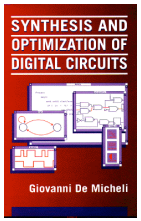
CSE372
Digital Systems Organization and Design Lab

Prof. Milo Martin

Unit 5: Hardware Synthesis

Overview

- **CAD (Computer Aided Design)**
  - Use computers to design computers
  - Virtuous cycle
  - Architectural-level, logical-level, physical-level
- **Goal: simplify the design of efficient hardware**
  - Much like high-level languages
- Source: "Synthesis and Optimization of Digital Circuits"
  - Giovanni De Micheli, 1994
  - Somewhat dated, but an important guide
  - Some pictures from his slides
- Active area of research



Hardware CAD

- Our focus
  - Digital, synchronous systems
- Synthesis
  - Convert "code" into "logic blocks"
  - Logic blocks is implementation technology specific
  - Reduce logic delays and area
- Place & Route
  - Physical-level layout of circuits on chip
  - Reduce wire delays and area

Two Types of Chips

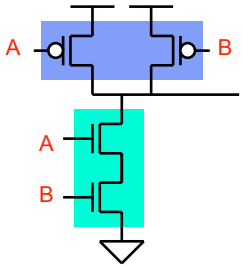
- **General-purpose processors**
  - Design for specific domains, but still "general-purpose"
  - Program using a traditional programming language
- **Special-purpose chips**
  - ASICs (application-specific integrated circuits)
  - Hard-code logic
  - Example domains: media encoding, decoding, signal processing
  - General purpose chips slow take over ASICs
    - ASICs find new niche (bleeding edge applications)
- Aside: hardware/software co-design (or co-synthesis)
  - Given a software algorithm, make a processor for running it
  - Advanced hardware synthesis and compiler optimizations

Design Implementation Approaches

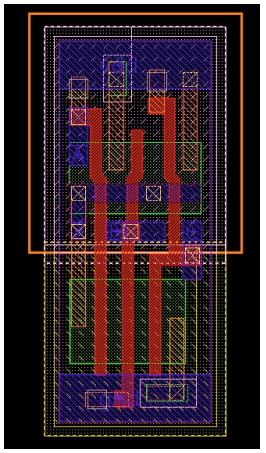
- Full-custom
  - Designs specify every transistor
  - Used for performance-critical parts of high-volume designs
- Semi-custom
  - Mixture of full-custom and synthesis design flows
  - Use of macro-cells: special highly-optimized components
    - Examples: memory arrays, adders, multipliers, etc.
- Standard cells (all synthesis)
  - Synthesis and place & route does everything
  - Much like our FPGA design flow
  - Our project Verilog + $$$$ = actual chip

Standard Cell Example

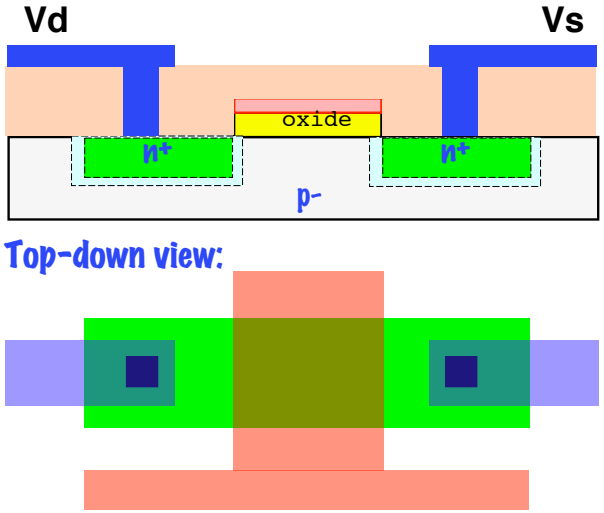
- Standard cell libraries
  - Blocks of logic primitives
  - "AOI" gates
    - Any and/or/invert operations
  - Might include all three-input gates
  - Also includes flip-flops, etc.
- Much like a FPGA design flow
  - Faster
  - Not programmable



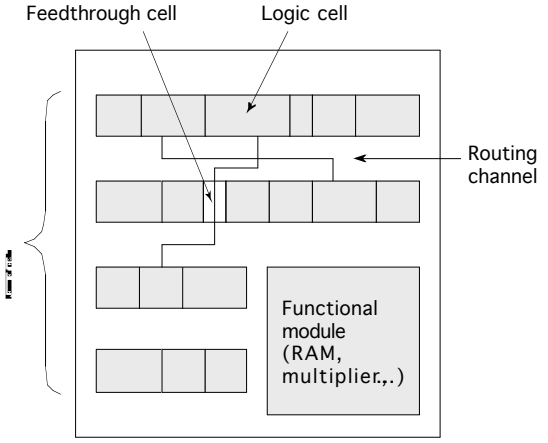
3-input NAND cell
(from ST Microelectronics):



