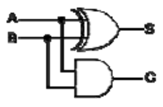
2. SRAM-based FPGA

Current/Future Directions

- FPGA (Field-programmable gate arrays) - mid 1980s
  - Misleading name - there is no array of gates
  - Array of fine-grained configurable components
    - Will discuss architecture shortly
  - Currently support millions of gates
- Coarse-grained RC architectures
  - Array of coarse-grained components
    - Multipliers, DSP units, etc.
  - Potentially, larger capacity than FPGA
    - But, applications may not map well
      - Wasted resources
      - Inefficient execution

FPGA Architectures

- How can we implement any circuit in an FPGA?
  - First, focus on combinational logic
  - Example: Half adder
    - Combinational logic represented by truth table
    - What kind of hardware can implement a truth table?

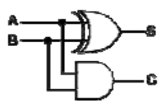


Input		Out
A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

Input		Out
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Look-up-tables (LUTs)

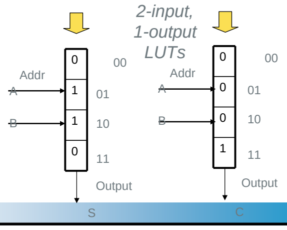
- Implement truth table in small memories (LUTs)
  - Usually SRAM



A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

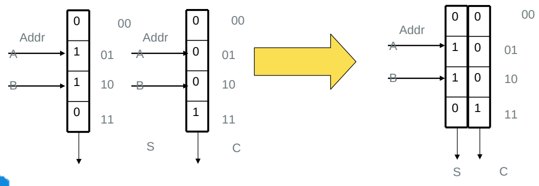
2-input, 1-output LUTs



Logic inputs connect to address inputs, logic output is memory output

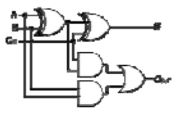
Look-up-tables (LUTs)

- Alternatively, could have used a 2-input, 2-output LUT
  - Outputs commonly use same inputs



Look-up-tables (LUTs)

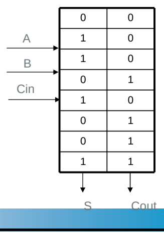
- Slightly bigger example: Full adder
  - ***Combinational logic can be implemented in a LUT with same number of inputs and outputs***
    - 3-input, 2-ouput LUT



Truth Table

Inputs			Outputs	
A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

3-input, 2-output LUT

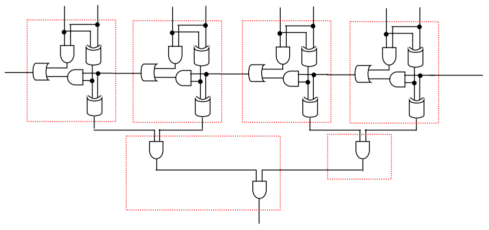


Look-up-tables (LUTs)

- Why aren't FPGAs just a big LUT?
  - Size of truth table grows exponentially based on # of inputs
    - 3 inputs = 8 rows, 4 inputs = 16 rows, 5 inputs = 32 rows, etc.
  - Same number of rows in truth table and LUT
  - LUTs grow exponentially based on # of inputs
- Number of SRAM bits in a LUT = $2^i * o$
  - i = # of inputs, o = # of outputs
  - Example: 64 input combinational logic with 1 output would require 2^{64} SRAM bits
    - 1.84×10^{19}
- Clearly, not feasible to use large LUTs
  - So, how do FPGAs implement logic with many inputs?

Look-up-tables (LUTs)

- Fortunately, we can map circuits onto multiple LUTs
  - Divide circuit into smaller circuits that fit in LUTs (same # of inputs and outputs)
  - Example: 3-input, 2-output LUTs

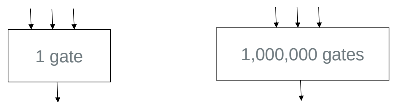


Look-up-tables (LUTs)

- What if circuit doesn't map perfectly?
  - More inputs in LUT than in circuit
    - Truth table handles this problem
    - Unused inputs are ignored
  - More outputs in LUT than in circuit
    - Extra outputs simply not used
      - Space is wasted, so should use multiple outputs whenever possible

Look-up-tables (LUTs)

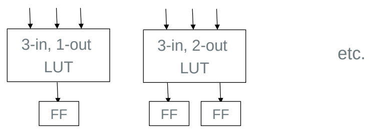
- Important Point
  - The number of gates in a circuit has no effect on the mapping into a LUT
    - All that matters is the number of inputs and outputs
    - Unfortunately, it isn't common to see large circuits with a few inputs



