EECS151 : Introduction to Digital Design and ICs

Lecture 3 – Design Process, Verilog I

Bora Nikolić

August 2021: Esperanto at HotChips
The ET-SoC-1 is fabricated in TSMC 7nm
• 24 billion transistors
• Die-area: 570 mm²
1088 ET-Minion energy-efficient 64-bit RISC-V processors
• Each with an attached vector/tensor unit
• Typical operation 500 MHz to 1.5 GHz expected
4 ET-Maxion 64-bit high-performance RISC-V out-of-order processors
• Typical operation 500 MHz to 2 GHz expected
1 RISC-V service processor
Over 160 million bytes of on-die SRAM used for caches and scratchpad memory
Root of trust for secure boot
Power typically < 20 watts, can be adjusted for 10 to 60+ watts under SW control

D. Ditzel, HotChips'21

Review

• Design process involves specification, modeling, architecture design, RTL
design, physical design, manufacturing
  • Validation: Have we built the right thing?
  • Verification: Have we built the thing right?

• Combinational logic: Outputs depend only on inputs

• Sequential logic: Outputs depend on the inputs and the state
How Did We Get Here?

IC Design in the 70’s and early 80’s

- Circuit design, layout, and processing tightly linked.
- Logic design and layout was all done by-hand
- Chip design was typically done by vertically integrated companies, who designed and fabricated their own chips
  - Fabs weren’t that expensive back then

The Intel 4004 microprocessor, which was introduced in 1971. The 4004 contained 2300 transistors and performed 60,000 calculations per second. Courtesy: Intel.
Early Design Practice

• Initially, designs were represented by hand drawings. Then masks were made by transferring drawings to rubylith.
  • Base layer of heavy transparent dimensionally stable Mylar. A thin film of deep red cellophane-like material covers the base layer. Patterns formed by cutting (often by hand) the transparent covering.
  • Later transition to an electronic format (CIF, GDS) meant: Layouts easily be stored and transmitted. Written to tape and transferred to manufacturer (tape-out). Transmitted over the network (new idea back then). Software could automatically check for layout errors. Generated from a program - huge idea.

The Start of the IC Design Revolution
Geometric Design Rules

• Early on, to generate the mask information for fabrication, the designer needed intimate knowledge of the manufacturing process. Even once this knowledge was distilled to a set of “Geometric Design Rules”, this set of rules was voluminous.

• Academics (C. Mead/L.Conway and others) came up with a much simplified set of design rules (single page description)

  ▶ Sufficiently small set that designers could memorize. Sufficiently abstract to allow process engineers to shrink the process and preserve existing layouts. Process resolution becomes a “parameter”, λ.

Key Development: Silicon Foundries

• Separate the designer from the fabricator: Modeled after the printing industry. (Very few authors actually own and run printing presses!)

• Simple standard geometric design rules where the key: these form the “contract” between the designer and manufacturer.

• Designer sends the layout, foundry manufactures the chip and send back.

  ▶ A scalable model for the industry: IC fab is expensive and complex. Amortizes the expense over many designers. Designers and companies not held back by need to develop and maintain large expensive factories. “Fabless” semiconductor companies - lots of these and very few foundries.

TSMC, Global Foundries, UMC, Samsung, SMIC, ...
Computer Aided Design (1)

Several advances lead to the development of interactive tools for generating layout:

- Computer-based layout representation (CIF, GDS).
- Advances in computer graphics (Ivan Sutherland) and display devices.
- Personal “workstation” (Xerox Alto - Chuck Thacker). “Back room” computers didn’t have the necessary bandwidth to the display.
- Berkeley version - MAGIC

Early ’80’s Design Methodology and Flow

Schematic + Full-Custom Layout

- SPICE for timing,
- Switch-level simulation for overall functionality,
- Hand layout,
- No power analysis,
- Layout verified with geometric design rule checker (DRC) and later also layout versus schematic (LVS) checkers
Computer Aided Design (2)

- For some time after CIF was invented: Layout was generated by hand, then typed in as a CIF file with a text editor.

