

# Hardware Description Language Based on Message Passing and Implicit Pipelining

Dmitri Boulytchev and Oleg Medvedev

St.Petersburg State University

EWDTs'09, September 18-21, 2009, Moscow

# Hardware-Software CoDesign

- ▶ Adjust *hardware* to *software* instead of optimizing software for predefined hardware.
- ▶ Needed for many systems that perform special tasks and must comply with efficiency, cost and power requirements, for example
  - ▶ video, audio encoding/decoding devices;
  - ▶ various channel protocol implementations (CDMA wireless modems, coaxile cable modems, etc.)
  - ▶ network devices (routers, switches, etc.)

# Project Goals

- ▶ Develop a language to allow algorithm description at a high level of abstraction.
- ▶ Design a hardware description language as a subset of that high-level programming language in order to facilitate automated hardware-software partitioning.
- ▶ Develop a toolkit to partition high-level description into
  - ▶ a description of a reconfigurable hardware at a hardware description level;
  - ▶ a binary code that implements the algorithm for that hardware.
- ▶ Eliminate compiler retargeting step.

# Hardware Description Languages (HDLs)

## ▶ Conventional HDLs (VHDL, Verilog):

- ▶ low-level description in terms of electrical signals and their interconnections;
- ▶ high-level structural description in terms of blocks and their interconnections.

## ▶ Modern HDLs:

- ▶ **Handel-C**: the program is assembled as a parallel and sequential composition of primitive one-cycle assignments and communications.
- ▶ **Bluespec**: a system is described as a set of modules communicating by messages and by accessing read-only ports of each other. An internal description of a module is based on a concept of *guarded atomic actions*. The actions operate on the module's local data.
- ▶ **SpecC** — C-like control flow, procedures and functions based on low-level signals.

# Hardware-Software Codesign Language (HaSCoL): Hardware Description Level

- ▶ A multi-level cycle-accurate HDL.
- ▶ A core language for programming in terms of structural design, blocking and non-blocking message passing and implicit pipelining of straight-line code.
- ▶ Many orthogonal extensions to describe control flow, procedures and functions, monitors for synchronization, global variable with blocking and non-blocking assignment, processor decoder and instruction set etc.
- ▶ These extensions are automatically converted into the core language.

# HaSCoL Example: Pipelined Polynomial Evaluator

— a three stage pipeline for quadratic polynomial evaluation

```
local poly (a, b, c, x : int(32)) {
```

— two multiplications are performed on this stage

— and their results are saved to pipeline registers

```
x2 = x * x | bx = b * x;
```

— the precomputed values from pipeline registers

— are used to proceed with evaluation

```
ax2 = a * x2 | bxc = bx + c;
```

— the final result is sent as a message to some channel

```
send result(ax2 + bxc)
```

```
}
```

## HaSCoL Example: Fast Fourier Transform

```
s, m, wm, started := 0, 1, 128, true;
while s < 7 do
  m, wm := m << 1, wm >> 1 |
  k, w, j, s := 0, 0, 0, s + 1;
  while k < 128 do
    while j < m >> 1 do
      skip;
      send butterfly'A(k, j, m, w) |
      w, j := w + wm, j + 1
    done;
    k, w, j := k + m, 0, 0
  done
done;
final := true
```

# HaSCoL Example: Instruction Set Description

- this instruction has 2 parameters, which are
- numbers of a source and destination register

```
insn (src, dst : uint(3))
```

- the instruction assigns one register to another

```
does { regs'A[dst] := regs[src] }
```

- this is a binary pattern for the instruction code

```
coded { 0b01 src dst 0b0 _ : 20 }
```

- this is a syntax pattern for the instruction

- "r5 := r7;" is an example of an assembler code

```
looks { "r" dst " := " "r" src }
```

- this instruction assigns a 24 bit immediate to a register

```
insn (dst : uint(3), val : uint(24))
```

```
does { regs'A[dst] := ext(val, 32) }
```

```
coded { 0b00 dst val }
```

```
looks { "r" dst " := " val }
```

# HaSCoL Evaluation: Benchmark Synthesis Results

	FFT		polynomial	
	VHDL	HaSCoL	VHDL	HaSCoL
Lines of code	531	290	95	18
4-input LUTs	566	2320	129	162
RAMB16s	40	40	0	0
DSP48s	16	16	18	18
Clock, MHz	147	150	120	122

# HaSCoL Evaluation: Processor Design

IceBrick — general-purpose processor written in HaSCoL. Features:

- ▶ a potential ability to run Linux (work in progress);
- ▶ executes arithmetic and bus commands in parallel;
- ▶ integrated into GRLIB environment as an AMBA bus master;
- ▶ runs at 70MHz on Virtex 5 xc5vlx50t-1 FPGA;
- ▶ general instruction core + MMU described by about 700 lines of code;
- ▶ total FPGA resource consumption is about 2500 slices.

Thank you!