SYSTEM-LEVEL MODELING OF DYNAMICALLY RECONFIGURABLE HARDWARE WITH SYSTEMC

Antti Pelkonen, VTT Electronics

Kostas Masselos, INTRACOM SA

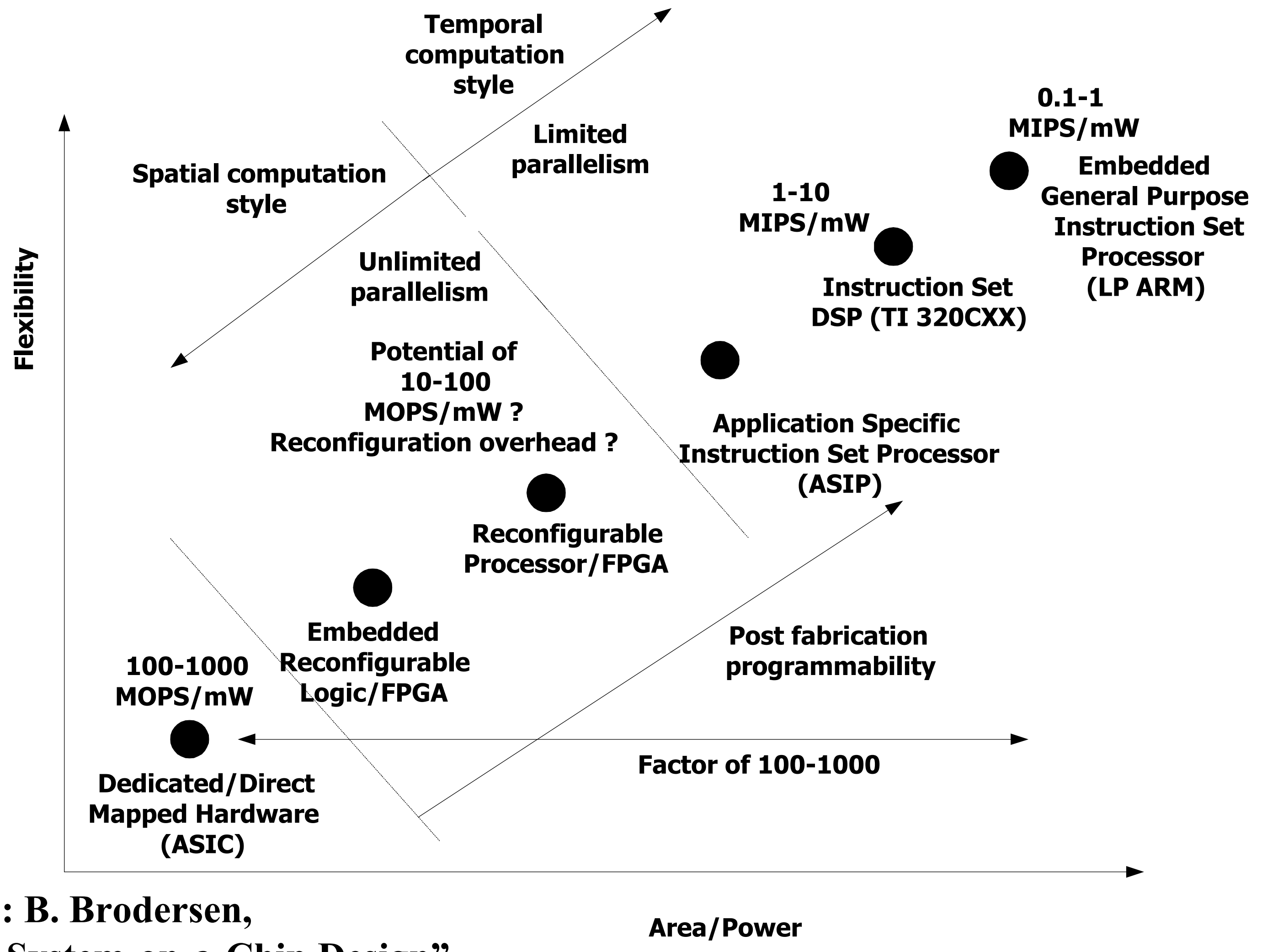
Miroslav Cupák, IMEC

OUTLINE

- Background
- Implementation technologies
- The ADRIATIC design flow
- DRCF modeling principles and tools
- Future work
- Conclusions

