EECS 151/251A
Spring 2020
Digital Design and Integrated Circuits
Instructor: J. Wawrzynek
Lecture 2: Design
Outline

- Details of Design Metrics
- Digital Logic – Basic Concepts
- Design Implementation Alternatives
- Design Flows
- ASICs
Review from Lecture 1

- Moore’s law is slowing down
  - There are continued improvements in technology, but at a slower pace
- Dennard’s scaling has ended a decade ago
  - All designs are now power limited
- Multi-cores, specialization and customization provides added performance
  - Under power constraints and stagnant technology
- Design costs are high
  - Methodology and better tools to rescue!
- All design decisions involve tradeoffs between performance, cost, and power
  - Pareto optimally defines the best designs.
Digital Logic
Implementing Digital Systems

- Given a functional description and performance, cost, & power constraints, come up with an implementation using a set of primitives.
- Digital systems are implemented as a set of *combinational logic and state elements*:

![Diagram](image)

- What is the methodology we use to implement a digital system?
Design Process through layers of abstractions

- Specification (e.g. in plain text)
- Model (e.g. in C/C++)
- Logic Description (e.g. in Verilog)
- Physical design (layout; ASIC, FPGA)
- Micro-Architecture (e.g. in-order, out-of-order)

Validation:
- Validation: is model implementing the specification and meeting the performance?
- Verification: logic/physical design correct?
- Test: Does the part work?

Validation: Have we built the right thing?
Verification: Have we built the thing correctly?

The key to success is that each layer preserves the essential functionality and constraints from above, but adds more details.
Modern (Mostly) Digital System-On-A-Chip (SOC)

- Apple A12 Bionic
  - 2x Large CPUs
  - 4x Small CPUs
  - GPUs
  - Neural processing unit (NPU)
  - Lots of memory
  - DDR memory interfaces

- 7nm CMOS
- Up to 2.49GHz
Design Metrics
Basic Design Tradeoffs

- Improve on one at the expense of the others
- Tradeoffs exist at every level in the system design
- Design Specification
  - Functional Description
  - Performance, cost, power constraints
- Designer must make the tradeoffs needed to achieve the function within the constraints
Performance

• **Throughput**
  - Number of tasks performed in a unit of time (operations per second)
  - E.g. Google TPUv3 board performs 420 TFLOPS ($10^{12}$ floating-point operations per second, where a floating point operation is BFLOAT16)
  - Watch out for ‘op’ definitions – can be a 1-b ADD or a double-precision FP add (or more complex task)
  - Peak vs. average throughput

• **Latency**
  - How long does a task take from start to finish
  - E.g. facial recognition on a phone takes 10’s of ms
  - Sometime expressed in terms of clock cycles
  - Average vs. ‘tail’ latency
Energy and Power

- **Energy** (in joules (J))
  - Needed to perform a task (energy efficiency)
  - Ex: add two numbers or fetch a datum from memory
  - Battery stores certain amount of energy (in Ws = J or Wh)
  - That is what utility charges for (in kWh)

- **Power** (in watts (W))
  - Energy dissipated per unit time (W = J/s)
  - Sets cooling requirements
    - Heat spreader, size of a heat sink, forced air, liquid, …
Cost

- **Non-recurring** engineering (NRE) costs
  - Cost to develop a design (product)
    - Amortized over all units shipped
    - E.g. $20M in development adds $.20 to each of 100M units

- **Recurring** costs
  - Cost to manufacture, test and package a unit
  - Processed wafer cost is ~10k (around 16nm node) which yields:
    - 50-100 large FPGAs or GPUs
    - 200 laptop CPUs
    - >1000 cell phone SoCs

\[
\text{cost per IC} = \text{variable cost per IC} + \frac{\text{fixed cost}}{\text{volume}}
\]

\[
\text{variable cost} = \frac{\text{cost of die} + \text{cost of die test} + \text{cost of packaging}}{\text{final test yield}}
\]
Die Cost

\[
\text{cost of die} = \frac{\text{cost of wafer}}{\text{dies per wafer} \times \text{die yield}}
\]

From: http://www.amd.com
Yield

\[ Y = \frac{\text{No. of good chips per wafer}}{\text{Total number of chips per wafer}} \times 100\% \]

Die cost = \[ \frac{\text{Wafer cost}}{\text{Dies per wafer} \times \text{Die yield}} \]