Aside: Transistor Physical Layout

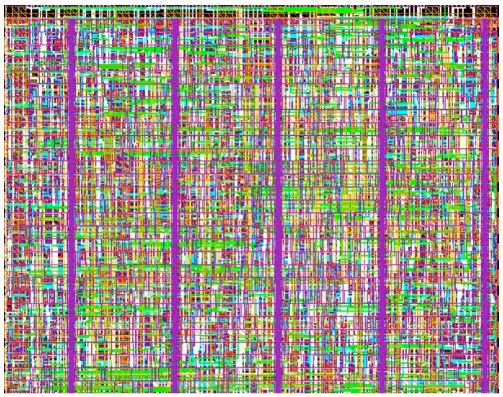


Cell-based Design (or standard cells)



Routing channel requirements are reduced by presence of more interconnect layers

Cell-based Layout

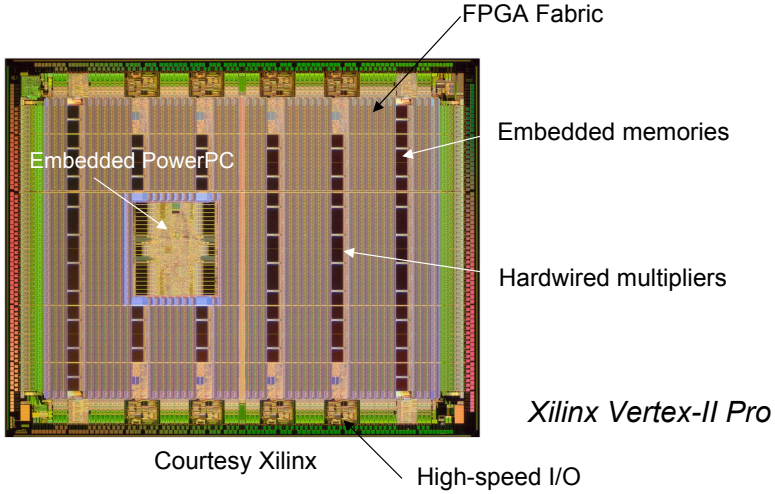


Cell-structure hidden under interconnect layers

Combinational Logic Optimizations

- Constant propagation
  - Any hard-coded value pre-computed
- Common sub-expression elimination
  - Avoid redundant re-computation of signals
- Strength reduction
  - Transform harder operations into simpler operations
  - Example: "x * 2" into "x << 1"
- Resource sharing
  - Reduce logic by reuse
  - Mux and adder example

