ILLINOIS INSTITUTE OF TECHNOLOGY
Transforming Lives. Inventing the Future.

The SOCks Design Platform

Johannes Grad

System-on-Chip (SoC) Design

- Combines all elements of a computer onto a single chip
  - Microprocessor
  - Memory
  - Address- and Databus
  - Periphery
  - Application specific logic
- Software development must take place on simulation models or FPGAs until the actual chip is fabricated
- Hardware/Software Co-Design issues: Need to make educated guess on what becomes hardware and what is done in Software early in the design process

“Soft” IP Blocks

- Synthesizable HDL code (commercial HDL is usually encrypted)
- From Synopsys Designware, Opencores, MIPS, etc.
- Can be implemented on any Library
- HDL for 8051, 6800 available
- Usually highly configurable
  - Cash (Yes, No, How big, Code/Data separate or unified)
  - Pipelined (Yes, No)
  - SRAM interface (single cycle, multi cycle)
  - User Defined Instructions
- Timing, Area and Power depend on process, CAD tools used, and user skills
- Popular Example: Synopsys DesignWare

“Hard” IP Blocks

- Fully implemented, verified mask layout block
- Available only for specific process
- Not configurable
- Guaranteed Timing, Area, Power Consumption
- E.g. MIPS Hard-IP cores

Note:

SOCks uses Soft-IP cores

The Leon Core

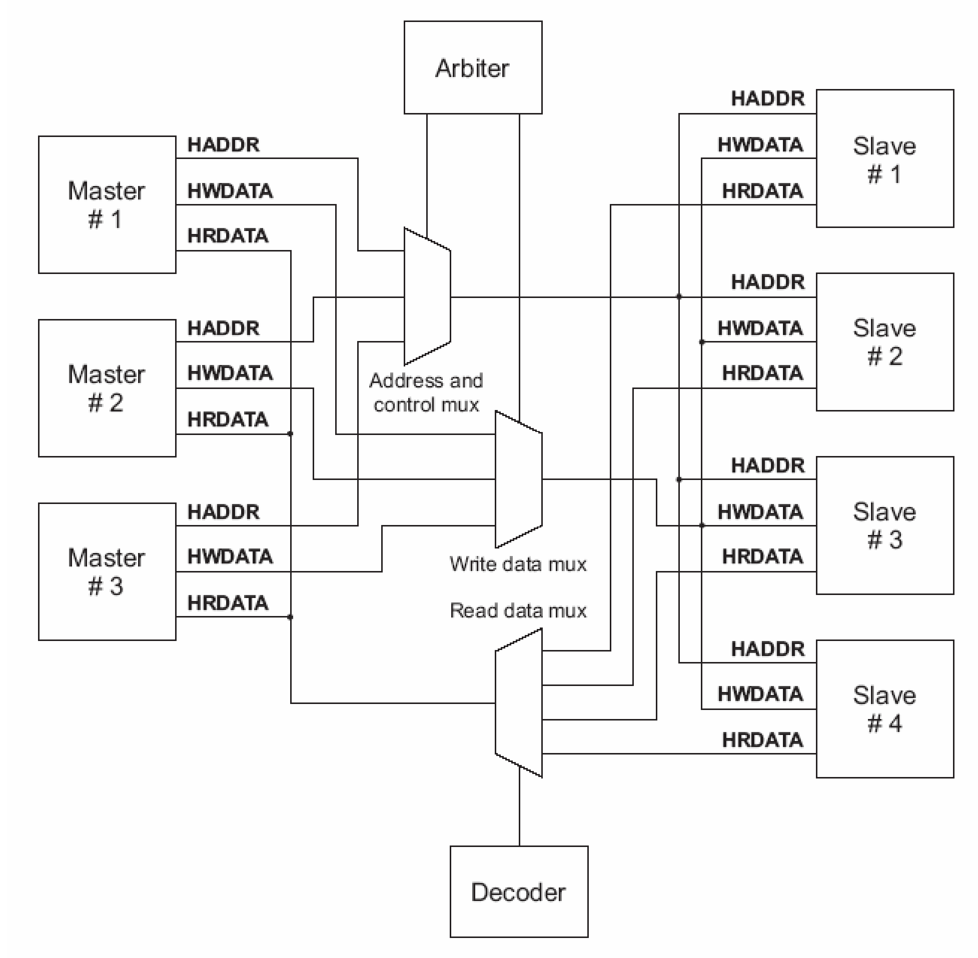
- Available in the public domain (under GNU License)
- http://www.gaisler.com/
- Synthesizable VHDL soft-core
- Highly configurable for many different scenarios
- Verified in several silicon implementations
- Contains AMBA controller
- Turbo-Eagle uses 2 instances of Leon:
  - Master CPU called “Leon”
  - Slave CPU called “DSP” (a Leon core configured for DSP)
- “Leon” has been modified by Cadence for this project to run at twice the bus frequency and to use 2-port instead of 3-port RAMs
- “DSP” still runs at bus frequency