Both of these circuits can be implemented in a single 3-input, 1-output LUT

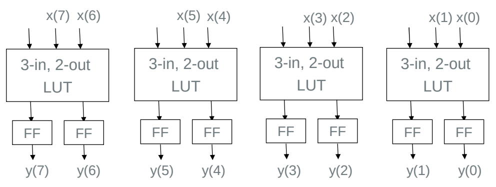
Sequential Logic

- Problem: How to handle sequential logic
  - Truth tables don't work
- *Possible solution*:
  - Add a flip-flop to the output of LUT



Sequential Logic

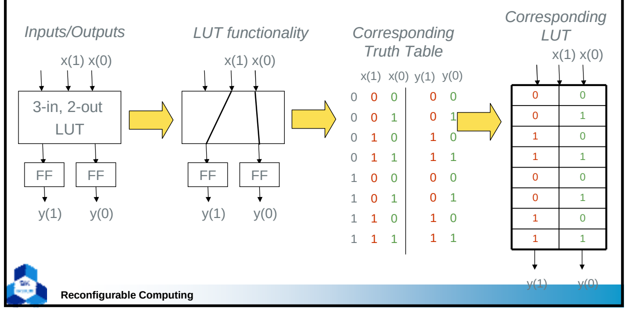
- Example: 8-bit register using 3-input, 2-output LUTs
  - Input: x, Output: y



  - **What does LUT need to do to implement register?**

Sequential Logic

- Example, cont.
  - LUT simply passes inputs to appropriate output

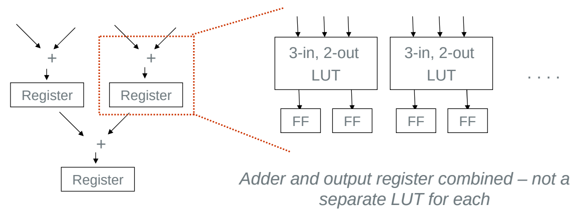


	x(1)	x(0)	y(1)	y(0)
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

0	0
0	1
1	0
1	1
0	0
0	1
1	0
1	1

Sequential Logic

- Isn't it a waste to use LUTs for registers?
- YES! (when it can be used for something else)
  - Commonly used for pipelined circuits
    - Example: Pipelined adder



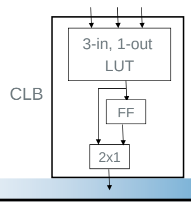
Adder and output register combined – not a separate LUT for each

Sequential Logic

- Existing FPGAs don't have a flip flop connected to LUT outputs
- Why not?
  - Flip flop has to be used!
    - Impossible to have pure combinational logic
  - Adds latency to circuit
- ***Actual Solution:***
  - Configurable Logic Blocks (CLBs)

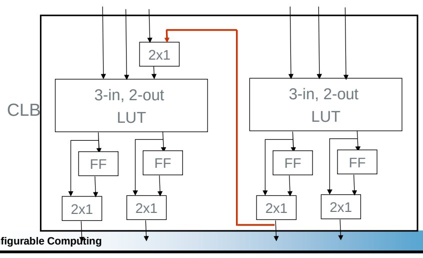
Configurable Logic Blocks (CLBs)

- CLBs: the basic FPGA functional unit
  - First issue: How to make flip-flop optional?
    - Simplest way: use a mux
      - Circuit can now use output from LUT or from FF
      - Where does select come from? (will be answered shortly)



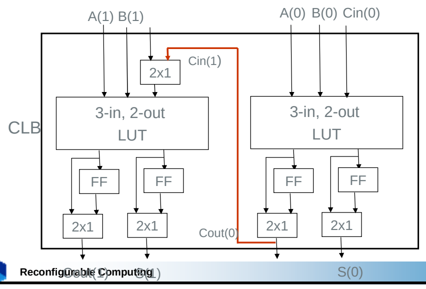
Configurable Logic Blocks (CLBs)

- CLBs usually contain more than 1 LUT
  - Why?
    - Efficient way of handling common I/O between adjacent LUTs
    - Saves routing resources (we haven't discussed yet)



Configurable Logic Blocks (CLBs)

- Example: Ripple-carry adder
  - Each LUT implements 1 full adder
  - Use efficient connections between LUTs for carry signals



Configurable Logic Blocks (CLBs)

- CLBs often have specialized connections between adjacent CLBs
  - Further improves carry chains
  - Avoids routing resources
- Some commercial CLBs even more complex
  - Xilinx Virtex 4 CLB consists of 4 “slices”
    - 1 slice = 2 LUTs + 2 FFs + other stuff
    - 1 Virtex 4 CLB = 8 LUTs
  - Altera devices has LABs (Logic Array Blocks)
    - Consist of 16 LEs (logic elements) which each have 4 input LUTs

What Else?