- Layout compilers
  - Soon some designers started embedding CIF primitives in conventional programming languages: LISP, pascal, fortran, C.
  - This allows designers to write programs that generated layout. Such programs could be parameterized:

  ```
  define GENERATE_RAM(rows, columns) {
    for I from 1 to rows
      for J from 1 to columns
        (GENERATE_BITCELL)}
  GENERATE_RAM(128, 32);
  ```

  - Lead to circuit/layout generation from higher level descriptions.

- Eventually, Cadence and Synopsys (formed out of Berkeley) built tools to generate layout from HDLs.

Administrivia

- Everyone needs to be enrolled by now!
  - If you are still waitlisted, move to a different lab, discussion

- Lab 1 this week
- Lab 2 is more involved
  - Be prepared
  - Verilog primer

- Homework 1 posted this week, due next Friday
  - Start early

- Labor day on Monday, no lecture/discussion
Implementation Alternatives

### Implementation Alternative Summary

<table>
<thead>
<tr>
<th>Alternative</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full-custom:</td>
<td>Every transistors layout hand-drawn and optimized.</td>
</tr>
<tr>
<td>Standard-cell:</td>
<td>Logic gates and “macros” automatically placed and routed.</td>
</tr>
<tr>
<td>Gate-array (structured ASIC):</td>
<td>Partially prefabricated wafers with arrays of transistors customized with metal layers or vias.</td>
</tr>
<tr>
<td>FPGA:</td>
<td>Prefabricated chips that can be customized “in the field”</td>
</tr>
<tr>
<td>Microprocessor (e.g. Single-board computers):</td>
<td>Function customized through software.</td>
</tr>
<tr>
<td>Domain-specific processor:</td>
<td>Special instruction set interpreters (ex: DSP, NP, GPU).</td>
</tr>
</tbody>
</table>

These days, “ASIC” almost always means standard-cell design.
**Full-Custom**

- Circuit styles and transistors are custom sized and drawn to optimize performance and power.
- High NRE (non-recurring engineering) costs
  - Time-consuming layout iterations
- Common today for **analog design**.
- In digital world – for **memories** and **arrays**.

**Standard Cells**

- Library of logic gates (1000-2000)
  - Combinational: NANDs, NORs, ... Sequential: Flip-flops, latches, ...
- Each cell comes complete with:
  - layout, schematic, symbol
  - Logic (Verilog), timing, power models.
- Chip layout is automatic, reducing NREs (usually no manual layout).
- Memories, I/O and analog components complete the ASIC
**Gate Array**

- Prefabricated wafers of, rows of transistors. Customize as needed with “back-end” metal processing (contact cuts, metal wires). Could use a different factory.
- CAD software understands how to make gates and registers.

**Diagram:**
- Two-step manufacture:
  - First (deep) processing steps lead to standard masks, then to base wafers.
  - Customization: contacts & metal layers lead to custom masks, then to ASIC.

**Gate Array**

- Shifts large portion of design and mask NRE to vendor.
- Shorter design and processing times, reduced time to market for user.
- Highly structured layout with fixed size transistors leads to large sub-circuits (ex: Flip-flops) and higher per die costs.
- Memory arrays are particularly inefficient, so often prefabricated.
- Displaced by field-programmable gate arrays

**Images:**
- Sea-of-gates, structured ASIC, master-slice.
Field Programmable Gate Arrays (FPGA)

- Two-dimensional array of simple logic- and interconnection-blocks.
- Typical architecture: Look-up-tables (LUTs) implement any function of n-inputs ($n=3$ in this case).
- Optional flip-flop with each LUT.
- Fuses, EPROM, or Static RAM cells are used to store the "configuration".
  - Here, it determines function implemented by LUT, selection of Flip-flop, and interconnection points.
- Many FPGAs include special circuits to accelerate adder carry-chain and many special cores: RAMs, MAC, Ethernet, USB, PCIe, CPUs, ...

FPGA versus ASIC

- **ASIC**: Higher NRE costs (10's of $M$). Relatively low cost per die (10's of $ or less).
- **FPGAs**: Low NRE costs. Relatively low silicon efficiency $\Rightarrow$ high cost per part (> 10's of $ to 1000's of $).
- **Cross-over volume** from cost effective FPGA design to ASIC was often in the 100K range.
- **ASICs** often have $>10x$ higher performance and 10x lower power for the same application.
Microprocessors

- Where relatively low performance and/or high flexibility is needed, a viable implementation alternative:
  - Software implements desired function
  - “Microcontroller”, often with built in nonvolatile program memory and used as single function.