The AMBA bus (1)

- Developed by ARM
- http://www.arm.com/products/solutions/AMBAHomePage.html
- Common bus interface for rapid SoC development
- Paradigm: “Design Reuse”
  - Only need to code and verify a block once
  - Can use over and over in other AMBA systems
- Avoids glue logic between blocks with custom busses
- “AMBA Compliance Testbench” to certify blocks as compatible
- “AHB” – High Speed Version
- “APB” – Peripheral low speed version
  - Use Bridge to interface to “AHB”
  - Less stringent requirements for low throughput blocks
  - Isolates critical bus segments from slower blocks

The AMBA Bus (2)

- Multiplexed, not tri-stated
  - Uses dedicated Point-2-Point links between blocks
  - Uses Multiplexer to establish link and grant bus
  - Avoid long busses that connect to many blocks
  - Possible because wire density much greater than with discrete components
- 1 transfer takes 2 cycles:
  1. Address Phase
  2. Data Phase
- Transfer Types
  - Single Word
  - Burst
  - Split
- Only positive-edge logic
  - Easier timing analysis
  - Supports more libraries



SoC Design Flow
1) Firmware Design

- “Firmware” is the code that is executed on an embedded system
- Not visible to the consumer
- Typically resides in Flash memory
  - Can update code in the field
  - Can offer user to download firmware
- Tools used:
  - GCC SPARC compiler, linker and assembler
  - Installed on the ECE servers “skew” and “vulcan”
  - Include “/opt/rtems/bin” in your $PATH variable:
    setenv path (/opt/rtems/bin:$PATH)

SoC Design Flow
2) RTL Design

- VHDL, Verilog or mixed design (SystemC in the future)
- Instantiate memories and PHYs in testbench
- Load RAM and ROM images into testbench
- Run Simulation, capture output in file
- Compare file to golden file (known good output)
- Tools used:
  - Cadence Incisive Platform:
  - NC-VHDL, NC-Verilog, NC-SIM

SoC Design Flow
3) Synthesis

- Generate timing models for all RAMs
- Partition design into blocks
- Create timing constraints
- Synthesize blocks and toplevel
- Output netlist and toplevel timing constraints
- Tools used
  - Cadence Encounter Platform:
  - PKS, BuildGates or RTL Compiler

