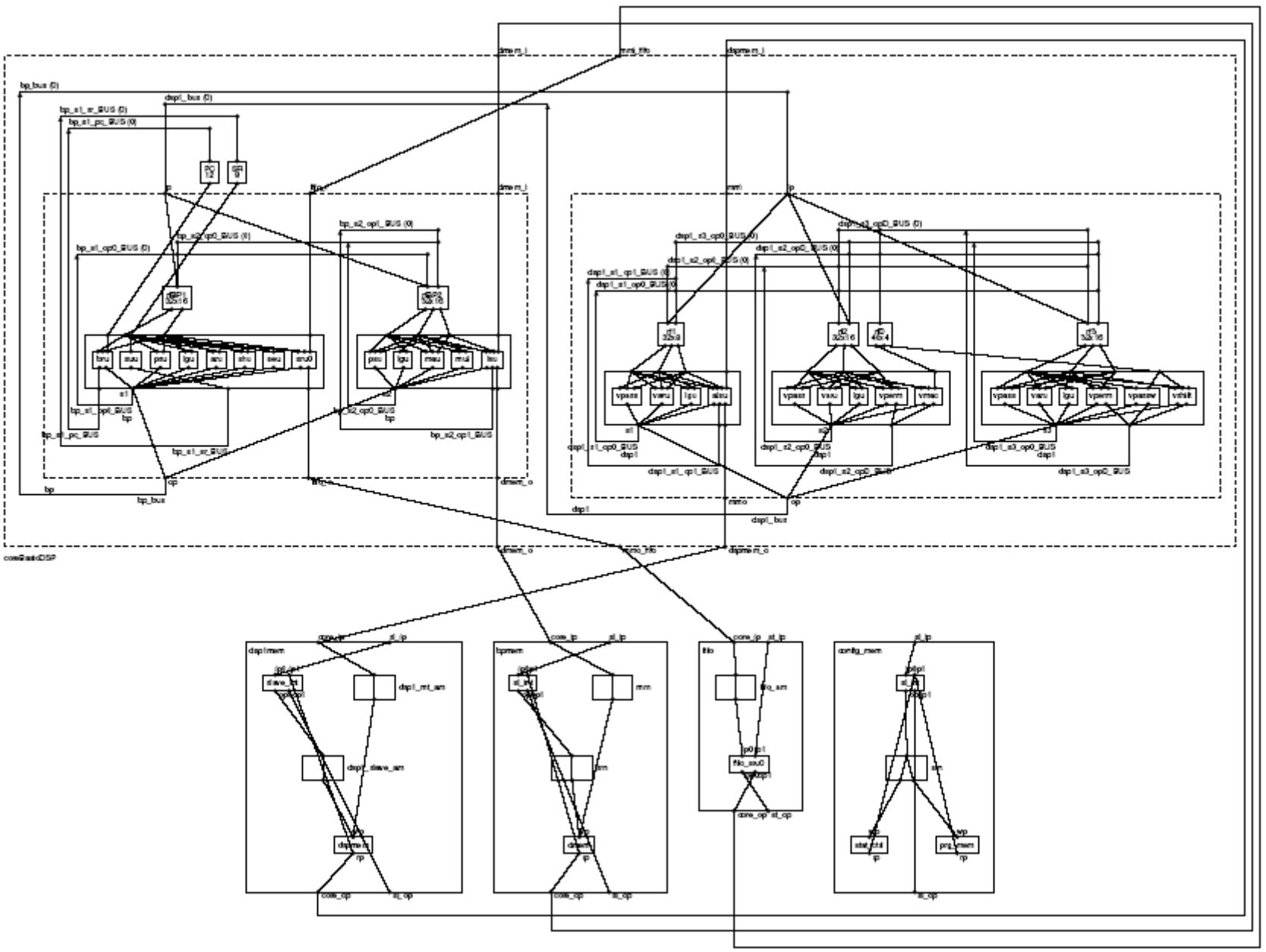
Processor Design Choices

During design time, the following design choices can be made:

- number of issue slots
- function units: type, number, distribution
- register files: number, sizes, and porting
- Interconnect: within issue slots
- Interconnect: between issue slots
- generic instruction set
- application-specific instruction set
- local memories: number, connection, ports
- interfaces: number, type, protocol

© 2008 Silicon Hive BV

example core: “pearl_coral”