For the rest of the presentation:
DRCF = Dynamically ReConfigurable Fabric

BACKGROUND


Temporal computation style
Spatial computation style
Limited parallelism
Unlimited parallelism
0.1-1 MIPS/mW
Embedded General Purpose Instruction Set Processor (LP ARM)
1-10 MIPS/mW
Instruction Set DSP (TI 320CXX)
Flexibility
Potential of 10-100 MOPS/mW ?
Reconfiguration overhead ?
Application Specific Instruction Set Processor (ASIP)
Reconfigurable Processor/FPGA
Embedded Reconfigurable Logic/FPGA
Post fabrication programmability
100-1000 MOPS/mW
Dedicated/Direct Mapped Hardware (ASIC)
Factor of 100-1000
Area/Power

Reference: B. Brodersen, "Wireless System-on-a-Chip Design", http://bwrc.eecs.berkeley.edu/

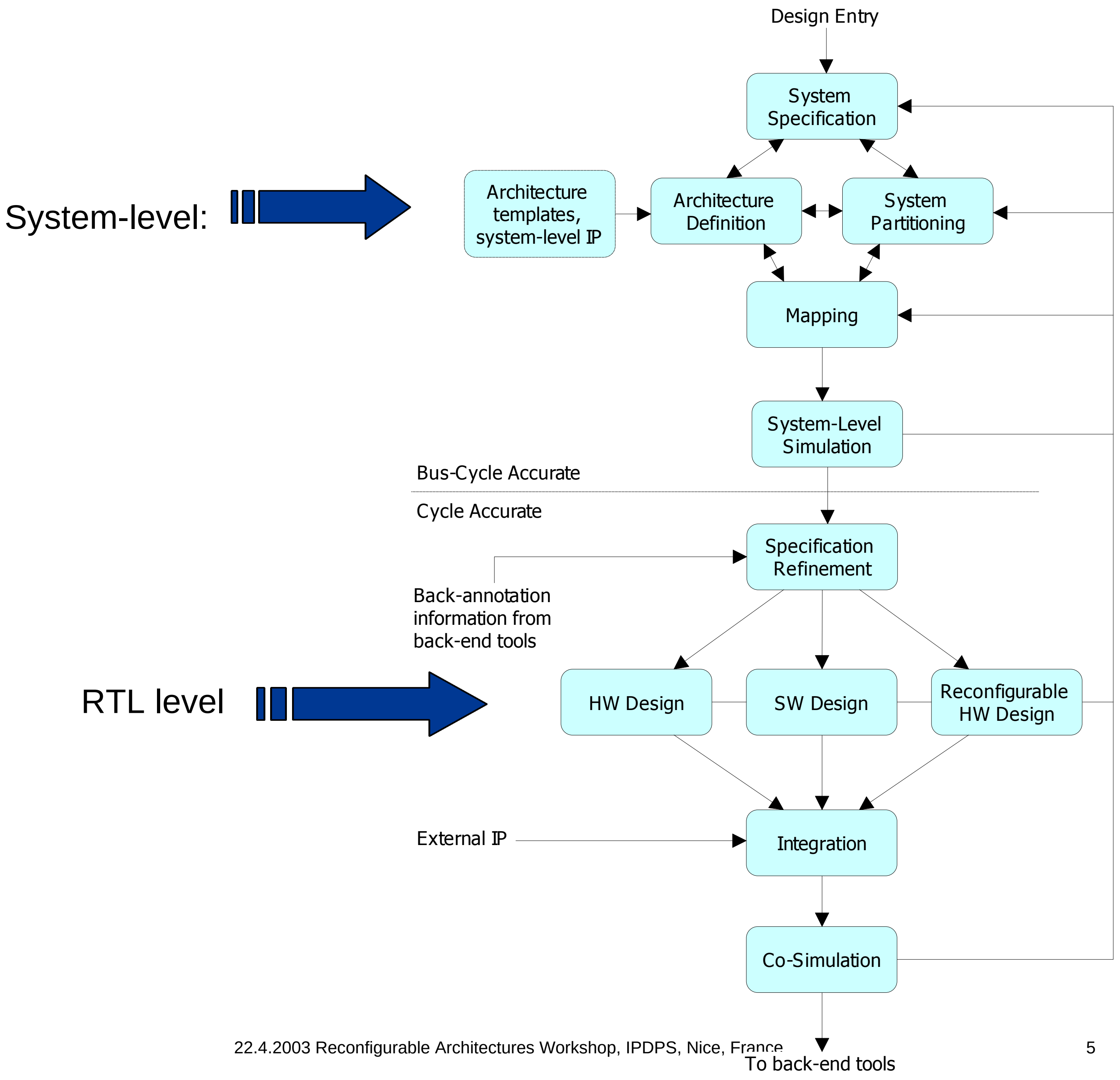
RECONFIGURABLE IMPLEMENTATION TECHNOLOGIES

Classification:

