The CompSOC Design Flow for Virtual Execution Platforms

FPGAWorld
10-09-2013

Sven Goossens*, Benny Akesson*, Martijn Koedam*, Ashkan Beyranvand Nejad‡, Andrew Nelson‡, Kees Goossens*

* TU/e
Technische Universiteit
Eindhoven
University of Technology

‡ TUDelft
Delft University of Technology

Where innovation starts
Introduction

- Embedded system complexity is increasing
- Power budget is limited:
  - Resource sharing
  - Heterogeneous multiprocessor architectures
- Some applications on the system have real-time requirements, others do not
  \[ \rightarrow \] Mixed time-criticality systems

www.compsoc.eu
Introduction: The Verification Problem

- Without special measures:
  - Resource sharing makes functional and timing behavior interdependent
  - Verification effort grows exponentially with the number of applications
    - Can only be done after integration (and may need to be repeated!)

Verification scenarios:

1. App 1
2. App 2
3. App 3
4. App 4
5. App 5
6. App 3
7. App 3

= Use-case switch

Non-real-time applications:
- App 3
- App 1
- App 4

Real-time applications:
- App 2
- App 5

SoC:
- Core 1
- Core 2
- NoC
- Peripherals
- SDRAM

www.compsoc.eu
Introduction: Virtual Execution Platforms

- Isolation between applications to reduce the number of verification scenarios:
  - Performance isolation (resource budgets): **Predictable virtual platforms**
    - E.g.: you get 30/100 cycles
  - Complete cycle-level temporal isolation: **Composable virtual platforms**
    - E.g.: you always get cycle 0-30 in every 100 cycles period, independent of other virtual platforms

- Applications run in their own virtual platform
- The physical SoC resources are designed to eliminate interference
- Allows independent application development and verification
- The **CompSOC platform template demonstrates this design philosophy**
Contributions

• Overview of the CompSOC FPGA tool flow:
  1. **Hardware flow:**
     − High level platform description $\rightarrow$ synthesized implementation
  2. **System software flow:**
     − Generates software stack, including micro kernel, resource managers, drivers, and virtual platform boot loader
  3. **Application flow:**
     − Maps data flow applications, generates virtual platform configuration

• **Goal:** Reduce design and verification effort $\rightarrow$ reduce the time-to-market
• Integration of individual elements of CompSOC project
CompSOC Platform Components

- CompSOC has a library of hardware and software components

Software
- Drivers
- Resource managers (RM)
- Composable Micro Kernel (CoMik)

Processing
- µBlaze
- Timer / Interrupt (TIFU)

Memory
- S(D)RAM
- CMEM
- IMEM
- DMEM

Communication
- DMA
- Mux / Demux
- Adapters
- NoC

Peripherals
- TFT
- UART
- Speaker
- Buttons

Software stack

Processor tile

Communication core

Memory mapped peripheral
Components are combined into a template, used to create instances.
High-level Input Files

- High-level description of IPs:
  - Dimensioning (i.e. 2 DMAs per tile)
  - Memory elements / sizes
- Their master/slave ports
  - Protocol, data /address width
- Clock domains
- NoC dimensions

- Fan-in and fan-out of ports
- Select Direct or NoC-based connection
Overview of Hardware Flow Tools

Typical run time:

Architecture

Add implicit components, default parameters and interfaces

< 1 min

IP Expansion

Generate NoC, (de) multiplexers, Protocol adapters, width converters, clock domain crossings

Completed Architecture

Gather HDL code, create top-level file, wrap in Xilinx PCore

Communication

Xilinx Project Generation

Generate Xilinx platform description (MHS-file) + constraints

Communication Core Generation

HDL Instantiation

make

Synthesize Platform

Bitstream

4/23
• No user input is required after the initial specification at the top of the flow*

* Note: this is a research tool, garbage in, garbage out applies to some extent
Outline