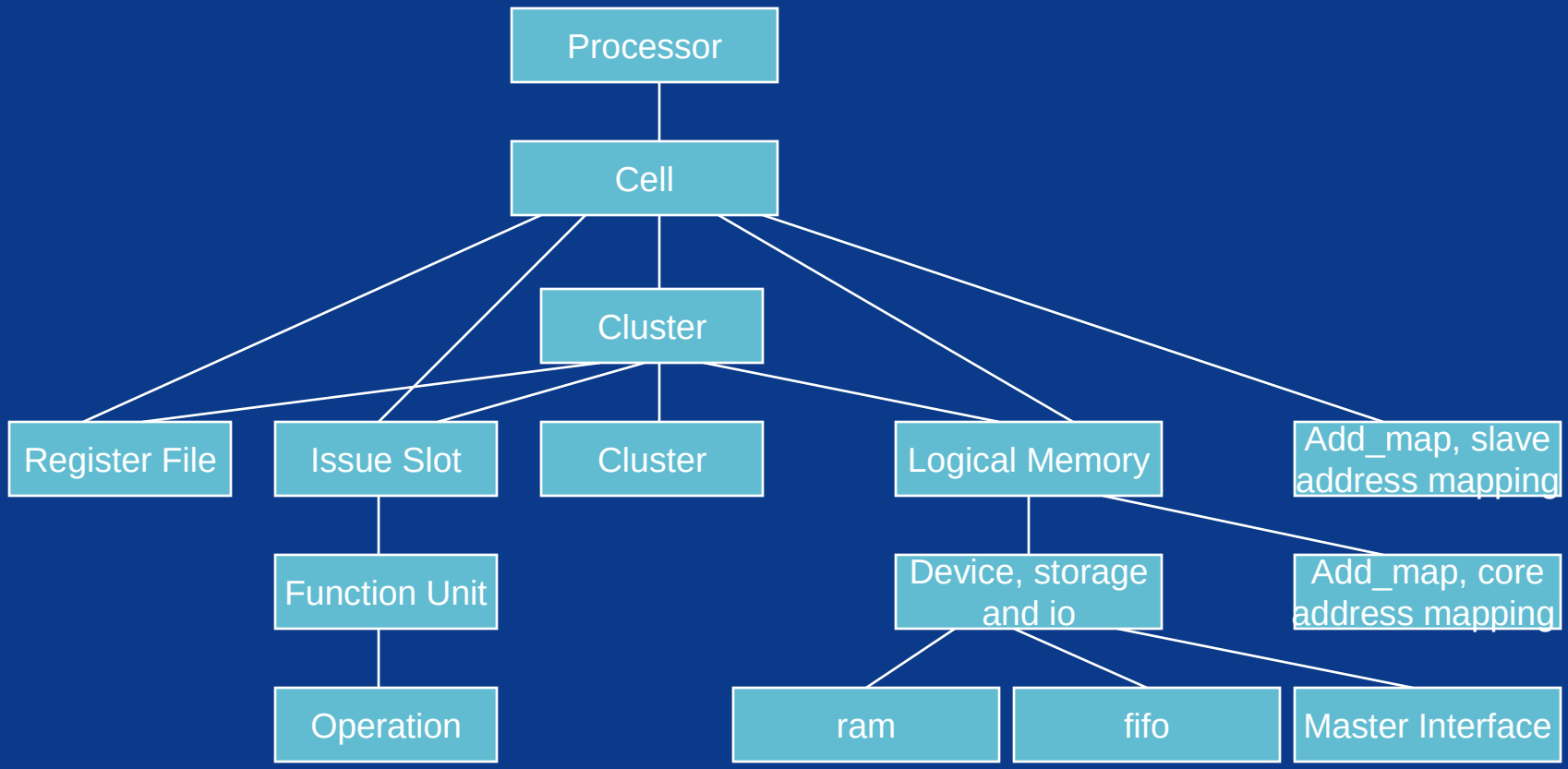


Silicon Hive

Machine Description

© 2008 Silicon Hive BV

TIM Hierarchy

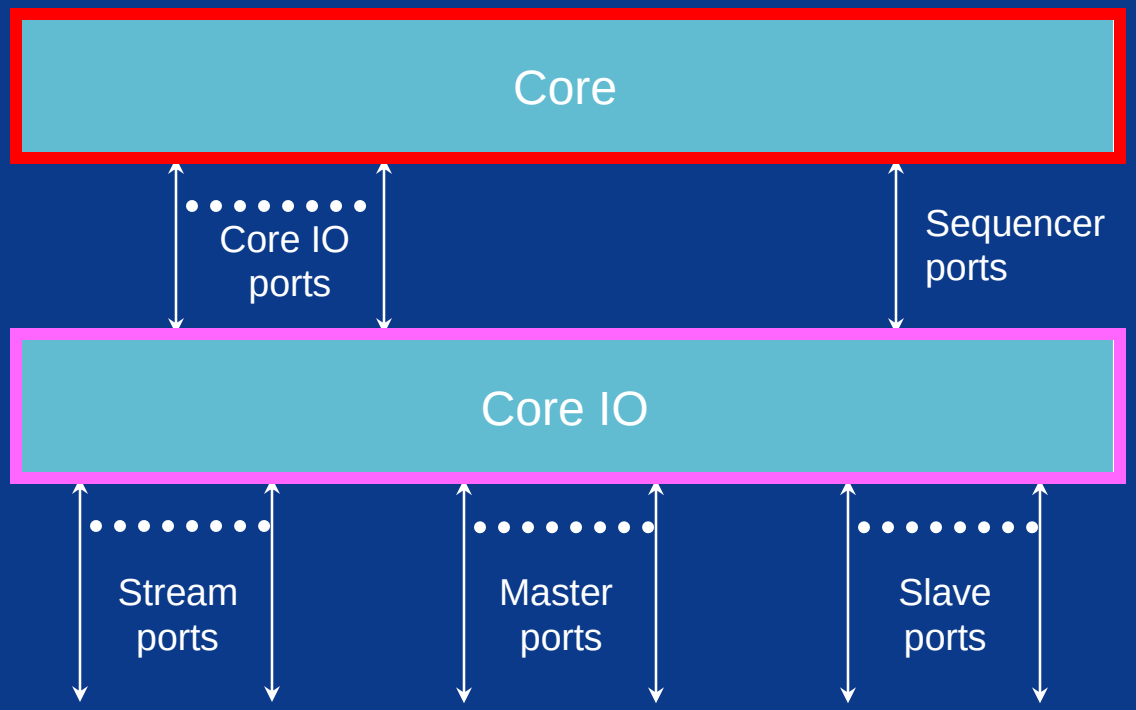


Core

Core IO

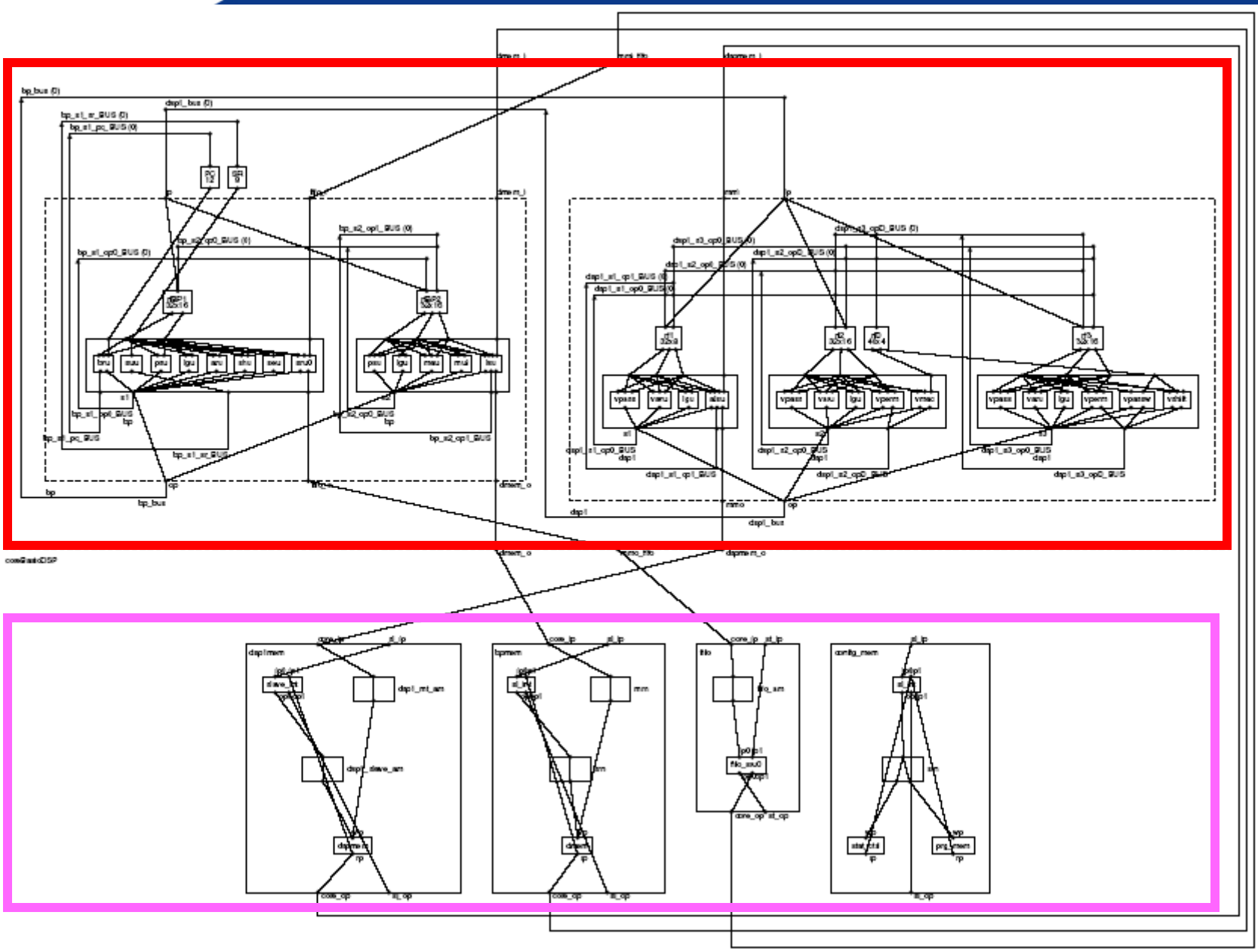
© 2008 Silicon Hive BV

Core & Core IO



© 2008 Silicon Hive BV

Core & Core IO



TIM building block

- At the heart of the tim languange is the “building block” syntax.

- It defines a new building block name,
- with specified inputs and outputs,
- instantiating one or more other building blocks,
- optionally defining specific building block properties,
- optionally taking parameters.

© 2008 Silicon Hive BV

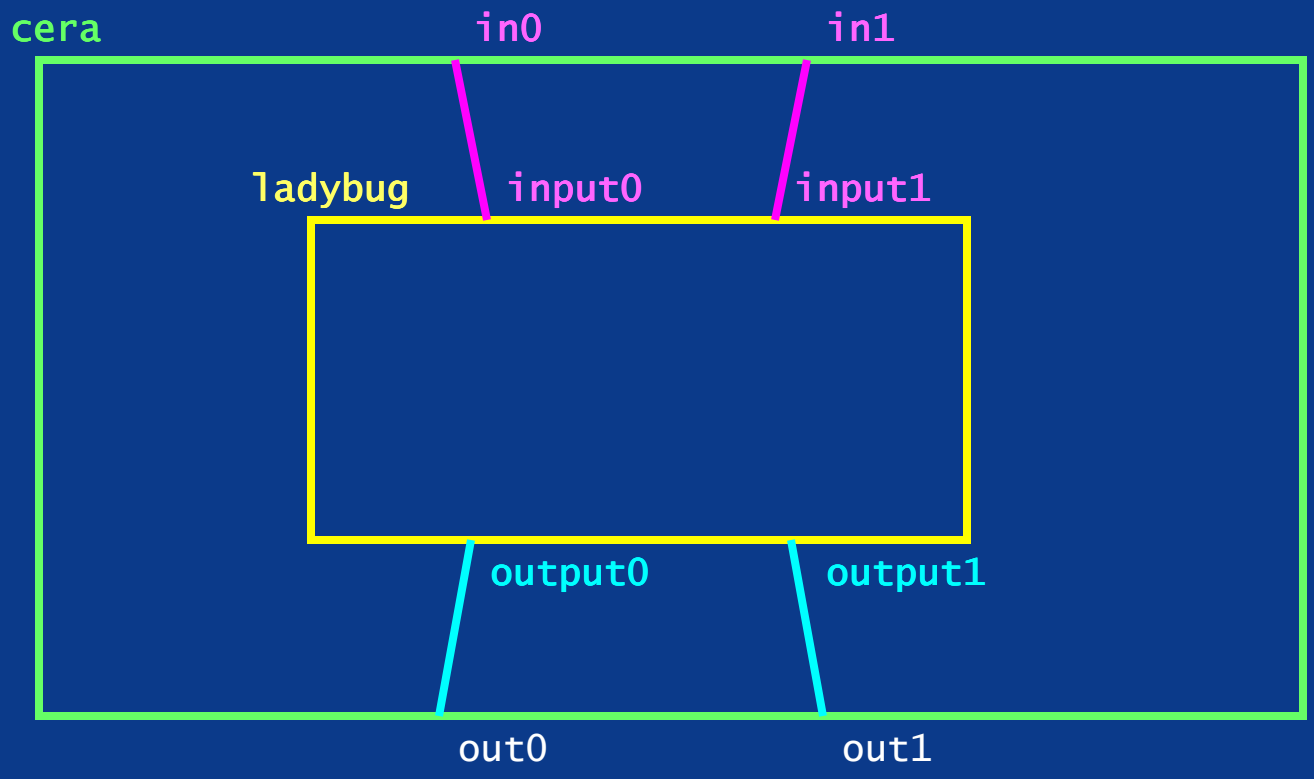
TIM building block syntax

```
keyword name       ( PortType0 in0, in1,   PortType2 in2 )
                -> ( PortType0 out0, out1, PortType2 out2 )
{
  body
};
```

- Keyword can be “Cluster”, “IS”, “Logical_memory”, …
- Name can be freely chosen.

© 2008 Silicon Hive BV

TIM building block syntax



© 2008 Silicon Hive BV

TIM building block syntax

```
keyword insect    ( PortType in0,  in1 )
                -> ( PortType out0, out1 )
{
  // instantiation of building block ladybug with name cera:
  ladybug cera (in0, in1 );

  // list of outputs
  out0 = cera.output0;
  out1 = cera.output1;
};

keyword ladybug ( PortType input0,  input1 )
              -> ( PortType output0, output1 )
{
  // instantiation of other units
  ant ant1 (input0, input1);
  ant ant2 (input0, input1);
  bee bee1 (input0, input1);

  output0 = { ant1.output0, ant2.output0, bee1.output0 };
  output1 =   bee1.output1;
};
```


© 2008 Silicon Hive BV

Properties in building block

- Different building blocks of the tim hierarchy can set different properties
- examples:

```
Cell pearl ( … ) -> ( … )
{
  CharBits  := 8;
  ShortSize := 2;
  DefaultMem := bp.dmem
  ...
};

Bus Bus32 (Port32 ip) -> (Port32 op)
{
  Width := 32;
};
```


© 2008 Silicon Hive BV

Parameters in building block

- example declaration of parameterized building block:

```
keyword insect < signed arms, signed legs >  ( … ) -> ( … )
{
  signed limbs := arms + legs; // parameter expression
  ...
};
```

- Instantiating a parameterized building block:

```
  signed a := 2;
  signed b := 2;
  insect bug_with_4limbs < a, b > ( <inputs> );
```

- Some examples of operands allowed in parameter expression:

```
+, -, *, /, %, <<, >>
```


© 2008 Silicon Hive BV

parameters and properties

- a parameterized building block with properties

```
Port my_svecNxB < signed nway, signed elembits >  /* no inputs/outputs */
{
  Signed := 1;
  Packed := nway;                  // parameter expression
  Width  := nway * elembits;       // parameter expression
};
```


© 2008 Silicon Hive BV

Silicon Hive

Machine Description

- **Processor**
- Cell
- Cluster
- Issue slot
- Functional unit
- Operation

© 2008 Silicon Hive BV

Processor

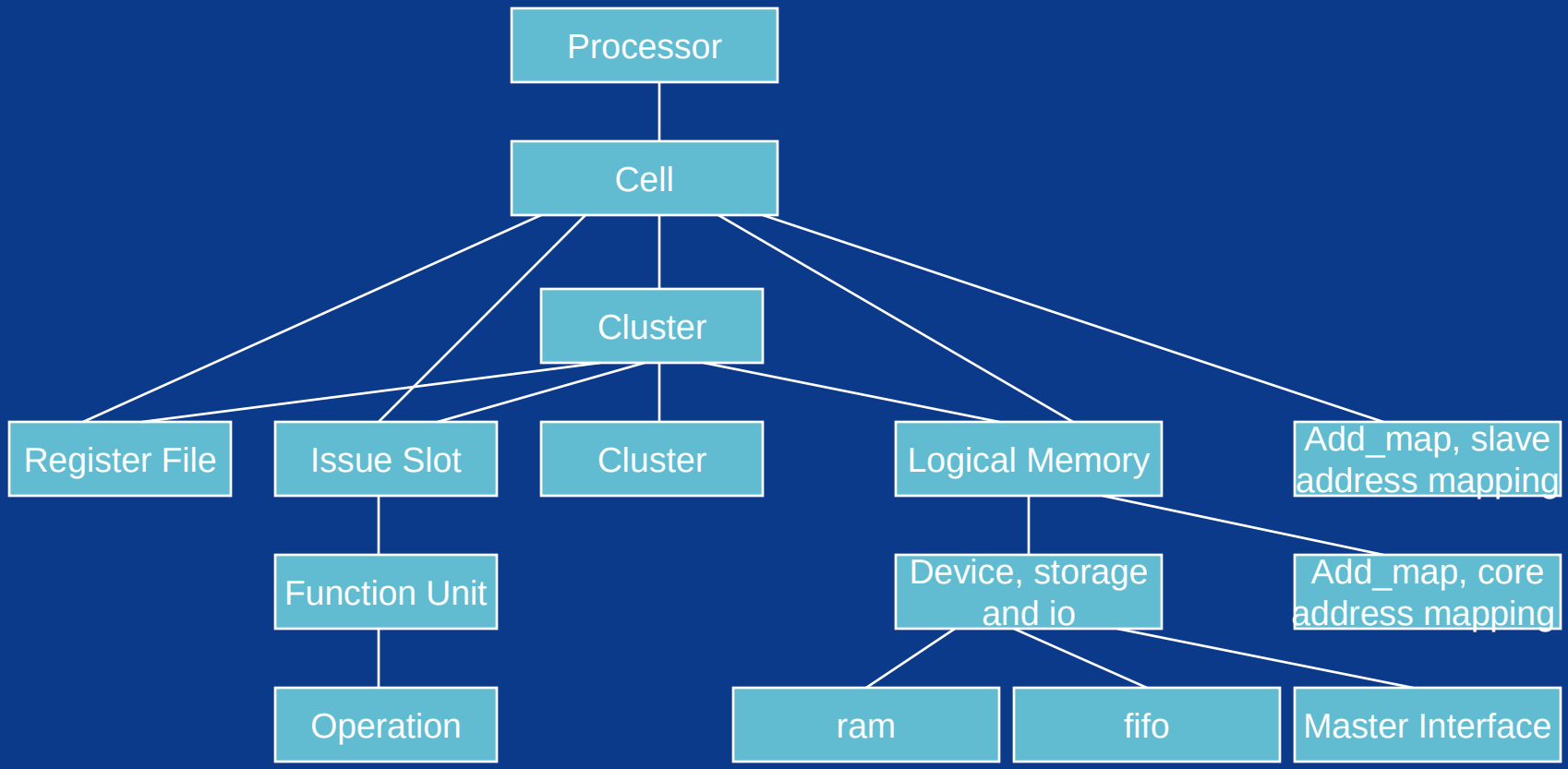
TIM keyword: **Processor**

- Specification of input and output ports of processor:
  master IF, slave IF, fifo IF
- Instantiation of **cell**
- Specification of inter-Cell connectivity
  (currently 1 cell only: processor I/O = cell I/O)

