Digital System Design

by

Dr. Lesley Shannon

Email: lshannon@ensc.sfu.ca

Course Website: http://www.ensc.sfu.ca/~lshannon/courses/ensc350

Simon Fraser University

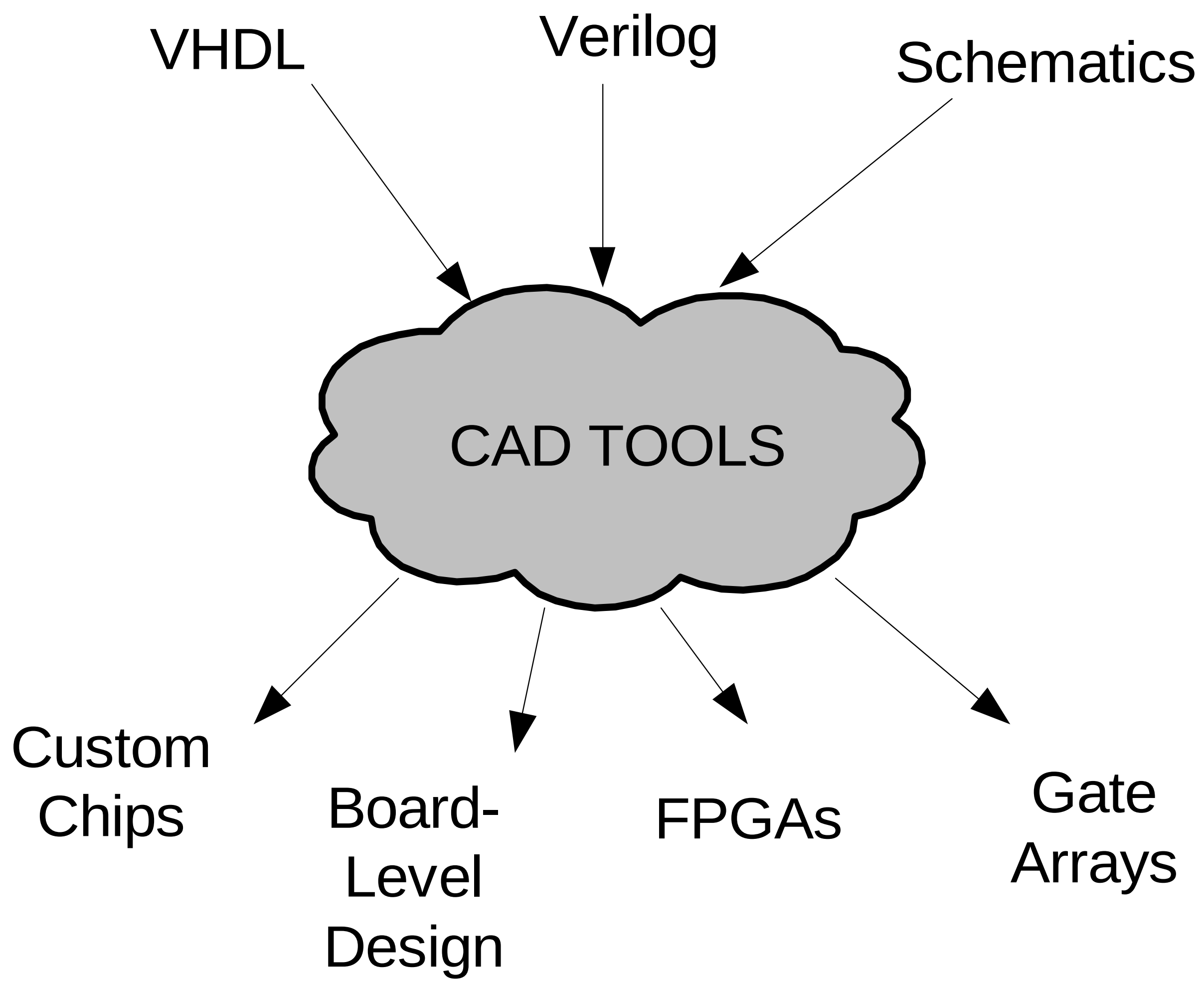
Slide Set: 6

Date: February 2, 2009

Slide Set Overview

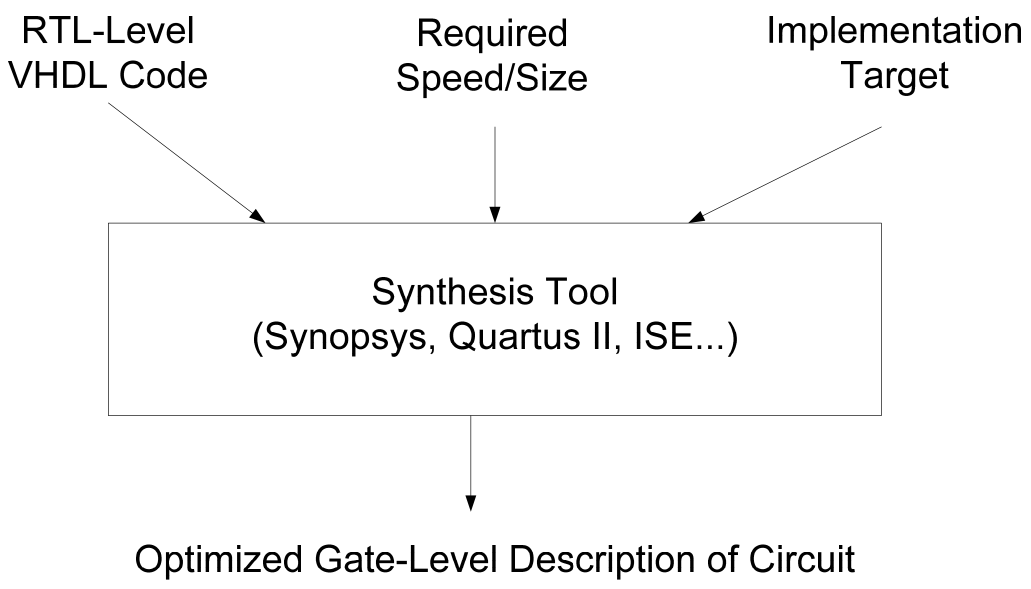
- How do we configure an FPGA
- How do CAD tools transform HDL into configurations for FPGAs?