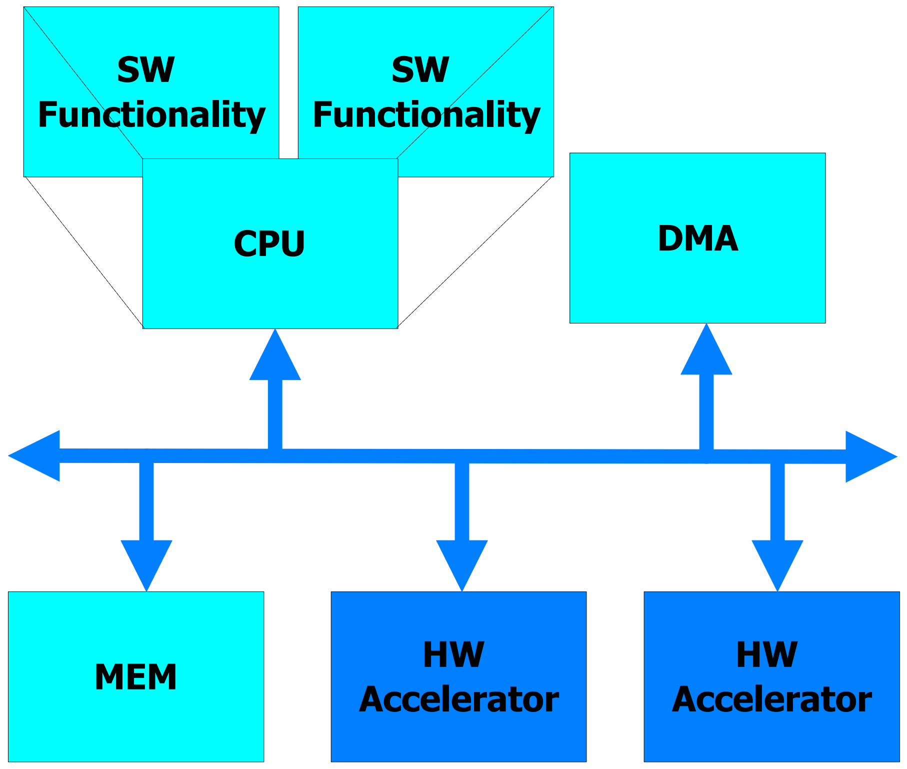
- **Reconfiguration scheme**
  - static, dynamic
- **Coupling**
  - in data-path of a processor, independent coprocessor
- **Granularity**
  - bit-oriented, word oriented

Technologies:

- **System-level FPGA**
  - Xilinx, Altera etc.
- **Embedded FPGA**
  - Actel, Leopard Logic etc.
- **Arrays of processing elements**
  - MorphoSys, PACT etc.


Design Entry
System Specification
System-level:
Architecture templates, system-level IP
Architecture Definition
System Partitioning
Mapping
System-Level Simulation
Bus-Cycle Accurate
Cycle Accurate
Specification Refinement
Back-annotation information from back-end tools
RTL level
HW Design
SW Design
Reconfigurable HW Design
External IP
Integration
Co-Simulation
To back-end tools

