CS250
VLSI Systems Design

Fall 2020

John Wawrzynek

with

Arya Reais-Parsi
Digital Design

Given a functional description and performance, cost, & power constraints, come up with an appropriate implementation using a set of primitives.

- How do we learn how to do this?
  1. Learn micro-architectural templates
  2. Learn about primitives
  3. Learn about design representations and tools
     Map translate between representations
     Transform for optimization (move towards or along the Pareto Frontier)
  4. Use trial and error - CAD tools and prototyping. Practice!

- Digital design is in some ways more an art than a science. The creative spirit is critical in combining primitive elements & other components in new ways to achieve a desired function.

- However, unlike art, we have objective measures of a design:
  
  \textbf{Performance} \hspace{1cm} \textbf{Cost} \hspace{1cm} \textbf{Power}
Limitation of Full-custom Design

Labor intensive.

Key metric is the number of leaf cells required to efficiently use a given area of silicon.

Memory arrays and FPGAs are a good fit.
Standard cell design

Layout considerations

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

Clock Rail
(not typical)

Power Rails in M1

GND Rail

NAND2

Flip-flop

Well Contact under Power Rail

Cell I/O on M2
**Standard Cell Design**

- **Gate Library.** Fixed-height, to be placed in rows. Vdd and Gnd rails connect by abutment.

- **I/O Ports.** Auto-router places wires over cell to connect them.

**Logic schematics using library gates.**

**Process Design Kit: Design Rules and Device Models**
Place & Route

Software places cells into rows, to “optimize” area, performance, and power constraints.

Router connects gate ports to match schematic.

Router “optimizes” relative lengths of wire to meet constraints.

We put “optimize” in quotes to reflect the NP-hard nature of the algorithms behind place & route.
Standard cell layout

- 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”
12 x 16 Multiplier in Structured Custom and Standard Cell

Benchmark by a custom design house (Obsidian).

In general, they claim: “30% of the power, twice the speed, and 4 times the density of standard cells”.

Custom layout (left) is a factor of 2.2 smaller than standard cell layout (right).
Logic Synthesis - The Automation of Logic Design

At the start of the 1980s, the standard-cell flow was driven by hand-drawn schematics.

By the early 1990s, schematics were replaced with Verilog/VHDL, to drive logic synthesis, whose output was integrated into standard cell back ends.
Logic design
... as I learned it in 1981 ...

\[ f = m_1 + m_2 + m_5 + m_6 + m_7 \]
\[ = y'y + x'y'z + xyz + xy'z + xyz \]
\[ = \frac{x}{y} (y'z + y'z) + \frac{x}{y} (y'z + y'z) + xy \]
\[ = \frac{1}{y} (y'z + y'z) + \frac{5}{y} (y'z + y'z) + xy \]
\[ = y'z + y'z + xyz \]
In the early 1980s, progress in academia and industrial labs made the problem domain tractable ...

**Given:** Finite-State Machine $F(X,Y,Z, \lambda, \delta)$ where:

- $X$: Input alphabet
- $Y$: Output alphabet
- $Z$: Set of internal states
  - $\delta: X \times Z \rightarrow Z$ (next state function)
  - $\lambda: X \times Z \rightarrow Y$ (output function)

**Target:** Circuit $C(G, W)$ where:

- $G$: set of circuit components $g \in \{\text{Boolean gates, flip-flops, etc}\}$
- $W$: set of wires connecting $G$
In the second half of the 1980s, the startup that became Synopsys developed Design Compiler (dc) ...

Eventually, Verilog/VHDL.

