A design methodology for TTA protocol processors

Presentation by
Seppo Virtanen
seppo.virtanen@utu.fi
http://users.utu.fi/seaavi

Embedded Systems lab,
Turku Centre for Computer Science (TUCS)
http://www.tucs.fi
Introduction

- TACO goal: specify a methodology for designing programmable processors with hardware optimized for a given protocol processing application
  - Start with an app spec, result of flow is a VHDL model
  - Find frequently appearing operations in application
  - Explore a set of protocol processor architectures to find candidates for implementation
    - Candidates must be able to execute the target application correctly within given timing constraints
    - Candidates must be feasible in terms of area and power use

- Building blocks needed
  - Base architectural framework: TTA
  - Rules for suggesting parts of app to be implemented in hardware
  - Models for simulation, estimation and synthesis
  - Mechanism for model integration and interaction
Architectural framework

- TTA
- Each functional unit performs a protocol processing task
- The number and types of FU’s used chosen depending on requirements of target application

Some supported tasks

- Checksum calculation
- Up/down counters
- Boolean evaluations
- Bit field matching
- Bit field masking
- Shifting
Design flow: application analysis

- Analysis of specification
  - List of required modules
    - Create new module (SystemC, Matlab, VHDL)
  - Sequential assembler for virtual processor
    - Architecture instance model composition
      - Simulation and physical parameter estimation
        - Synthesizable processor model
Application analysis

- Identify frequently appearing operations in the protocol processing application
  - e.g. CRC, Boolean, counting, timing, matching
  - These become FU’s of processor
  - A method for finding such operations is described in D. Truscan’s presentation

- Compare to existing modules in our VHDL and SystemC module library

- Add module into library if operation can not be performed with existing modules, and create a Matlab representation of it
SystemC Simulation framework

- Implementations of FU’s, sockets, interconnection buses, dispatch logic written in SystemC 1.0.1
- Heterogenous level of abstraction
  - Inter-module communication at RTL level
  - Internal functionality of modules at higher levels
- Object oriented techniques used:
  - Inheritance
  - Polymorphism
SystemC Model Details

- Parent class encompasses all mutual features of subclasses, both functionality and interfaces
- Leaf classes contain distinctive additions to functionality and interface $\Rightarrow$ leaf classes remain relatively simple
- Benefits: more compact and readable code, fewer errors, design of new FU’s is faster
- Some modules are dynamically created and connected in the SystemC simulation setup
  - Makes construction of simulators easier and faster $\Rightarrow$ Faster design space exploration
SystemC model: main.cpp example

- The following code in main.cpp creates three buses, one matcher unit and four sockets
- Sockets are created when needed, always connected

```cpp
Bus* bus1: new Bus(“Bus1”);
Bus* bus2: new Bus(“Bus2”);
Bus* bus3: new Bus(“Bus3”);
Matcher* m1: new Matcher(“Matcher1”);
bus1 -> insertOperand(m1);
bus2 -> insertData(m1);
bus3 -> insertTrigger(m1);
bus1 -> insertResult(m1);
bus2 -> insertResult(m1);
```
Matlab estimation model

- Set of scripts and functions in M-language
  - equations for delay, area, power estimation
- Delay: estimate pipe stage lengths
- Area: estimate based on delay constraints (gate / repeater size)
- Power/task energy: estimate based on supply voltage, cycle length, cycle count
- More details in Tero Nurmi’s presentation
VHDL synthesis model

- Hybrid model: common operations of modules modeled in structural VHDL, module-specific parts modeled in behavioral VHDL
- Code reuse possible
  - e.g. functional units: replace module-specific part (module interface structure remains unmodified)
  - e.g. module duplicates: only module identification information needs to be changed
Design flow: virtual assembler

- Analysis of specification
- List of required modules
  - Create new module (SystemC, Matlab, VHDL)
  - Sequential assembler for virtual processor
    - Architecture instance model composition
    - Simulation and physical parameter estimation
      - Synthesizable processor model
Virtual Processor Assembler