- inter-Processor connectivity is done with “hive system description” (HSD)

© 2008 Silicon Hive BV

TIM Hierarchy



Core

Core IO

© 2008 Silicon Hive BV

Processor: tim code

```
Processor pearl_coral_processor
    ( mmioW_DTL<intWidth> sl_ip, fifoW_DTL<bpFifoWidth> st_ip0, st_ip1)
 -> ( mmioW_DTL<intWidth> sl_op, fifoW_DTL<bpFifoWidth> st_op0, st_op1)
{
  /* parameters that hold throughout the whole cell */
  signed intWidth     := 32;
  signed llWidth      := 32;
  signed bpFifoWidth  := 32;

  /* instantiation of Cell */
  pearl_coral_cell pearl_coral <intWidth, llWidth, bpFifoWidth>
                               (sl_ip, st_ip0, st_ip1);

  /* list of outputs of this building block (Processor) */
  sl_op  = pearl_coral.sl_op;
  st_op0 = pearl_coral.st_op0;
  st_op1 = pearl_coral.st_op1;
};
```


© 2008 Silicon Hive BV

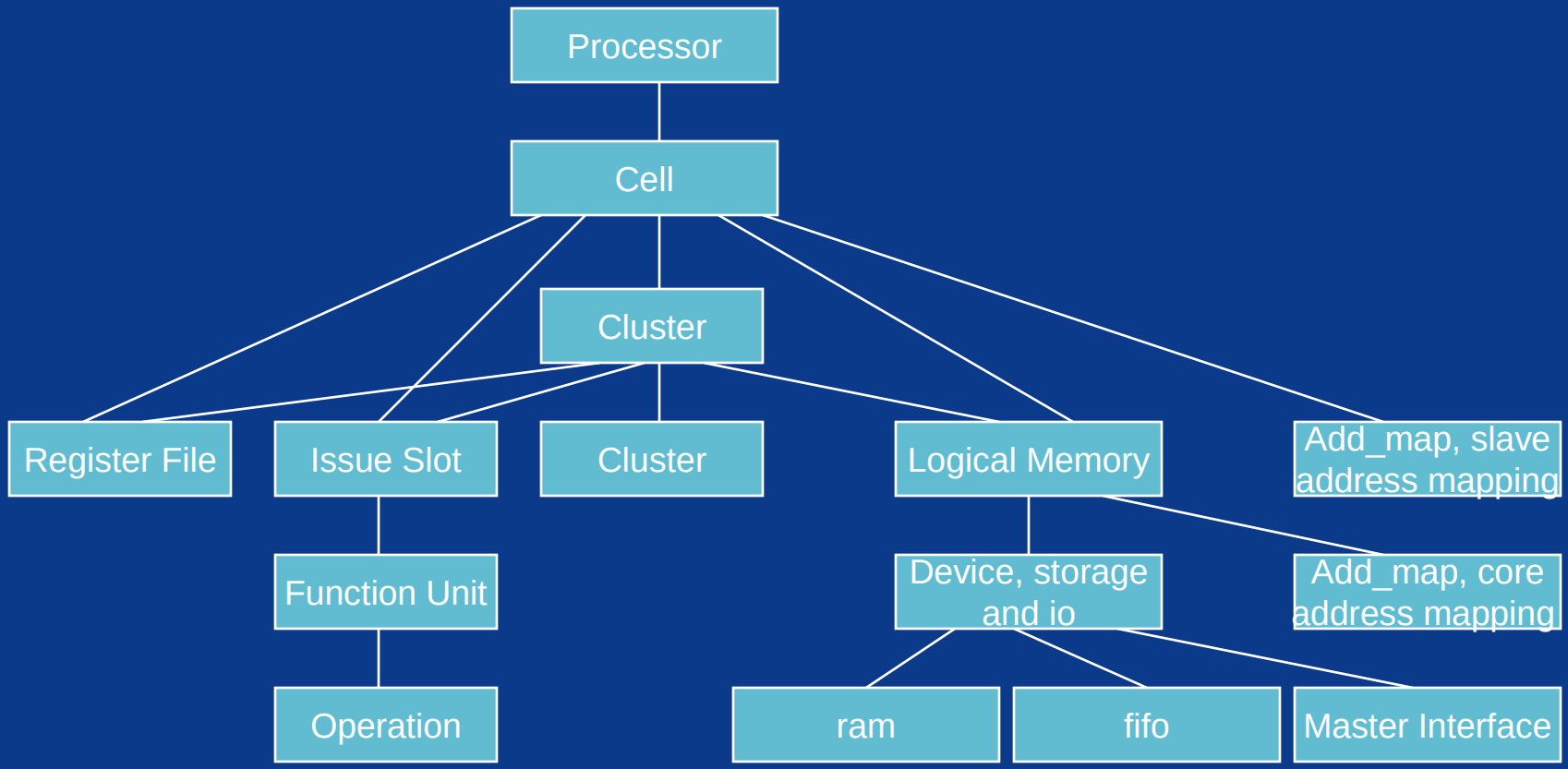
Silicon Hive

Machine Description

- Processor
- **Cell**
- Cluster
- Issue slot
- Functional unit
- Operation

© 2008 Silicon Hive BV

TIM Hierarchy



Core

Core IO

© 2008 Silicon Hive BV

Cell

TIM keyword: **Cell**

- Specification of input and output ports of cell
  (currently 1 cell only: processor I/O = cell I/O → master IF, slave IF, fifo IF)

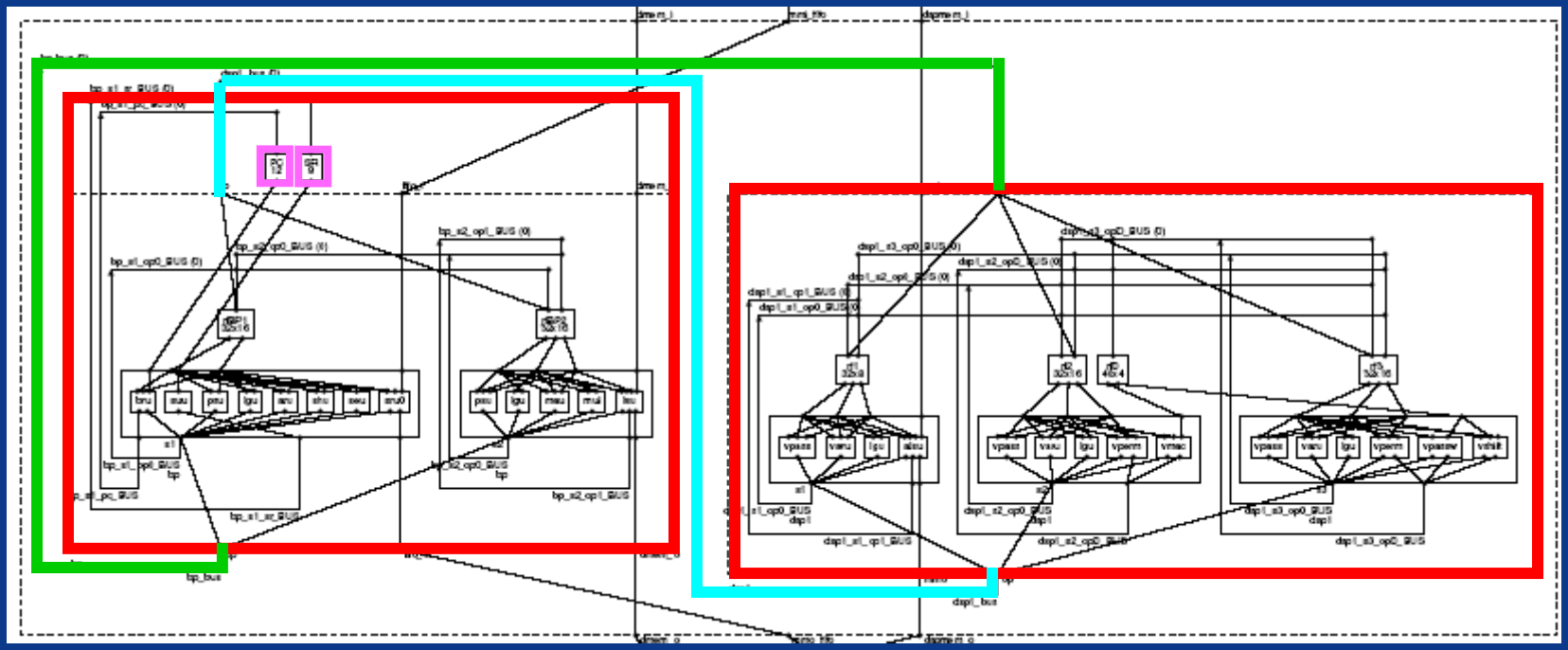
Instantiation of:

- clusters
- Issue slots
- register files
- logical memories
- slave address mapping
- interfaces

- instantiation of **PC, SR** can be done at Cell level or within a cluster
  (BPSE clusters have PC and SR).

© 2008 Silicon Hive BV

Cell



© 2008 Silicon Hive BV

Cell: tim code (1)

```
Cell pearl_coral_cell
    < signed intWidth, signed llWidth, signed bpFifoWidth >
    ( mmioW_DTL<intWidth> sl_ip, fifoW_DTL<bpFifoWidth> st_ip0, st_ip1 )
 -> ( mmioW_DTL<intWidth> sl_op, fifoW_DTL<bpFifoWidth> st_op0, st_op1 )
{
  /* C-types size properties */
  CharBits      := 8;
  ShortSize     := 2;
  IntSize       := intWidth/8;
  LongSize      := 4;
  LongLongSize  := llWidth/8;

  /* Default memory */
  DefaultMem    := bp.dmem;

  /* Stack pointer and function call return pointer */
  SP_rf         := bp.rf2;
  SP_idx        := 0;
  RP_rf         := bp.rf2;
  RP_idx        := 1;
```


© 2008 Silicon Hive BV

Cell: tim code (2)

```
  /* optional: function call parameter passing registers */
  PP_regs[0]    := bp.rf1[2];
  PP_regs[1]    := bp.rf1[3];

/* parameters for the cluster pearl */
  signed bpRF1cap       := 16;
  signed bpRF2cap       := 16;
  signed bpMemCap       := 16384;

  /* instantiation of cluster bpse_pearl with name bp */
  bpse_pearl  bp    < intWidth, bpRF1cap, bpRF2cap, bpMemCap,... >
                    ( coral1.op, sl_ip, sl_ip, sl_ip, st_ip0, st_ip1 );

  /* instantiation of cluster cpse_coral with name coral1 */
  cpse_coral coral1 < parameters... >
                    ( bp.op, sl_ip );

  /* list of outputs of this building block (Cell), only one output */
  sl_op = { bp.sl_op_config, bp.sl_op_pmem,
            bp.sl_op_dmem, coral1.sl_op };
};
```


© 2008 Silicon Hive BV

Bus

TIM keyword: **Bus**

- Busses are generated implicitly: every issue slot output creates a bus
- Bus keyword is used to merge multiple issue slot outputs into a single bus

```
Bus Bus32 (Port32 ip) -> (Port32 op) { Width := 32; }; // declaration

Bus32       out_bus;      // instantiate bus
out_bus    = { is1.op0, is2.op0 };
op          = out_bus;    // op is cluster output
```

