Performance Verification for ESL Design Methodology from AADL Models

Hugues Jérome
Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO)
Université de Toulouse
31055 TOULOUSE Cedex 4
Jerome.huges@isae.fr

Monteiro Fellipe
Space Codesign Systems Inc.
450 rue St-Pierre, Suite 1010
Montreal, QC, Canada H2Y 2M9
Fellipe.monteiro@spacecodesign.com

Gaudron Mathieu, Bois Guy
Computer and Software Eng. Department
Polytechnique Montreal
Montreal (Quebec),
Mathieu.gaudron@polymtl.ca, Guy.bois@polymtl.ca
Agenda

- Problem
- Our approach
- Proposed ESL flow based on AADL
- Case study and QoR Constraints
- Experimental Results
- Conclusion and Future Work
Problem

- Systems on chip (SoCs) development faces a number of **tough requirements**

- Demand to **shrink time-to-market** and keep costs under control

- **Errors are introduced early** but detected (too) lately and at higher cost
Our approach

- **Shift-left**
  - Perform hardware/software validation earlier in the design flow

- **Integration** of different methodologies and technologies
  1. Model-Based Engineering (MBE) with AADL
  2. ESL Virtual Platform (VP)
  3. Mapping from AADL to VP components
  4. ESL Flow and Architectural Exploration

- In this work we will focus on the performance verification process
Model-Based System/Software Engineering

Architecture helps you focusing on the actual system

Architectural patterns

Link to code/model

Non-functional properties
Space Codesign

SpaceStudio graphical interface representing the Virtual Prototype

IRT Workshop Hardware/Software co-development 2015
Goals & contribution

- Perform design space exploration in an automatic way
  - Leverage AADL description + constraints to guide exploration instead of manual selection of candidates

- Bring SpaceStudio capabilities to AADL design evaluation

Two-step contribution
1. Bridge AADL and SystemC
2. Generate candidates
AADL entities communicate through ports
- Uniform API from comp.
  - PoV: send/receive calls

Ocarina generates API based on AADL model information:
- Process, Task and port id
- From component PoV, only local ports are visible
- Other elements are used internally to route message

```c
__po_hi_gqueue_store_out (self, port, &request);
```

Instance handle  Port variable  Data sent
Step #1: Capturing SystemC designs in AADL

- Mapping of regular SpaceStudio entities back to AADL

<table>
<thead>
<tr>
<th>AADL Construct</th>
<th>SpaceStudio Construct/ (Examples of component subtypes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus</td>
<td>Bus/ (APBBus, AMBA_AXIBus, AMBA_AXIBus_LT, AMBA_AHBBus, LMBBus and OPBBus)</td>
</tr>
<tr>
<td>Processor</td>
<td>Processor/ (Xilinx MicroBlaze, Gaisler LEON3, ARM, MIPS, Freescale and Intel)</td>
</tr>
<tr>
<td>Memory</td>
<td>Memory/ (GaislerOnChipRAM, GaislerOnChipROM,GaislerRAM, LMBRAM, BRAM and DDR)</td>
</tr>
</tbody>
</table>
| Thread         | Module/ Source code (.cpp and .h) is optional  
|                | - SW Module when Actual_Processor_BINDING property is specified  
|                | - HW module otherwise |
| Device         | Device/ Need source code (.cpp and .h) |

- Combined with property sets to be discussed for configuration

- Solution#1: minimal example being developed,
  - Iterations required to enrich property sets
  - Discussions to build library of reusable designs
Aim is to facilitate binding of C algorithms directly to SystemC wrappers, generated from component architecture.

Design of a SystemC skeleton backend that

- Maps component scheduling policy (sporadic, periodic) to corresponding SystemC template
- Maps component ports to interaction with port variables

```cpp
system_adder_adder::system_adder_adder(sc_module_name zName, ...) : SpaceBaseModule(zName, ...) {
    SC_THREAD(thread);
}

void system_adder_adder::Thread(void) {
    while (1) {
        DeviceWrite(Timer1_ID, offset, &initValue);
        //Execution of functional algorithm
        DeviceRead(Timer1_ID, offset, &timerValue);
    }
```
Step#3: Revisiting SpaceStudio API

- SpaceStudio API has similarities with AADL one:
  - API to send/receive data/events, FIFO-based message passing, Shared-memory communication, Register-based communication
  - But API lists destination block, not local port
- E.g. `ModuleRead(EXTR_ID, SPACE_NON_BLOCKING, &inmsg);`
- This limits component reuse in case of change in architecture

- Solution #3: add intermediate routing table
  - Like in regular AADL code generation: a module interacts with local ports, a routing table acts as a proxy to remote ports
  - Implemented in Ocarina by M. Gaudron, in AADL-to-SpaceStudio back-end
ESL Flow Based on AADL

Requirements

AADL

C/C++ Source Code (.h and .cpp)

Configuration and code generation with OCARINA

Python Script

Performance verification and Architectural Exploration

Requirements Met?