- Virtual processor = a processor with one interconnection bus and one of each required type of functional units.
- After establishing that required modules exist:
  - Refine application specification until it becomes a list of consecutive data moves between modules
    - e.g. move counter result to input of a Boolean unit
  - List of consecutive moves = virtual processor assembler code
  - virtual assembler used as basis for deriving architecture instances
Design flow: Design iteration cycle

- Analysis of specification
- List of required modules
- Create new module (SystemC, Matlab, VHDL)

Sequential assembler for virtual processor

Architecture instance model composition

Simulation and physical parameter estimation

Synthesizable processor model
Design Iteration Cycle

- Construct SystemC simulation model of an architecture instance based on virtual ASM
  - Either manually or using the Design Tool
  - Application code must be tuned for the instance
- Verify functionality through simulation
- Provide simulation results to Matlab model for estimating physical characteristics
- Analyze simulation and estimation results
- Explore more instances or proceed to VHDL model synthesis
Setting up model instances

- Three different models in three different OS's
- Simulation, estimation and synthesis setup takes time!
- Top level code needed for every model for every architecture
- In practice 3 persons needed to carry out experiments
- Unwanted features possible due to coding errors

a) Generated SystemC code
```systemc
sc_clock clk("clock", 20);
NetControl nc("NetCtrl1");
nc.clk(clk);
Bus* bus1 = new Bus("Bus1");
Bus* bus2 = new Bus("Bus2");
Bus* bus3 = new Bus("Bus3");
Matcher* m1 = new Matcher("Matcher1", clk);
bus1->insertOperand(m1);
bus2->insertOperand(m1);
bus3->insertOperand(m1);
bus1->insertData(m1);
bus2->insertData(m1);
bus3->insertData(m1);
bus1->insertTrigger(m1);
bus2->insertTrigger(m1);
bus3->insertTrigger(m1);
bus1->insertResult(m1);
bus2->insertResult(m1);
bus3->insertResult(m1);
nc.initialize();
```

b) Generated Matlab code
```matlab
matcher = [2 4 129 0.5]
```

c) Generated VHDL code
```vhdl
signal highway : tacbus_matrix(2 downto 0);
signal matcher1_01, matcher1_02, matcher1_R : std_logic_vector(1 downto 0);
signal matcher1_1Stm1_act, matcher1_1Sopm1_act : std_logic;
signal matcher1_1Sodm2_act, matcher1_0Srm1_act : std_logic;
--matcher 1
TM1: input_socket
  generic map(idmatcher1m1, 1, 0, 1, "i11")
  port map(clk, rst, Clock, squash, dst_id, highway, matcher1_1Stm1_act, matcher1_T, open);
  OM1: input_socket
  generic map(idmatcher1opm1, 1, 0, 1, "i11")
  port map(clk, rst, Clock, squash, dst_id, highway, matcher1_1Sopm1_act, matcher1_O1, open);
  DM1: input_socket
  generic map(idmatcher1dcm2, 1, 0, 1, "i11")
  port map(clk, rst, Clock, squash, dst_id, highway, matcher1_1Sodm2_act, matcher1_O2, open);
  matcher1: matcher_1u
  port map(clk, rst, Clock, matcher1_T, matcher1_01, matcher1_02,
    matcher1_1Stm1_act,matcher1_1Sopm1_act,
    matcher1_1Sodm2_act, matcher1_0Srm1_act, matcher1_R, a);
RM1: output_socket
  generic map(ido_matcher1m1, 1, 0, 1, "i11")
  port map(clk, rst, Clock, squash, src_id, matcher1_R, matcher1_0Srm1_act, highway, open);
```
Solution: TACO design tool

- First version in Delphi for Windows (Pascal)
  - Not very customizable
  - Only one supported OS
  - However, made it clear that a tool of this kind is needed

- Current version in Java
  - Very customizable; adding a HW block requires only creating templates for code generation
  - Platform independent
  - Generates correct code every time; no debugging
  - Designer can visually identify key hardware blocks and their interconnections
  - Helps the designer in evaluating design quality
TACO Design Tool
Tool integration in design flow

- Design a prototype using the tool
- Generate top level files for SysC, Mlab, VHDL
- Simulate prototype in SystemC
  - Obtain clock constraint, bus util, register stats
- If simulations are OK, estimate physical characteristics in Mlab using simulation results
  - Estimates for area and power
- Analyze simulation and estimation results
- Explore more instances or proceed to VHDL model synthesis
Turn-around Time