- Two “abstraction” levels:
  - Instruction Set Architecture (ISA)
  - “Synthesizable” RTL model (“soft core”, available in HDL)

- Their implementation can be both ASIC or FPGA

System-on-Chip (SOC)

- Brings together: standard cell blocks, custom analog blocks, processor cores, memory blocks, embedded FPGAs, ...

- Standardized on-chip buses (or hierarchical interconnect) permit “easy” integration of many blocks.  
  - Ex: AXI, ...

- “IP Block” business model: Hard- or soft-cores available from third party vendors.

- ARM, inc. is the example.  Hard- and “synthesizable” processors.

- ARM and other companies provide, Ethernet, USB controllers, analog functions, memory blocks, ...

- Pre-verified block designs, standard bus interfaces (or adapters) ease integration - lower NREs, shorten TTM.
Quiz

True or false?

a) FPGAs are usually faster than ASICs

b) Verilog syntax is different for FPGAs and ASICs

c) One cannot change the FPGA function in a deployed product

ASIC Design
Verilog to ASIC Layout Flow

- “Push-button” approach

```verilog
module adder64 (a, b, sum);
  input [63:0] a, b;
  output [63:0] sum;
  assign sum = a + b;
endmodule
```

Standard cell layout methodology

- With limited # metal layers, dedicated routing channels were needed
- Currently area dominated by wires

Old times: 1μm, 2-metal process
Modern sub-100nm process
“Transistors are free things that fit under wires”
Standard Cells (ASAP7)

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Schematic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1x Inverter</td>
<td>INVx1</td>
<td><img src="image" alt="INVx1 Schematic" /></td>
</tr>
</tbody>
</table>

Verilog for logic sim

```verilog
`celldefine
module INVx1_ASAP7_75t_R (Y, A);
output Y;
input A;
// Function
not (Y, A);
// Timing
specify
(A => Y) = 0;
endspecify
endmodule
`endcelldefine
```

Netlist for Spice sim

```
.lib for synthesis
cell (INVx11_ASAP7_75t_R) {
area : 3.03264;
call Leakage_power : 562.699;
pin (VDD) {
pin_type : primary_power;
voltage_name : "VDD";
}
pin (VSS) {
pin_type : primary_ground;
voltage_name : "VSS";
}
leakage_power () {
value : 538.867;
when : "(A * !Y)"
related_pg_pin : VDD;
}
```

+ .lef for place and route, and other files for LVS and DRC

Standard Cells

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Schematic</th>
</tr>
</thead>
<tbody>
<tr>
<td>1x Inverter</td>
<td>INVx2</td>
<td><img src="image" alt="INVx2 Schematic" /></td>
</tr>
<tr>
<td>2x Inverter</td>
<td>INVx2</td>
<td><img src="image" alt="INVx2 Schematic" /></td>
</tr>
<tr>
<td>1x NAND2</td>
<td>NAND2x1</td>
<td><img src="image" alt="NAND2x1 Schematic" /></td>
</tr>
</tbody>
</table>
Standard cell placement

- Standard cells abut and flip
- Router connects inputs and outputs

Macro modules

256\times32 (or 8192 bit) SRAM generated by hard-macro module generator

- Generate highly regular structures (entire memories, multipliers, etc.) with a few lines of code
- Verilog models for memories automatically generated based on size
Hardware Description Languages

**Verilog:**
- Simple C-like syntax for structural and behavior hardware constructs
- Mature set of commercial tools for synthesis and simulation
- Used in EECS 151 / 251A

**VHDL:**
- Semantically very close to Verilog
- More syntactic overhead
- Extensive type system for “synthesis time” checking

**System Verilog:**
- Enhances Verilog with strong typing along with other additions

**BlueSpec:**
- Invented by Prof. Arvind at MIT
- Originally built within the Haskell programming language
- Proprietary, available commercially: bluespec.edu

