Introduction

- A digital circuit design is just an idea, perhaps drawn on paper
- We eventually need to implement the circuit on a physical device
  - How do we get from (a) to (b)?

Manufactured IC Technologies

- We can manufacture our own IC
  - Months of time and millions of dollars
  - (1) Full-custom or (2) semicustom

  (1) Full-custom IC
  - We make a full custom layout
    - Using CAD tools
    - Layout describes the location and size of every transistor and wire
  - A fab (fabrication plant) builds IC for layout
    - Hard!
      - Fab setup costs ("non-recurring engineering", or NRE, costs) high
      - Error prone (several "respins"
  - Fairly uncommon
    - Reserved for special ICs that demand the very best performance or the very smallest size/power

- (2) Semicustom IC
  - "Application-specific IC" (ASIC)
  - (a) Gate array or (b) standard cell

  (2a) Gate array
  - Series of gates already layed out on chip
  - We just wire them together
    - Using CAD tools
  - Vs. full-custom
    - Cheaper and quicker to design
    - But worse performance, size, power
  - Very popular

Gate Array equations:
- \( s = a'b + ab' \)
- \( c = ab \)
Manufactured IC Technologies – Standard Cell ASIC

- (2) Semicustom IC
  - “Application-specific IC” (ASIC)
  - (a) Gate array or (b) standard cell
- (2b) Standard cell
  - Pre-laid-out “cells” exist in library, not on chip
  - Designer instantiates cells into pre-defined rows, and connects
  - Vs. gate array
    - Better performance/power/size
    - A bit harder to design
  - Vs. full custom
    - Not as good of circuit, but still far easier to design

Implementing Circuits Using NAND Gates Only

- Gate array may have NAND gates only
  - NAND is universal gate
  - Any circuit can be mapped to NAND only
- Convert AND/OR/NOT circuit to NAND-only circuit using mapping rules
  - After converting, remove double inversions
- Shortcut when converting by hand
  - Use inversion bubbles rather than drawing inverters as 2-input NAND
  - Then remove double inversions as before

Implementing Circuits Using NOR Gates Only

- NOR gate is also universal
- Converting AND/OR/NOT to NOR is done using similar rules
Implementing Circuits Using NOR Gates Only

- Example: Half adder

Programmable IC Technology – FPGA

- Manufactured IC technologies require weeks to months to fabricate
  - And have large (hundred thousand to million dollar) initial costs
- Programmable ICs are pre-manufactured
  - Can implement circuit today
  - Just download bits into device
  - Slower/faster/more-power than manufactured ICs
    - But get it today, and no fabrication costs
- Popular programmable IC – FPGA
  - "Field-programmable gate array”
    - Developed late 1980s
    - Though no “gate array” inside
    - Name was gate arrays were very popular in the 1980s
    - Programmable in seconds

FPGA Internals: Lookup Tables (LUTs)

- Example: Seat belt warning light on a NOR-based gate array
  - Note: if using 2-input NOR gates, first convert AND/OR gates to 2-inputs

- Basic idea: Memory can implement combinational logic
  - e.g., 2-address memory can implement 2-input logic
  - 1-bit wide memory – 1 function, 2-bits wide – 2 functions
- Such memory in FPGA known as Lookup Table (LUT)
  - Example: 2-address memory can implement 2-input logic
  - But get it today, and no fabrication costs

- Lookup tables become inefficient for more inputs
  - 3 inputs → only 8 words
  - 8 inputs → 256 words
  - 16 inputs → 65,536 words
- FPGAs thus have numerous small (3, 4, 5, or even 6-input) LUTs
  - If circuit has more inputs, must partition circuit among LUTs
  - Example: Extended seat belt warning light system.
**FPGA Internals: Lookup Tables (LUTs)**

- Partitioning among smaller LUTs is more size efficient
  - Example: 9-input circuit

  ![Original 9-input circuit](image1)

  Requires only 4 3-input LUTs (8x1 memories) — much smaller than a 9-input LUT (512x1 memory)

- Example: 9-input circuit

  ![Example: Partitioning a circuit among 3-input LUTs](image2)

- Mapping a 2x4 decoder to 3-input 2-output LUTs

  ![Mapping a 2x4 decoder to 3-input 2-output LUTs](image3)

