Part 1: ASIC Design Overview

Topic 1: Hardware Description Languages

Topic 2: CMOS Devices

Topic 3: CMOS Circuits

Topic 4: Full-Custom Design Methodology

Topic 5: Automated Design Methodologies

Topic 6: Closing the Gap

Topic 7: Clocking, Power Distribution, Packaging, and I/O

Topic 8: Testing and Verification
Design Domains, Abstractions, and Principles

- Modularity
- Hierarchy
- Encapsulation
- Regularity
- Extensibility

Full-Custom Design
Behavioral, Structural, and Physical Abstractions

Adapted from [Ellervee'04]
Behavioral, Structural, and Physical Abstractions

Adapted from [Ellervee'04]
Computer Engineering Stack Abstractions

- Application
- Algorithm
- Programming Language
- Operating System
- Instruction Set Architecture
- Microarchitecture
- Register-Transfer Level
- Gate Level
- Circuits
- Devices
- Technology

- Processors
- Memories
- Networks

- Control
- Datapath
- Memories

- Logic
- State
- Interconnect

- Transistors
- Wires
Design Principles in VLSI Design

- **Modularity** – Decompose into components with well-defined interfaces
- **Hierarchy** – Recursively apply modularity principle
- **Encapsulation** – Hide implementation details from interfaces
- **Regularity** – Leverage structure at various levels of abstraction
- **Extensibility** – Include mechanisms/hooks to simplify future changes
Design Principle: Modularity

- Separate design into components with well-defined interfaces
- Reason, design, and test components in isolation
- Interface may or may not encapsulate implementation
Design Principle: Modularity

Modularity can also impact electrical and physical characteristics

**Electrical Modularity**

What happens if we cascade many of these transmission gate multiplexers?

**Physical Modularity**

pwr/gnd rails & wells in fixed locations so they connect via abutment

Adapted from [Weste’11]
Design Principle: Hierarchy

Recursively apply modularity principle until complexity of submodules is manageable
Design Principle: Hierarchy
Design Principle: Encapsulation

- Modularity requires well-defined interfaces, but these interfaces might still expose significant implementation details (e.g., interface in control/datapath split reveals many details of the implementation)

- Choose interfaces that hide implementation details where possible to enable more robust composition

- Lab 1 multipliers all use a latency-insensitive val/rdy message interface to hide timing details, any one of these can be swapped into a processor and should work without modification
  - Fixed-latency iterative multiplier
  - Variable-latency iterative multiplier
  - Pipelined multiplier
Design Principle: Regularity

- Modularity, hierarchy, and encapsulation can still lead to many different kinds of modules which can increase design complexity.

- Choose a hierarchical decomposition to leverage structure and thus facilitate reuse and reduce complexity.

- Both structural and physical regularity can be exploited.
Design Principle: Regularity

- **Physical Regularity**
  - in Datapaths
  - in Memories

- **Structural Regularity**
  - in Ripple-Carry Adder
  - in Network

- **Common Network Design**
  - Common Cache Design

Diagram showing network designs with regularities in physical and structural aspects.
Design Principle: Extensibility

Simple form of polymorphism enables varying bitwidth of operands

Difficult with full-custom design methodology!

Parameterization of network and caches enables reuse; static elaboration could enable varying the number of cores and the types of components
Agenda

Design Domains, Abstractions, and Principles

Full-Custom Design
  Cells
  Datapaths
  Memories
  Control
Full Custom Design

Key is that all circuits and transistors are optimized for specific context

Intel 4004
Overview of Full Custom Design Methodology

Design Domains, Abstractions, and Principles • Full-Custom Design •

Adapted from [Weste’11]
Custom Cells: Lambda-Based Design Rules

One lambda = one half of the “minimum” mask dimension, typically the length of a transistor channel. Usually all edges must be “on grid”, e.g., in the MOSIS scalable rules, all edges must be on a lambda grid.
Custom Cells: Sample “Lambda” Layout

Adapted from [Terman'02]
Lambda-based design rules are based on the assumption that one can scale a design to the appropriate size before manufacture. The assumption is that all manufacturing dimensions scale equally, an assumption that “works” only over some modest span of time. For example: if a design is completed with a poly width of $2\lambda$ and a metal width of $3\lambda$ then minimum width metal wires will always be 50% wider than minimum width poly wires.

Consider the following data from Weste, Table 3.2:

<table>
<thead>
<tr>
<th>contacted metal pitch</th>
<th>lambda rule</th>
<th>lambda</th>
<th>micron rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2 * contact size</td>
<td>1\lambda</td>
<td>0.5\mu</td>
<td>0.375\mu</td>
</tr>
<tr>
<td>contact surround</td>
<td>1\lambda</td>
<td>0.5\mu</td>
<td>0.5\mu</td>
</tr>
<tr>
<td>metal-to-metal spacing</td>
<td>3\lambda</td>
<td>1.5\mu</td>
<td>1.0\mu</td>
</tr>
<tr>
<td>contact surround</td>
<td>1\lambda</td>
<td>0.5\mu</td>
<td>0.5\mu</td>
</tr>
<tr>
<td>1/2 * contact size</td>
<td>8\lambda</td>
<td>4\mu</td>
<td>2.75\mu</td>
</tr>
</tbody>
</table>

Scaled design is legal but much larger than it needs to be!