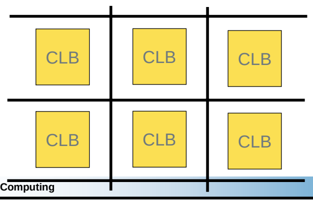
- Basic building block is CLB
  - Can implement combinational+sequential logic
  - All circuits consist of combinational and sequential logic
- So what else is needed?

Reconfigurable Interconnect

- FPGAs need some way of connecting CLBs together
  - Reconfigurable interconnect
  - But, we can only put fixed wires on a chip
- Problem: How to make reconfigurable connections with fixed wires?
  - Main challenge:
    - Should be flexible enough to support almost any circuit

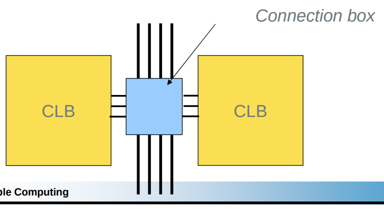
Reconfigurable Interconnect

- Problem 2: If FPGA doesn’t know which CLBs will be connected, where does it put wires?
- Solution:
  - Put wires everywhere!
    - Referred to as channel wires, routing channels, routing tracks, many others
  - CLBs typically arranged in a grid, with wires on all sides



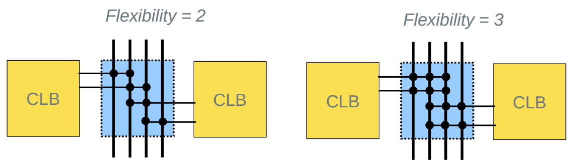
Reconfigurable Interconnect

- Problem 3: How to connect CLB to wires?
- Solution: Connection box
  - Device that allows inputs and outputs of CLB to connect to different wires



Reconfigurable Interconnect

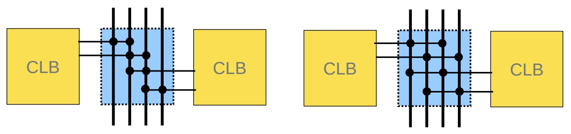
- Connection box characteristics
  - Flexibility
    - The number of wires a CLB input/output can connect to



*Dots represent **possible** connections*

Reconfigurable Interconnect

- Connection box characteristics
  - Topology
    - Defines the specific wires each CLB I/O can connect to
    - Examples: same flexibility, different topology



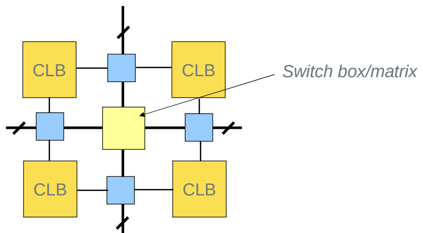
*Dots represent **possible** connections*

Reconfigurable Interconnect

- Connection boxes allow CLBs to connect to routing wires
  - But, that only allows us to move signals along a single wire
  - Not very useful
- Problem 4: How do FPGAs connect wires together?

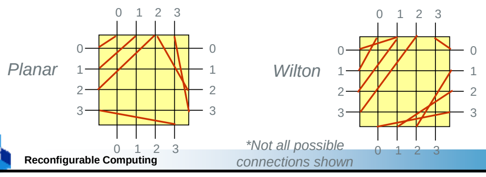
Reconfigurable Interconnect

- Solution: Switch boxes, switch matrices
  - Connects horizontal and vertical routing channels



Reconfigurable Interconnect

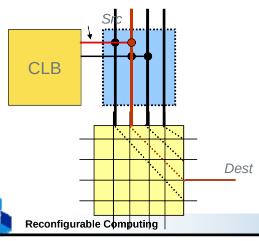
- Switch boxes
  - Flexibility - defines how many wires a single wire can connect to
  - Topology - defines which wires can be connected
    - Planar/subset switch box: only connects same channels (e.g. 0 to 0, 1 to 1, etc.)
    - Wilton switch box: connects different channels