Recall: the relationship between HDLs and Chips



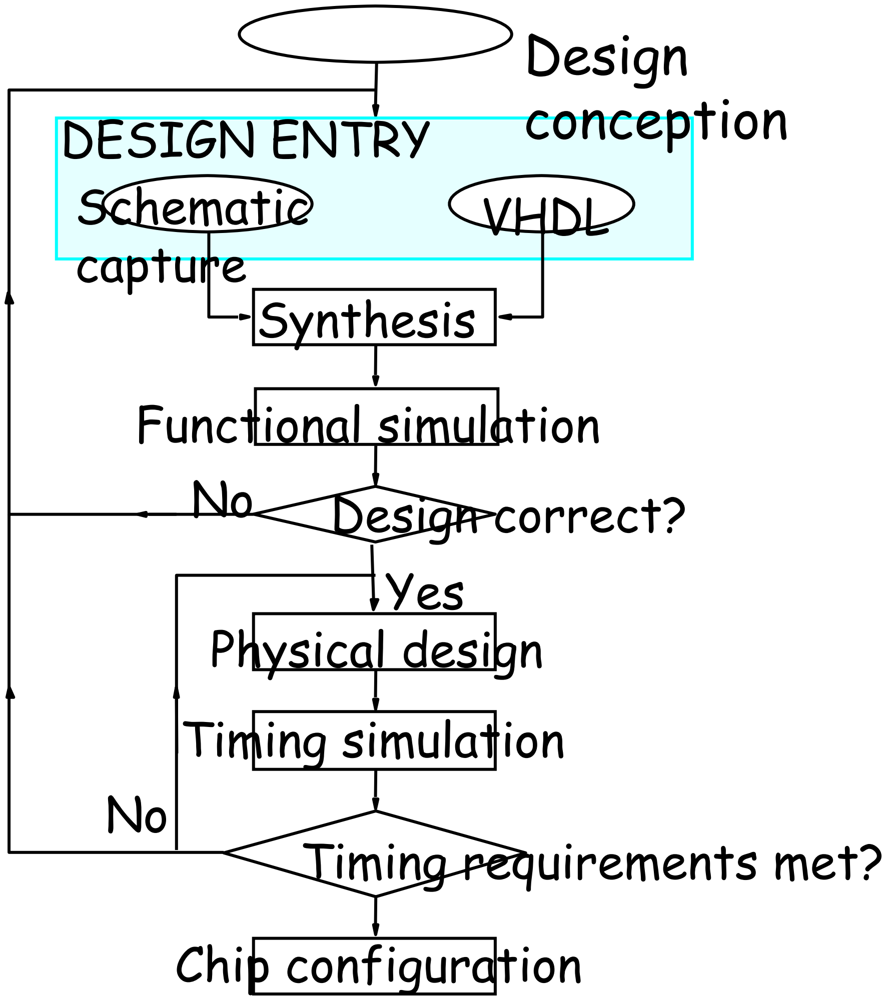
Recall: Synthesis (“HW Compilation”)

- Today’s definition of Synthesis:
  - Automatically creating an optimized gate-level description from an RTL-level description:


RTL-Level VHDL Code
Required Speed/Size
Implementation Target
Synthesis Tool (Synopsys, Quartus II, ISE...)
Optimized Gate-Level Description of Circuit

- Tomorrow’s definition of Synthesis:
  - Automatically creating an optimized gate-level description from a behavioural description

