Theme: Towards Reconfigurable HPC
Lecture **8**

Reconfigurable HPC II - HW Design Methodology, Theory & Tools

Manfred Mücke

University of Vienna
Research Lab Computational Technologies and Applications

Inverted CERN School of Computing, 3-5 March 2008

Outline

- **Objectives**
  - Understanding CPU vs. FPGA trade-offs
  - Understanding hardware synthesis and constraints
  - Understanding categories of synthesis tools
- **Contents**
  - Compilation targets
  - What fits into an FPGA?
  - Hardware synthesis
  - FPGA Design Flow
  - Algorithmic Synthesis Tool
  - Outlook

Choose Your Target

- Preferred target for HPC applications: **many tightly coupled cores**

⇨ Applications (x+MPI) are written for distributed machines

Enter FPGAs:

- In EE, FPGAs combine diverse functionality
- (several) EEs design for a single FPGA

⇨ Classically **single-FPGA tools only!!**

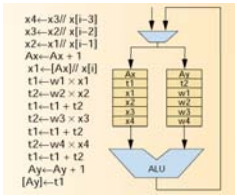
- Some tools consider CPU+FPGA (HW/SW Codesign)

A not so simple question

What fits into an FPGA?

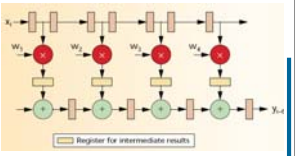
Temporal Decomposition

- CPUs provide a defined set of functions (their instruction set)
- A program on a classic CPU uses one function per time step.
- It solves a problem by sequential application of given functionality
  ⇨Temporal decomposition
- Memory limits complexity
- ISA defines efficiency



Spatial Decomposition

- FPGAs implement a fully parallel function set, custom-tailored for your application.
- All functionality operates *in parallel*.
  ⇨ Spatial Decomposition
- All required functionality needs to fit in the given area (save run-time configuration)
- Complexity is limited by silicon area
- Efficiency is limited by FPGAs basic building blocks (and their use)



Now, HOW MUCH?

- You can fit arbitrary complex functionality in an FPGA by implementing a CPU (softcore) and running code on it (slowest Nios II requires ~3% of an EP2C35).
- Functionality in LEs requires area, but brings speed.
- Only good tools achieve good implementations, and best balance!
- Consider data and control flow separately

Again, Your Target (On-Chip)?

Two basic approaches:

- **Start with all-Software**
  ⇨ design small accelerators for specific functionality (extract data flow)
  ⇨ repeat till you run out of area or reach speed

HPC starts here

- **Start with all-Hardware**
  ⇨ extract more complex FSMs
  ⇨ generalize functional units
  ⇨ schedule operations
  ⇨ repeat till your design fits FPGA

HW-Accelerated Software

Algorithm	Speed Increase (vs. Nios II CPU)	System f_{MAX} (MHz)	System Resource Increase (1)
Autocorrelation	41.0x	115	124%
Bit Allocation	42.3x	110	152%
Convolution Encoder	13.3x	95	133%
Fast Fourier Transform (FFT)	15.0x	85	208%
High Pass Filter	42.9x	110	181%
Matrix Rotate	73.6x	95	106%
RGB to CMYK	41.5x	120	84%
RGB to YIQ	39.9x	110	158%

A complex issue

Hardware Synthesis

Logic Synthesis

- Digital Systems ⇨ Logic Synthesis
- **Logic synthesis = Inferring an implementation from a more abstract description**
- Already a*b is far from straightforward:

FPGA	CPU
How many bits?	Fixed data paths
Power budget?	Fixed
How fast?	Given clock
How much area?	Fixed

redesign often, make perfect match

design once, use many years

Algorithmic Synthesis - Constraints

How to start:

1. Pick your input language
2. Model your application
3. Take a compiler/synthesizer
4. Give model and tell the compiler, which out of $10^{100000..}$ versions you want!