- Bus keyword can also be used to give a bus an other name

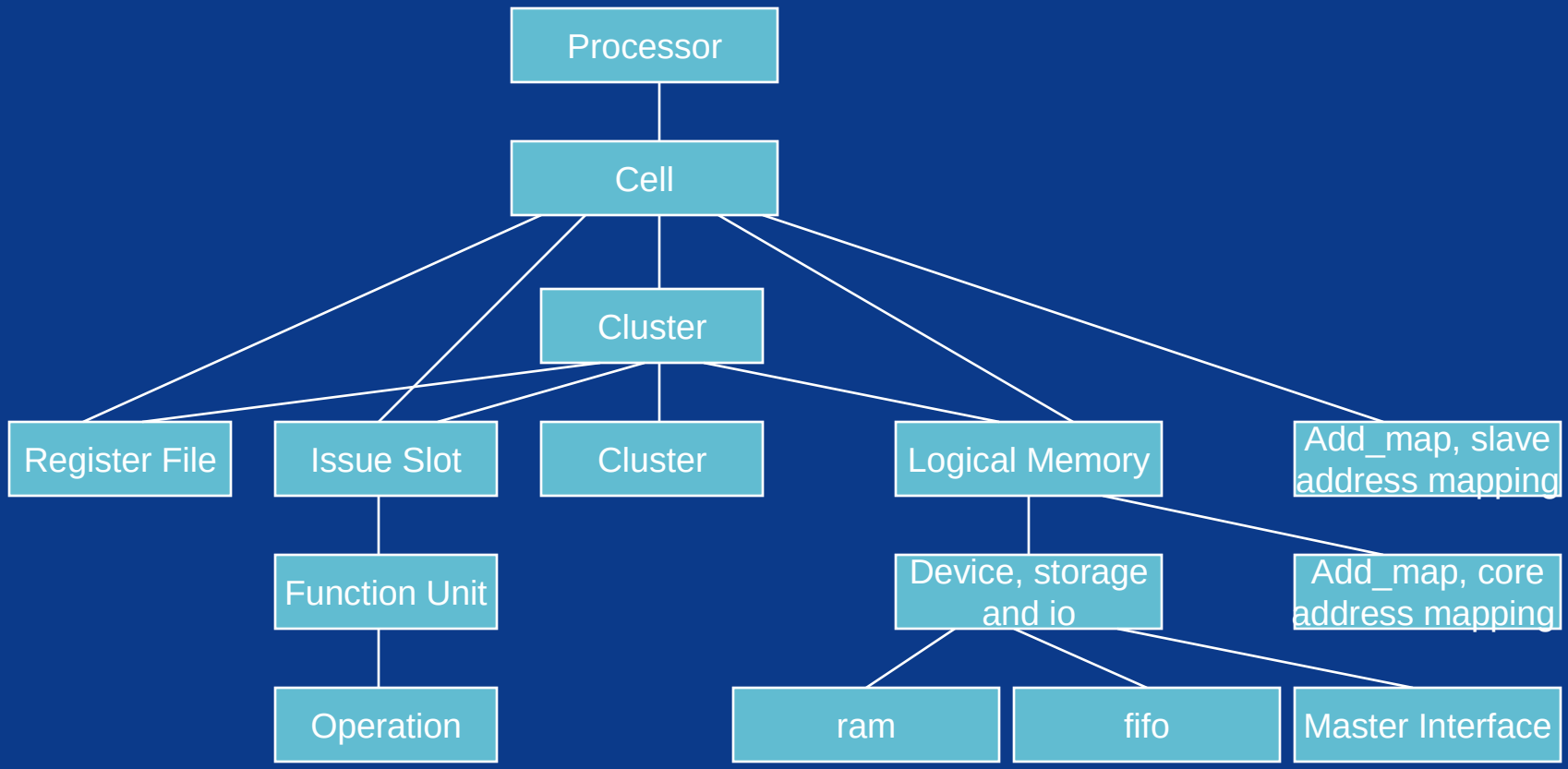
© 2008 Silicon Hive BV

Machine Description

- Processor
- Cell
- **Cluster**
- Issue slot
- Functional unit
- Operation

© 2008 Silicon Hive BV

TIM Hierarchy



Core

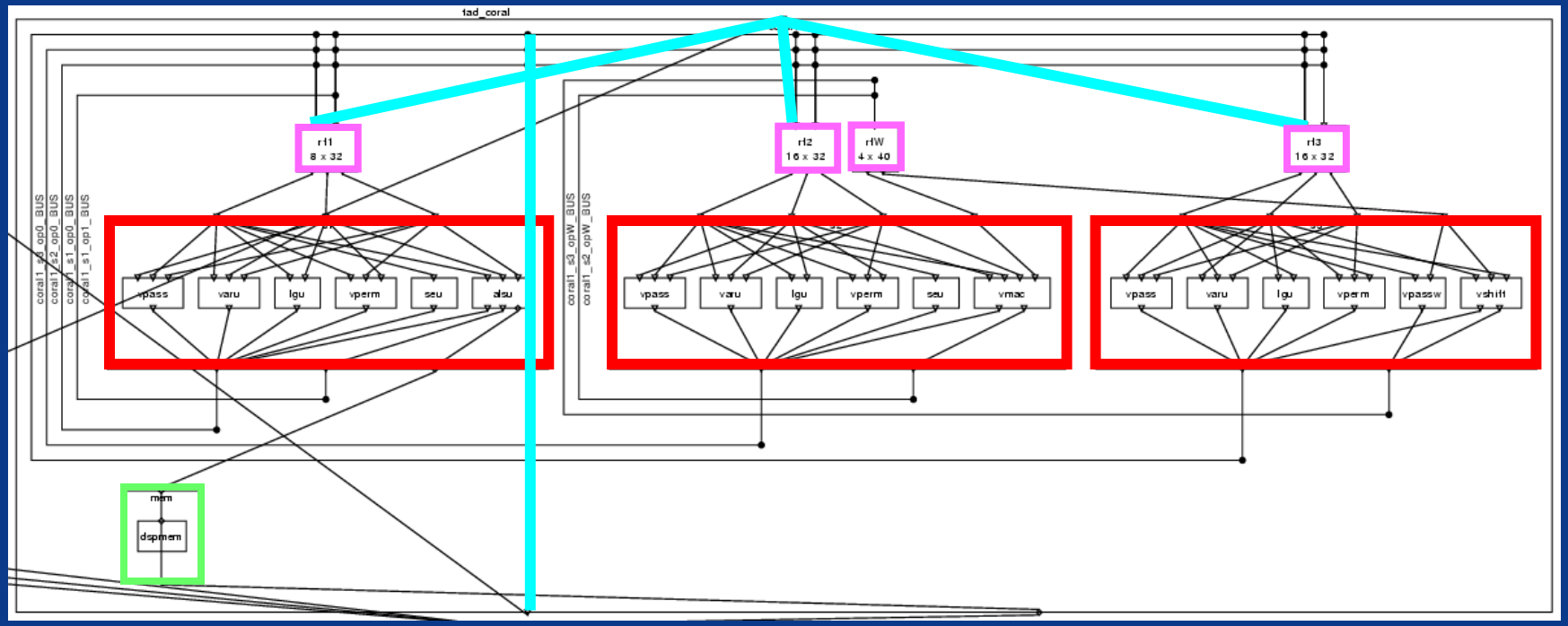
Core IO

© 2008 Silicon Hive BV

Cluster Coral (compute pse)

TIM keyword: **Cluster**

- Optional instantiation of **issue slots**, **register files,** and **logical memories**
- and **interconnect** between them
- and **interconnect** to cluster input and outputs.



© 2008 Silicon Hive BV

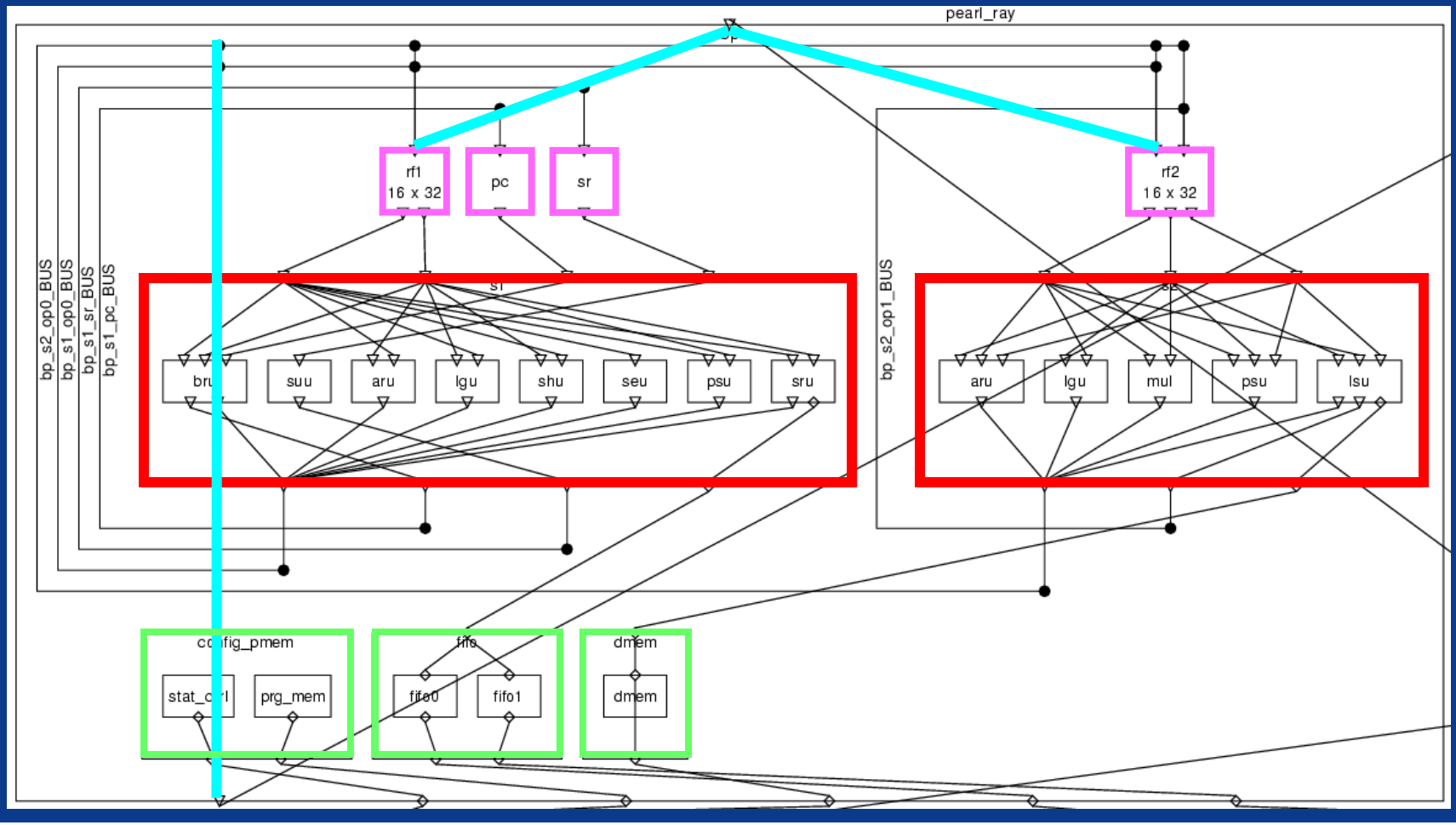
Cluster Coral: TIM code

```
Cluster cpse_coral <   parameters ... >
    ( portW<intWidth> ip, mmioW<intWidth> sl_ip )
 -> ( portW<intWidth> op, mmioW<intWidth> sl_op )
{
#define ALL s1.op0, s2.op0, s3.op0     /* all 1-st outputs */

  /* register file instantiation */
  RFwc2x3          rf1  <intWidth,rf1cap>            ( {ALL, ip}, {ALL, s1.op1} );
  RFwc2x3          rf2  <intWidth,rf2cap>            ( {ALL, ip}, {ALL} );
  RFwc2x3          rf3  <intWidth,rf3cap>            ( {ALL, ip}, {ALL} );
  RFwc1x2          rfW  <llWidth, rfWcap>            ( {s2.opW, s3.opW} );

  /* isue slot instantiation */
  cpse_coral_SLOT1  s1 <intWidth, s1immBits>       ( rf1.op0, rf1.op1, rf1.op2, mem.core_op );
  cpse_coral_SLOT2  s2 <intWidth, llWidth, s2immBits>   ( rf2.op0, rf2.op1, rf2.op2, rfW.op0 );
  cpse_coral_SLOT3  s3 <intWidth, llWidth, s3immBits>   ( rf3.op0, rf3.op1, rf3.op2, rfW.op1 );

  /* logical memory instantiation */
  cpse_coral_LM_mem mem <intWidth, memCap>             (s1.mmo, sl_ip);

  op               = {s1.op0, s2.op0, s3.op0};
  sl_op            = mem.sl_op;
};
```


© 2008 Silicon Hive BV

Cluster Pearl (base PSE)

- Instantiation of **issue slots** and **register files**,
- and **Logical Memories** (incl. config_pmem for a base PSE)
- and **interconnect** between them
- and **interconnect** to cluster input and outputs.