Design Flow



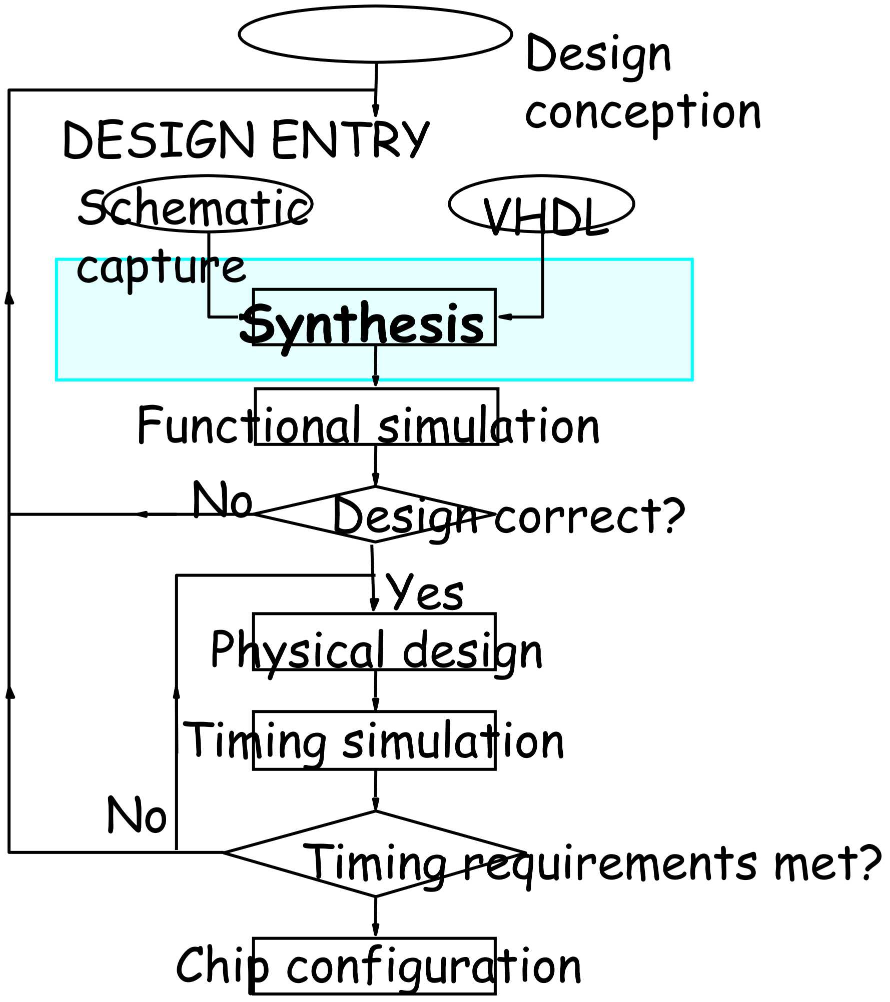
B&V: Figure 2.29. A typical CAD system.

So far we've been talking about:

Design Entry: How do you write your HDL so that the tools can synthesize it.

FPGA Architecture: Where does our design finally end up running

Design Flow



B&V: Figure 2.29. A typical CAD system.

This lecture: We'll talk about how the "magical" transformation occurs

Steps of Synthesis

1. Behavioural level Synthesis
2. Technology Mapping
3. Placement
4. Routing

Step 1: Behavioural level Synthesis

- Everybody writes HDL differently

- The HDL you have written needs to be transformed into a more generic format
  - Can you think of why?

Step 1: Behavioural level Synthesis

- In this phase, the parser looks for recognizable constructs
  - Remember the 3 forms of synthesizable processes
- Is each process a flipflop? Combinational logic? Combination of the two?
- The flipflops will be mapped to flipflops, the combinational logic will be mapped to LUTs

Step 1: Behavioural level Synthesis

- Final notes:
  - Structural logic is easier to map than behavioural logic (the synthesizer is more likely to "get it right")
    - The exception being flipflops – use processes the way we talked about otherwise the tools won't recognize the flipflops
  - If you use any special embedded blocks (e.g. multipliers BRAMs, etc.), they will not be mapped to LUTs,
    - Don't worry about it
  - If you use library functions/IP cores, those are also special cases
    - Don't worry about it

Step 2: Technology Mapping

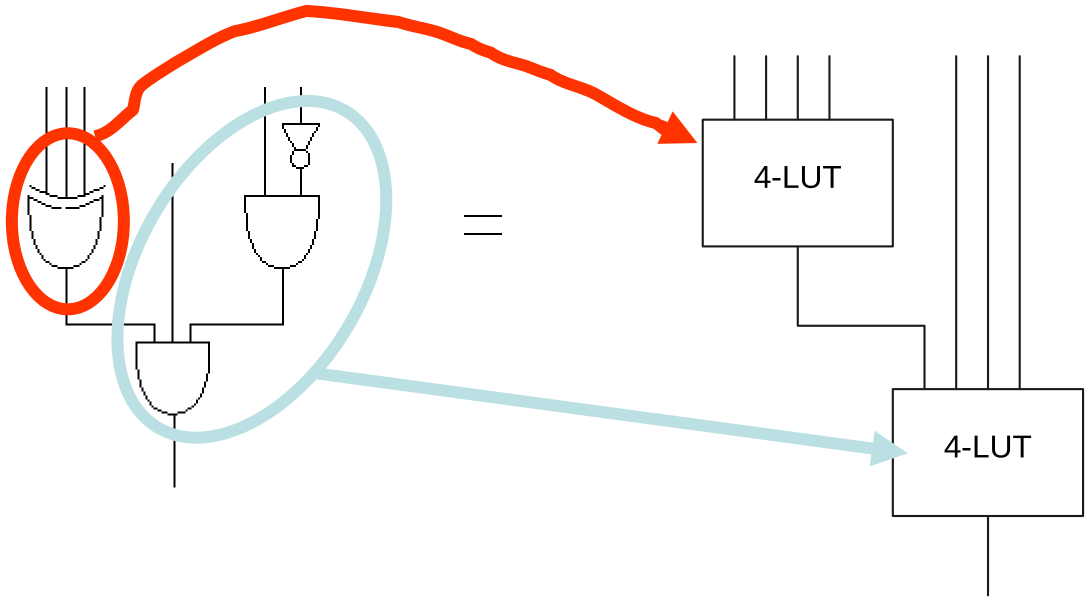
- You now have a netlist of logic gates
- This needs to be mapped into the technology available on your chip
  - In the case of an FPGA, the majority of the logic is implemented using logic tables
- Question:
  - How many different functions can be implemented by a 4-input lookup table?

Step 2: Technology Mapping

- You now have a netlist of logic gates
- This needs to be mapped into the technology available on your chip
  - In the case of an FPGA, the majority of the logic is implemented using logic tables
- Question:
  - How many different functions can be implemented by a 4-input lookup table?
    - Answer: $2^{2^4} = 65\ 536$
    - WHY?

