HDLs & FPGAs

Hey! I wanted to **build** a real Beta, not just simulate one...

There's no accounting for taste...

Quiz 4: this Friday.  Final Deadline: next Thu @ 5p
The Need for HDLs

A **specification** is an engineering contract that lists all the goals for a project:

- goals include area, power, throughput, latency, functionality, test coverage, costs (NREs and piece costs), ... Helps you figure out when you’re done and how to make engineering tradeoffs. Later on, goals help remind everyone (especially management) what was agreed to at the outset!

- top-down design: partition the project into modules with well-defined interfaces so that each module can be worked on by a separate team. Gives the SW types a head start too! *(Hardware/software codesign is currently all the rage...)*
The Need for HDLs (cont’d.)

A **behavioral model** serves as an executable functional specification that documents the exact behavior of all the individual modules and their interfaces. Since one can run tests, this model can be refined and finally verified through simulation.

We need a way to talk about what hardware should do without actually designing the hardware itself, i.e., we need to separate behavior from implementation. We need a **Hardware Description Language**

If we were then able to **synthesize** an implementation directly from the behavioral model, we’d be in good shape!
Using an HDL Description

So, we have an executable functional specification that
- documents exact behavior of all the modules and their interfaces
- can be tested & refined until it does what you want

An HDL description is the first step in a mostly automated process to build an implementation directly from the behavioral model.

Using an HDL Description

HDL description ➔ Logic Synthesis ➔ Gate netlist ➔ Place & route ➔ CPLD FPGA Stdcell ASIC

- HDL → logic
- map to target library
- optimize speed, area
- create floorplan blocks
- place cells in block
- route interconnect
- optimize (iterate!)

This is what you draw in Jade

6.004 Computation Structures
L24: HDLs & FPGAs, Slide #4
A Tale of Two HDLs

**VHDL**

- ADA-like verbose syntax, lots of redundancy (which can be good!)
- Extensible types and simulation engine. Logic representations are not built in and have evolved with time (IEEE-1164).
- Design is composed of entities each of which can have multiple architectures. A configuration chooses what architecture is used for a given instance of an entity.
- Behavioral, dataflow and structural modeling. Synthesizable subset...
- Harder to learn and use, not technology-specific, DoD mandate

**Verilog**

- C-like concise syntax
- Built-in types and logic representations. Oddly, this led to slightly incompatible simulators from different vendors.
- Design is composed of modules.
- Behavioral, dataflow and structural modeling. Synthesizable subset...
- Easy to learn and use, fast simulation, good for hardware design.

New: Bluespec, System Verilog...
Verilog Example: Beta Register File

```verilog
// 2-read, 1-write 32-location register file
module regfile(
    input [4:0] ra1,  // address for read port 1 (Reg[RA])
    output [31:0] rd1, // read data for port 1
    input [4:0] ra2,  // address for read port 2 (Reg[RB], Reg[RC] for ST)
    output [31:0] rd2, // read data for port 2
    input clk,
    input werf,      // write enable, active high
    input [4:0] wa,  // address for write port (Reg[RC])
    input [31:0] wd   // write data
);

reg [31:0] registers[31:0]; // the register file itself (local)

// read paths are combinational, check for reads from R31
assign rd1 = (ra1 == 31) ? 0 : registers[ra1];
assign rd2 = (ra2 == 31) ? 0 : registers[ra2];

// write port is active only when WERF is asserted
always @(posedge clk)
    if (werf) registers[wa] <= wd;
endmodule
```
module pc(
  input clk,
  input reset,                // forces PC to 0x80000000
  input [2:0] pcsel,        // selects source of next PC
  input [15:0] offset,      // inst[15:0]
  input [31:0] jump_addr,   // from Reg[RA], used in JMP instruction
  output [31:0] branch_addr, // send to datapath for LDR instruction
  output reg [31:0] pc,      // used as address for instruction fetch
  output [31:0] pc_plus_4,   // saved in regfile during branches, JMP, traps
);

wire [30:0] pcinc;
wire [31:0] npc;

// the Beta PC increments by 4, but won’t change supervisor bit
assign pcinc = pc + 4;
assign pc_plus_4 = {pc[31],pcinc};

// branch address = PC + 4 + 4*sxt(offset)
assign branch_addr = {0,pcinc + {{13{offset[15]}},offset[15:0],2'b00}};

assign npc = reset ? 32'h80000000 :  
  (pcsel == 0) ? {pc[31],pcinc} :           // normal
  (pcsel == 1) ? {pc[31],branch_addr[30:0]} : // branch
  (pcsel == 2) ? {pc[31] & jump_addr[31],jump_addr[30:0]} : // jump
  (pcsel == 3) ? 32'h80000004 : 32'h80000008; // illop, trap

// pc register, pc[31] is supervisor bit and gets special treatment
always @(posedge clk) pc <= npc;
endmodule
FPGA configuration is stored in RAM and is loaded from a high-density serial FLASH ROM during power up. Change the ROM contents and you change the circuit implemented by the FPGA!