1986 “Pitch Slide” for EDA startup Optimal Solutions

Logic Synthesis

Productivity the key “value-add”

Technology Mapping
Typical ASIC Implementation Path

- Behavioral Model (C++, python, ...)
- HDL represents RTL
  - Behavioral specification
- Structural
- Instantiation of special blocks - “black boxes” (ex: RAM, ROM)
- Library cells from the PDK (standard cells)
- GDSII is the modern standard file format for representing layout information (layers and geometry)
HDL 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
```
Macro (Black Box) 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
Typical ASIC Implementation Path

- Verification
  - Discrete Event Simulator does most the work (Modelsim, VCS, verilator)
- Input test Vectors and expected output vectors developed with behavioral model
- Pre-synthesis, function only
- Post-synthesis, approximate timing
- Post-layout, accurate timing
  - “back annotated” with layout wiring parasitics
Standard cell characterization

<table>
<thead>
<tr>
<th>Path</th>
<th>1.2V - 125°C</th>
<th>1.6V - 40°C</th>
</tr>
</thead>
<tbody>
<tr>
<td>(I_{n1}-t_{pLH})</td>
<td>0.073+7.98C+0.317T</td>
<td>0.020+2.73C+0.253T</td>
</tr>
<tr>
<td>(I_{n1}-t_{pHL})</td>
<td>0.069+8.43C+0.364T</td>
<td>0.018+2.14C+0.292T</td>
</tr>
<tr>
<td>(I_{n2}-t_{pLH})</td>
<td>0.101+7.97C+0.318T</td>
<td>0.026+2.38C+0.255T</td>
</tr>
<tr>
<td>(I_{n2}-t_{pHL})</td>
<td>0.097+8.42C+0.325T</td>
<td>0.023+2.14C+0.269T</td>
</tr>
<tr>
<td>(I_{n3}-t_{pLH})</td>
<td>0.120+8.00C+0.318T</td>
<td>0.031+2.37C+0.258T</td>
</tr>
<tr>
<td>(I_{n3}-t_{pHL})</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.

Similarly for power models
Typical ASIC Implementation Path

- Verifying the Synthesis Netlists for SoC is a big job
- FPGA based emulators
- Runs 100’s times faster than software simulator
- Boot OS on processor

**Synthesis netlist**

- **Library Cell models**
- **FPGA Logic Synthesis/Mapping**
- **Configuration Bit Stream**
- **Black Box models**
- **Test Vectors**

- **FPGA-based Emulator**
- **Output Vectors**

Nvidia’s Emulation Lab
The “timing closure” problem

- Biggest problem are wires (signals and clock)

*Iterative Removal of Timing Violations (white lines)*
Closing Timing

- Netlist Simulation post layout (includes wire delay)
- Alternative is **static timing analysis tools**
  - Traces through all circuit paths without input patterns
  - A version of this in logic syntheser to help optimize, based on approximate layout information
  - Similar analysis for power
  - Potentially more comprehensive than simulation
  - Finds all paths, but some may be “false”. Never used when real input is applied
GDSII Checks

- Layout Versus Schematic
  - extracts circuit topology
  - compares to pre-layout netlist
- Geometric Design Checker verifies conformance to foundary rules
“Automated Flow”. Easy?

- For good results:
  - Verification at every level (exercise in paranoia)
  - Layout, floorplanning major blocks
  - Vdd/GND: distribution strategy, layers, topology, decoupling
  - clock: PLL/DLL, buffering, skew/jitter spec
  - I/O: pad drive strengths, ESD protection
- Why can't this all be automated?
  - Functional Specification -> Optimized Correct Layout
Today’s chips are mosaics. The chip design process often consists of licensing “intellectual property (IP)” from other companies (large like CPUs and GPUs, small like DRAM controllers & analog blocks).

On chip buses, IP blocks are often designed to hook up to standardized on-chip buses, defined by CPU IP vendors like ARM.
Apple TV SoC

Chip designed by Apple, but many blocks are licensed from third parties. Some are standard cells, others full custom.
Circuits Topics: Basic (review?)

What you need to know as a VLSI Systems designer.

- Processing/devices: planar, finfets, GDR
- Device models: switch, RC, Vth
- Logic circuits: gates, muxes, transmission gates, FFs
- Circuit Delay: gate delay, wire delay, FET sizing
- Circuit Power: formulation/factors
- System Delay: factors, optimization
- System Power: factors, optimization
IC Processing

- Chip physical features
- Planar process steps
- Geometric Design Rules
- Masks / lithography
Fabrication
Silicon “ingots” are grown from a “perfect” crystal seed in a melt, and then purified to “nine nines”.

![Silicon ingots growth diagram](image)
Ingots sliced into 450 μm thick wafers, using a diamond saw.
An n-channel MOS transistor

Polysilicon gate, dielectric, and substrate form a capacitor.

nFet is off (I is “leakage”)

Vg = 1V, small region near the surface turns from p-type to n-type.

nFet is on.
Mask set for an n-Fet (circa 1986)

\[ V_d = 1V \]
\[ V_g = 0V \]
\[ V_s = 0V \]

- \( I \approx nA \)
- \( n^+ \) diffusion
- poly (gate)
- diff contact
- metal

Top-down view:

Layers to do p-Fet not shown. Modern processes have 6 to 10 metal layers (or more) (in 1986: 2).
“Design rules” for masks, 1986 ...

Poly overhang.
So that if masks are misaligned, we still get channel.

Minimum gate length.
So that the source and drain depletion regions do not meet!

#1: n+ diffusion
#2: poly (gate)

Metal rules:
Contact separation from channel, one fixed contact size, overlap rules with metal, etc ...