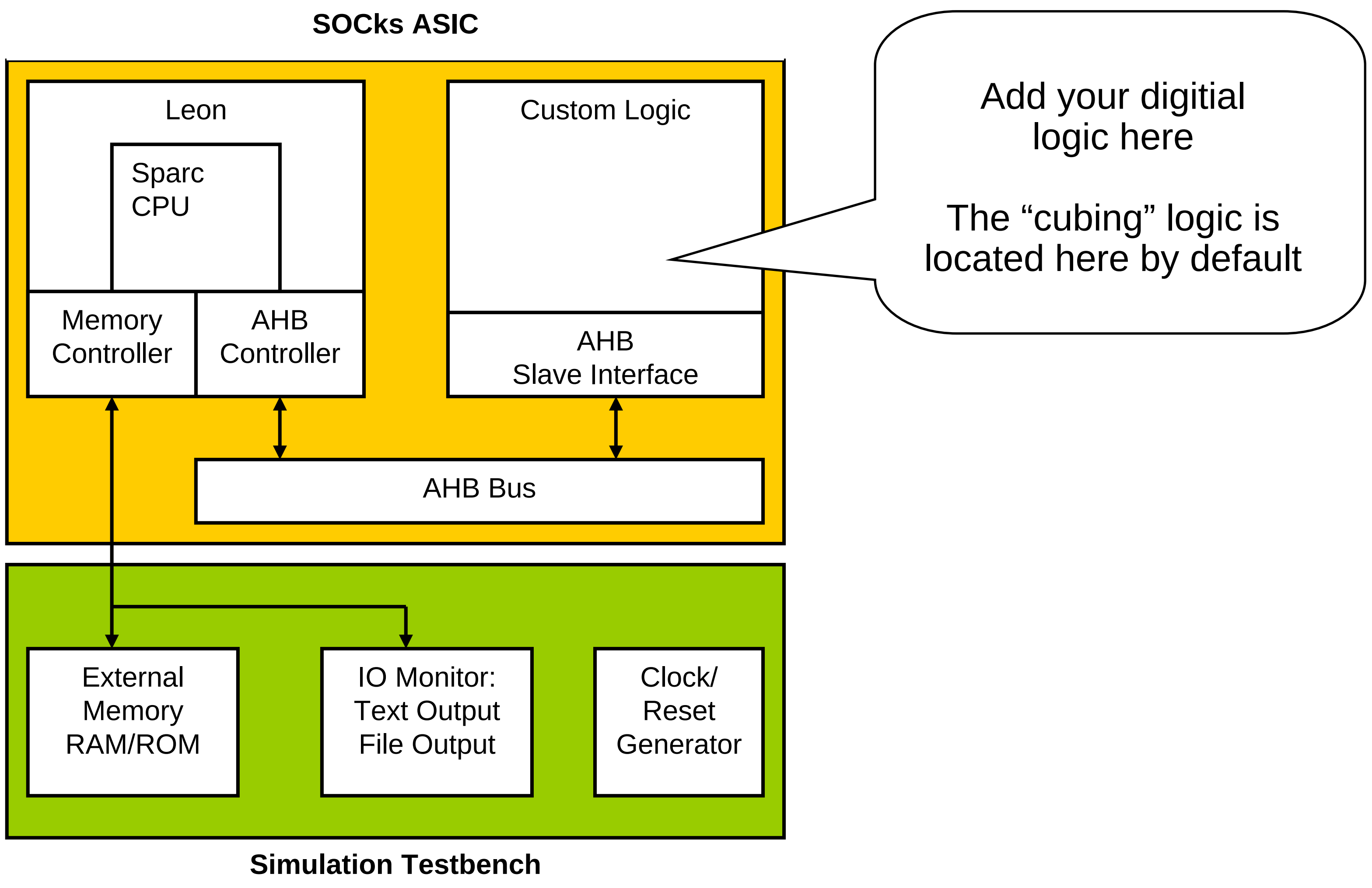
Note: Synthesis not part of SOCks Project

SoC Design Flow
4) Physical Implementation

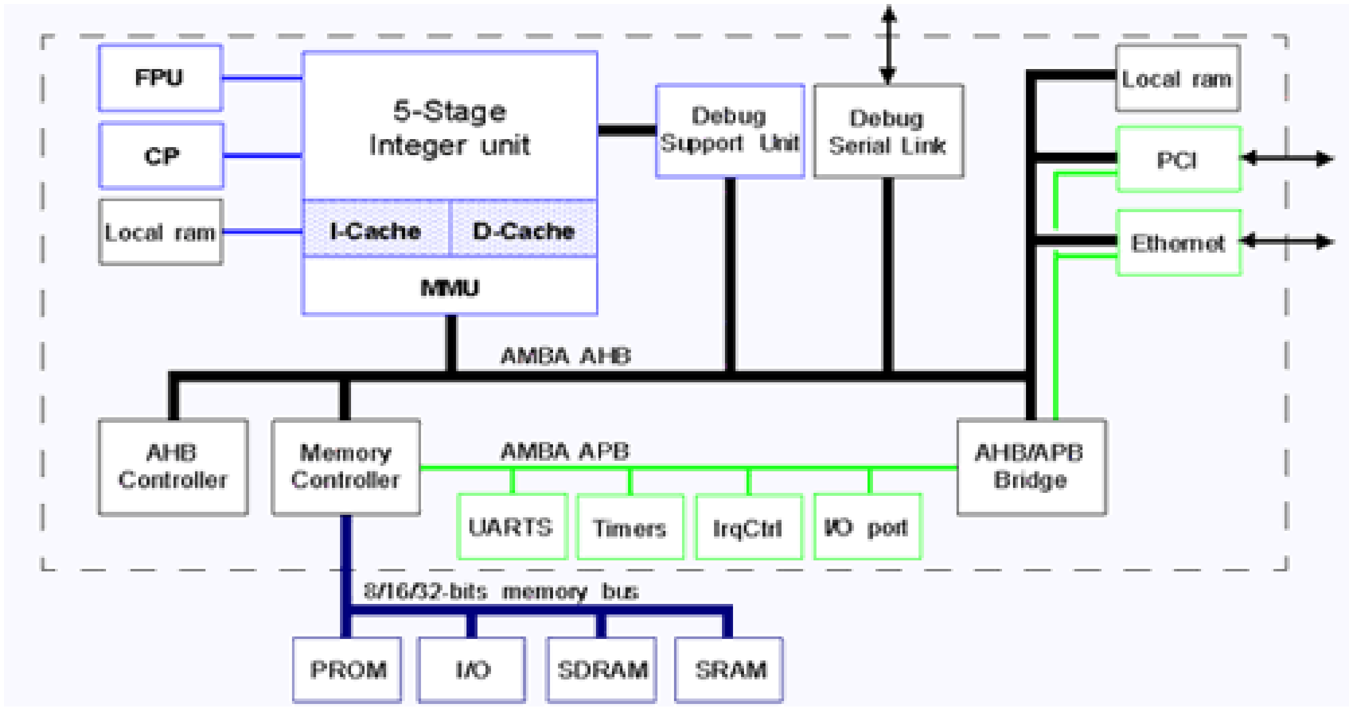
- Generate geometry abstracts for all RAMs
- Create floorplan, place RAMs, crate power structures
- Partition design into blocks and implement each block
- Load blocks, flatten toplevel
- Run final timing and DRC analysis
- Tools used:
  - Cadence Encounter Platform:
  - SOC Encounter, Nanoroute, Fire&Ice

