Project: HW/SW Synthesis

Embedded Systems Group
Department of Computer Science
University of Kaiserslautern, Germany

March 23, 2015

Goal of the Project

- Development of a HW/SW System: accelerating numerical computations
- Design: Coprocessor
- Embedded Software

Host API

Host Computer

- Managing Coprocessor
- Managing Device Memory
- starting and stopping Coprocessor kernels

Execution Model

PCP

- processes kernels
- executes large number of concurrent threads
- branching: threads can diverge within a warp → some threads will be inactive
- within warps: SIMD
- synchronize warps explicitly: `SYNC` instruction

PCP Properties

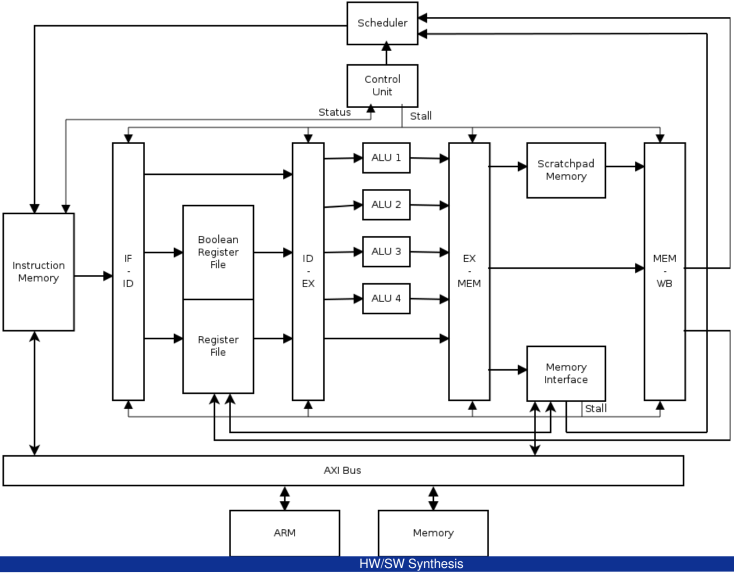
Device Memory 512MiB of DDR3-SDRAM on the FPGA-board

Scratchpad Memory 16kiB, low latency, can load or store a 4-byte word per thread in each cycle

Instruction Memory 8kiB, for up to 2000 instructions

Registers full set of registers per thread (32 general purpose 32-bit registers, 8 boolean condition registers)

Pipeline - Overview



Loading a program into the Instruction Memory

- load instructions via **AXI bus**
- last three instructions: `SYNC`, `TERM`, `START`
- `START` is not written into the Instruction Memory
- `START` last instruction → start after program is loaded

Control Instructions:

`SYNC`:	Synchronize all Warps
`TERM`:	Terminate the Program
`START`:	Start the Program

PCP - Composition

Main Components of the PCP:

Control Unit: Control Signals to and from the processor

Scheduler: 3 FIFO-queues containing warps

Pipeline: 5 stages, standard RISC pipeline

Control Unit and Scheduler

Control Unit Tasks:

Initially	fill the scheduler with warps
TERM	stall the pipeline write to the AXI bus that the program is finished

Scheduler Tasks:

FIFO 1:	Warps that have completed the **WB** stage
FIFO 2:	Warps that have completed a **Memory Access** Instruction
FIFO 3:	Warps that are **inactive** (have arrived at a SYNC)

Pipeline - Instruction Fetch

All pipeline stages get general inputs that are forwarded to the next stage:

warp: id, PCs, active flags of the current warp

valid: data in this stage is valid

stall: the pipeline is being stalled

IF - Inputs:

PC: Program Counter, from active warp

IF - Outputs:

Instruction: 32 Bit instruction word

Pipeline - Instruction Decode

- contains the **Register File**
- for 32 Bit registers, for boolean registers

ID - Inputs:

Instruction: 32 Bit instruction word

ID - Outputs:

Cond: predicated execution

OpCode: 5 Bit operation code

depending on Instruction:

Register Adresses: registers and target registers

Immediate: 16 Bit immediate

Pipeline - Execute

- contains 4 **ALUs**
- calculations for **R**-type instructions
- other instruction types: data is passed on to the next stage

EX - Outputs:

ALU out: computation results

Pipeline - Memory Access

- contains **ScratchPad**
- communicates with **Memory** via the AXI bus
- **Memory** is connected **asynchronously**
- in case of memory access instruction: write warps to the scheduler (when the reply arrives on the bus)

MEM - Outputs:

MEM data: data from Memory/ScratchPad

Warp: reply from bus → warp to scheduler

Pipeline - Write Back

- write back to the **register file**
- write warp to scheduler
- update PCs

Instructions - Examples

Register Type (ADD, SUB, AND, OR, XOR)						
Cond	Instr	RegTrg	Unused	RegA	RegB	Unused
0..2	3..7	8..12	13	14..18	19..23	24..31

Mem Type (LW/LWS)					
Cond	Instr	RegTrg	Unused	AddrR	Unused
0..2	3..7	8..12	13	14..18	19..31

Mem Type (SW/SWS)					
Cond	Instr	Unused	AddrR	DataR	Unused
0..2	3..7	8..13	14..18	19..23	24..31

Further instructions: `LOADI`, `LTID`, `EQ`, `LT`, `SRL`, `SLL`, `JD`, `JDN`, `TERM`, `SYNC`

Device Interface

- Implemented on the FPGA board’s ARM processor
- lwIP for TCP/IP
- Access to AXI bus via memory mapped I/O
- Memory management using C standard library
- No further logic

Network Protocol

- Communication initiated by host
- Packet size is multiple of 4 bytes

Host

PCP{S,O,T,U,M,F,D,H,C}
[Payload]

Device

PCPA
[Payload]

or

PCPE
Error Code

Assembler

- Strips comments
- Replaces parameters with immediate values
- Expands immediate loads $>$16bit
- Resolves jump labels to addresses

Host Interface

- Small Pyhton library providing PCP class
- Abstracts network communication
- Kernels are input as text
- Automatically assembles kernels before sending

Does it work?

Almost...

Simulation

- works fine

Reality

- problems with memory access

What did we learn?

- Designing Hardware is different from Software
- Broken tools are broken
- Additional problems occur after simulation
- Testing takes a lot of time

Thank You!