Cluster Pearl: TIM code (1)

```
Cluster bpse_pearl < parameters ... >
    ( portW<intWidth> ip, mmioW<intWidth> sl_ip_config, sl_ip_pmem, sl_ip_dmem, fifoW<ffgWidth> st_ip0, st_ip1 )
 -> ( portW<intWidth> op, mmioW<intWidth> sl_op_config, sl_op_pmem, sl_op_dmem, fifoW<ffgWidth> st_op0, st_op1 )
{
  /* Program counter and status register instantiation */
  RFpc          pc  <pcWidth, branchLatency> (s1.pc);
  RFsr          sr  <branchLatency>          (s1.sr);

  RFwc1x2       rf1 <intWidth,rf1Cap>  ( {s1.op0, s2.op0, ip}                    );
  RFwc2x3       rf2 <intWidth,rf2Cap>  ( {s1.op0, s2.op0, ip},  {s2.op0, s2.op1} );

  bpse_pearl_SLOT1    s1 <intWidth,s1ImmBits,pcWidth,ffgCount,ffgWidth>
                         (rf1.op0, rf1.op1,          pc.rp, sr.rp, fifo.core_op );
  bpse_pearl_SLOT2    s2 <intWidth,s2ImmBits>
                         (rf2.op0, rf2.op1, rf2.op2,              dmem.core_op );
```


© 2008 Silicon Hive BV

Cluster Pearl: tim code (2)

```
  /* logical memory instantiation */
  bpse_pearl_LM_config_pmem config_pmem <intWidth, pmemWidth, pmemCap, statusControlRFcap>
                                                          (sl_ip_config, sl_ip_pmem);
  bpse_pearl_LM_2fifos      fifo        <ffgWidth, ffgCap>  (s1.mmo, st_ip0, st_ip1);
  bpse_pearl_LM_dmem        dmem        <intWidth, dmemCap> (s2.mmo, sl_ip_dmem);

  /* outputs of the cluster */
  op             = { s1.op0, s2.op0 };
  sl_op_config   = config_pmem.sl_op_config;
  sl_op_pmem     = config_pmem.sl_op_pmem;
  sl_op_dmem     = dmem.sl_op;
  st_op0         = fifo.st_op0;
  st_op1         = fifo.st_op1;
};
```


© 2008 Silicon Hive BV

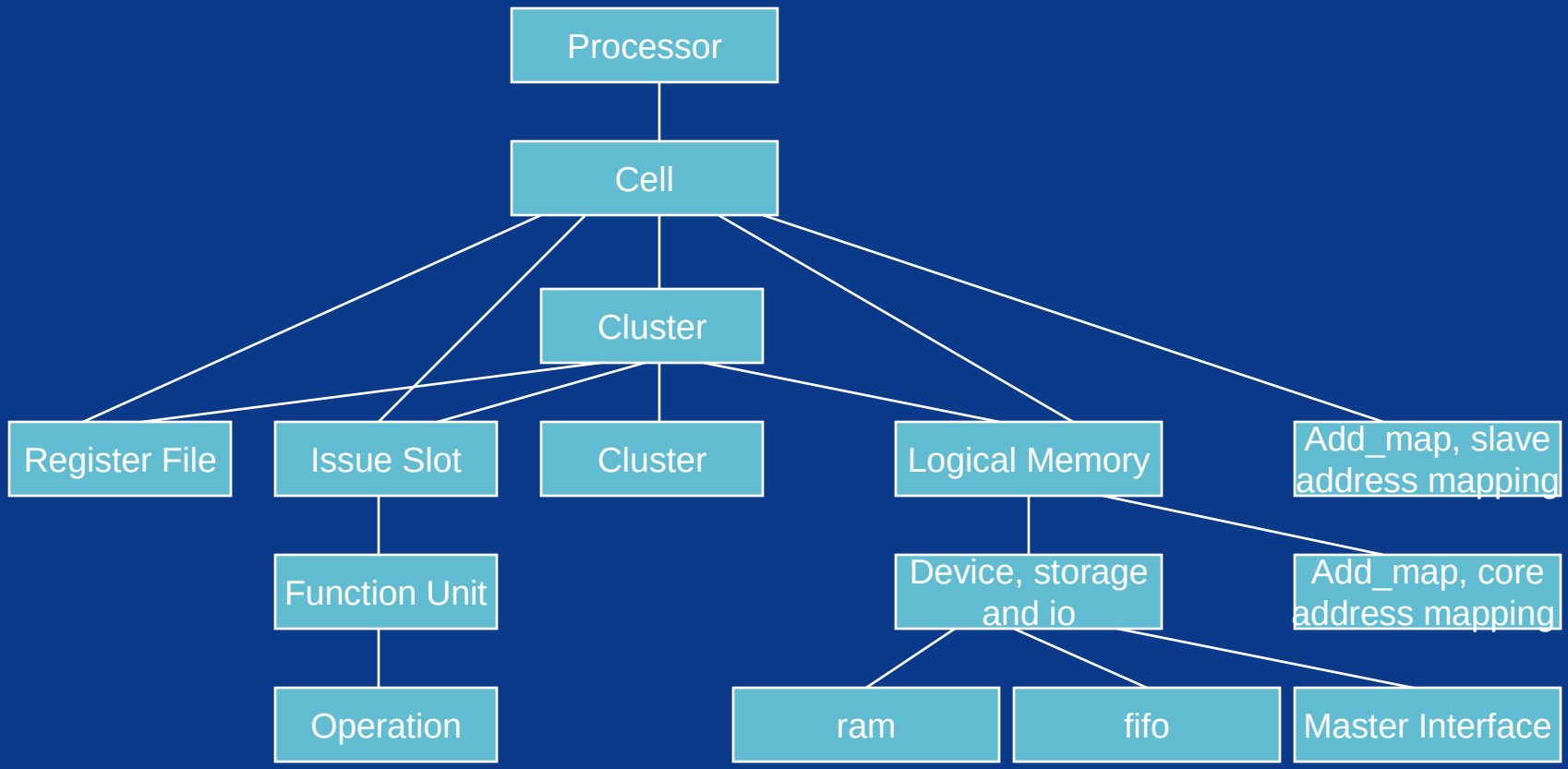
Silicon Hive

Machine Description

- Processor
- Cell
- Cluster
- **Issue slot**
- Functional unit
- Operation

© 2008 Silicon Hive BV

TIM Hierarchy



Core

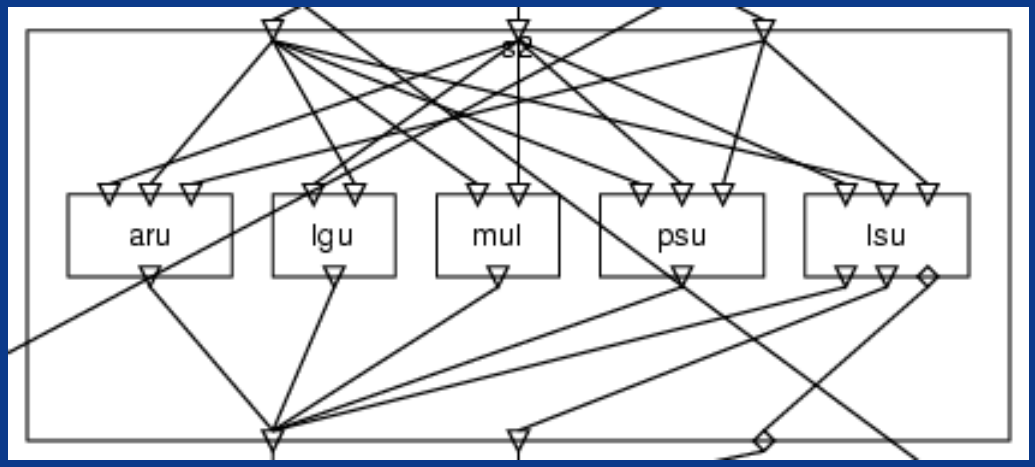
Core IO

© 2008 Silicon Hive BV

Issue slot

tim keyword: **IS**

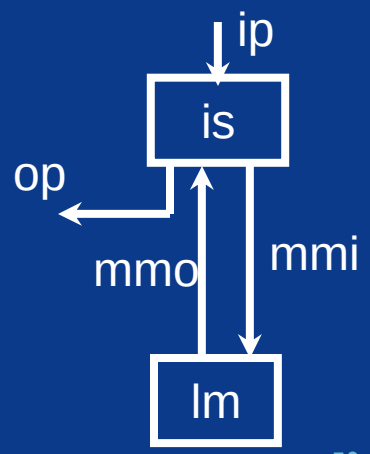
- Instantiation of Functional Units
- input/output connection between issue slot and the functional units it contains



© 2008 Silicon Hive BV

Issue Slot

```
IS bpse_pearl_SLOT2
    < signed intWidth, signed immBits >
    ( portW<intWidth> ip0, ip1, ip2, mmioW<intWidth> mmi )
 -> ( portW<intWidth> op0, op1,      mmioW<intWidth> mmo )
{
  /* configured functional units instantiation */
  cfu_std_ARU_mod_i1        aru <intWidth, immBits>      ( ip1, ip0, ip2 );
  cfu_std_LGU_i1            lgu <intWidth, immBits>      ( ip1, ip0 );
  cfu_std_MPU_i0_L1         mul <intWidth, immBits>      ( ip0, ip1);
  cfu_std_PSU_3in_i0        psu <intWidth, immBits>      ( ip0, ip1, ip2);
  cfu_std_ALSU32_oa_i1_L1   lsu <intWidth, immBits>      ( ip1, ip0, ip2,  mmi);

  /* outputs of the Slot */
  op0 = { aru.op0, lgu.op0, mul.op1, psu.op0, lsu.op0 };
  op1 = { lsu.op1 };
  mmo = lsu.mmo;
};
```



