ECE 551
System on Chip Design

Introduction

Garrett S. Rose
Fall 2018
Outline

- Review of IC design basics
- Design partitioning and design approaches
- Introduction to semi-custom design approach
- Introduction to automated approach
- A snapshot of the tools you will use
## Types of Digital Systems

- **ASIC**: Application Specific Integrated Circuits
  - Usually fabricated in silicon, custom or synthesized

- **Microprocessor**

- **FPGA**: Field Programmable Gate Arrays
  - Reconfigurable logic blocks configured for use

### General Purpose
- Von Neumann Processors
- Bit-Sliced Processors
- Array/Vector Processors

### Application Specific
- Reconfigurable Logic
- Custom Logic
MOS Integrated Circuits

- 1970’s processes usually had only nMOS transistors
  - Inexpensive, but consume power while idle

- 1980s-present: CMOS processes for low idle power

Intel 1101 256-bit SRAM  Intel 4004 4-bit µProc

Adapted from Harris’ slides from Harvey Mudd. Copyright 2005 Addison-Wesley.
Moore’s Law

- 1965: Gordon Moore plotted transistor on each chip
  - Fit straight line on semilog scale
  - Transistor counts have doubled every 26 months

Integration Levels

SSI: 10 gates
MSI: 1000 gates
LSI: 10,000 gates
VLSI: > 10k gates

Adapted from Harris’ slides from Harvey Mudd.
Why Scaling?

• Technology shrinks by 0.7/generation
• Every generation can integrate 2x more functions per chip; chip cost does not increase significantly
• Cost of a function decreases by 2x
• But …
  – How to design chips with more and more functions?
  – Design engineering population does not double every two years…
• Hence, a need for more efficient design methods
  – Exploit different levels of abstraction
A Microprocessor Example

- Intel Coffee Lake
  - Up to 6 cores (12 threads)
  - 2.9 to 4.8GHz clock (Core i9)
  - Still 14nm technology...
    - Cannon Lake (shrink) is 10nm
    - Moore's Law at an end?
  - Over 1 billion transistors
  - 12MB SmartCache
  - Built-in GPU – a “system on chip”
  - Some custom (“neat” areas);
    many synthesized components (“messy” areas)
Designing a Microprocessor

• Requires a large team and a lot of time

• Some components can be synthesized
  – Design written in code (e.g. VHDL)
  – Synthesis tool generates netlist from code

• Other components must be custom designed
  – Blocks on critical path (i.e. critical to system performance)
An ASIC Example: Avalon Bitcoin Mining ASIC

- Application very specific: device that mines bitcoins
- Bitcoin mining can be time consuming – ASIC works fast
- Almost entirely synthesized from standard cells
  - “messy” look of layout
  - Not as fast as full custom
Structured Design

- **Hierarchy**: Divide and Conquer
  - Recursively system into modules
- **Regularity**
  - Reuse modules wherever possible
  - Ex: Standard cell library
- **Modularity**: well-formed interfaces
  - Allows modules to be treated as black boxes
- **Locality**
  - Physical and temporal
Design Partitioning

- **Architecture:** User’s perspective, what does it do?
  - Instruction set, registers
  - MIPS, x86, Alpha, PIC, ARM, …
- **Microarchitecture**
  - Single cycle, multicycle, pipelined, superscalar?
- **Logic:** how are functional blocks constructed
  - Ripple carry, carry lookahead, carry select adders
- **Circuit:** how are transistors used
  - Complementary CMOS, pass transistors, domino
- **Physical:** chip layout
  - Datapaths, memories, random logic
Design Abstraction Levels

Adapted from Harris’ slides from Harvey Mudd. Copyright 2005 Addison-Wesley.
Gajski Y-Chart

Adapted from Harris' slides from Harvey Mudd. Copyright 2005 Addison-Wesley.
Design Metrics

How to evaluate performance of a digital circuit (gate, block, …)?

- Cost
- Reliability
- Scalability
- Speed (delay, operating frequency)
- Power dissipation
- Energy to perform a function
VLSI Design Approaches

- **Custom**: A design that has been carefully crafted by the designer including manual layout. This is Intro. to VLSI (ECE 433/533).

- **Semi-custom**: Some custom crafted blocks including custom blocks using automation (i.e., SKILL) along with some fully automated blocks. Usually makes use of predefined, well optimized blocks that cannot be changed.

- **Automated**: The use of a hardware description language (VHDL, Verilog, SystemC, etc.) and CAD tools to synthesize a design from code to Silicon.
Custom: Tightly packed but irregular regions

Automated: Very irregular “rats’ nest” regions

Memory (very regular & dense) is custom but can easily be drawn using SKILL

Image courtesy of Intel Corp.
ASIC/SOC Design Flow

- SoC design should always start with a clear **design flow**
- An integrated circuit design flow shows the particular steps taken for each design stage and highlights CAD tools used
ASIC/SOC Design Flow

1. Requirements
2. RTL Model
3. HDL and/or Schematic Entry
4. HDL Code
5. Synthesis of HDL Code
6. Gate-Level Netlist
7. Place & Route and/or Mapping
8. Physical Design
9. Fabricate (ASIC) or Configure (FPGA)