Step 2:The Technology Mapping Problem

- Input: A netlist of gates
- Output: A netlist of logic blocks that implement the function

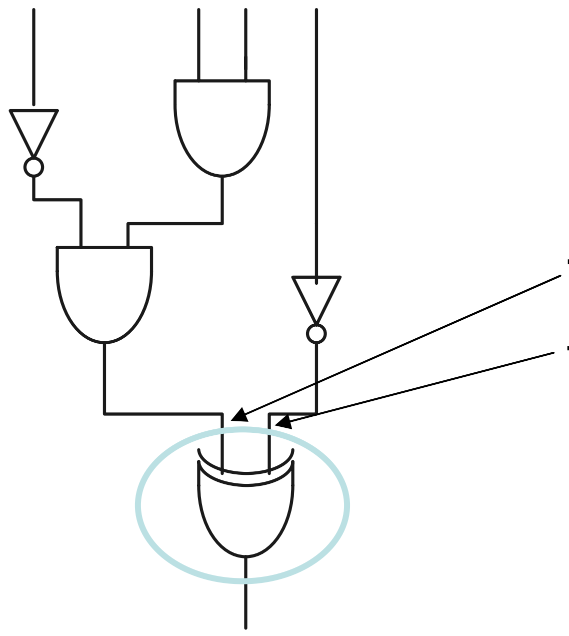


Step 2: Technology Mapping

- This also called “packing” as we try to fit (pack) as many gates as possible into one LUT
  - Why do we want to pack as much logic as possible into one LUT?
    - Hint: Think about how this affects circuit delay
- How many logic gates can you fit into one LUT?
    - Answer: It’s not the number of gates, it’s the number of logic equation ***inputs***

Step 2: Technology Mapping Example

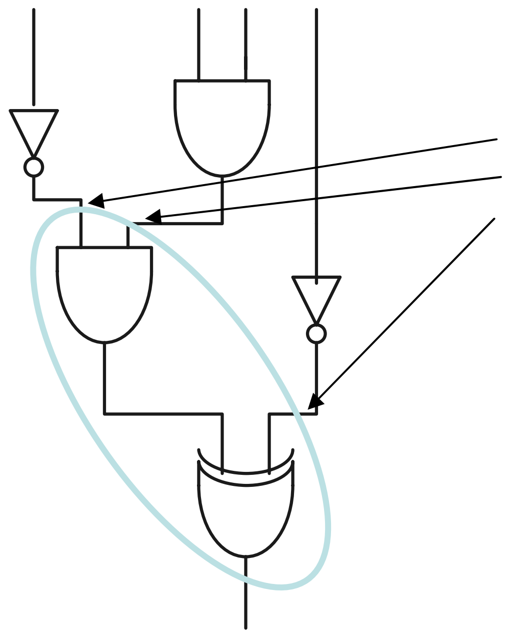
- Example: consider packing into 3-input lookup tables



Start with "root" (output)