⇨ Constraints
- ⇨ inlined or separate
- ⇨ high- or low-level
- ⇨ quantitative or qualitative

Better modelling language ⇨ less (low-level) constraints needed

Short History of Synthesis

- **Logic minimization**
  Same abstraction level ⇨ minimize resources
- **Register-Transfer Level (RTL) Synthesis**
  Registers (global synchronisation) + combinational logic
  Freedom between registers
- **Behavioural synthesis**
  Untimed description
  Freedom everywhere
- **High(er)-Level Synthesis**
  more decent term
- **Algorithmic Synthesis**
  "Care more about specifying your problem, than the hardware"

Real-World Synthesis

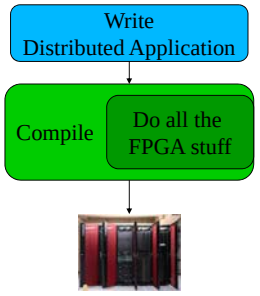
- STILL VHDL (20+ years)
- Register-Transfer Level (<=)
- Describe functional blocks on a per-cycle basis
- Make sure communication works correctly
- Low-level data types (bitvectors)
- Simulation is very precise, but very slow
- Good (and cheap) tools

It's a long way...

FPGA Design Flow

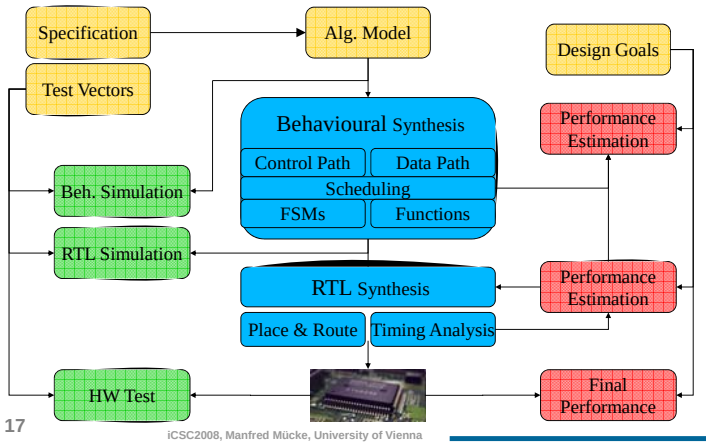
Design Flow for HPC Users

- Do you want to know the details about FPGA design flows?
  ⇨ NO (just want to run my C/Fortran app...)
- ... but you need to know for the following slides ☺



unfortunately, it is
a bit more difficult...

Algorithmic HW Design Flow



A Design-Flow Wishlist

- **Modeling**
  Expressing your problem in a machine-readable way
  Can you read it, too?
- **Verification**
  Is this really, what you wanted?
- **Simulation**
  ⇨ Testvectors
  ⇨ SW development
  ⇨ Sufficiently fast?
- **Synthesis**
  Hardware implementation
  Achieving performance goals?

Executable Software Specification goes Hardware

Algorithmic Synthesis Tools

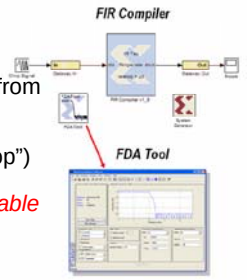
(Selection of) Existing Design Tools

- **Block-based tools/generators**
  - Vendor IP Blocks (LPM), Simulink
- **Annotated/reduced/extended C**
  - Handel C, Impulse-C, Mitrion-C
- **C-based Design Explorer**
  - Catapult C, C2H
- **Modeling Languages**
  - SystemC, Esterel
- **Binary Synthesis**

Block-Based

- **Xilinx System Generator**
- Blockset for Simulink
  - ⇨ DSP (FIR filters, FFTs, ..)
  - ⇨ Error correction (Viterbi, Reed-Solomon, ..)
  - ⇨ Memories (FIFO, RAM, ROM,..)
  - ⇨ Arithmetic and digital logic
- Automatic generation of VHDL or Verilog from Simulink
- Hardware co-simulation ("FPGA-in-the-loop")

⇨ Highly efficient *if your functionality is available*



Annotated C

Handel-C by Celoxica

- C-based HDL
  Handel-C = ANSI-C
  – side-effects, floating-point, recursion
  + signals, channels, parallelism, bits, RAM, HW library
- Originally developed at Oxford University Computing Laboratory (~1996)