© 2008 Silicon Hive BV

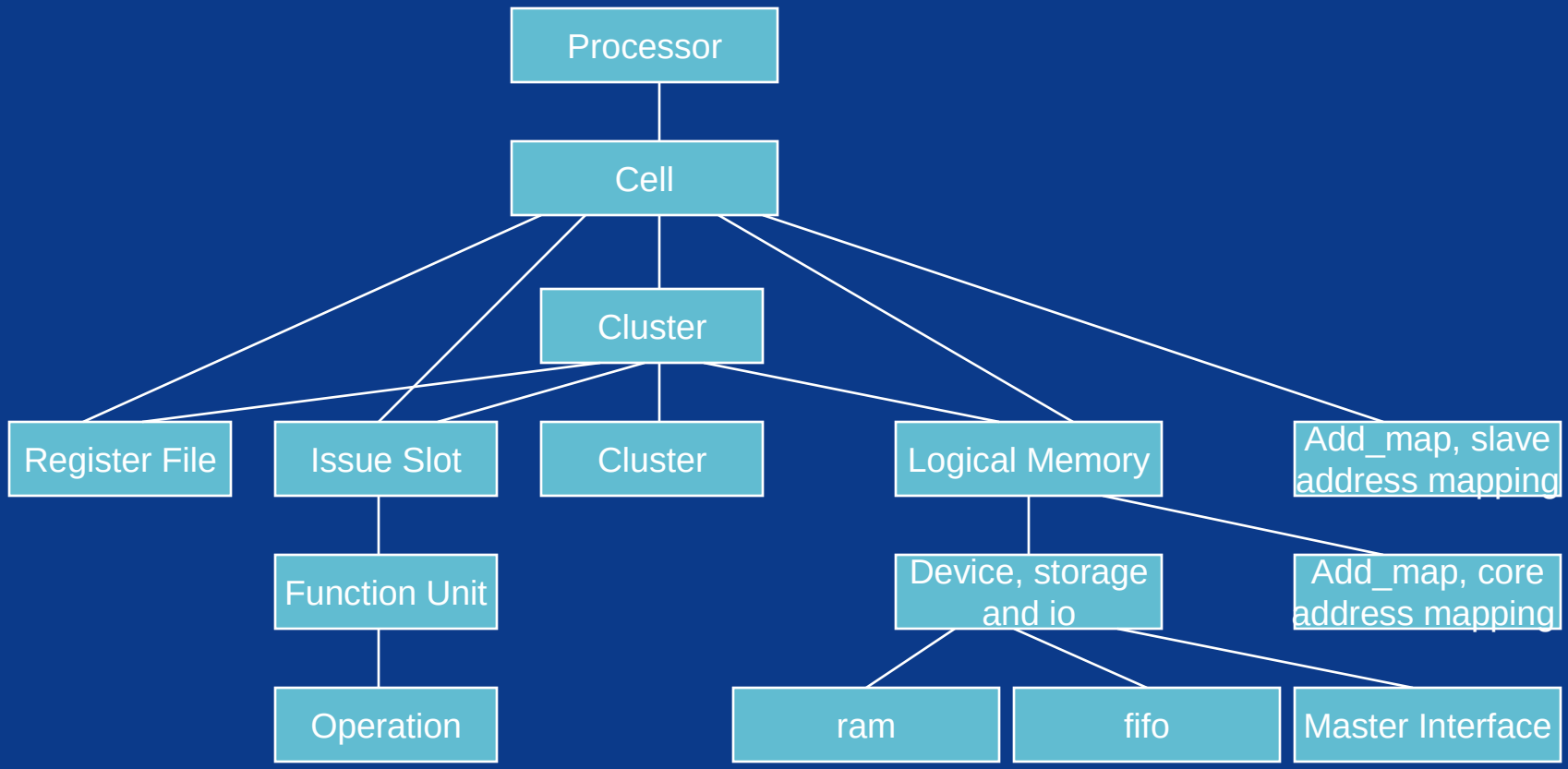
Silicon Hive

Machine Description

- Processor
- Cell
- Cluster
- Issue slot
- **Functional unit**
- Operation

© 2008 Silicon Hive BV

TIM Hierarchy



Core

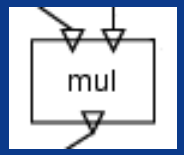
Core IO

© 2008 Silicon Hive BV

Functional Unit

tim keyword: **FU**

- Instantiation of Operations
- input/output connection between functional unit and operations it contains
- Specification of Time Shape



© 2008 Silicon Hive BV

Functional Unit

- **multiply unit (MPU) example**

```
FU STD_MPU
    ( Port ~ ip0, Port ~ ip1 )
 -> ( Port op1 )
{
  freeze(Cycle);  // do not allow timeshape to be changed in derived fu-s

  /* time shape */
  Cycle[0] := {ip0, ip1};
  Cycle[1] := {op1};

  /* operations available in this functional unit */
  std_mul_u:    op1 =  std_mul_u     (ip0, ip1);
  std_mul:      op1 =  std_mul       (ip0, ip1);
  std_mulsat_u: op1 =  std_mulsat_u  (ip0, ip1);
  std_mulsat:   op1 =  std_mulsat    (ip0, ip1);
};
```


© 2008 Silicon Hive BV

Functional Unit

- **load store unit (LSU) example**

```
FU STD_LSU
    ( Port ~ ip0, Port ~ ip1, Port ~ ip2, Port ~ mmi )
 -> ( Port op0,                           Port   mmo )
{
  std_ld:     mmo, op0      = std_ld     (ip0, mmi);
  std_st:     mmo           = std_st     (ip0, ip2);
  std_ldo:    mmo, op0      = std_ldo    (ip0, ip1, mmi);
  std_sto:    mmo           = std_sto    (ip0, ip1, ip2);
};
```

- The mmi port is connected to the read data port of a logical memory
- The mmo port is connected to the address & write data port of a logical memory
- mmi and mmo have special port type (`Properties := MMIO;`)

© 2008 Silicon Hive BV

Functional Unit, Configured

- A functional unit can be configured by deriving it from an other functional unit

```
STD_MPU cfu_std_MPU_imm8
  ( portW<intWidth> ip0, ip1 ) -> ( portW<intWidth> op1 )
{
  use
  {
    std_mul;
    std_mul_u;
    std_mul_i:  op1 =  std_mul   (Immediate(ip0,[(-128),..,127]), ip1);
    std_mul_iu: op1 =  std_mul_u (Immediate(ip0,[0,..,255]), ip1)
  };
};
```


© 2008 Silicon Hive BV

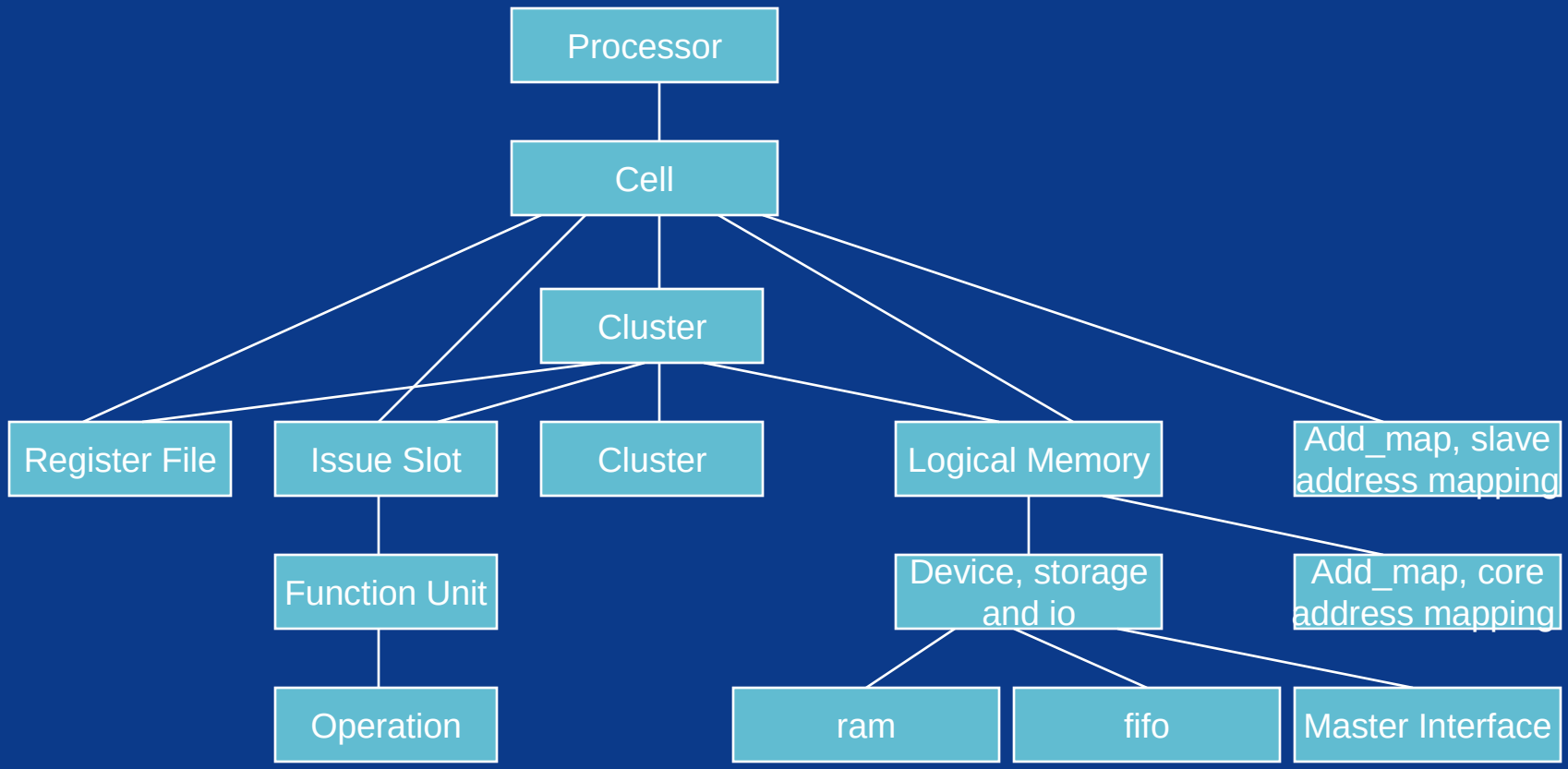
Silicon Hive

Machine Description

- Processor
- Cell
- Core
- Cluster
- Issue slot
- Functional unit
- **Operation**

© 2008 Silicon Hive BV

TIM Hierarchy



Core

Core IO

© 2008 Silicon Hive BV

Operation Semantic

- tim keywords: **OP** and **SEM**

- Specification of semantic of operation.
- Optionally information for documentation generation can be added

- semantics are described using expressions and statements
  following examples show some possibilities

© 2008 Silicon Hive BV

Operation Semantics

- scalar multiplication example

```
OP std_mul (Signed A, B) -> (Signed R)
{
  SEM R (A,B) =
  {
    R = (sR)A * (sR)B;
  };
  <DOC>
    <SHORT> Signed multiplication </SHORT>
    <SEM> R = A*B </SEM>
    <DESCRP> This operation returns the signed product of the arguments A
   and B. </DESCRP>
  </DOC>;
};
```

Where `sR` is defined as: `signed<Width:=(Width(R))>`

© 2008 Silicon Hive BV

Operation semantics

- vector multiplication example

```
OP vec_mul <signed nway>
    (svecN<nway> A, B) -> (svecN<nway> R)
{
  SEM R <nway> (A,B) =
  {
    /* loop over each vector element */
    for i [(nway-1), 0]
    {
      R[i] = (sRi)A[i] * (sRi)B[i];
    }
  };
};
```


© 2008 Silicon Hive BV

Operation semantics

- semantic call example

```
OP vec_mul <signed nway>
    (svecN<nway> A, B) -> (svecN<nway> R)
{
  SEM R <nway> (A,B) =
  {
    /* loop over each vector element */
    for i [(nway-1), 0]
    {
      /* semantic call */
      R[i] = (sRi)std_mul.R( A[i], B[i] );
    }
  };
};
```


© 2008 Silicon Hive BV

Operation semantics

- more complex example

```
OP std_asrrnd   (Signed A, no_sign B ) -> (Signed R)
{
  SEM R(A,B) =
  {
    sRPlus t;              // temporary variable
    uB b = (unsigned)B;       // temporary variable that get's value of
   input B assigned

    if b==0
    then { t = (sRPlus)A; }
    elif  (A&((2<<b)-1)) == (1<<(b-1))
      then { t = (sRPlus)(A >> b); }
      else { t = (sRPlus)(((sAPlus)A + (sAPlus)(1<<(b-1))) >> b); }
    fi
    R = (sR) t;
  };
};
```

Where `sRPlus` is defined as: `signed<Width:=(Width(R)+1)>`

© 2008 Silicon Hive BV

expression operators

- List of supported operators
  (most of them are C like)

Unary

- `+, -, !, ~`

Binary

- `&&, ||, ==, !=, <, <=, >, >=`
- `&, |, ^`
- `+, -, *, /, %, <<, >>`
- `++ (concatenation)`

TIM checks if the operands are of the proper type

© 2008 Silicon Hive BV

Port types

- **port types for units, examples:**

```
Port     portW      <signed w> { Width := w; };
Port     portPC     <signed w> { Width := w; };
Port     portSR                { Width := 9; };

Port     mmioW      <signed w> { Width := w; Properties := MMIO; AddressWidth := 32; };
mmioW    mmioW_DTL             { Protocol := DTL; };
mmioW    mmioW_CIO             { Protocol := CIO; };
mmioW    mmioW_AHB             { Protocol := AHB; };
```

- **port types for semantics, examples:**

```
Port     no_sign               { Float  := 0; };
no_sign  Signed                { Signed := 1; };
no_sign  Unsigned              { Signed := 0; };
Port     vecN     <signed nway> { Packed := nway; };
Port     svecN    <signed nway> { Packed := nway; Signed := 1; };
Port     uvecN    <signed nway> { Packed := nway; Signed := 0; };
```

- To enhance reuse semantic do not set the width property
- The signess, and packedness property should not be set for a building block