Step 2: Technology Mapping Example

- Example: consider packing into 3-input lookup tables

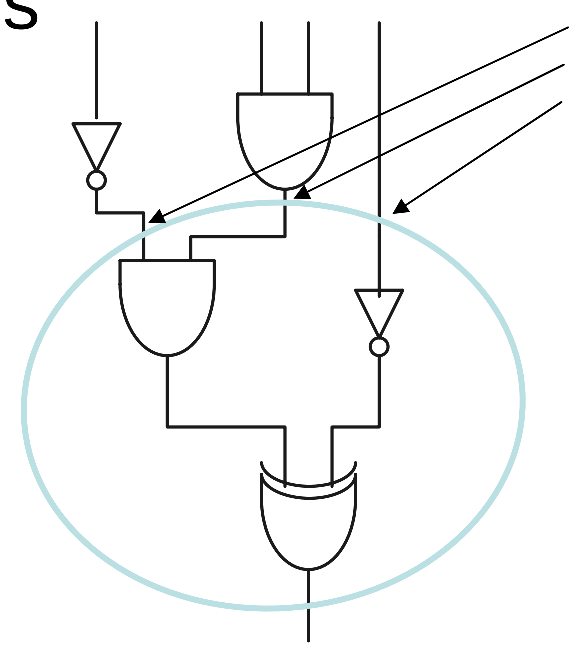


Three inputs, so AND gate can be implemented in same LUT

Consider each input

Step 2: Technology Mapping Example

- Example: consider packing into 3-input lookup tables

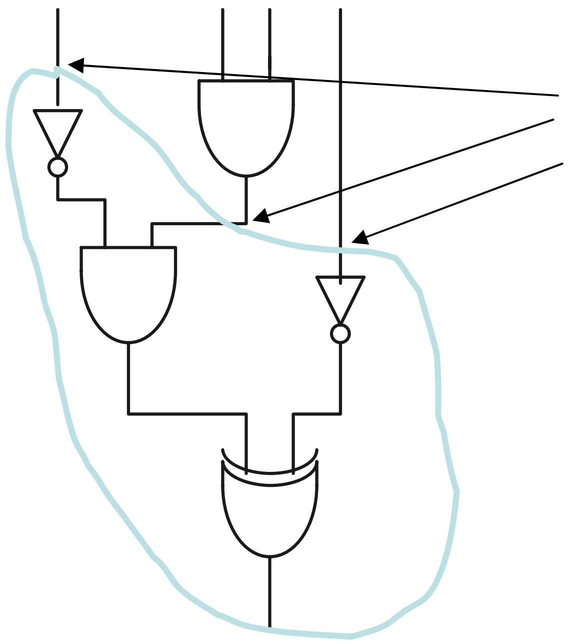


Three inputs, so Inverter can be implemented in same LUT

Consider each input

Step 2: Technology Mapping Example

- Example: consider packing into 3-input lookup tables

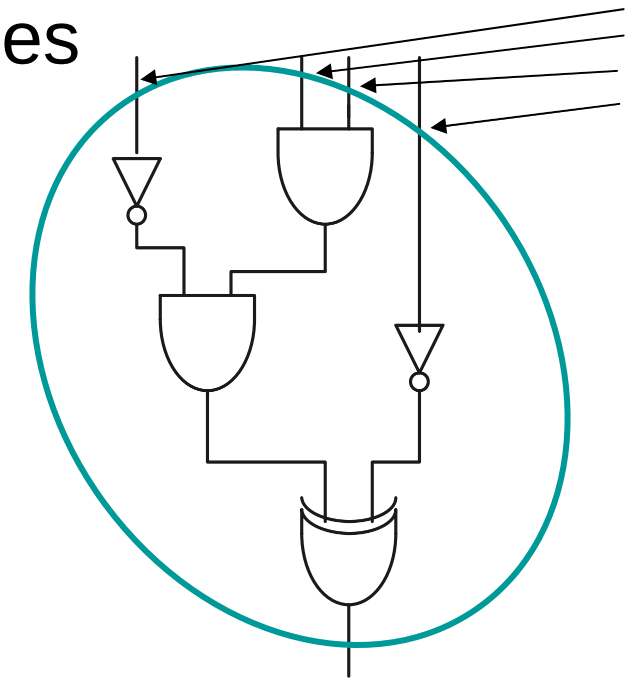


Three inputs, so Inverter can be implemented in same LUT

Consider each input

Step 2: Technology Mapping Example

- Example: consider packing into 3-input lookup tables

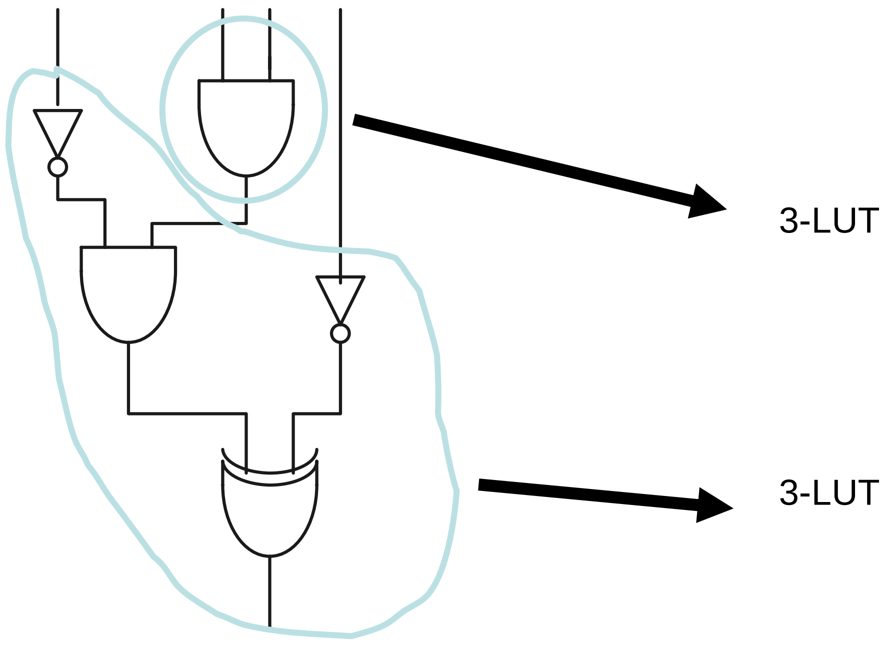


Four Inputs, so we <u>cannot</u> pack this AND gate

Consider each input

Step 2: Technology Mapping Example

- Example: consider packing into 3-input lookup tables



Go back, and start with AND gate

Step 2: Technology Mapping

- The good news:
  - We can do a good job of this
    - We can get Delay optimal mapping without having to wait forever
      - In other words, this is not the hard part
      - Caveat, other types of embedded blocks make this a harder problem

Step 3: Placement

- Now we have a set of LUTs, flipflops, etc
  - Where do we put them on the chip?
    - This is the "placement" problem and it is a ***VERY*** hard problem
      - 4N!
      - This is called an NP-complete problem (Cannot be completed in polynomial time)
      - We can't guarantee that we'll get the "right" (optimal) answer here