Dies per wafer = \[ \frac{\pi \times (\text{wafer diameter}/2)^2}{\text{die area}} - \frac{\pi \times \text{wafer diameter}}{\sqrt{2} \times \text{die area}} \]
Defects

\[ \text{Yield} = 0.25 \]

\[ \text{Yield} = 0.76 \]

\[
\text{die yield} = \left( 1 + \frac{\text{defects per unit area} \times \text{die area}}{\alpha} \right)^{-\alpha}
\]

\[ \alpha \text{ is approximately 3} \]

\[ \text{die cost} = f(\text{die area})^4 \]
Digital Logic
Basic Concepts
Logic Gates

<table>
<thead>
<tr>
<th></th>
<th>AND</th>
<th>OR</th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>a</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- Logic gates are often the primitive elements out of which combinational logic circuits are constructed.
  - In some technologies, there is a one-to-one correspondence between logic gate representations and actual circuits (ASIC standard cells have gate implementations).
  - Other times, we use them just as another abstraction layer (FPGAs have no real logic gates).

- How about these gates with more than 2 inputs?
- Do we need all these types?
Multi-Input Gates

3-Input NAND

NAND3 Boolean equation

\[ \text{Out} = \overline{A \cdot B \cdot C} \]

And-Or-Invert

AOI21 Boolean equation

\[ \text{Out} = \overline{A \cdot B} + \overline{C} \]

• Single gate in modern CMOS usually doesn’t have more than 3-4 inputs
Logic circuits have been built out of many different technologies. If we have a basic logic gate (AND or OR) and inversion we can build a complete logic family.

**CMOS Gate**

**DTL**

**Hydraulic**

**Mechanical LEGO logic gates.** A clockwise rotation represents a binary “one” while a counter-clockwise rotation represents a binary “zero.”
A necessary property of any suitable technology for logic circuits is "Restoration" or "Regeneration"

Circuits need:
- to ignore noise and other non-idealities at their inputs, and
- generate "cleaned-up" signals at their output.

Otherwise, each stage propagates input noise to their output and eventually noise and other non-idealities would accumulate and signal content would be lost.
- Inverter acts like a “non-linear” amplifier
- The non-linearity is critical to restoration
- Other logic gates act similarly with respect to input/output relationship.
Combinational Logic Blocks

Example four-input Boolean function:

- Output a function only of the current inputs (no history).
- Truth-table representation of function. Output is explicitly specified for each input combination.
- In general, CL blocks have more than one output signal, in which case, the truth-table will have multiple output columns.

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>y</th>
<th>F(a,b,c,d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>F(0,0,0,0)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F(0,0,0,1)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>F(0,0,1,0)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>F(0,0,1,1)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>F(0,1,0,0)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F(0,1,0,1)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>F(0,1,1,0)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>F(0,1,1,1)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>F(1,1,1,1)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>F(1,0,0,0)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>F(1,0,0,1)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>F(1,0,1,0)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>F(1,0,1,1)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>F(1,1,0,0)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>F(1,1,0,1)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>F(1,1,1,0)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>F(1,1,1,1)</td>
<td></td>
</tr>
</tbody>
</table>

Truth Table
Example CL Block

- 2-bit adder. Takes two 2-bit integers and produces 3-bit result.

- Think about truth table for 32-bit adder. It’s possible to write out, but it might take a while!

<table>
<thead>
<tr>
<th>a1</th>
<th>a0</th>
<th>b1</th>
<th>b0</th>
<th>c2</th>
<th>c1</th>
<th>c0</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
<td>001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>10</td>
<td>10</td>
<td>010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>01</td>
<td>01</td>
<td>010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>00</td>
<td>10</td>
<td>010</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>01</td>
<td>01</td>
<td>10</td>
<td>011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>00</td>
<td>11</td>
<td>011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>01</td>
<td>01</td>
<td>11</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>10</td>
<td>11</td>
<td>101</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>110</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Theorem: Any combinational logic function can be implemented as a networks of logic gates.
Example Logic Circuit

How do we know that these two representations are equivalent?

Will come back to this later!
Sequential Logic Blocks

- Output is a function of both the current inputs and the state.

- “State” represents the memory.
- State is a function of previous inputs.
- In synchronous digital systems, state is updated on each clock tick.
- “F” is just a combinational logic block.

This means the way the block responds to a particular input depends on what it has seen previously.
State Elements: circuits that store info

- Examples: registers, memories
- Register: Under the control of the “load” signal, the register captures the input value and stores it indefinitely.
- The value stored by the register appears on the output (after a small delay).
- Until the next load, changes on the data input are ignored (unlike CL, where input changes change output).
- These get used for short term storage (ex: register file), and to help move coordinate data movement.