#3: diff contact
#4: metal
How a fab uses a mask set to make an IC

Vd = 1V
Vg = 1V
Vs = 0V

Id = \mu A

Top-down view:

#1: n+ diffusion
#2: poly (gate)
#3: diff contact
#4: metal
Start with an un-doped wafer ...

UV hardens exposed resist. A wafer wash leaves only hard resist.

Steps
#1: dope wafer p-
#2: grow gate oxide
#3: deposit polysilicon
#4: spin on photoresist
#5: place positive poly mask and expose with UV.
Wet etch to remove unmasked ...

HF acid etches through poly and oxide, but not hardened resist.

After etch and resist removal
Use diffusion mask to implant n-type

accelerated donor atoms

Notice how donor atoms are blocked by gate and do not enter channel.

Thus, the channel is “self-aligned”, precise mask alignment is not needed!
Metallization completes device

Grow a thick oxide on top of the wafer.

Mask and etch to make contact holes.

Put a layer of metal on chip. Be sure to fill in the holes!
Final product ...

Top-down view:

“The planar process”
Jean Hoerni, Fairchild Semiconductor 1958
MOSFET Devices

- Conductance Curves
- FinFET / wraparound-gates
- Models
  - switch
  - Resistor
MOSFET Threshold Voltage

Transistor “turns on” when $V_{gs}$ is $> V_t$. 

$I_{ds}$

$0.25 = V_t$

$I_{off} = 0$ ???

$0.7 = V_{dd}$

$1.2 \text{ mA} = I_{on}$
Transistor “resistance”

- Actually, nonlinear I/V characteristic:
Plot on a “Log” Scale to See “Off” Current

Process engineers can:

- increase $I_{on}$ by lowering $V_t$ - but that raises $I_{off}$
- decrease $I_{off}$ by raising $V_t$ - but that lowers $I_{on}$.

$I_{off} = 10 \text{ nA}$

$1.2 \text{ mA} = I_{on}$

$0.25 = V_t$

$0.7 = V_{dd}$
Switch Model of MOS Transistor

Useful for reasoning about “function” of circuits.
Resistor Model of MOS Transistor

Useful for approximating timing of circuits.
Latest Modern Process

Transistor channel is a raised fin.
Gate controls channel from sides and top.
**Recent Process Evolution**

As of September 2018, mass production of 7 nm devices has begun. The first mainstream 7 nm mobile processor intended for mass market use, the **Apple A12 Bionic**, was released at their September 2018 event. Although **Huawei** announced its own 7 nm processor before the Apple A12 Bionic, the **Kirin 980** on August 31, 2018, the **Apple A12 Bionic** was released for public, mass market use to consumers before the Kirin 980. Both chips are manufactured by **TSMC**. **AMD** is currently working on their "Rome" workstation processors, which are based on the 7 nanometer node and feature up to 64 cores.

The 5 nm node was once assumed by some experts to be the end of **Moore's law**. Transistors smaller than 7 nm will experience quantum tunnelling through the gate oxide layer. Due to the costs involved in development, 5 nm is predicted to take longer to reach market than the two years estimated by Moore's law. Beyond 7 nm, it was initially claimed that major technological advances would have to be made to produce chips at this small scale. In particular, it is believed that 5 nm may usher in the successor to the **FinFET**, such as a **gate-all-around** architecture.

Although **Intel** has not yet revealed any specific plans to manufacturers or retailers, their 2009 roadmap projected an end-user release by approximately 2020. In early 2017, **Samsung** announced production of a 4 nm node by 2020 as part of its revised roadmap. On January 26th 2018, **TSMC** announced production of a 5 nm node by 2020 on its new fab 18. In October 2018, TSMC disclosed plans to start risk production of 5 nm devices in April 2019.

3.5 nm is a name for the first node beyond 5 nm. In 2018, **IMEC** and **Cadence** had taped out 3 nm test chips. Also, **Samsung** announced that they plan to use Gate-All-Around technology to produce 3 nm FETs in 2021.
Logic Circuit

- Logic gates in transistors
- Transmission Gates
- Tri-state Buffers
- Multiplexor Circuits
- Latch / Flip-flop circuits
- SRAM circuits
Inverter (NOT gate):

NAND gate:

Note:
- out = 0 iff a AND b = 1 therefore out = (ab)'
- pFET network and nFET networks are duals of one another.

How about AND gate?
Transistor-level Logic Circuits

Simple rule for wiring up MOSFETs:

- nFET is used only to pass logic zero.
- pFET is used only to pass logic one.

For example, consider the NAND gate:

Note: This rule is sometimes violated by expert designers under special conditions.
CMOS Logic Gates in General

Pull-up network conducts under conditions to generate a logic 1 output.

Pull-down network conducts for logic 0 output.

Conductance must be mutually exclusive - else, short circuit!

Pull-up and pull-down networks are “topological duals”
Transmission Gate

- Transmission gates are the way to build “switches” in CMOS.
- In general, both transistor types are needed:
  - nFET to pass zeros.
  - pFET to pass ones.
- The transmission gate is bi-directional (unlike logic gates).
Transmission-gate Multiplexor

2-to-multiplexor: 
\[ C = s a + s'b \]

Switches simplify the implementation:

Compare the cost to logic gate implementation.
Tri-state Buffers

- Transistor circuit for inverting tri-state buffer:

Tri-state Buffer:

```
<table>
<thead>
<tr>
<th>OE</th>
<th>IN</th>
<th>OUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Z</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

“high impedance” (output disconnected)

Variations:

- Inverting buffer

- Inverted enable

Transmission gate useful in implementation
Tri-state Buffers

Tri-state buffers enable “bidirectional” connections. Tri-state buffers are used when multiple circuits all connect to a common wire. Only one circuit at a time is allowed to drive the bus. All others “disconnect” their outputs, but can “listen”.

Tri-state Based Multiplexor

Multiplexor

If \( s=1 \) then \( c=a \) else \( c=b \)

Transistor Circuit for inverting multiplexor:
Latches and Flip-flops

Positive Level-sensitive **latch**:

Positive Edge-triggered **flip-flop** built from two level-sensitive latches:

**Latch Implementation:**
SRAM Cell Array Details

Most common is 6-transistor (6T) cell array.

Word selects this cell, and all others in a row.

For write operation, column bit lines are driven differentially (0 on one, 1 on the other).
Values overwrites cell state.

For read operation, column bit lines are equalized (set to same voltage), then released. Cell pulls down one bit line or the other.
Generic Memory Block

- **Word lines** used to select a row for reading or writing
- **Bit lines** carry data to/from periphery
- **Core aspect ratio** keep close to 1 to help balance delay on word line versus bit line
- **Address bits** are divided between the two decoders
- **Row decoder** used to select word line
- **Column decoder** used to select one or more columns for input/output of data

*Storage cell could be either static or dynamic*
End of Lecture 8