Step 3: Placement

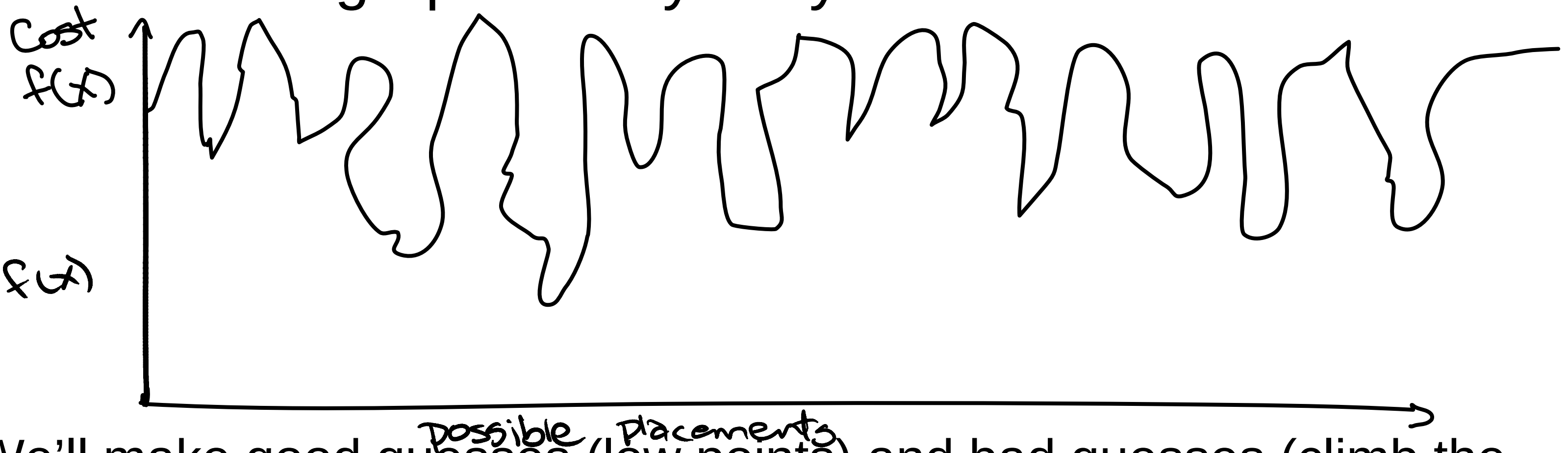
- For example, say we have a 5x4 placement problem
  - An exhaustive search would require checking 20! solutions and choosing the best
    - If finding each solution took 1ms, then it would take 77 million years to find the solution
      - ~10 million years before the last dinosaur
    - So if the dinosaurs started their computer to try and solve this placement problem, we would just be getting the final solution

- We have to place 10s to 100s of thousands of LUTs
  - Obviously, this would take way too long to search for the optimal solution

Step 3: Placement

- Instead we use heuristics:
  - A computational method that uses trial and error methods to approximate a solution for computationally difficult problems
  - The solution graph is very "hilly"

The Better placement The lower the cost f(x)



  - We'll make good guesses (low points) and bad guesses (climb the hill)

Step 3: Placement

- A popular heuristic is Simulated Annealing:
  - Annealing is a process by which method is slowly cooled over time allowing atomic structures to settle into low energy states (strong but not brittle)
  - At high "temperatures", we can make bad decisions
    - Gets us out of local minima so we can explore more of the graph
    - At low temperatures, we make mostly good choices and try and settle on a good choice
      - Note: This is a good choice and not necessarily the best choice

Step 3: Placement

- Summary:
  - We try to get a good placement because the closer the LUTs are to each other, the shorter the wire length
    - The shorter wire length means shorter circuit delays and better clock rates
    - We aren't doing a good job
      - Normalized for increased computing power and gate density, our ability to place and route a circuit is 11x slower than it was about a decade ago
      - This is an area of active research: we're trying to do better

Step 4: Routing

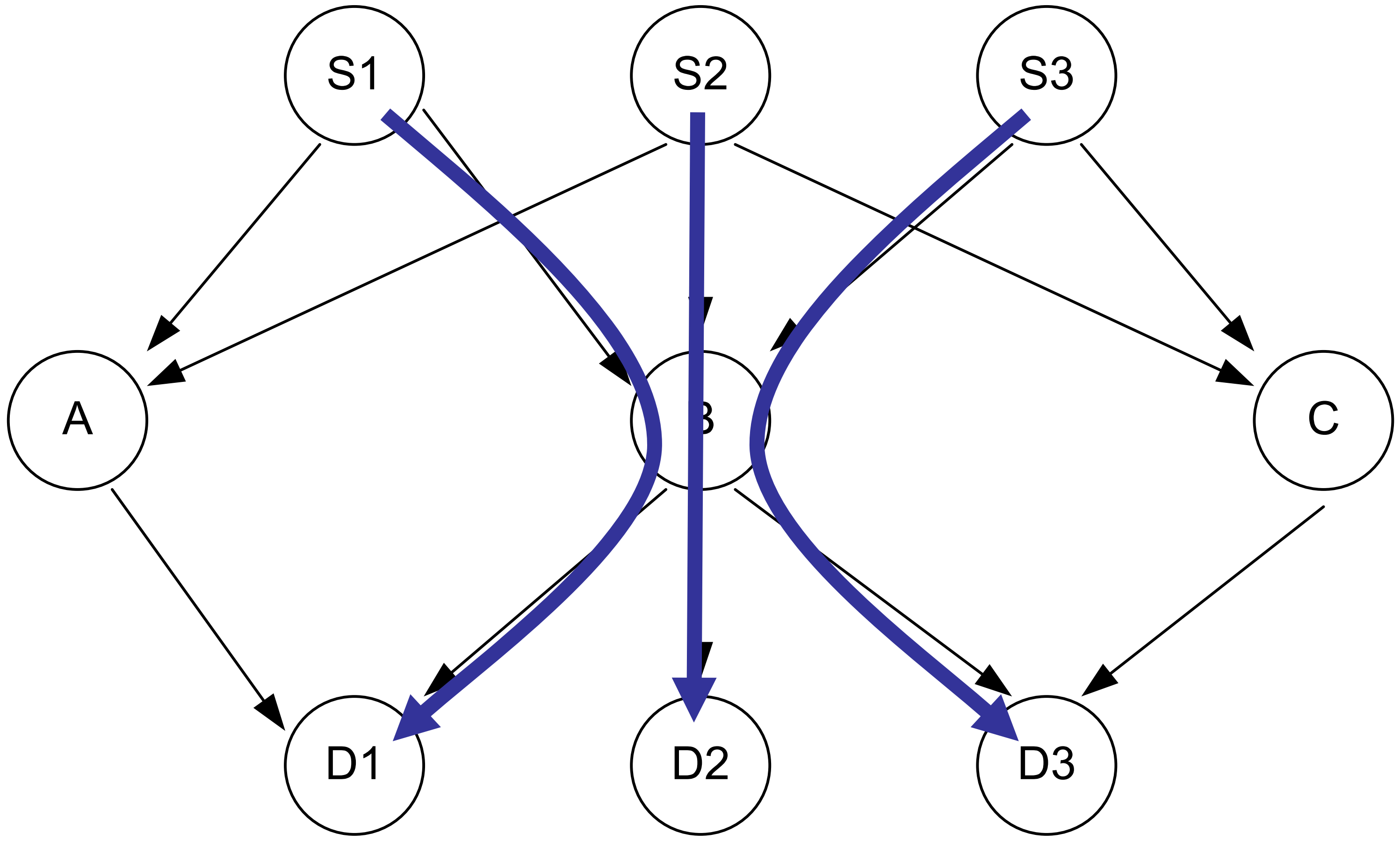
- All the logic components (LUTs, etc) have been placed on the fabric
- Now we need to wire up the connections between the blocks
  - In other words, we need to route wires on the available routing tracks
  - Want to use the shortest possible path between LUTs
- Sometimes this will cause congestion (if lots of blocks are placed close together that need to connect to each other)