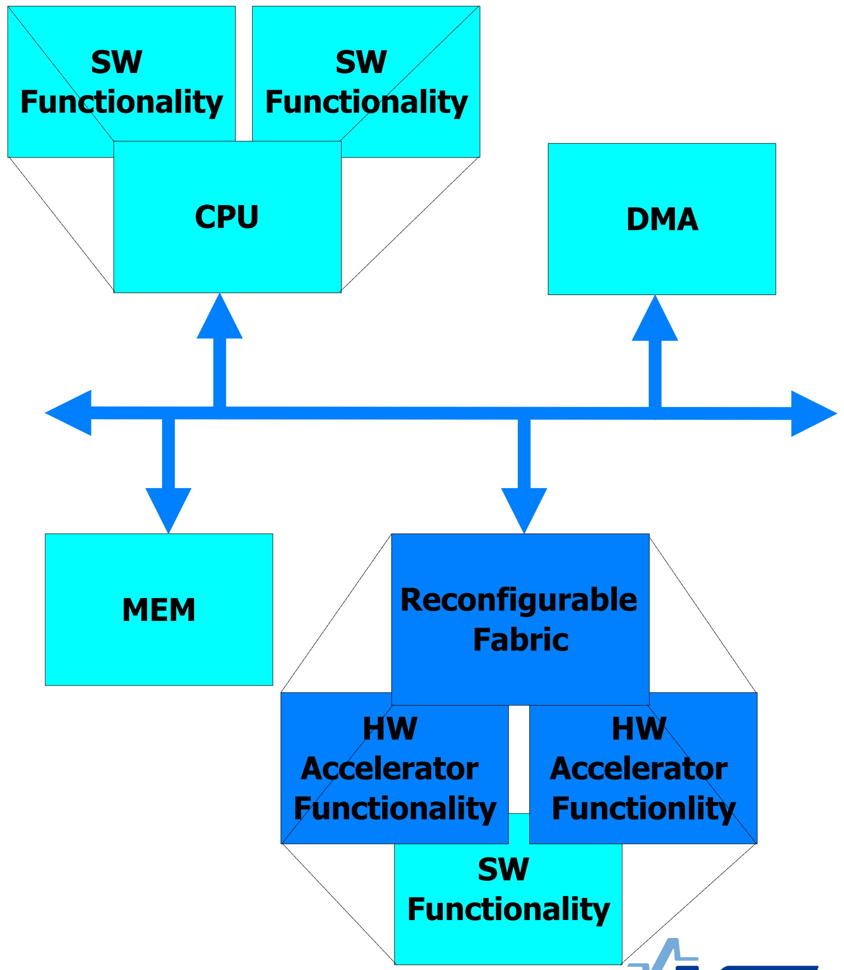
DRCF MODELING PRINCIPLES 1/2: What to model with DRCF



- Typical architecture
- Some software processes, some hardware processes
- What to implement with DRCF...
- **HW processes**?
  - Small granularity DRFC similar to ASIC or FPGA gate
  - Estimation, design and modeling similar
- **SW processes**?
  - Large granularity DRCF similar to simple processor ALU
  - No generalized methods for estimation and modeling