C-Based

Impulse C by Impulse Accelerated Technologies

- Originally developed as Streams-C at LANL
- C subset + library extension
- Partition your application in processes.
  Connect tasks with streams.
  (Interface logic inferred by compiler)
- Tasks can be executed either on CPU or FPGA.
- designed for dataflow-oriented, streaming applications

C-Based

Mitrion-C by Mitrionics

- "Implicitly parallel C-family programming language"
- Infers an optimized CPU for your application.
  This "Mitrion Virtual Processor" is described in VHDL, to be implemented on FPGA.
- Extracts heavily used functionality from your code and moves it into custom CPU instructions
- Acceleration typically 10x to 30x
  (compared to execution on a sequential CPU)

C-Based Design Explorer

C2H Compiler by Altera

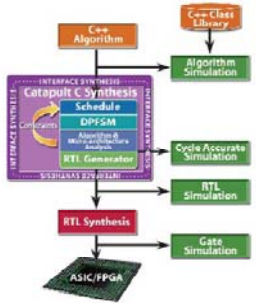
1. Profile your (ANSI-C) software
2. Highlight the desired functions within the Nios II IDE and "right-click to accelerate".
3. Review the compiler report file to determine simple C code optimizations.
4. Optimize and iterate until desired performance is met.

- Highly integrated!
  Compiler guides user.

C-Based Design Explorer

Catapult C Synthesis by MentorGraphics

- Accepts C++ as input
  Creates a range of design variants
  User picks by specifying constraints
- SystemC simulation models
- Excels in graphical feedback
- "The price [..] currently ranges from $89,000 to $275,000"



Matlab-based Design Explorer

AccelDSP by Xilinx

- "If the Xilinx blockset is not sufficient"
- Synthesizeable blocks from Matlab code
  Architectural exploration of high-level DSP algorithms
  Floating- to fixed-point conversion
  C++ simulation model generation
- Originally developed by AccelChip, acquired in 2006

Modelling Language

SystemC

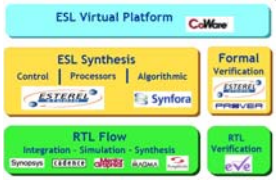
- = C++ libraries and macros enabling system-level modeling and simulation of digital systems (IEEE Std. 1666™-2005).
- = description language + simulation kernel
- Description at different levels of abstraction (RTL, C, TLM)
- Very fast simulation at higher abstraction levels
- SoC architectural exploration (busses, memory, responsiveness)
- TLM (protocol modeling) is very succesful
- Hardware synthesis from SystemC is in its infancy

Esterel-based Synthesis

Esterel Studio

- "The leading design and verification suite for control-intensive hardware IP"
- Esterel = synchronous programming language developed at Ecole des Mines & INRIA (1980s). Well-suited for describing reactive systems.
  - \+ captures concurrency
  - \+ formal verification possible
  - \- data-driven designs
- VHDL/Verilog/SystemC from Esterel specification

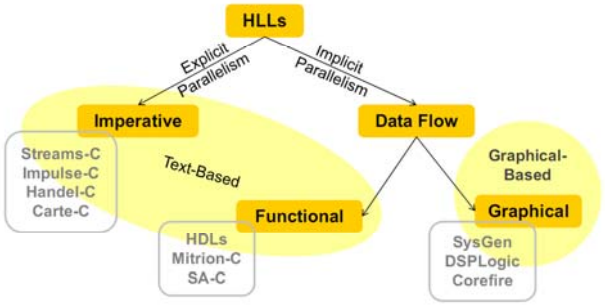
⇨ Semantically sound foundation (Lustre is used for Airbus FCS)



Binary Synthesis

- Synthesis from executing binaries
- Replaces CPU by "CPU-VM"
  Extracts, implements and calls HW accelerators on the fly
  Compare to Java JIT Compiler approaches
  - \+ Connects to all existing tools
  - \+ Transparent to the user
  - \- Limited scope
  - \- Reverse Compiler Optimizations
- Requires closely coupled hybrid system

A Classification



Taken from: " Productivity of high-level languages on reconfigurable computers: An HPC perspective"

Is this the end of the story?