**Chisel:**
- Open, developed at UC Berkeley
- Used in CS152, CS250
- Available at: chisel.eecs.berkeley.edu
Verilog: Brief History

- Invented as simulation language. Synthesis was an afterthought. Many of the basic techniques for synthesis were developed at Berkeley in the 80’s and applied commercially in the 90’s.
- Around the same time as the origin of Verilog, the US Department of Defense developed VHDL (A double acronym! VSIC (Very High-Speed Integrated Circuit) HDL). Because it was in the public domain it began to grow in popularity.
- Afraid of losing market share, Cadence opened Verilog to the public in 1990.
- An IEEE working group was established in 1993, and ratified IEEE Standard 1394 (Verilog) in 1995. We use IEEE Std 1364-2005.
- Verilog is the language of choice of Silicon Valley companies, initially because of high-quality tool support and its similarity to C-language syntax.
- VHDL is still popular within the government, in Europe and Japan, and some Universities.
- Most major CAD frameworks now support both.
Verilog Introduction

- A **module** definition describes a component in a circuit
- Two ways to describe module contents:
  - **Structural Verilog**
    - List of sub-components and how they are connected
    - Just like schematics, but using text
    - Tidious to write, hard to decode
    - You get precise control over circuit details
    - May be necessary to map to special resources of the FPGA/ASIC
  - **Behavioral Verilog**
    - Describe what a component does, not how it does it
    - Synthesized into a circuit that has this behavior
    - Result is only as good as the tools
  - Build up a hierarchy of modules. Top-level module is your entire design (or the environment to test your design).

Logic Synthesis

- Verilog and VHDL started out as simulation languages but soon programs were written to automatically convert Verilog code into low-level circuit descriptions (netlists).
- Synthesis converts Verilog (or other HDL) descriptions to a logic mapping by using technology-specific primitives:
  - For FPGA: LUTs, FlipFlops, and BRAMs.
  - For ASICs: standard cells and memory macros.
- In addition, synthesis algorithms optimize the implementation for delay and power.
Verilog Modules and Instantiation

- Modules define circuit components.
- Instantiation defines hierarchy of the design.

```verilog
module addr_cell (a, b, cin, s, cout);
    input   a, b, cin;
    output  s, cout;
endmodule
```

Note: A module is not a function in the C sense. There is no call and return mechanism. Think of it more like a hierarchical data structure.

Structural Model - XOR example

```verilog
module xor_gate (out, a, b);
    input   a, b;
    output  out;
    wire    aBar, bBar, t1, t2;
    not    invA (aBar, a);
    not    invB (bBar, b);
    and    and1 (t1, a, bBar);
    and    and2 (t2, b, aBar);
    or      or1 (out, t1, t2);
endmodule
```

Notes:
- The instantiated gates are not “executed”. They are active always.
- xor gate already exists as a built-in (so really no need to define it).
- Undeclared variables assumed to be wires. Don't let this happen to you!
Structural Example: 2-to1 mux

```verilog
module mux2 (in0, in1, select, out);
  input in0, in1, select;
  output out;
  wire s0, w0, w1;
  not (s0, select);  // C++ style comments
  and (w0, s0, in0), (w1, select, in1);
  or  (out, w0, w1);
endmodule // mux2
```

Multiple instances can share the same "master" name.

```verilog
module mux4 (in0, in1, in2, in3, select, out);
  input in0, in1, in2, in3;
  input [1:0] select;
  output out;
  wire w0, w1;
  mux2
  m0 (.select(select[0]), .in0(in0), .in1(in1), .out(w0)),
  m1 (.select(select[0]), .in0(in2), .in1(in3), .out(w1)),
  m3 (.select(select[1]), .in0(w0), .in1(w1), .out(out));
endmodule // mux4
```

Signal array. Declares select[1], select[0]

Named ports. Highly recommended.
Summary

• The design process has changed over time

• ASIC design process is dominant today:
  • Logic synthesis
  • Automated place and route
  • Abstraction layer: Standard cells
  • The design flow involves translating an abstracted RTL design into physical logic gates, and verifying that it has been done correctly

• Verilog is commonly used to describe RTL
  • Structural or
  • Behavioral