Adapted from [Terman'02]
Custom Cells: Sticks and Compaction

Stick diagram

Horizontal constraints for compaction in X

Compact X then Y

Compact Y then X

Compact X with jog insertion, then Y

Adapted from [Terman'02]
Custom Cells: Example Stick Diagram
Custom Cells: Example Stick Diagram
Custom Cells: Cell “Styles”

Vertical Gates
Good for circuits where fet sizes are similar and each gate has limited fanout. Best choice for multiple input static gates and for datapaths.

Horizontal Gates
Good for circuits where long and short fets are needed or where nodes must control many fets. Often used in multiple-output complex gates (e.g., sum/carry circuits).

What about routing signals between gates? Note that both layouts block metal/poly routing inside the cell. Choices: metal2 routing over the cell or routing above/below the cell.

- avoid long (> 50 squares) poly runs
- don’t “capture” white space in a cell
- don’t obsess over the layout, instead make a second pass, optimizing where it counts

Adapted from [Terman’02]
Custom Cells: Optimizing Connections

Which does this gate do? Which is better considering node capacitances?

Adapted from [Terman’02]
Custom Cells: Optimizing Large Transistors

Which is better considering wire resistances?
Which is better considering node capacitances?

Adapted from [Terman'02]
Custom Cells: Optimizing Diffusion Sharing

Adapted from [Terman'02]
Custom Cells: Optimizing Across Cells

What does this cell do? What if we want to replicate this cell vertically to process many bits in parallel?

Adapted from [Terman'02]
Custom Cells: Optimizing Across Cells

- Place shared geometry symmetrically about shared boundary.
- Place items that aren’t to be shared 1/2 minspacing rule from shared boundary.
- Reflect cell about X axis so that Pfets are next to each other: this avoids large ndiff/pdiff spacing.
- Run shared control signals vertically -- they'll wire themselves up automatically?

Adapted from [Terman’02]
Custom Cells: Merging Simple Cells

Two latch implementations: Left implementation composes primitive gates, while right implementation uses single tightly integrated gate.

Adapted from [Weste’11]
Design Domains, Abstractions, and Principles

Custom Datapaths, Memories, Control

- Full-Custom Design

Fetch (F)  Decode (D)  Execute (X)  Memory (M)  Writeback (W)

ECE 5745  T04: Full-Custom Design Methodology
Custom Datapaths, Memories, Control

Adapted from [Weste’11]
Custom Datapaths: PC Generation Unit

Routing every path as a separate 32-bit bus would take a large area.

Adapted from [Terman'02]
Custom Datapaths: Bitslices

- Implement datapath as single bit slices, contain one bit from each functional unit
- Route each bus bit position within bitslice

Adapted from [Terman’02]
Custom Datapaths: Bit Pitch

- Height of each bit slice depends on:
  - height of tallest cell in entire bitslice
  - maximum number of buses running through any cell in bitslice

Two layers metal, buses run by side of cells

Three+ layers metal, buses run over cells

Adapted from [Terman'02]
Custom Datapaths: PC Gen Example

In 1.0\(\mu\)m, 2-metal CMOS process

Bus rip out at right angles

Buses routed by side of cells

Adapted from [Terman'02]
Custom Datapaths: Datapath Library Cells

- **Have to choose maximum datapath cell height**
  - Too high, wastes area in simple cells
  - Too small, squeezes complex cells. Grow superlinearly in length dimension, so also wastes area.

- **Compromise, around 8-12 metal tracks works OK**

  ![Datapath Diagram](example)

  - Power run vertically in M2
  - Control lines run vertically in M2, e.g., mux selects, clocks
  - Width varies according to cell type
  - 12 track pitch in M3 (buses fly over cell horizontally)

Adapted from [Terman'02]
Custom Datapaths: Optimizing Datapath Layout

Reduce congestion by rearranging datapath components to minimize required number of vertical tracks per bitslice.

Count number of wires between each component.

Datapath Input or Output

Bus Writer
Bus Reader

Bit cells must be >5 tracks high

Count number of wires crossing over each cell.

Carefully account for buses going in opposite directions.

Bit cells must be >4 tracks high.
Custom Datapaths: MIPS Datapath Track Allocation

Adapted from [Weste’11]
Custom Datapaths: MIPS Datapath Example (1)

Adapted from [Weste’11]
Custom Datapaths: MIPS Datapath Example (2)

- Full-Custom Design

MIPS Datapath Example

- 32 bits
- MIPS register file
- Load data arrives here
- Multiply/Divide Unit
- Log shifter layout
- Instruction bits arrive here (also in bypass)
- Program counter generation
- Fetch address leaves here
- Memory address leaves here
- Adder
- Bypass network for six-stage Pipe

Adapted from [Terman’02]
Custom Memories: Array Structure

Adapted from [Weste’11]
Custom Memories: Register File Circuits

Adapted from [Weste’11]
Custom Memories: Register File Circuits

Design Domains, Abstractions, and Principles

• Full-Custom Design •

Adapted from [Weste’11]
Custom Memories: SRAM Circuits

Adapted from [Weste'11]
Custom Memories: SRAM Layout

Design Domains, Abstractions, and Principles

- Full-Custom Design -

Adapted from [Weste'11]
Custom Memories: SRAM

- Regular arrays built with cells that abut in two dimensions
  - Have to pitch match in both dimensions

---

Adapted from [Terman'02]
Finite-State-Machine Control Unit

Adapted from [Weste’11]
Full Custom Control Logic with PLA

Adapted from [Weste’11]

ECE 5745 T04: Full-Custom Design Methodology 50 / 53
Top-Level Chip Floorplan

Adapted from [Weste’11]
Final Full-Custom MIPS Processor

Adapted from [Weste’11]
Acknowledgments