Yes

Project Generation for EDA tool (e.g. Xilinx, Altera, Synopsys)

No \(\Rightarrow\) Feedback to the user

Implementation

Space Library

SpaceStudio – ESL Virtual Platform
ESL Virtual Platform (VP)

- ESL design and verification
  - **Emerging** electronic design methodology
  - Focuses on the **higher abstraction** level concerns.

- In this work we use *SpaceStudio™* [8]
  - Functional/non-functional specification (C/C++/SystemC)
  - Scriptable tool
  - System performance prediction
  - HW/SW co-design
  - Architectural exploration
  - Non-intrusive monitoring
  - Fast FPGA prototyping
ESL Flow and Architectural Exploration

- 3 levels of abstraction:
  - High Level
    - Elix - Functional validation
      - Application and Algorithm Optimisation
    - Simtek - Architectural validation
      - Exploration loop
      - Architecture parameters evaluation:
        - Clocks frequency
        - FIFO sizes
        - Bus latency
        - Cache configuration
        - Etc.
  - Low Level
    - GenX - Implementation
      - RTL project generation
      - Xilinx
      - Altera
      - Etc.

IRT Workshop Hardware/Software co-development 2015
SpaceStudio Elix – Functional validation

Requirements

AADL

C/C++
Source code (.h .and cpp)

Configuration and code generation with OCARINA

Python Script

Elíx - Functional validation

Module A

Module B

Module C

Module D
Simtek - Architectural validation

Potential Architectures \( \{A_1, A_2, \ldots, A_n\} \)

- HW/SW Co-Synthesis
- Embedded Software
- Switching Activities
- HW Resource Estimation
- Power Metrics
- Performance Metrics

Database (DB)

- ISS
  - QEMU
  - ARM Fast Models
  - Simics
  - OVP
- ASIC
  - Calypso Catapult
- FPGA
  - Vivado (Xilinx)
  - Vivado HLS
  - Quartus (Altera)
- OS / RTOS
  - Linux
  - uC
  - RTEMS
  - Bare Metal
  - VxWorks
- Power Analysis
  - Xpower (Xilinx)
  - Docea Power
- Third Party ESL Products
  - Supported
  - Not supported yet
Simtek Profiling and Monitoring Database

- **Resource consumption**
  - Bus bandwidth
  - FIFO bandwidth
  - Memory accesses
  - CPU Time
  - Power Consumption

- **Resource estimation**
  - Hardware resource usage estimation

- **Performance**
  - Deadlock
  - Starvation
  - Latency
  - Execution time
  - Deadline
  - Communication bottlenecks

- **Software Events**
  - Processor load
  - Task switching
SpaceStudio

GenX - Implementation

IP Libraries
- Vivado (Xilinx)
- Quartus (Altera)

HLS
- Calypto Catapult
- Vivado HLS

RTL Hardware Platform

TLM Virtual Platform

IP Mapping / Reuse

Embedded Software

High Level Synthesis

Firmware and Drivers Generation

Software Firmware

Software Application

Target
- Zynq SoC
- Virtex FPGA Family
- Spartan FPGA Family
- Cyclone V Soc

OS / RTOS
- Linux
- uC
- RTEMS
- Bare Metal
- VxWorks

Third Party ESL Products
- Supported
- Not supported yet
**Case Study: Motion-JPEG**

- **System:** Remote video monitoring to assure proper behavior using thumbnails
  - The MJPEG (as a subsystem) is part of the complete video processing system on FPGA

- **Objective:** Performance verification of a M-JPEG video decoder application for video thumbnails

---

**A complete video processing system**

**The M-JPEG subsystem**
QoR Constraints

- The target is a Xilinx Zynq-7000 with three types of processing:
  - ARM Cortex-A9 dual-core processor running Linux
  - MicroBlaze soft-core processors running μCOS/II RTOS
  - FPGA fabric as coprocessors

- As the subsystem is part of a larger system, we must:
  - Minimize the FPGA resources
  - Not exceed 10% of ARM processor maximum load

- A minimum of 4 frames per second (FPS) as sufficient
Experimental Results (1)

- **Five Hardware/Software architectures verified**

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Mapping on SW</th>
<th>Mapping on HW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ARM</td>
<td>MicroBlaze</td>
</tr>
<tr>
<td>Arch #1</td>
<td>IDCT / IQZZ / VLD</td>
<td>DEMUX</td>
</tr>
<tr>
<td>Arch #2</td>
<td>IDCT</td>
<td>DEMUX</td>
</tr>
<tr>
<td>Arch #3</td>
<td>IQZZ</td>
<td>DEMUX</td>
</tr>
<tr>
<td>Arch #4</td>
<td>VLD</td>
<td>DEMUX</td>
</tr>
<tr>
<td>Arch #5</td>
<td>IDCT</td>
<td>DEMUX / LIBU</td>
</tr>
</tbody>
</table>