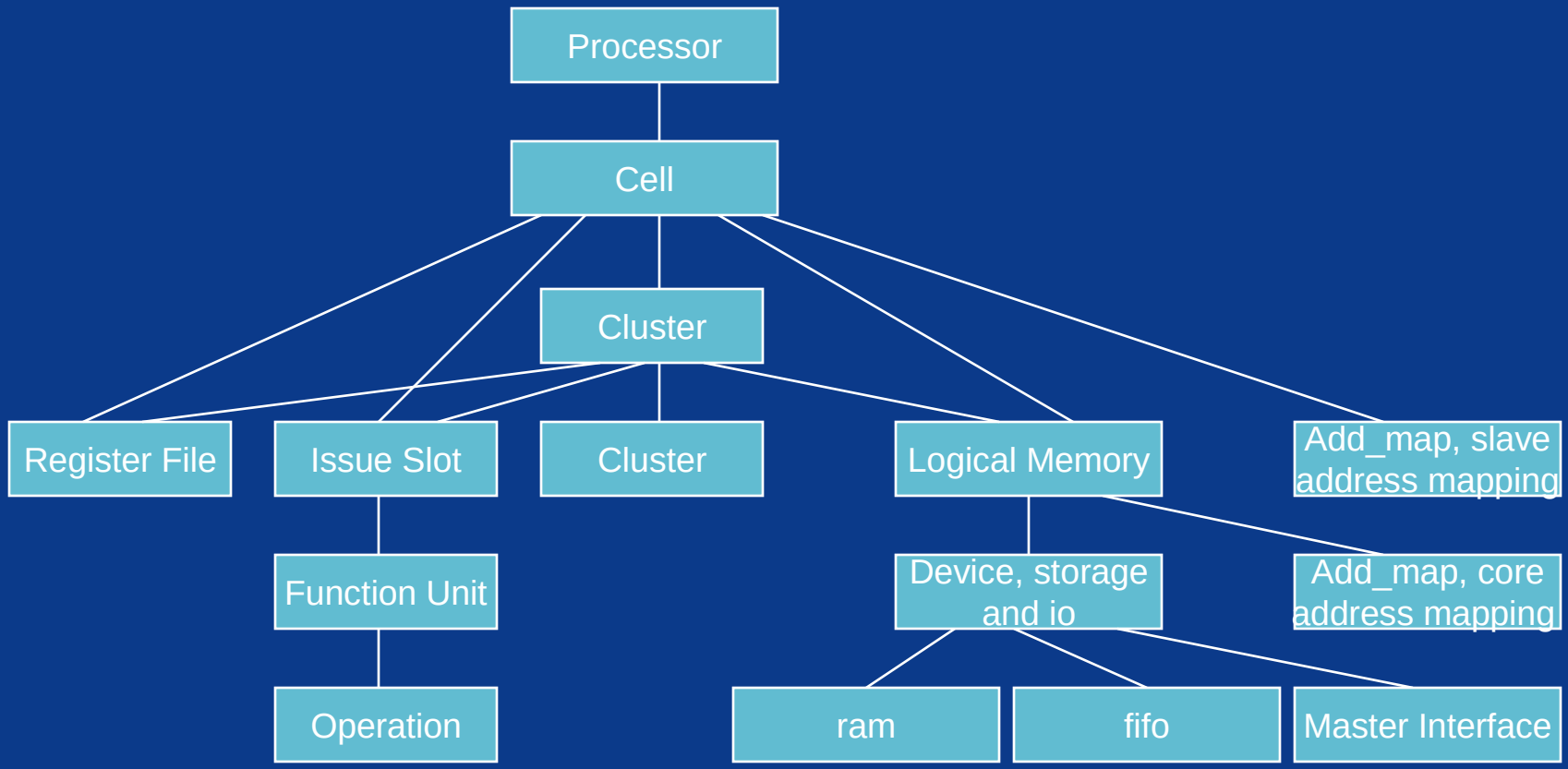
© 2008 Silicon Hive BV

Silicon Hive

Core IO

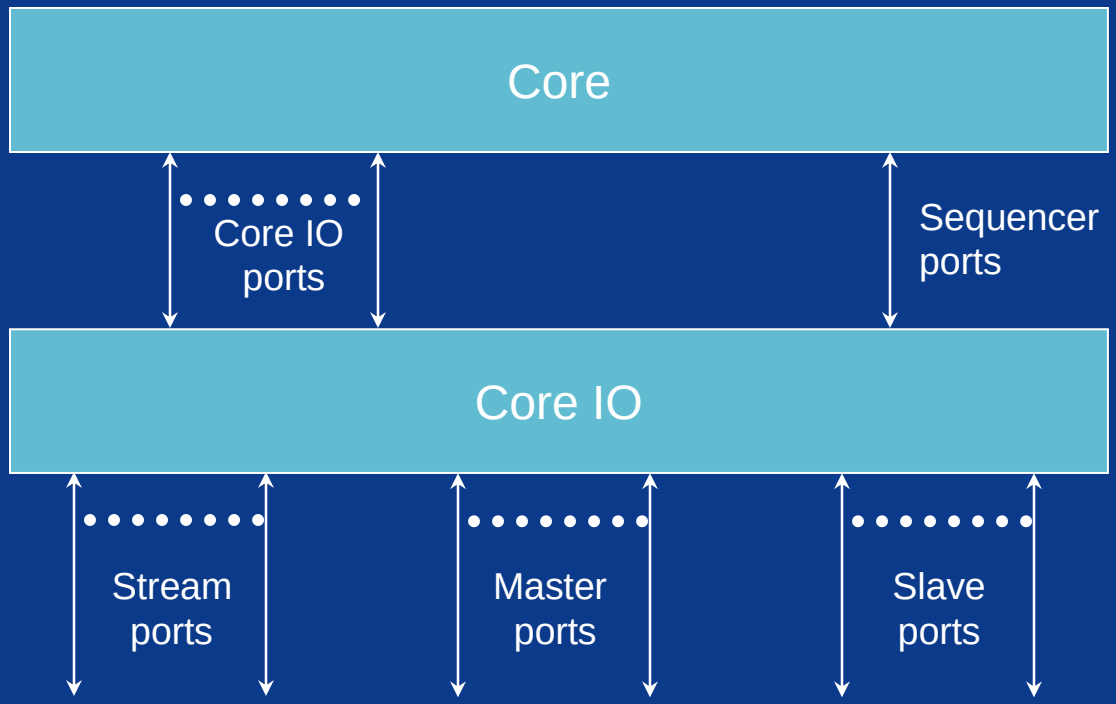
© 2008 Silicon Hive BV

TIM Hierarchy



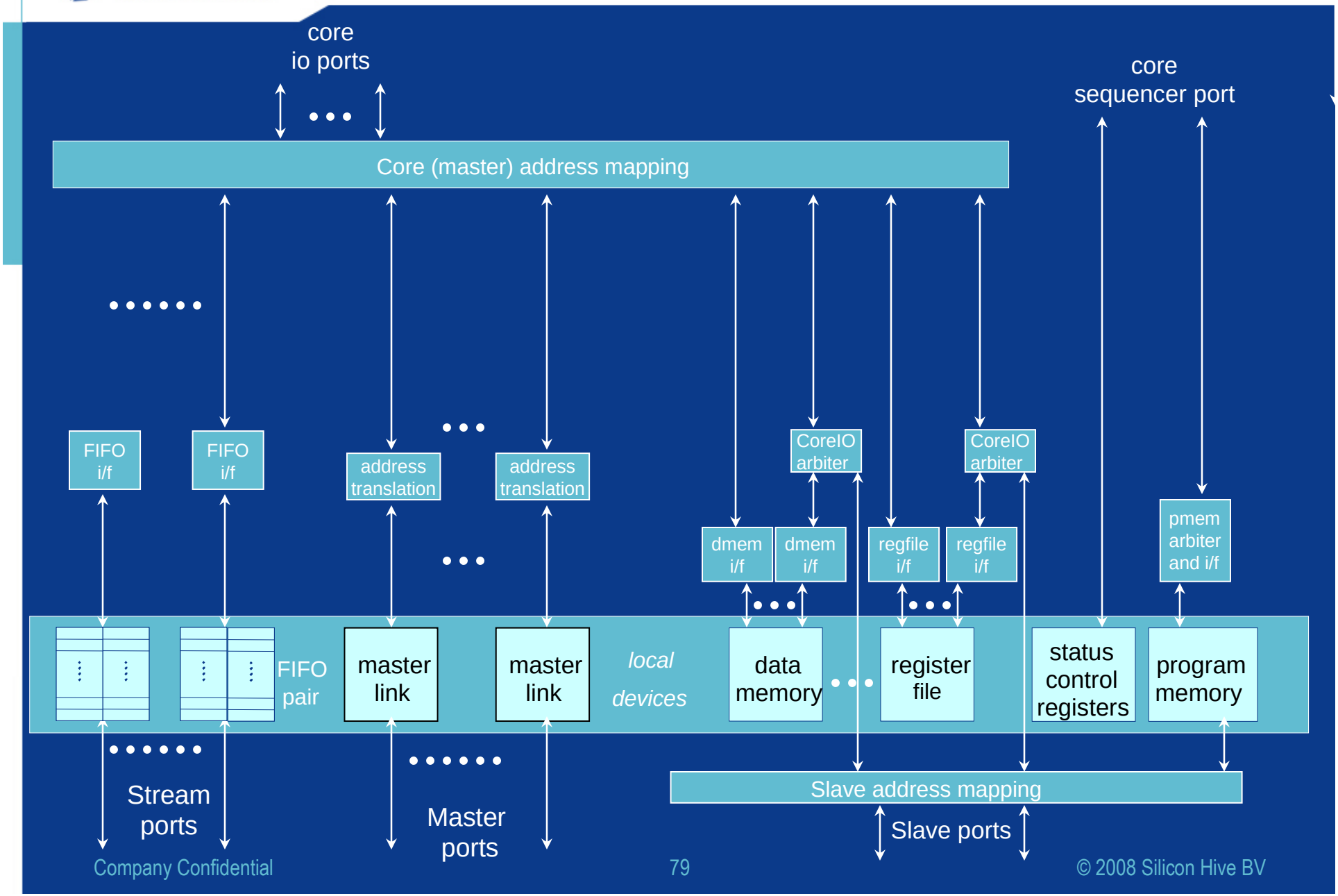
© 2008 Silicon Hive BV

Core & Core IO



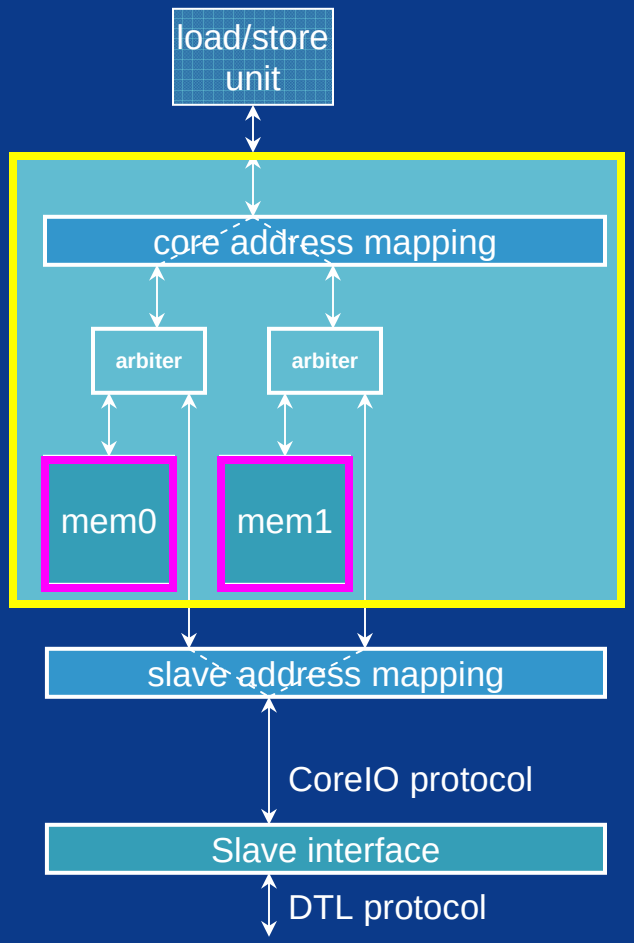
© 2008 Silicon Hive BV

Core IO



© 2008 Silicon Hive BV

Logical Memory (ram)