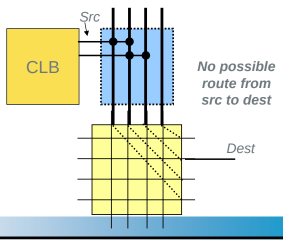


Not all possible connections shown

Reconfigurable Interconnect

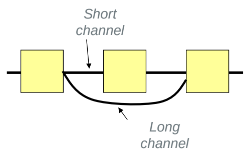
- Why do flexiblity and topology matter?
  - Routability: a measure of the number of circuits that can be routed
    - Higher flexibility = better routability
    - Wilton switch box topology = better routability





Reconfigurable Interconnect

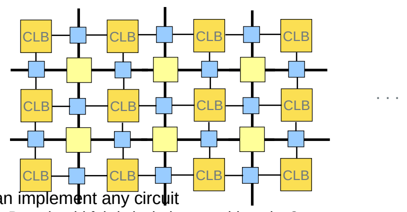
- Switch boxes
  - Short channels
    - Useful for connecting adjacent CLBs
  - Long channels
    - Useful for connecting CLBs that are separated
    - Allows for reduced routing delay for non-adjacent CLBs



dce 2010

FPGA Fabrics

- FPGA layout called a "fabric"
  - 2-dimensional array of CLBs and programmable interconnect
  - Sometimes referred to as an "island style" architecture



  - Can implement any circuit
    - But, should fabric include something else?

dce 2010

FPGA Fabrics

- What about memory?
  - Could use FF's in CLBs to create a memory
    - Example: Create a 1 MB memory with:
      - CLB with a single 3-input, 2-output LUT
    - Each CLB = 2 bits of memory (because of 2 outputs)
    - Total CLBs = (1 MB * 8 bits/byte) / 2 bits/CLB
      - 4 million CLBs!!!!
      - FPGAs commonly have tens of thousands of LUTs
        - » Large devices have 100-200k LUTs
        - » State-of-the-art devices ~800k LUTs
      - Even if FPGAs were large enough, using a chip to implement 1 MB of memory is not smart
  - Conclusion:
    - Bad Idea!! Huge waste of resources!

dce 2010

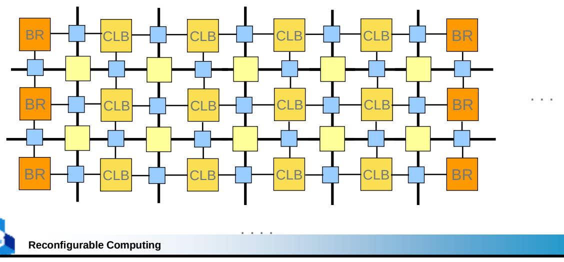
FPGA Memory Components

- Solution 1: Use LUTs for logic or memory
  - LUTs are small SRAMs, why not use them as memory?
  - Xilinx refers to as distributed RAM
- Solution 2: Include dedicated RAM components in the FPGA fabric
  - Xilinx refers to as Block RAM
    - Can be single/dual-ported
    - Can be combined into arbitrary sizes
    - Can be used as FIFO
      - Different clock speeds for reads/writes
  - Altera has Memory Blocks
    - M4K: 4k bits of RAM
    - Others: M9K, M20k, M144K

dce 2010

FPGA Memory Components

- Fabric with Block RAM
  - Block RAM can be placed anywhere
  - Typically, placed in columns of the fabric



dce 2010

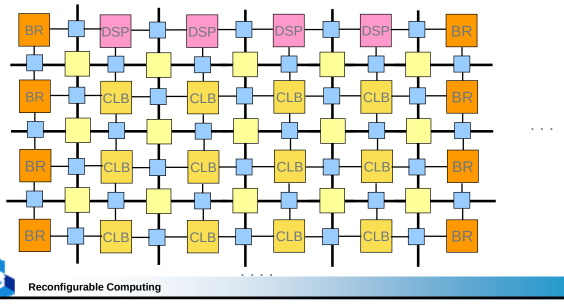
DSP Components

- FPGAs commonly used for DSP apps
  - Makes sense to include custom DSP units instead of mapping onto LUTs
    - Custom unit = faster/smaller
- Example: Xilinx DSP48
  - Includes multipliers, adders, subtractors, etc.
    - 18x18 multiplication
    - 48-bit addition/subtraction
  - Provides efficient way of implementing
    - Add/subtract/multiply
    - MAC (Multiply-accumulate)
    - Barrel shifter
    - FIR Filter
    - Square root
    - Etc.
- Altera devices have multiplier blocks
  - Can be configured as 18x18 or 2 separate 9x9 multipliers

dce 2010

Existing Fabrics

- Existing FPGAs are 2-dimensional arrays of CLBs, DSP, Block RAM, and programmable interconnect
  - Actual layout/placement differs for different FPGAs