- **DEMUX, IDCT and IQZZ in SW and the rest in HW**

```plaintext
properties
Actual_Processor_Binding => (reference (uBlaze)) applies to demux;
Actual_Processor_Binding => (reference (a9)) applies to idct;
Actual_Processor_Binding => (reference (a9)) applies to iqzz;
Actual_Memory_Binding => (reference (mem)) applies to demux;
end integration.demuxub_coloridctiqzza9;
```
Experimental Results (2)

- Mapping of the DEMUX thread (AADL to Python)

```aadl
thread Demux
features
  to_vld : out event data port Uint_data;
  to_iqzz : out event data port Uint_data;
  to_idct : out event data port Uint_data;
  to_libu : out event data port Uint_data;
end Demux;
thread implementation Demux.impl
calls
  loop : {code : subprogram Demux_Function;};
properties
  Period => 50ms;
  Dispatch_Protocol => Periodic;
end Demux.impl;
```

```python
MODULE_DEMUX = (system demux P1,'proc',8,'Periodic',50)
module = project.createModule(MODULE_DEMUX[0])
shutil.copyfile(os.path.join(resource_dir, 'system_demux_P1.h'), module.getHeaderPath())
shutil.copyfile(os.path.join(resource_dir, 'system_demux_P1.cpp'), module.getSourcePath())
moduleInstance = design.createModuleInstance(MODULE_DEMUX[0])
moduleInstance.mapTo(processor[MODULE_DEMUX[1]])
```
Experimental Results (3)

- Mapping of the DEMUX thread
  - AADL to SpaceStudio through python scripting

```python
MODULE_DEMUX = (system_demux_P1, 'proc', 8, 'Periodic', 50)
module = project.createModule(MODULE_DEMUX)
shutil.copyfile(os.path.join(resource_dir, 'system_demux_P1.h'), module.getHeaderPath())
shutil.copyfile(os.path.join(resource_dir, 'system_demux_P1.cpp'), module.getSourcePath())
moduleInstance = design.createModuleInstance(MODULE_DEMUX)
moduleInstance.mapTo(processor[MODULE_DEMUX[1]])

void system_demux_P1::thread(vcid) // Can be either SW or HW
{
    double initVal;
    while(1)
    {
        DeviceWrite(XILINX_TIMER1_ID, offset, &initVal);
        demux_funct();
        DeviceRead(XILINX_TIMER1_ID, offset, &timerVal);
        if (timerVal < 50)
            wait(50 - timerVal); // Wait minimum latency
        else
            sc_stop(); // Miss Deadline
    }
}
```

IRT Workshop Hardware/Software co-development 2015
# Experimental Results (4)

## Performance

<table>
<thead>
<tr>
<th>Arch.</th>
<th>FPS</th>
<th>Maximum load on the ARM processor (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arch #1</td>
<td>2.5</td>
<td>66</td>
</tr>
<tr>
<td>Arch #2</td>
<td>17.54</td>
<td>48</td>
</tr>
<tr>
<td>Arch #3</td>
<td>17.4</td>
<td>49</td>
</tr>
<tr>
<td>Arch #4</td>
<td>4.7</td>
<td>50</td>
</tr>
<tr>
<td>Arch #5</td>
<td>4.2</td>
<td>10</td>
</tr>
</tbody>
</table>

## Hardware resources

<table>
<thead>
<tr>
<th>Arch.</th>
<th>LUT (%)</th>
<th>FF (%)</th>
<th>RAM (%)</th>
<th>DSP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arch #1</td>
<td>31</td>
<td>27</td>
<td>37</td>
<td>21</td>
</tr>
<tr>
<td>Arch #2</td>
<td>45</td>
<td>34</td>
<td>47</td>
<td>22</td>
</tr>
<tr>
<td>Arch #3</td>
<td>46</td>
<td>33</td>
<td>47</td>
<td>27</td>
</tr>
<tr>
<td>Arch #4</td>
<td>34</td>
<td>30</td>
<td>43</td>
<td>47</td>
</tr>
<tr>
<td>Arch #5</td>
<td>22</td>
<td>13</td>
<td>45</td>
<td>8</td>
</tr>
</tbody>
</table>

## Architectures 1
- Doesn’t meet the performance requirements of 4 FPS

## Architectures 2, 3 and 4
- Too much load (over 10%)
- Too much hardware resources.

## Architecture 5
- Meet all requirements
Conclusion and Future Work

- AADL high-level model mapped on a virtual platform
- AADL allowing *early analysis* and to *avoid late* re-engineering efforts
- Performance verification of *different architectures achieved in hours*
  - In RTL at least few days

Next?
- Supporting additional properties
  - Cache size
  - Communication buffer size
  - Power constraints
- Verify scheduling properties of the system model