or slave routing network (SRN)
is in Cell level, not in Logical_memory

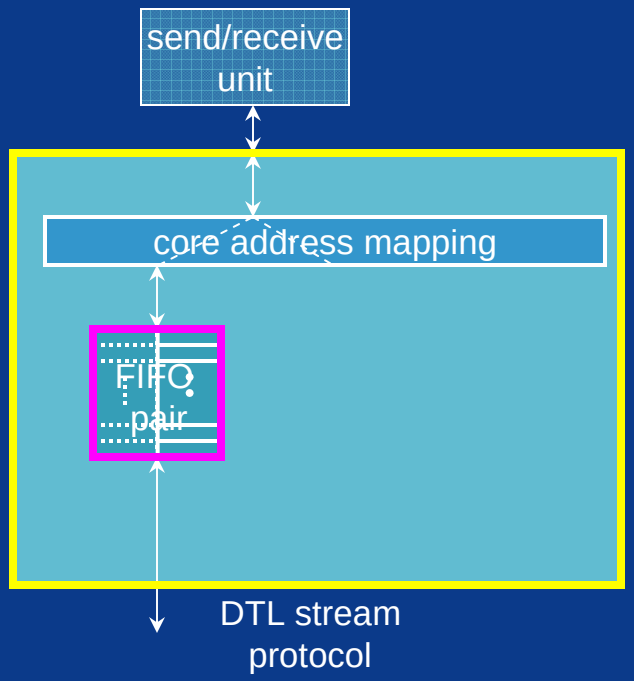
© 2008 Silicon Hive BV

Logical Memory (ram)

```
Device dev_dmem_arb <signed width, signed memCap>
    ( mmioW_p_in<width> wp )
 -> ( mmioW<width>      rp )
{
  Width := width;
  Capacity := memCap;
  Kind := ram;
  Latency := 1;
};

Logical_memory lm_2mem_sl <signed width, signed memCap>
    ( mmioW_p_out<width> core_ip, sl_ip0, sl_ip1 )
 -> ( mmioW<width>       core_op, sl_op0, sl_op1 )
{
  dev_dmem_arb  mem0 <width, memCap> ({ core_ip, sl_ip0 });
  dev_dmem_arb  mem1 <width, memCap> ({ core_ip, sl_ip1 });

  core_op     = { mem0.rp, mem1.rp };
  sl_op0      = mem0.rp;
  sl_op1      = mem1.rp;
};
```


© 2008 Silicon Hive BV

Logical Memory (fifo)



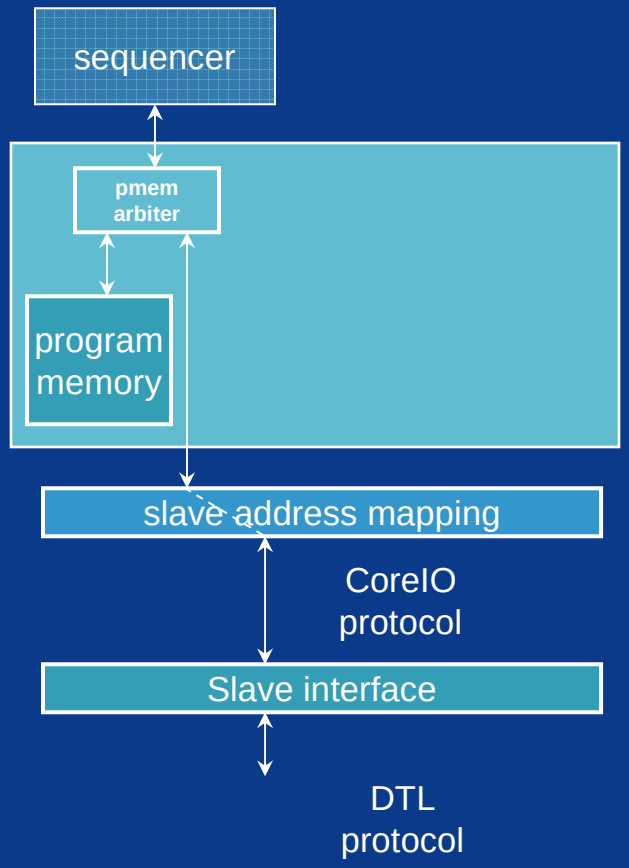
© 2008 Silicon Hive BV

Logical Memory (fifo)

```
Device fifo_2way <signed fifoWidth, signed fifoCap>
    (fifoW<fifoWidth> ip0, ip1) -> (fifoW<fifoWidth> op0, op1)
{
  Width := fifoWidth;
  Capacity := fifoCap; // minumum is 2. In words of Width bits
  Kind := fifo;
  Latency := 0;
};


Logical_memory lm_1fifo <signed ffWidth, signed ffCap>
    (fifoW<ffWidth> core_ip, st_ip) -> (fifoW<ffWidth> core_op, st_op )
{
  fifo_2way     fifo0 <ffWidth, ffCap> ( core_ip, st_ip );

  // note: at these output assignments the "wiring is crossed" deliberately:
  st_op       = fifo0.op0;
  core_op     = fifo0.op1;
};
```


© 2008 Silicon Hive BV

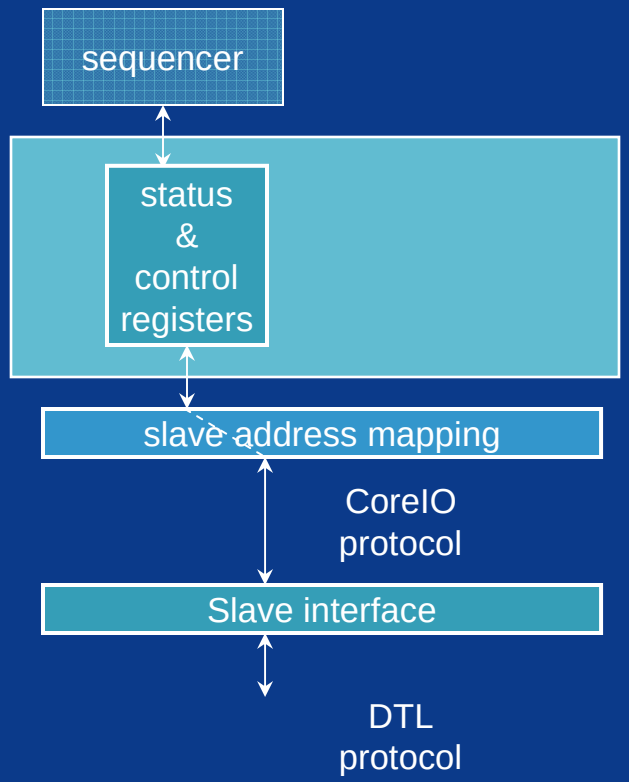
Logical Memory (program mem)



adress mapping is in Cell level

© 2008 Silicon Hive BV

Logical Memory (stat&ctrl)



adress mapping is in Cell level

© 2008 Silicon Hive BV

Address Mapping

- general syntax:

```
Add_map (inputs) -> (outputs)
{
    output_list = input_list [ <start_address> , .. , <end_address> ];
};
```

- example:

```
Add_map mrn_data_mem (MMIO32 ip0) -> (MMIO32 op0, MMIO32 op1)
{
    op0 = ip0 [0x00000, .., 0x03FFF];
    op1 = ip0 [0x04000, .., 0x07FFF];
};
```


© 2008 Silicon Hive BV

Device

- general syntax:

```
Device ( input ) -> ( output )
{
    Width = expr;
    Capacity = expr;
    Kind = [ram | rf_ram | master_int | stat_ctrl | prg_mem | fifo];
    Latency = expr;
    /* for master interface only: */
    BaseAddrs = expr;
    MaxBurstSize = expr;
    Protocol = expr;
};
```


© 2008 Silicon Hive BV

tim file structure

To get some structure in the files for a core, a general partition of the tim source file could look like:

- corename.tim — processor and cell
- corename_is.tim — issue slots
- corename_fu.tim — (derived) functional units (if not from lib)
- corename_lm.tim — logical memories

If the core is partitioned with clusters the tim source code for each cluster could be partioned the same way.

© 2008 Silicon Hive BV

Silicon Hive

PBB library

© 2008 Silicon Hive BV

PBB library: overview

PBB: Processor Building Blocks

- contains course to fine grain granularity building blocks described in TIM
- use course blocks for quick starting point
- use fine grain blocks for more control/optimization

- The building blocks become available using #include in your core files
- They are self-contained: the blocks themselfs include the lower level blocks they need.

© 2008 Silicon Hive BV

PBB library: structure (1)

directory structure (located in $HIVEBIN/../include)

```
pbb
+-- pse                 processing and storage elements
    +-- bpse            base pse
    +-- cpse            compute pse
+-- pe                  processing elements
    +-- bpe             base pe
    +-- cpe             compute pe
+-- lm                  logical memories
+-- fu_configured       configured functional units
+-- fu                  base functional units
+-- semantics
    +-- std             standard (scalar) operation semantics
    +-- vec             vector operation semantics
    +-- types           semantic-port types and declaration defines
+-- types               bb-port types and register file types
```


© 2008 Silicon Hive BV

PBB library: structure (2)

subdirectories:

[fu_configured | fu | semantics]

- **+-- std**
- **+-- vec**
- **+-- vec_v** (not in semantics)
- **+-- asp**

and more (depending on delivery what is in)

file name structure examples in **fu_configured/std** and **semantics/std**:

- **sem_std_arith.tim**
- **sem_std_io.tim**
- **sem_std_mult.tim**

and more

© 2008 Silicon Hive BV

PBB library: PSEs

There are four PSEs in pbb/pse:

- tad — minimal base pse,
  1 issue slot, no multiplier
- pearl — more performance, supports single-cycle copy loop,
  2 issue slot pse, auto increment LSU
- ray — LSU with master interface,
  1 issue slot, also with arithmetic and logic
- coral — pse with typical DSP characteristics, 3 issue slots,
  mac with wide acc reg., saturation, rounding, simd4, simd2

- pbb/pe contains the processing part of the PSEs (non-storage part).
  This allows freedom in connecting different memory configuration.

© 2008 Silicon Hive BV

PBB library: usage

examples of pse usage (instantiate at Cell level)

```
#include <pbb/clusters/bpse/tad/bpse_pearl.tim>
#include <pbb/clusters/cpse/coral/cpse_coral.tim>

/* set parameters for base pse pearl */
signed bpRF1cap       := 32;
signed bpRF2cap       := 16;
more parameters...

bpse_pearl  bp    < bpRF1cap, bpRF2cap, more parameters... >
                  ( coral1.op, other inputs );

/* set parameters for compute pse coral */

cpse_coral coral1 < parameters... >
                  ( bp.op, other inputs );
```


© 2008 Silicon Hive BV

Silicon Hive

Operation name scheme

© 2008 Silicon Hive BV

Operations vs. semantic

Difference between operation and semantic:

- An operation is visible to the programmer of the core (semantic not)

A configured functional unit instantiates **operations** using **semantics**:

```
std_pass:    op0 =  std_pass  ( ip0 );
std_imm:     op0 =  std_pass  ( Immediate(ip0, [(-128),..,127]) );

std_mul_u:   op1 =  std_mul_u ( ip0, ip1 );
std_mul_iu:  op1 =  std_mul_u ( Immediate(ip0,[0,..,255]), ip1 );

std_mac:     op0 =  std_mac    ( ip0, ip1, ip2 );
std_mac_w:   opW =  std_mac    ( ip0, ip1, ipW );
```


© 2008 Silicon Hive BV

Operation naming scheme

An operation name can have 4 fields:

- namespace_operationname_modifiers_postfix

namespace	std, vec, cpx, gal, sfp, dfp, asp or "custommer string"
operationname	indicates function of operation imm, pass, add, mul, st ... *rnd, *sat, *oi ... ← some standarization strings
modifiers	indicate changes w.r.t. the default one i, u, s, e, t, o#, v# ...
postfix	free format string (use when none of other fiels apply)

Examples:

std_st160i
std_mulsat_iu
std_mul_o2_bw
vec_asr_cv8

© 2008 Silicon Hive BV

Semantics naming scheme

A semantic can serve multiple operations,

therefore in semantic:

- i, w, v8, v16 are not used
- e, t are only used sometimes
- “__” (modifier field empty) must be substituted with “_x_”

© 2008 Silicon Hive BV

Silicon Hive

www.siliconhive.com

© 2008 Silicon Hive BV