Step 4: Routing

- The algorithm Pathfinder does a good job of finding ways to route wires around the congestion by keeping track of the congestion at a node
  - If there is too much congestion at a node (ie if you run out of wiring tracks), it rips up the routing and tries again
    - On this second try, it makes the over used routing tracks more expensive to use. This forces the router to use a different path for some of the connections to relieve congestion

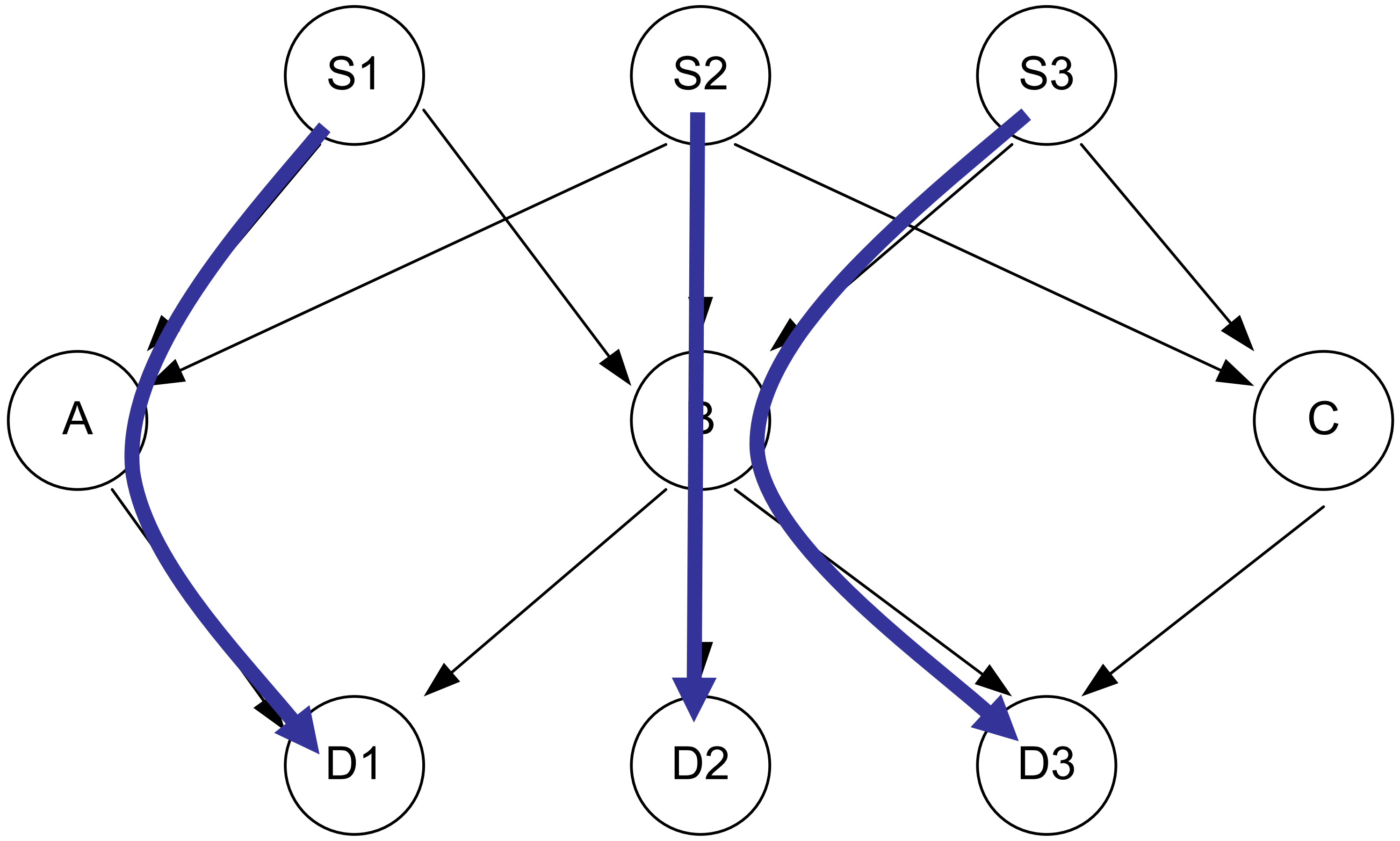
Step 4: Routing Congestion Example

- Pathfinder detects congestion from sources S1,S2 and S3 through routing resource B to destinations D1, D2, and D3
  - In other words, all 3 paths want to use wire segment B