- LUT typically has 2 (or more) outputs, not just one
  - Example: Partitioning a circuit among 3-input 2-output lookup tables

  ![Example: Mapping a 2x4 decoder onto an FPGA with a switch matrix](image4)

**FPGA Internals: Switch Matrices**

- Previous slides had hardwired connections between LUTs
  - Instead, want to program the connections too
- Use switch matrices (also known as programmable interconnect)
  - Simple mux-based version — each output can be set to any of the four inputs just by programming its 2-bit configuration memory

  ![FPGA Internals: Switch Matrices](image5)

- Mapping the extended seatbelt warning light onto an FPGA with a switch matrix
  - Recall earlier example (let's ignore d input for simplicity)

  ![FPGA Internals: Switch Matrices](image6)
FPGA Internals: Configurable Logic Blocks (CLBs)

- LUTs can only implement combinational logic
- Need flip-flops to implement sequential logic
- Add flip-flop to each LUT output
  - Configurable Logic Block (CLB)
  - LUT = flip-flops
  - Can program CLB outputs to come from flip-flops or from LUTs directly

FPGA Internals: Sequential Circuit Example using CLBs

- Connections can be "programmed" to create custom circuit
- All configuration memory bits are connected as one big shift register
- Known as scan chain
- Shift in "bit file" of desired circuit

FPGA Internals: Overall Architecture

- Consists of hundreds or thousands of CLBs and switch matrices
  (SMs) arranged in regular pattern on a chip

FPGA Internals: Programming an FPGA

- All configuration memory bits are connected as one big shift register
- Known as scan chain
- Shift in "bit file" of desired circuit

Other Technologies

- Simple Programmable Logic Devices (SPLDs)
  - Developed 1970s (thus, pre-dates FPGAs)
  - Prefabricated IC with large AND-OR structure
  - Connections can be "programmed" to create custom circuit
    - Circuit shown can implement any 3-input function of up to 3 terms
      - e.g., \( F = abc + a'c' \)

Programmable Nodes in an SPLD

- Fuse based – "blown" fuse removes connection
- Memory based – 1 creates connection

Conceptual view of configuration bit scan chain
PLD Drawings and PLD Implementation Example

- Common way of drawing PLD connections:
  - Uses one wire to represent all inputs of an AND
  - Uses “x” to represent connection
  - Crossing wires are not connected unless “x” is present

- Example: Seat belt warning light using SPLD

![Two ways to generate a 0 term](image)

PLD Extensions

Two-output PLD

PLD with programmable registered outputs

More on PLDs

- Originally (1970s) known as Programmable Logic Array – PLA
  - Had programmable AND and OR arrays
- AMD created “Programmable Array Logic” – “PAL” (trademark)
  - Only AND array was programmable (fuse based)
- Lattice Semiconductor Corp. created “Generic Array Logic – GAL” (trademark)
  - Memory based
- As IC capacities increased, companies put multiple PLD structures on one chip, interconnecting them
  - Become known as Complex PLDs (CPLDs), and older PLDs became known as Simple PLDs (SPLDs)
- GENERALLY SPEAKING, difference of SPLDs vs. CPLDs vs. FPGAs:
  - SPLD: tens to hundreds of gates, and usually non-volatile (saves bits without power)
  - CPLD: thousands of gates, and usually non-volatile
  - FPGA: tens of thousands of gates and more, and usually volatile (but no reason why couldn’t be non-volatile)

Technology Comparisons

<table>
<thead>
<tr>
<th>Technology</th>
<th>Full-custom</th>
<th>Standard cell (semicustom)</th>
<th>Gate array (semicustom)</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easier design</td>
<td>(1)</td>
<td>(2)</td>
<td>(3)</td>
<td>(4)</td>
</tr>
<tr>
<td>More optimized</td>
<td>(2)</td>
<td>(3)</td>
<td>(4)</td>
<td>(4)</td>
</tr>
<tr>
<td>Custom processor</td>
<td>(1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programmable</td>
<td>(2)</td>
<td>(3)</td>
<td>(4)</td>
<td></td>
</tr>
</tbody>
</table>

Key Trend in Implementation Technologies

- Transistors per IC doubling every 18 months for past three decades
  - Known as "Moore’s Law"
  - Tremendous implications – applications infeasible at one time due to outrageous processing requirements become feasible a few years later
  - Can Moore's Law continue?