DRCF MODELING PRINCIPLES 2/2
How to model DRCF

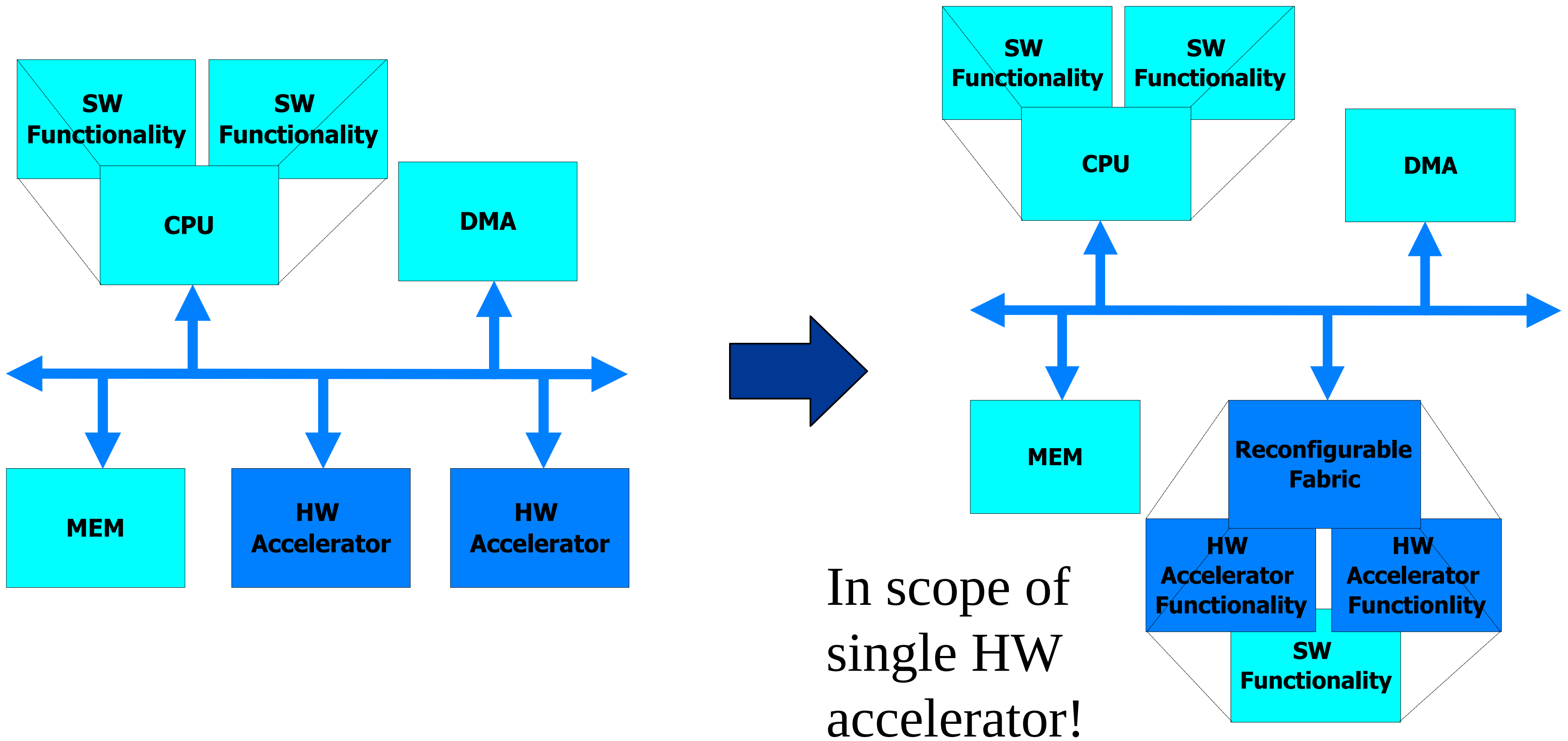
- DRCF contains contexts which implement the functionality
- In the beginning, reconfiguration automatic
    - does not require support from architecture
    - Relies on information on re-configuration speed etc.
- At later and more detailed levels
    - re-configuration process initiated by DMA or CPU
    - Contexts allocated to memory
    - Real bus traffic
- When implementation technology is chosen, co-simulation with vendor back-end



DRCF MODELING WITH SystemC

- DRCF implemented as hierarchical module
  - instrumentation and management processes
  - General DRCF interface (if possible) or collection of all interfaces of included models
- Contexts implemented inside hierarchical module
  - a wrapper to module that is simulated as a member functionality in DRCF
- Does not require any change of code (as long as…)
  - Bus interfaces are “suitable”
  - Non-blocking or splittable bus interfaces

DRCF MODELING EXAMPLE



In scope of single HW accelerator!

DRCF MODELING WITH SystemC 1/7: Analysis of component

HWACC.H:
```
class hwacc : public sc_module,
              public bus_slv_if
{
 public:
  sc_in_clk clk;
  sc_port<bus_mst_if> mst_port;
  …
```

2. Analyze interfaces

1. Analyze ports

BUS_SLV_IF.H
```
class bus_slv_if : public virtual sc_interface
{
 public:
  virtual sc_uint<ADDW> get_low_add()=0;
  virtual sc_uint<ADDW> get_high_add()=0;
  virtual bool read(sc_uint<ADDW> add, sc_int<DATAW> *data)=0;
  virtual bool write(sc_uint<ADDW> add, sc_int<DATAW> *data)=0;
};
```

3. Analyze interface methods

DRCF MODELING WITH SystemC 2/7: Analysis of instance

```
SC_MODULE(top){
  sc_in_clk clk;

  hwacc *hwa;
  bus   *system_bus;

  SC_CTOR(top) {
    system_bus = new bus("BUS");
    system_bus->clk(clk);

    hwa = new hwacc("HWA", HWA_START, HWA_END);
    hwa ->clk(clk);
    hwa ->mst_port(*system_bus);
    system_bus->slv_port(*hwa);
  }
};
```