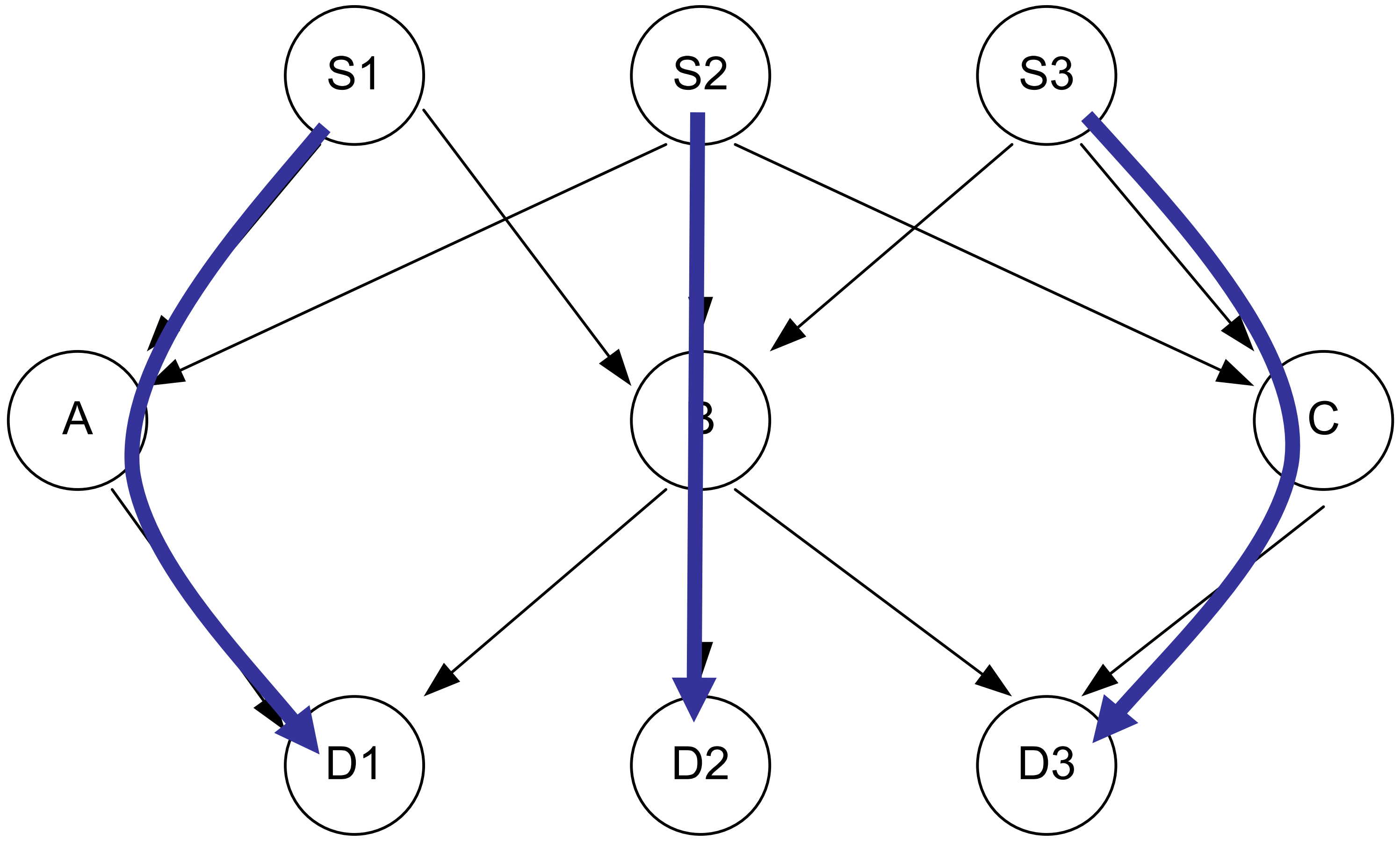
Step 4: Routing Congestion Example

- Pathfinder Rips up *all* of the nets, and tries to route again



Step 4: Routing Congestion Example

- Pathfinder will keep trying until it successfully routes the paths or until some defined expiry time is reached at which point it will return a "failure to route"



Bitstream Generation- the final phase

- Bitstream generation is not considered part of synthesis
  - After a circuit has been routed, its location and connectivity are defined
  - Bitstream generation translates this map into the necessary configuration bits to activate/deactivate the appropriate switches and LUT values that will implement this circuit
    - This bitstream is what is loaded onto the FPGA when you download your circuit

Bitstream Generation- the final phase

- This is the really easy part
  - There's a one-to-one mapping between the placed and routed circuit and the final bitstream
  - The analogy is the translation of assembly code into machine code
- Remember:
  - Bitstreams are device specific
  - If you want to download the same circuit onto a different part, you have to generate a new bitstream for a new placement and routing for that new device

Questions

- What is the first step in synthesis called?

- What does the first step do?

Questions

- What is the second step in synthesis called?

- What does the second step do?

Questions

- What is the third step in synthesis called?

- What does the third step do?

Questions

- What is the fourth step in synthesis called?

- What does the fourth step do?

Questions

- What is the easiest step in the synthesis process?

- What is the hardest step in the synthesis process?

Questions

- After we’ve completed synthesis, what’s left before we can download the circuit?

- Is this hard (ie time consuming)?