FPGAs Are Locked In The Hardware World

- Currently, almost all FPGA designs are made by traditional hardware design methods
- FPGAs emulate an empty, re-configurable silicon surface
The Development Of Hardware Design

• Electrical engineering started off hand-wiring circuits
• Schematics
  – Shifted from planning device to design device
• RTL, global clock
  – The only abstraction lately added
• Synthesis
  – When schematics became too complex
The Design Cycle Of ASIC-hardware

1. Specify exact requirements
2. Define architecture
3. Select technology and foundry
   • These three can’t be changed later
4. Implement
   • Focus on Routability, testability, verifiability, power, yield, and many other factors
5. Verify design
There is more...

6. Generate physical layout
   • Same cost as implementation
7. Do tape-out
   • 3 months, $200k – $1M USD
8. Test and verify
   • Same cost as implementation
9. Give customer samples
10. Sell volume quantities
11. Improve yield
Some Of The Design Issues When Doing Hardware Implementation

- Achievement of target clock
- Floor planning
- Intimate knowledge of target platform
- Interfacing clock domains
- Balancing of combinatorial paths
- Achieving deterministic execution
- Synchronization of data
- Pipe length balancing
- Implementing external communication
Hardware design has never abstracted away from electricity
Hardware Design Techniques

- Static timing analysis
- Re-timing
- Formal verification
- Re-usability
- Floorplanning
Software Design Techniques

- Object Oriented Programming
- XP
- UML
- Design Patterns
- Functional Programming
Problem Domains For Hardware

- Memory controllers
- Fir-filters
- Display controllers
- SER/PAR converters
- PCI, USB, IDE, etc-controllers
- 3g-coding, video-coding, software defined radio
Problem Domains For Software

- Weather simulation
- Gene-matching
- Computational fluid dynamics
- Neural networks
- Geological simulations
- FEM
- Financial systems
The Hardware Paradigm

- Development focus
  - Silicon cost / performance
- Development techniques
  - Detailed control of every bit and clock-cycle
- Problem domains
  - Interface logic, signal handling
- Tools
  - Simulators, synthesis and place & route
- Support systems
  - IP-blocks
The Software Paradigm

- Focus
  - Development cost / maintainability
- Development techniques
  - Readability, abstraction, encapsulation
- Problem domains
  - Large complex systems, e.g. MS-word
- Tools
  - Compilers, debuggers, UML-charts
- Support systems
  - OS, system libraries
Two Worlds

- Interfacing
- Real World
- Embedded
- Silicon Cost

- Computation
- Simulation
- Computers
- Maintenance Cost
The Divider

• Hardware design
  – Driven by the design cycle
  – Precise control over electrical signals

• Software design
  – Driven by the code-base life-cycle
  – Abstract description of problem and algorithm
An FPGA In An HPC-computer Fundamentally Changes The Conditions

- The FPGA is reconfigurable
  - Design for life-cycle, not design-cycle
- The FPGA is in a specific environment
  - The interface is set, no need to design it
- The FPGA is unspecified
  - One chip – many designs, instead of one design
  - many chips
- The FPGA is intended for computation – not interfacing
The Mitrion Processor
Computing Performance Vs. Programming Level

- VHDL
- Verilog
- DSP
- Assembly
- C / C++
- Java
- Mitrion

Level of programming

Hardware Construction Languages

Low Level Languages

High Level Languages
The Purpose Of A Processor Architecture

- A processor is an abstraction layer
- Completely separates hardware from software
- A machine, built in hardware that performs your program, written in software
- Allows the programming language to ignore hardware design considerations
The traditional von Neumann processor is based on a state machine. It operates on one instruction at a time from an instruction stream.

+ Easily programmable
+ Executes programs of any size
- Sequential
- Low silicon utilization
- I/O intensive

➢ Needs very high clock frequency
The Philosophy Behind The Mitrion System

• Do not attempt to build hardware directly from a program
• Instead:
  – Insert an abstract machine between the program and the hardware
  – Compile language into a configuration of the abstract machine
  – Instantiate the adapted machine in an FPGA
A New Processor Architecture Specifically For FPGAs

Architecture design goal:
- High silicon utilization
- Take advantage of FPGA re-configurability

Goal achieved by:
- Allow processor to be massively parallel
- Allow processor to be fully adapted to algorithm
A Super-computer On A Chip

- Fully distributed architecture
- Similar in design to parallel computers, but within a single chip
- High chip utilization through FPGA
  - Fine grain parallelism
  - Each PE fully adapted to algorithm
  - Network topology specific for algorithm
- Highly efficient synchronization and scheduling mechanisms give low overhead
Fine Grain Parallelism

- Fine Grain, Intrinsically parallel, Adaptable, architecture
- No Instruction Stream, instead Data Stream
- Provably deterministic
  - No race-conditions
  - No deadlocks
Using The Mitrion Processor
Compiling A Mitrion Program

Mitrion-C Source code

Compiler

Processor Specification

Processor Configurator

Processor Architecture

Processor HW-Design (VHDL IP Core)

FPGA

Mitrion Development Toolkit
No User Hardware Implementation Or Integration

- Fixed HW interface
  - No integration considerations, just `main()`
- Runs at fixed clock-rate
  - No timing considerations
  - No combinatorial path balancing, pipe length balancing, data flow synchronization, selection of architecture specific features, etc
Mitrion Offers Portability And Scalability

Any Mitrion program can be re-compiled for a new target platform. The architecture allows automatic scaling with FPGA size and features.

As FPGA sizes continue to grow, Mitrion will allow for easy upgrades to the next generation of performance available.

Using HDLs, a fundamental redesign will typically be needed for each hardware upgrade.
Mitrion-C

• The Mitrion processor needs a fully parallel programming language
  – Languages with vector parallel extensions or simple parallel instructions not sufficient

• Main considerations
  – High parallelism
  – High programmability
  – No hardware design considerations
Mitrion-C

- In general, one can think as usual. The processor will sort things up and handle the parallelism for you
- The language is designed to preserve any parallelism in your program
- The language is designed to help you reveal parallelism in your program
A C-family Language

- Basic syntax is the same as for other C-family languages
- Examples:
  - Blocks are surrounded by `{ }`
  - Assignment with `=`
  - Statements end with `;`
  - `If, for, while`
  - Most of the usual c operators
  - C-style comments (though nestable)
Scalar Types

• Basic types
  - `int/uint` signed/unsigned integer
  - `boolean` boolean value (true/false)
  - `float` Floating point real value
  - `bits` Bit vector format

• Free bit width
  - `int:24` 24 bit signed integer
  - `uint:19` 19 bit unsigned integer
  - `float:24.8` IEEE-754 single precision float

• Collections
  - `int:24[100]` Vector (indexable collection)
  - `int:14<100>` List (no index)
A C-family Language

• Important differences
  – No pointers
  – No dynamic allocation
  – Static general recursion only
    • Though loop structures may be dynamic
Contact Information

Mitrionics AB
Ideon Science Park
SE-223 70 Lund
www.mitrion.com

stefan.mohl@mitrion.com
On Show At SC04: Accelerating The Tree Of Life

- MCMC (Markov Chain Monte Carlo) simulation of phylogenetic trees
- No speed-up on clusters
- Order of magnitude performance gain
- Development time: 13 man-days
- Worlds fastest implementation