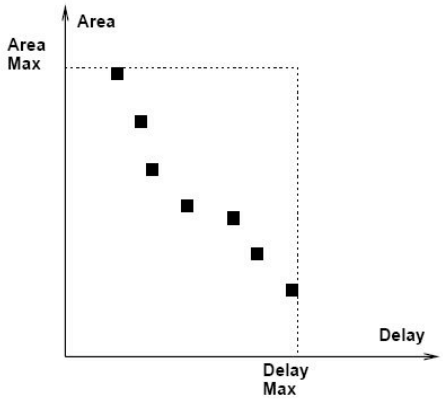
Heterogeneous Programmable Platforms



Xilinx Vertex-II Pro

CAD Tool Goals

- Maximize performance
- Minimize area
- Minimize power
- Always a trade off



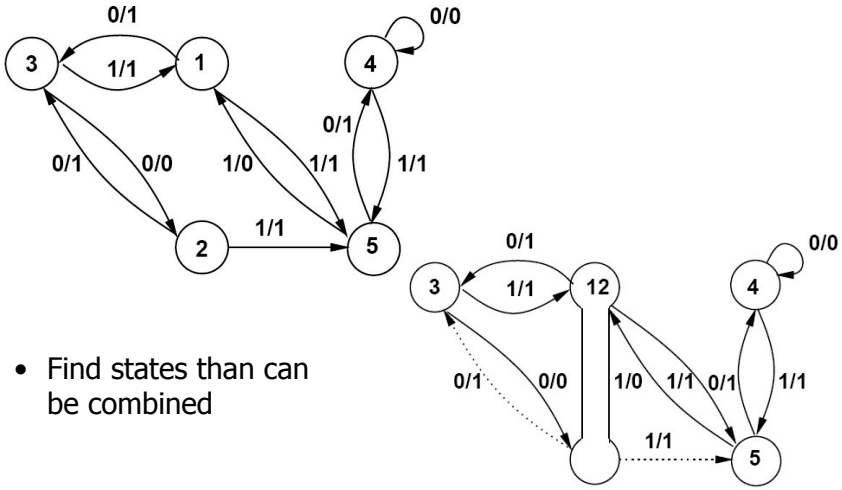
Combinational Logic Minimization

- Given a truth table…
  - Create fast and small logic to compute it
  - Example: may be faster to calculate inverse of function, then just invert the output
- Two types
  - Two-level
    - Calculate truth table from HDL code
    - PLA-like, simple
    - Not practical for large number of inputs
  - Multi-level
    - Keep structure from code
    - Transform it to improve it

State Machine Optimization

- State minimization
  - Reduce number of states by coalescing identical states
- State encoding
  - Encoding of state affects combinational logic
    - Next-state logic and output logic
  - Assign binary encoding to each unique state
  - Common approach: one-hot encoding
    - Pro: simple logic
    - Con: One state bit per state
- Combinational optimization
  - As before

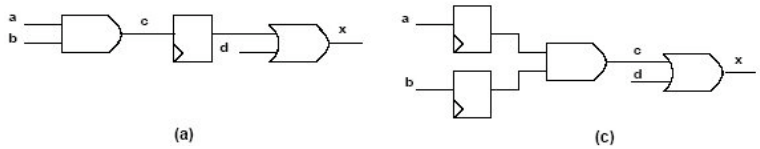
State Minimization Example



- Find states than can be combined

Sequential Logic Optimizations

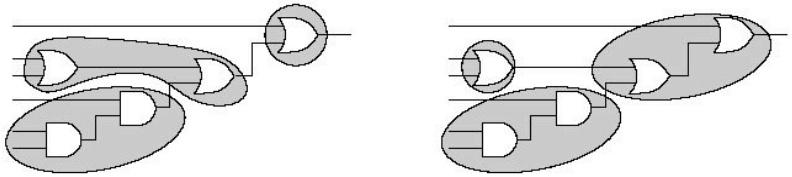
- Retiming
  - Moving logic and/or registers
  - Balance stages (move logic from one stage to another)
  - Which is better? (a) or (b)?



- Register insertion
  - Give tool combinational logic, ask for n-pipeline stages
  - Tool breaks the logic into stages
  - Xilinx will do this, actually
    - Example: memory arrays

Technology Mapping

- Map abstract logic to actual hardware primitives
  - Standard cells or FPGA lookup tables
  - Which is better?



  - Another opportunity: late arriving signals
- FPGA mapping
  - LUTs of any four-inputs
  - Many possible coverings
  - Note: 4-input P37X opcode, single LUT per output control signal

Place and Route

- Placing the primitives on the chip (or FPGA)
- Possible algorithm:
  - Start with guess of reasonable placement
  - Pick two elements, evaluate swapping them
    - If "better", perform the swap
    - If "worse", don't swap
- Evaluation metrics:
  - Average wire length (cheap to recalculate)
  - Re-evaluate circuit critical path (expensive to recalculate)
- Routing
  - Connecting the parts with wires
  - The fuller the FPGA, the harder the problem

Synthesis Under Constraints

- "Fast enough"
  - Given a delay goal, make it small (or low power)
- "Small enough"
  - Given an area (or power) budget, make it fast
- Why? No point in over-optimizing parts not on critical path
  - Example: pipelined processor
    - Once you know your slowest stage's delay
    - Re-optimize the other stages to save area
  - Example: top-down floor planning
    - Your design group has an area budget, can't exceed it

High-Level Synthesis

- Give the tool an algorithm
  - Tool creates logic to implement algorithm
    - Including cyclic computations
  - Examples: apply a low-pass filter to a stream of values
- More abstract than HDL
  - Not gate-level or cycle-level design
  - Sometimes Matlab's input language is used
    - Xilinx has a Matlab DSP (digital signal processing) design flow
- SystemC
  - A way to specify hardware using C++ library
  - Works like an HDL for hardware/software designs
- Research: C-code in, custom chip out

Miscellaneous Issues

- All synthesis and place & route phases interact
  - A decision later could make an earlier decision less good
  - Fundamentally hard problem
    - Iterative approaches can help
  - Sometimes manual intervention is needed
- Non-repeatable in some cases
  - Small design change, larger synthesis output change
- CAD tools are buggy
  - Lots of effort on synthesis verification
  - Take the output, verify it matches the input
    - Random or exhaustive test cases
    - Symbolic equivalence testing