- Introduction
- CompSOC platform template
- Hardware Flow
- **System Software Flow**
- Application Flow
- Example
- Conclusions
Overview of System Software Flow

Micro kernel: temporal isolation, stack / heap, tile HW API (timers, interrupts)

Which components exist (address map)

Initial platform setup, virtual platform boot loader, system application

Hardware Specification

CoMik

Completed Architecture

Resource Man. + Drivers

Compile

Xilinx libgen

libxil

rmlib

Compile

ELF

Tile 1

Data2Mem

Bitstream

Bootable bitstream

Initial platform setup, virtual platform boot loader, system application
Outline

- Introduction
- CompSOC platform template
- Hardware Flow
- System Software Flow
- Application Flow
- Example
- Conclusions
(Per-) Application Flow

1. **Completed Architecture**
2. **Application CSDF graph**
3. **Mapping (SDF³)**
4. **Mapping & budget requirements**
5. **Allocation**
6. **VP Configuration**
7. **Compile application bundle**
8. **Application Bundle**
9. **To SDRAM**
10. **Annotated with real-time requirements**
11. **Connect actor functions to communication API**
12. **Application source code**
13. **Data-flow-based design-space exploration**
14. **Entry point for other models of computation**
15. **Find resource configuration satisfying the requirements**
16. **Virtual Platform description + app. binary**

### Key Steps:
- **Completed Architecture**
- **Application CSDF graph**
- **Mapping (SDF³)**
- **Mapping & budget requirements**
- **Allocation**
- **VP Configuration**
- **Compile application bundle**
- **Application Bundle**
- **To SDRAM**
Outline

- Introduction
- CompSOC platform template
- Hardware Flow
- System Software Flow
- Application Flow
- Example
- Conclusions
Hardware Flow, 5-Tile Platform

- Small example to demonstrate the output of the hardware flow:

**Architecture**
- 5x processor Tile
  - 2 x DMA + 4 kB
  - 4 x CMEM x 16 kB
  - 128 kB IMEM
  - 64 kB DMEM
- Monitor Tile
  - 128 kB MEM

**Communication**
- 5x processor Tile
  - DMA
  - CMEM

**Tool flow**
- 400 lines of XML
- 56k lines of C++
- 42k lines of vhdl + verilog
- 3 hours (mostly synthesis)
5-Tile Platform

- Virtex-6 LX240T FPGA
- ML605 Board
- Clock domain per tile
  - 100 MHz
- SDRAM controller and TFT controller run at 150 MHz
- Utilization:
  - 19 % registers
  - 59 % LUTs
  - 87 % BRAMs
- Communication Core:
  - 8 % registers
  - 25 % LUTs (includes buffers)
- Placement is driven by the BRAM locations (horizontal stripes)

(Image generated using Xilinx PlanAhead tool)
Conclusions

- More functionality is integrated in single SoCs
- Resource sharing leads to behavioral / timing interdependence of applications
  - Both temporal and behavioral
  - Increases design and verification time, which increases time-to-market

- CompSOC tool flow reduces development and verification time by:
  - Providing tools to the system designer:
    - Hardware flow (high-level description to synthesized design)
    - System software flow (software stack)
    - Application flow (mapping, code generation, virtual platform dimensioning)
  - Based on CompSOC platform template, using virtual execution platforms:
    - Enables verification in isolation
    - Avoid re-verification
For further information:

www.compsoc.eu

Sven Goossens <s.l.m.goossens@tue.nl>
Benny Akesson <kessoben@fel.cvut.cz>
Martijn Koedam <m.l.p.j.koedam@tue.nl>
Andrew Nelson <a.t.nelson@tudelft.nl>
Ashkan Beyranvand <a.beyranvandnejad@tudelft.nl>
Kees Goossens <k.g.w.goossens@tue.nl>

Electronic Systems Group
Electrical Engineering Faculty
Eindhoven University of Technology

Computer Engineering Group
Faculty of Engineering, Mathematics and Computer Science
Delft University of Technology