*Often replace by clock signal (clk)*
Register Transfer Level Abstraction (RTL)

Any synchronous digital circuit can be represented with:

- Combinational Logic Blocks (CL), plus
- State Elements (registers or memories)

- State elements are mixed in with CL blocks to remember and to control the flow of data.

- Sometimes used in large groups by themselves for “long-term” data storage.
Digital Logic Delay

- Changes at the inputs do not instantaneously appear at the outputs
  - There are finite conductances and capacitances in each gate…
  - Propagation through a chain of gates is roughly the sum of the delay through the individual gates
Digital Logic Timing

• The longest propagation delay through CL blocks sets the maximum clock frequency

• To increase clock rate:
  • Find the longest path
  • Make it faster
Administrivia

• Make sure to compete Lab 1 by beginning of next lab session

• Lab 2 is more involved
  • Be prepared

• Homework 1 posted this week, due next Friday
  • Start early!
Implementation
Alternatives & Design Flow
### Implementation Alternative Summary

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Full-custom:</strong></td>
<td>All circuits/transistors layouts optimized for application.</td>
</tr>
<tr>
<td><strong>Standard-cell:</strong></td>
<td>Small function blocks/“cells” (gates, FFs) automatically placed and routed.</td>
</tr>
<tr>
<td><strong>Gate-array</strong> (structured ASIC):</td>
<td>Partially prefabricated wafers with arrays of transistors customized with metal layers or vias.</td>
</tr>
<tr>
<td><strong>FPGA:</strong></td>
<td>Prefabricated chips customized with loadable latches or fuses.</td>
</tr>
<tr>
<td><strong>Microprocessor:</strong></td>
<td>Instruction set interpreter customized through software.</td>
</tr>
<tr>
<td><strong>Domain Specific Processor:</strong></td>
<td>Special instruction set interpreters (ex: DSP, NP, GPU, TPU).</td>
</tr>
</tbody>
</table>

These days, “ASIC” almost always means Standard-cell.

**What are the important metrics of comparison?**


“The Important Distinction

“Instruction” Binding Time (cost of flexibility)

- When do we decide the functions (what operation is to be performed)?

- **General Principles**

  Earlier the decision is bound, the less area, delay/energy required for the implementation.

  Later the decision is bound, the more flexible the device.

A. DeHon
**Full-Custom**

- Circuit styles and transistors are custom sized and drawn to optimize die, size, power, performance.
- High NRE (non-recurring engineering) costs
  - Time-consuming and error prone layout
- Hand-optimizing the layout can result in small die for low per unit costs, extreme-low-power, or extreme-high-performance.
- Common today for **analog design**.
- Requires full set of custom masks.
- High NRE usually restricts use to high-volume applications/markets or highly-constrained and cost insensitive markets.
**Standard-Cell**

- Based around a set of pre-designed (and verified) cells
  - Ex: NANDs, NORs, Flip-Flops, counters slices, buffers, …
- Each cell comes complete with:
  - layout (perhaps for different technology nodes and processes),
  - Simulation, delay, & power models.
- Chip layout is automatic, reducing NREs (usually no hand-layout).
- (Slightly) less optimal use of area and power, leading to higher per die costs than full-custom.
- Commonly used with other predesigned blocks (large memories, I/O blocks, etc.)
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.
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, also:

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 connected 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, Enet, PCI, SERDES, CPUs, ...
ASICs: 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 ⇒ 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.
Microprocessors / Microcontrollers

- 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 a single function.

- Furthermore, instruction set processors (microprocessors) are an ubiquitous “abstraction” level.
  - “Synthesizable” RTL model (“soft core”, available in HDL)
  - Often mixed into other digital designs

- Their implementation hosted on a variety of implementation platforms: standard-cell, gate-array, FPGA, other processors?