Note: Physical Implementation not part of SOCks Project

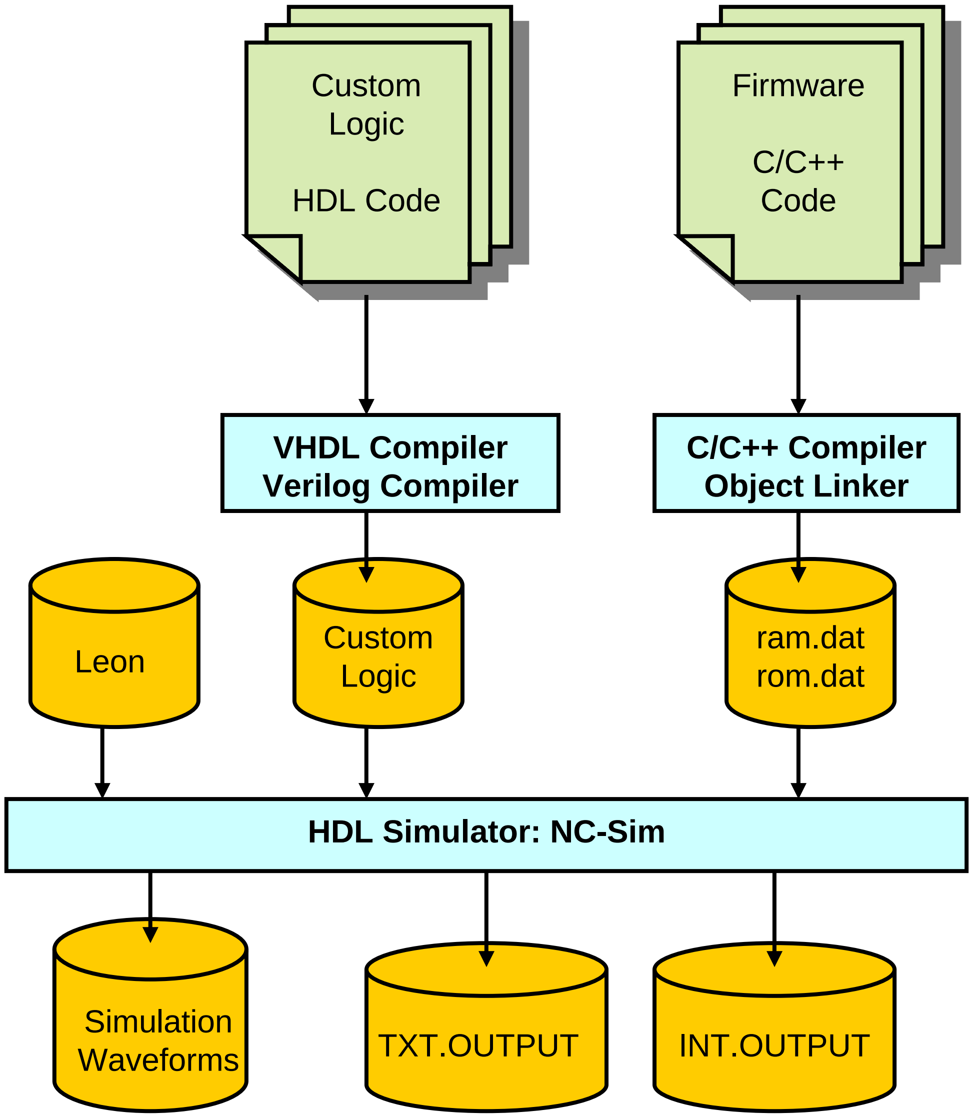
SOCks Overview



Leon Overview



SOCks Design Flow



Getting SOCks

- Make sure you have 50MB space available
- Use “quota –v” to check
- Install the data:

```
tar xvf /import/vlsi7/jgrad/socks/socks.tar
```

- This will create a folder ./socks
- Add this line to the bottom of your .cshrc file:

```
set path (/opt/rtems/bin $path)
```

- Test it by seeing if the compiler is found:

```
which sparc-rtems-g++
```

- Remember to run skew or vulcan. From other machines, do

```
ssh skew or ssh vulcan
```

SOCks Distribution Content

Directory	Description
./exe	Unix scripts to compile and simulate the SOCks ASIC
./doc	SOCks Documentation
./firmware	C/C++ Source Code Folder
./firmware/cube	Demo program #1
./firmware/bubblesort	Demo program #2
./sim	HDL Simulation Folder
./testbench	VHDL test-bench code
./testbench/include	Test-bench include files
./testbench/Tcl	TLC Scripts for NC-Sim
./hdl	HDL Folder
./hdl/custom	HDL for the Custom Logic
./hdl/TOP	HDL for the ASIC toplevel
./hdl/rtllib	Compiled HDL folder
./hdl/rtllib/custom	Compiled Custom logic
./hdl/rtllib/top	Compiled Top-level logic

SOCks Design Flow
1) Setting up a firmware folder

- Creating a new source code folder
  - cd ./firmware
  - cp –r bubblesort project1
- Use the “bubblesort” project as a template
- Put your C code into leon_test.c
- You can create as many folders as you want in the “firmware” folder
- For simulation you will then specify which firmware-folder to use

SOCks Design Flow
2) Compiling the Firmware

- Creating and compiling the source code

```
cd project1
[emacs | pico | etc.] leon_test.c
make
```

- All compilation instructions are in “Makefile”
- Simply type “make” and your code will be compiled and linked
- Use emacs or pico as your text editor
- The compilation result will be in “ram.dat” and “rom.dat”
- Those will be read into the testbench memories