Outlook

Open Questions

- Is C a good choice for hardware design entry?
  Technically: NO
  Economically: Maybe
  Legacy Code: Certainly

  Alternatives ⇨ Next Lecture
- Will FPGAs stay around?
  Generally: For sure
  In HPC: we believe yes ⇨ Hybrid Systems ⇨ Last lecture
- Is FPGA-centric design helpful for HPC? ⇨NO
  Unified design approach?
  Hybrid design approach (how to partition your app)?

Beware of C

- C (like any imperative, sequential language) is unsuitable as a specification language for efficient hardware designs (and therefore no good choice for hardware design entry)!
- ...but, not everything called "C" is C:
  ⇨ "C without side-effects",
  ⇨ "C without pointers",
  ⇨ "Single-assignment C",
  ⇨ "extended C-subset", ...

⇨ **Do not trust the labels, ask how it works!**

Further Reading

- Edwards, S. A., **The challenges of synthesizing hardware from C-like languages**. 2006. IEEE Design & Test of Computers 23 (5), 375-386. URL http://dx.doi.org/10.1109/MDT.2006.134
- Dehon, A., **The density advantage of configurable computing**. April 2000. Computer 33 (4), 41-49. URL http://portal.acm.org/citation.cfm?id=621452
- Wirth, N., **Hardware compilation: Translating programs into circuits.** 1998. Computer 31 (6), 25-31. URL http://dx.doi.org/10.1109/2.683004/
- Bryan Bowyer (Mentor Graphics), **Just What is Algorithmic Synthesis?** 2006. FPGA Journal http://www.fpgajournal.com/articles_2005/20051206_mentor.htm
- El-Araby, E., Nosum, P., El-Ghazawi, T., **Productivity of high-level languages on reconfigurable computers: An HPC perspective.** 2007. In: Field-Programmable Technology, 2007. ICFPT 2007. International Conference on. pp. 257-260. http://dx.doi.org/10.1109/FPT.2007.4439260

Q & A

Backup - HDCaml

More efficient (generic) HW description by narrowing HDL application domain (synchronous logic only).

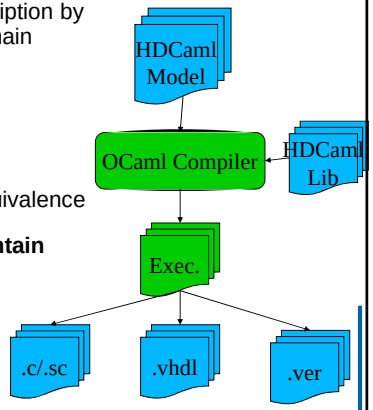
+ automatic bitwidth propagation

+ Software from HW description

+ Guaranteed C/VHDL model equivalence

+ Only one source code to maintain

- Still RTL



Backup – HThreads

= Hybrid (SW or HW) Threads by Kansas University

Andrews, D., Sass, R., Anderson, E., Agron, J., Peck, W., Stevens, J., Baijot, F., Komp, E., 2008. **Achieving programming model abstractions for reconfigurable computing.** Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 16 (1), 34-44.

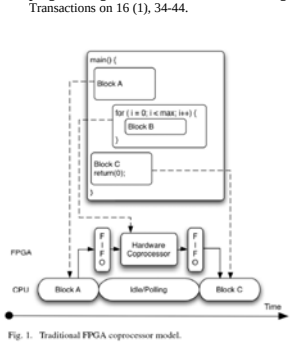


Fig. 1. Traditional FPGA coprocessor model.

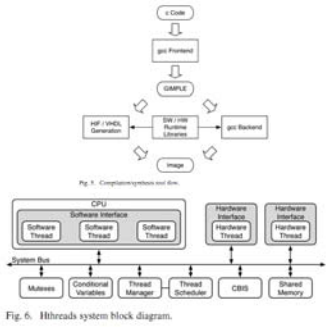


Fig. 6. Hthreads system block diagram.