Simulation at each step.
Automated Design

- Typically, an application specific integrated circuit (ASIC) is designed using commercial tools.
- A *standard cell library* is provided by a vendor (i.e., fab facility) which includes a set of basic gates.
- The design is specified using an HDL (such as Verilog or VHDL) and synthesized to a gate-level netlist in terms of the gates in the library.
- Finally, this netlist is used for floorplanning and place & route using another tool to produce layout.
- Custom blocks can be included but much of the flow is automated starting with the HDL.
Standard Cell Library

- The standard cell library is usually provided by a vendor.
- A lot is required for a library:
  - Optimized schematic and layout for each gate
  - Timing files
  - Input/output pin definitions

- Typical standard cells:
  INV, AND2, AND3, OR2, OR3, XOR2, AOI21, AOI22, etc.
HDLs

- Hardware Description Languages
  - Widely used in logic design
  - Verilog and VHDL

- Describe hardware using code
  - Document logic functions
  - Simulate logic before building
  - Synthesize code into gates and layout
    - Requires a library of standard cells
module fulladder(input a, b, c, 
 output s, cout);
 sum s1(a, b, c, s);
carry c1(a, b, c, cout);
endmodule

module carry(input a, b, c, 
 output cout)
 assign cout = (a&b) | (a&c) | (b&c);
endmodule

Verilog Example
Circuit Design

- How should logic be implemented?
  - NANDs and NORs vs. ANDs and ORs?
  - Fan-in and fan-out?
  - How wide should transistors be?

- These choices affect speed, area, power

- Logic synthesis makes these choices for you
  - Good enough for many applications
  - Hand-crafted circuits are still better
Example: Carry Logic

\texttt{assign} \ cout = (a\&b) \mid (a\&c) \mid (b\&c);

![Logic Diagram]

Transistors? Gate Delays?
Example: Carry Logic

assign cout = (a&b) | (a&c) | (b&c);

Transistors? Gate Delays?
module carry(input a, b, c, 
          output cout)

  wire x, y, z;
  and g1(x, a, b);
  and g2(y, a, c);
  and g3(z, b, c);
  or g4(cout, x, y, z);
endmodule
Place & Route

- Cadence SOC Encounter (right) is a tool used for:
  - Floorplanning
  - Placement
  - Routing
Some CAD Tools

- **ModelSim** – HDL (SystemVerilog, VHDL & Verilog) simulator
- **QuestaSim** (newer!) – HDL (SystemVerilog, VHDL & Verilog) simulator
- **Design Compiler** – Synthesis tool that takes HDL code and synthesizes to a gate-level netlist (typically a Verilog netlist) of standard cells
- **Encounter** – Physical design tool for generating layout from synthesized gate-level netlist
- **Xilinx Vivado** – Tools for synthesis, place & route, mapping, etc. for Xilinx FPGA implementations

- Will mostly use QuestaSim and Xilinx tools in this class
FPGA Based Design

- Field Programmable Gate Arrays (FPGA) can be used as a “cost effective” ASIC
- FPGA configured using HDL – SystemVerilog, Verilog or VHDL
- FPGAs are also useful for early prototyping of silicon-based ASIC implementations – save cost, mitigate risk

- We will primarily focus on FPGA based designs – think of designs as FPGA prototypes of small ASIC/SOC designs
Field Programmable Gate Arrays

Xilinx Spartan-II Block Diagram

Xilinx Spartan-II CLB
Digilent Nexys 4 Boards

- Form lab/project groups soon!
- Each group gets one Nexys 4 board
- Will use for labs and project

- Nexys 4 includes Xilinx Artix-7 FPGA
  - 15K logic slices (100K cells)
  - 4Mbits block RAM
  - 240 DSP slices
  - On-chip ADC
Basic FPGA Design Steps

- Follow the ASIC/SOC Flow
- Develop code using SystemVerilog
- Simulate SystemVerilog using:
  - Mentor Graphics QuestaSim, ModelSim, or Xilinx Vivado
  - Can also simulate VHDL and Verilog
- Synthesize with Xilinx Vivado
- Configure with Xilinx Vivado

Optional: Simulate post-synthesis netlist (usually Verilog)
Important To Do Items

• Login to one of the EECS “ada” servers:
  - There are 8 ada machines: ada1, ada2, ... ada7, ada8
  - All ada machines run the same OS with roughly the same setups
  - Example: ada5.eecs.utk.edu
  - User ID and password same as email

• QuestaSim and ModelSim run on the ada machines, can also run on laptop/PC
• If you're using Windows, may need MobaXterm for ssh connection
• The ada servers also support RealVNC connections (faster than ssh):
  https://www.eecs.utk.edu/resources/it/eecs-it-knowledge-base/remote-access/realvnc/

• More instructions posted on Canvas
Summary

- VLSI design occurs at several abstraction layers
- Can design manually (custom) or using automated CAD tools
- Most VLSI systems are in a “gray area” – some custom and some automation
- *This class focuses mostly on top-down, automated approach*
- We will study ASIC/SOC design in general and will use FPGAs as our primary platform for prototyping designs

Welcome... this should be enjoyable for us all!