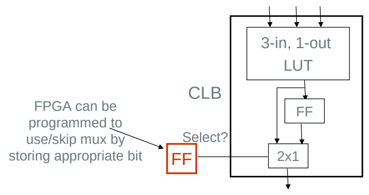


Programming FPGAs

- How to program/configure FPGA to implement circuit?
  - So far, we've mapped a circuit onto FPGA fabric
    - Known as technology mapping
      - Process of converting a circuit in one representation into a representation that corresponds to physical components
        - » Gates to LUTs
        - » Memory to Block RAMs
        - » Multiplications to DSP48s
        - » Etc.
  - But, we need some way of configuring each component to behave as desired
    - Examples:
      - How to store truth tables in LUTs?
      - How to connect wires in switch boxes?
      - Etc.

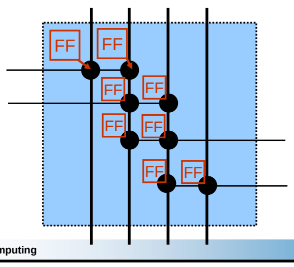
Programming FPGAs

- General Idea: include FF's in fabric to control programmable components
  - Example: CLB
    - Need a way to specify select for mux


3-in, 1-out LUT
CLB
FF
FPGA can be programmed to use/skip mux by storing appropriate bit
Select?
FF
2x1

Programming FPGAs

- Example 2:
  - Connection/switch boxes
  - Need FFs to specify connections

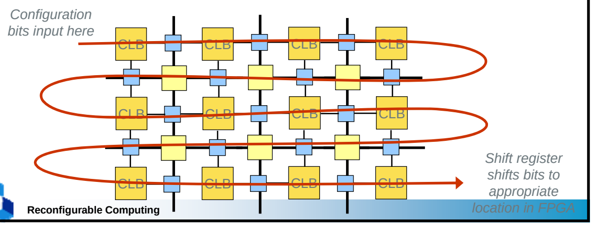


Programming FPGAs

- FPGAs programmed with a "bitfile"
  - File containing all information needed to program FPGA
    - Contains bits for each control FF
    - Also, contains bits to fill LUTs
- But, how do you get the bitfile into the FPGA?
  - > 10k LUTs
  - Small number of pins

Programming FPGAs

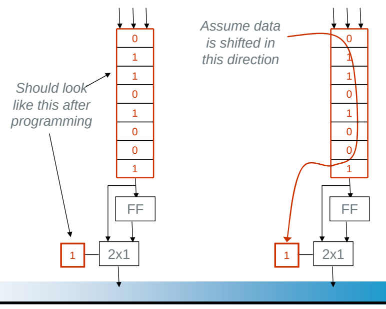
- Solution: Shift Registers
  - General Idea
    - Make a huge shift register out of all programmable components (LUTs, control FFs)
    - Shift in bitfile one bit at a time

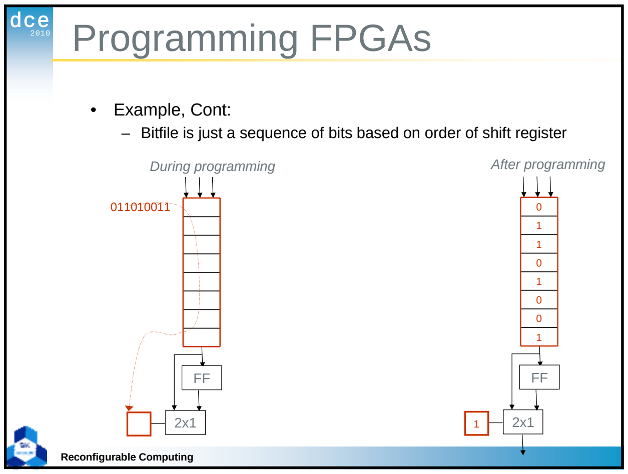

Configuration bits input here
CLB
CLB
CLB
CLB
CLB
CLB
CLB
CLB
CLB
CLB
CLB
CLB
Shift register shifts bits to appropriate location in FPGA