SOCks Design Flow
3) Creating Digital Logic

- In this step digital logic is created that will go into the “custom” block
- This block communicates with the Leon through the Amba bus
- Creating and compiling the custom logic HDL

```
cd ../../hdl/custom
[emacs | pico | etc.] custom_top.v
cd ..
../exe/socks_compile
```

- Your folder has to be called “custom”
- The compiler will compile all files that end in “.v”
- “Compile” in this context means to build a HDL simulation model, not a binary fom C code

SOCks Design Flow
4) Running the Simulation

ILLINOIS INSTITUTE OF TECHNOLOGY
Transforming Lives. Inventing the Future.

- Running the simulation (replace “project1” with the name of your firmware folder)

```
cd ../sim
../exe/socks_sim project1
```

- Running the simulation and creating waveforms for the custom logic

```
../exe/socks_sim project1 partial
simvision&
```

- Running the simulation and creating waveforms for the entire design

```
../exe/socks_sim project1 full
simvision&
```

SOCks Design Flow
5) Clean Up

- Removing all temporary data to save space

```
cd ..
exe/socks_clean
```

- This removes all temporary data
- Can be helpful to minimize disk space usage
- Also useful to force the tool to re-compile and build everything

Example: The Cube Example

- This a very simple SOC:
  - Software is running on the Leon
  - Hardware acceleration is provided for “$x=y^3$”
- Steps to run this:

```
cd firmware/cube
make
cd ../../hdl
../exe/socks_compile
cd ../sim
../exe/socks_sim cube
cat INT.OUTPUT
```

- The firmware used the function to output numbers
- That output can be found in INT.OUTPUT

Example
The Bubblesort Program

- This is a software-only example
- The custom-logic is still “$x=y^3$” but we will not use it
- All we do is compile firmware and run it on the Leon

```
cd firmware/bubblesort
make
cd ../../hdl
../exe/socks_compile
cd ../sim
../exe/socks_sim bubblesort
```

- The output is printed directly on the screen