<table>
<thead>
<tr>
<th>§</th>
<th>Assembler</th>
</tr>
</thead>
<tbody>
<tr>
<td>5E</td>
<td>ADD{cmd}{S} Rd, Rn, &lt;Operand2&gt;</td>
</tr>
<tr>
<td>5E</td>
<td>ADC{cmd}{S} Rd, Rn, &lt;Operand2&gt;</td>
</tr>
<tr>
<td>5E</td>
<td>QADD{cmd} Rd, Rm, Rn</td>
</tr>
<tr>
<td>5E</td>
<td>QDADD{cmd} Rd, Rm, Rn</td>
</tr>
<tr>
<td>5E</td>
<td>SUB{cmd}{S} Rd, Rn, &lt;Operand2&gt;</td>
</tr>
<tr>
<td>5E</td>
<td>SBC{cmd}{S} Rd, Rn, &lt;Operand2&gt;</td>
</tr>
<tr>
<td>5E</td>
<td>RSB{cmd}{S} Rd, Rn, &lt;Operand2&gt;</td>
</tr>
<tr>
<td>5E</td>
<td>RSC{cmd}{S} Rd, Rn, &lt;Operand2&gt;</td>
</tr>
<tr>
<td>5E</td>
<td>QSUB{cmd} Rd, Rm, Rn</td>
</tr>
<tr>
<td>5E</td>
<td>QDSUB{cmd} Rd, Rm, Rn</td>
</tr>
<tr>
<td>2</td>
<td>MUL{cmd}{S} Rd, Rm, Rs</td>
</tr>
<tr>
<td>2</td>
<td>MLA{cmd}{S} Rd, Rm, Rs, Rn</td>
</tr>
<tr>
<td>M</td>
<td>UMULL{cmd}{S} RdLo, RdHi, Rm, Rs</td>
</tr>
<tr>
<td>M</td>
<td>UMLAL{cmd}{S} RdLo, RdHi, Rm, Rs</td>
</tr>
<tr>
<td>6</td>
<td>UMAAL{cmd} RdLo, RdHi, Rm, Rs</td>
</tr>
</tbody>
</table>
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, AMBA, Sonics, …
- “IP Block” business model: Hard- or soft-cores available from third party designers.
- ARM, inc. is the shining example. Hard- and “synthesizable” RISC 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.
ASICs
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

1um, 2-metal process

Modern sub-100nm process
“Transistors are free things that fit under wires”
The ASIC flow

Design Capture
- Verilog (or VHDL)
  - Logic Synthesis
  - Floorplanning
    - Placement
      - Circuit Extraction
        - Post-Layout Simulation
          - Pre-Layout Simulation
  - Tape-out

Behavioral

Structural

Physical

Design Iteration
Modern ASIC Methodology and Flow

RTL Synthesis Based

- HDL specifies design as combinational logic + state elements
- Logic Synthesis converts hardware description to gate and flip-flop implementation
- Cell instantiations needed for blocks not inferred by synthesis (typically RAM)
- Event simulation verifies RTL
- “Formal” verification compares logical structure of gate netlist to RTL
- Place & route generates layout
- Timing and power checked statically
- Layout verified with LVS and GDRC
Standard cell design

- **Layout considerations**

  Cells have standard height but vary in width.
  Designed to connect power, ground, and wells by abutment.
Standard cell characterization

Power Supply Line (V_{DD})  Delay in (ns)!!

<table>
<thead>
<tr>
<th>Path</th>
<th>1.2V - 125°C</th>
<th>1.6V - 40°C</th>
</tr>
</thead>
<tbody>
<tr>
<td>In1-t_{pLH}</td>
<td>0.073+7.98C+0.317T</td>
<td>0.020+2.73C+0.253T</td>
</tr>
<tr>
<td>In1-t_{pHiL}</td>
<td>0.069+8.43C+0.364T</td>
<td>0.018+2.14C+0.292T</td>
</tr>
<tr>
<td>In2-t_{pLH}</td>
<td>0.101+7.97C+0.318T</td>
<td>0.026+2.38C+0.255T</td>
</tr>
<tr>
<td>In2-t_{pHiL}</td>
<td>0.097+8.42C+0.325T</td>
<td>0.023+2.14C+0.269T</td>
</tr>
<tr>
<td>In3-t_{pLH}</td>
<td>0.120+8.00C+0.318T</td>
<td>0.031+2.37C+0.258T</td>
</tr>
<tr>
<td>In3-t_{pHiL}</td>
<td>0.110+8.41C+0.280T</td>
<td>0.027+2.15C+0.223T</td>
</tr>
</tbody>
</table>

3-input NAND cell (from ST Microelectronics):
C = Load capacitance
T = input rise/fall time

Ground Supply Line (GND)

- Each library cell (FF, NAND, NOR, INV, etc.) and the variations on size (strength of the gate) is fully characterized across temperature, loading, etc.
Macro modules

256×32 (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
The “timing closure” problem

- Biggest problem are wires (signals and clock)

Iterative Removal of Timing Violations (white lines)
End of Lecture 2