Programming FPGAs

- Example:
  - Program CLB with 3-input, 1-output LUT to implement sum output of full adder

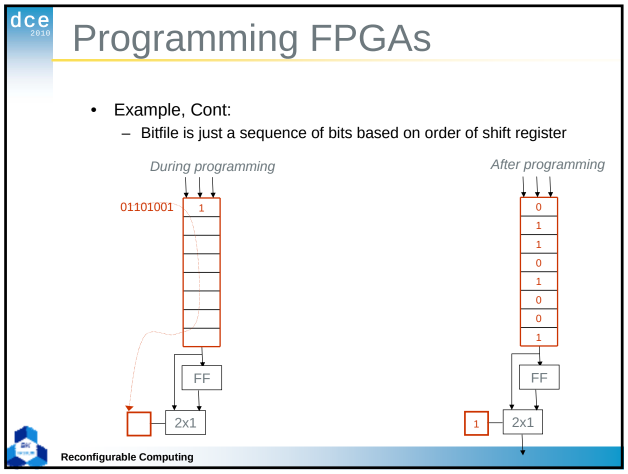
In			Out
A	B	Cin	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1


Should look like this after programming
0
1
1
0
1
0
0
1
FF
1
2x1
Assume data is shifted in this direction
0
1
1
0
1
0
0
1
FF
1
2x1