1. Locate declaration

2. Analyze constructor

3. Analyze port and interface bindings

DRCF MODELING WITH SystemC 3/7:
Modification of instance

```
SC_MODULE(top){
  sc_in_clk clk;

  drcf_own *drcf1;
  bus    *system_bus;

  SC_CTOR(top) {
    system_bus = new bus("BUS");
    system_bus->clk(clk);

    drcf1 = new drcf1(“DRCF1”);
    drcf1 ->clk(clk);
    drcf1 ->mst_port(*system_bus);
    system_bus->slv_port(*drcf1);
  }
};
```

1. Substitute declaration(s)

2. Substitute constructor(s)

3. Substitute substitute bindings and ports

DRCF MODELING WITH SystemC 4/7: Creation of the DRCF Component

1. Copy and rename template

```
class drcf_own : public sc_module
{
 public:
  sc_in_clk clk;
  SC_HAS_PROCESS(drcf_own);
  void arb_and_instr();

  SC_CTOR(drcf_own) {
    SC_THREAD(arb_and_instr);
    sensitive_pos << clk;
  }
};
```

DRCF MODELING WITH SystemC 5/7: Creation of the DRCF Component

2. Create componets

```
class drcf_own : public sc_module
{
 public:
  sc_in_clk clk;
  hwacc *hwa;

  SC_HAS_PROCESS(drcf_own);
  void arb_and_instr();

  SC_CTOR(drcf_own) {
    SC_THREAD(arb_and_instr);
    sensitive_pos << clk;
    hwa = new hwacc("HWA", HWA_START, HWA_END);
  }
};
```

DRCF MODELING WITH SystemC 6/7: Creation of the DRCF Component

3. Insert and bind ports

```
class drcf_own : public sc_module
{
 public:
  sc_in_clk clk;

  sc_port<bus_mst_if> mst_port;

  hwacc *hwa;
  SC_HAS_PROCESS(drcf_own);
  void arb_and_instr();
 SC_CTOR(drcf_own) {
    SC_THREAD(arb_and_instr);
    sensitive_pos << clk;
    hwa = new hwacc("HWA", HWA_START, HWA_END);

    hwa ->clk(clk);
    hwa ->mst_port(mst_port);
```

DRCF MODELING WITH SystemC 7/7: Creation of the DRCF Component

4. Insert interfaces

```
class drcf_own : public sc_module
{
 public:
  sc_in_clk clk;

  sc_port<bus_mst_if> mst_port;

  hwacc *hwa;
  SC_HAS_PROCESS(drcf_own);

  void arb_and_instr();

  sc_uint<ADDW> get_low_add();
  sc_uint<ADDW> get_high_add();
  bool read(sc_uint<ADDW> address, sc_int<DATAW> *data);
  bool write(sc_uint<ADDW> address, sc_int<DATAW> *data);
```

CONTEXT SCHEDULER AND PARAMETERS

Context scheduler:

- When an interface method is called, the context scheduler checks to **which component** the interface method **call was targeted to**.
- If the interface method call was targeted to the **active context**, the interface method call **is forwarded directly**.
- If the interface method call was targeted to a context which is **not active**, the **context switch is activated**
- During context switch, the arbitration and instrumentation process **generates proper data reads** in to the memory space that holds the required context.
- The scheduler will **keep track of** active time of each context as well as the time that the DRCF is in reconfiguring itself.

Parameters:

- The memory address, where the context is allocated.
- The size of the context.
- Delays associated with the re-configuration process (in addition to the delays of memory transfers).

- In the future, other parameter, such as dealing with partial reconfiguration or power consumption may be devised.

- Parameters are derived from data sheets

FUTURE WORK

- Final implementation of the tools (3Q2003)
- Validating the accuracy of the methodology (3Q2004)
  - TLM SystemC model of a test application
  - Implementation of the application on
    - System-level FPGA (Virtex IIPro)
    - Array of processing elements (to be defined)
- **Deriving proper parameters for these devices and analyzing the accuracy of the methodology!**

CONCLUSIONS

- Work is still in the early stages
- A modeling methodology was presented
    - for system-level
    - for estimating the impact on performance of different implementation technologies
    - automatic (requiring no code changes)
- The finalization of the tools and validation of the methodology using sample application is to be done during 2004

Thank you for your attention!

Any Questions?

Contact information:
Antti Pelkonen
VTT Electronics
P.O .Box 1100, FIN-90571
Oulu, Finland
antti.pelkonen@vtt.fi

http://www.imec.be/adriatic/