XC3S500E:
- 46x34 CLBs
- 9312 LUTs
- 500K “gates”
- 20 18k-bit BRAMs
- 20 18x18 MULs
- ~$30 w/ 256 pins
FPGA Die Photo

Digital Clock Management

Dual-port RAM

Embedded Multipliers

Configurable logic blocks

Multi-standard I/O blocks

Source: Xilinx Website.
Xilinx Slice

Source: Xilinx 7 Series CLB
Xilinx Synchronous Block Memory

- **Data_in** → **Data_out**
- **Address**
- **WE**
- **CLK**

18k bits total:
- 16k x 1
- 8k x 2
- 4k x 4
- 2k x (8+1)
- 1k x (16+2)
- 512 x (32+4)

Source: Xilinx App Note 463
Programmable Interconnect

Single- and Double-Length Lines, with Programmable Switch Matrices (PSMs)
Wires are not ideal!
Clocks Need Special Attention

Source: Xilinx Spartan-3 FPGA User Guide
Sandbox: Digilent Nexys 4 Board

http://www.digilentinc.com
Example: Beta on a Board
Beta2

An implementation of the 6.004 Beta processor with a 2-stage pipeline (IF, EXE) and one annulled branch delay slot. Memory operations (LD/LDR/ST) require two cycles in the EXE stage: the address is computed in the first cycle, the memory access made in the second (during which msel=1 and the IF stage is stalled). Other instructions take a single EXE cycle. Other changes:

- branch and LDR address arithmetic is performed by the ALU adder. To simplify the PCSEL logic JMP addresses are also routed through the ALU.
- a single main memory port is multiplexed between instruction fetch (msel=0) and memory accesses (msel=1).
- since regfile read and write are in the same stage, no bypassing is needed.
100: LD (R31, 6004, R2)
104: ADDC (R2, 47, R2)
108: ST (R2, 44, R31)
10C: XORC (R2, -1, R2)
110: ...

... 

6004: 123

Instruction Pipeline Diagram

CLK

Address

Data_out

Data_in

MWE

EXE stage inst.

msel

* Stalled in pipeline
Beta2 Floorplan

ALU adder
PC adder
2kx32 memory (4 BRAMs)

Font
ROM

Display mem
The Power of Engineering Abstractions

Good abstractions allow us to reason about behavior while shielding us from the details of the implementation.

Corollary: implementation technologies can evolve while preserving the engineering investment at higher levels.

Leads to hierarchical design:
- Limited complexity at each level ⇒ shorten design time, easier to verify
- Reusable building blocks

Cloud
Virtual Machines
Programming languages
Instruction set + memory
Bits, Logic gates
Lumped component model
Insulator, conductor, semiconductor
6.004: The Big Lesson

You’ve built, debugged, understood a complex computer from FETs to OS... what have you learned?

**Engineering Abstractions:**

- Understanding of their technical underpinnings
- Respect for their value
- Techniques for using them

But, most importantly:

The self assurance to discard them, in favor of new abstractions!

Good engineers *use* abstractions; GREAT engineers *create* them!
Things to look forward to...

6.004 is only an appetizer!

- Processors
  - Superscalars
  - Deep pipelines
  - Multicores

- Languages & Models
  - Python/Java/Ruby/...
  - Objects/Streams/Aspects
  - Networking

- Tools
  - Design Languages
  - FPGA prototyping
  - Timing Analyzers

- Algorithms
  - Arithmetic
  - Signal Processing
  - Language implementation

- Systems Software
  - Storage
  - Virtual Machines
  - Networking
Thinking Outside the Box

Will computers always look and operate the way computers do today?

Some things to question:

– Well-defined system “state”
– Silicon-based logic
– Logic at all
– Programming
Computing is slow...
The future is in your hands.
Start innovating!

-- 6.004 Staff

The only problem with Haiku is that you just get started and then

-- Roger McGough