- Setting up SystemC simulations and Matlab estimations is fast thanks to the Design Tool
- Simulations and estimations run fast
  - Design iteration and design space exploration at the system level is fast
- Logic synthesis setup is fast
  - VHDL code is generated by the design tool

- Design steps from logic synthesis onwards consume most of the design time
Design Experiment

- Protocol Processor for ATM AIS processing in a 622 Mbps network
- Alcatel 0.35 µm standard cell library
- A HEC FU had to be created into SystemC and VHDL component libraries
- Different architecture instances constructed by varying number of buses and FU’s
  - 1, 2 or 3 buses
  - Single or dual FU’s for required operations
Results of Experiment

- Add buses and/or FU’s → reduce clock cycles
- Add FU’s → consume more energy and area, use buses more efficiently
- Add buses → consume less energy (in the 0.35 μm technology generation)

<table>
<thead>
<tr>
<th>Archit. instance</th>
<th>Exec. cycles</th>
<th>Req. clock</th>
<th>Move slots</th>
<th>Unused slots</th>
<th>Bus util.</th>
</tr>
</thead>
<tbody>
<tr>
<td>single-1</td>
<td>121</td>
<td>178 MHz</td>
<td>121</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>single-2</td>
<td>68</td>
<td>100 MHz</td>
<td>136</td>
<td>15</td>
<td>89%</td>
</tr>
<tr>
<td>single-3</td>
<td>49</td>
<td>72 MHz</td>
<td>144</td>
<td>42</td>
<td>71%</td>
</tr>
<tr>
<td>double-1</td>
<td>90</td>
<td>132 MHz</td>
<td>90</td>
<td>0</td>
<td>100%</td>
</tr>
<tr>
<td>double-2</td>
<td>53</td>
<td>78 MHz</td>
<td>106</td>
<td>1</td>
<td>99%</td>
</tr>
<tr>
<td>double-3</td>
<td>36</td>
<td>53 MHz</td>
<td>108</td>
<td>2</td>
<td>98%</td>
</tr>
</tbody>
</table>

Table 1: Worst case clock frequencies and data bus utilizations. *Single-2* indicates the use of one FU of each needed type and two buses in the interconnection network, *double-3* the use of two FUs of each needed type and three buses.

<table>
<thead>
<tr>
<th>Archit. instance</th>
<th>Energy estim.</th>
<th>Logic area estimate</th>
<th>Logic area actual</th>
<th>μP area estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>single-1</td>
<td>187 nJ</td>
<td>2.08 mm²</td>
<td>-</td>
<td>2.63 mm²</td>
</tr>
<tr>
<td>single-2</td>
<td>74 nJ</td>
<td>0.89 mm²</td>
<td>-</td>
<td>1.20 mm²</td>
</tr>
<tr>
<td>single-3</td>
<td>58 nJ</td>
<td>0.89 mm²</td>
<td>0.87 mm²</td>
<td>1.27 mm²</td>
</tr>
<tr>
<td>double-1</td>
<td>179 nJ</td>
<td>1.88 mm²</td>
<td>-</td>
<td>2.39 mm²</td>
</tr>
<tr>
<td>double-2</td>
<td>105 nJ</td>
<td>1.41 mm²</td>
<td>-</td>
<td>1.96 mm²</td>
</tr>
<tr>
<td>double-3</td>
<td>81 nJ</td>
<td>1.41 mm²</td>
<td>1.25 mm²</td>
<td>2.09 mm²</td>
</tr>
</tbody>
</table>

Table 2: Estimated task energies, estimated and actual logic areas, and estimated processor area.
Conclusions

- The TACO design methodology supported this kind of application-specific system design and system level design space exploration well.
- The Design Tool considerably speeds up setting up simulations, estimations and synthesis.
- TACO system level simulations and estimations provided reliable results when compared to post-synthesis simulation results.
- Combining results of system level simulation and system level physical parameter estimation gives the designer reliable design feasibility feedback at early stages of the design process.