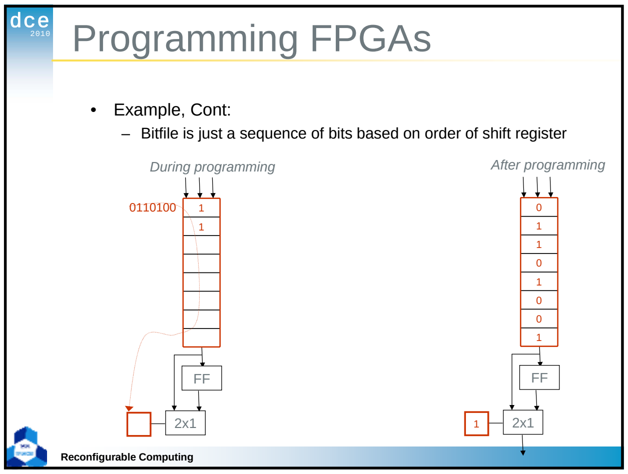
Programming FPGAs

- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register



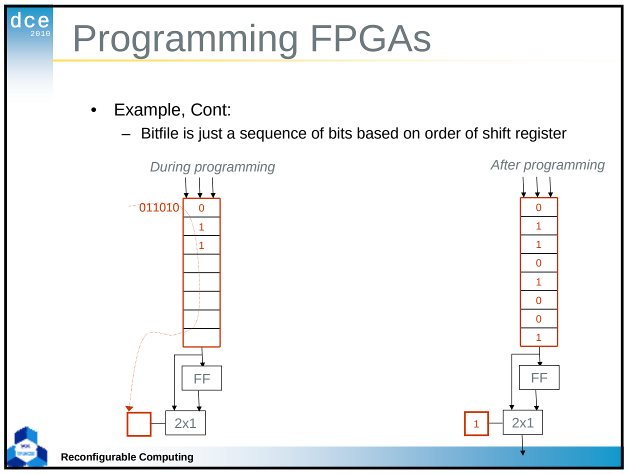
Programming FPGAs

- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register



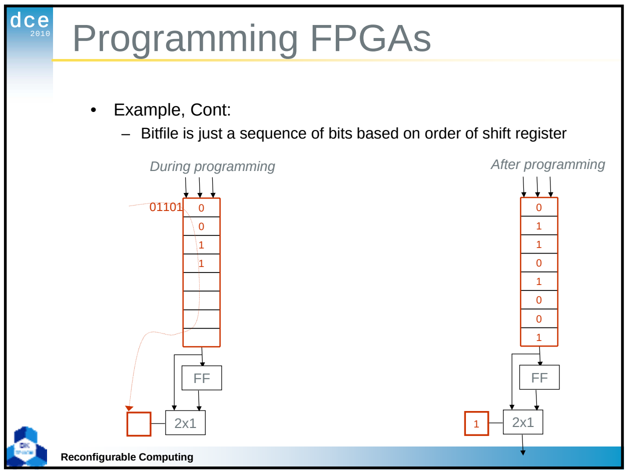
Programming FPGAs

- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register



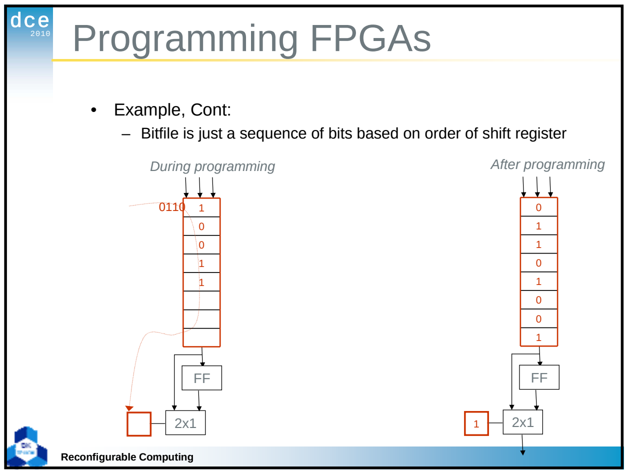
Programming FPGAs

- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register



Programming FPGAs

- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register

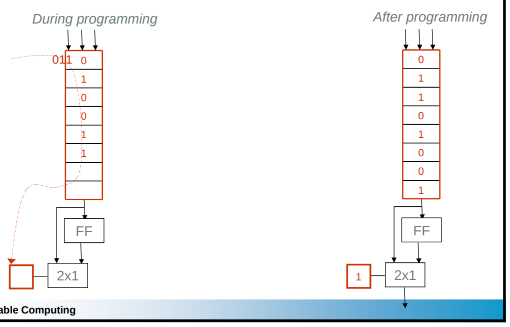


Programming FPGAs

- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register

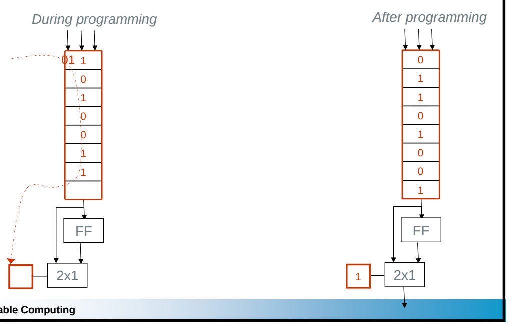
Programming FPGAs

- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register



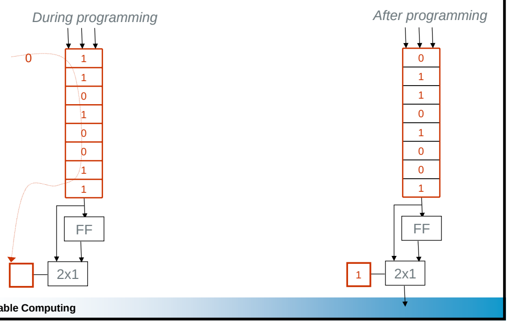
Programming FPGAs

- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register



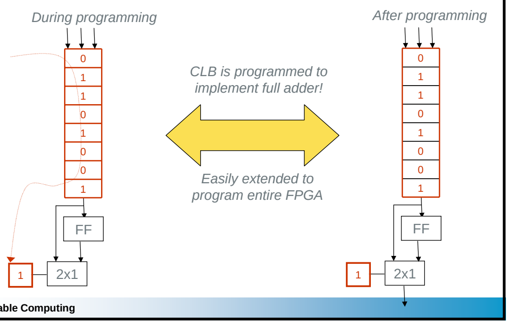
Programming FPGAs

- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register



Programming FPGAs

- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register


During programming
After programming
CLB is programmed to implement full adder!
Easily extended to program entire FPGA
0
1
1
0
1
0
0
1
FF
1
2x1
0
1
1
0
1
0
0
1
FF
1
2x1

Programming FPGAs

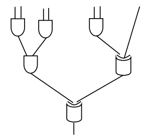
- Problem: Reconfiguring FPGA is slow
  - Shifting in 1 bit at a time not efficient
  - Bitfiles can be greater than 1 MB
  - Eliminates one of the main advantages of RC
    - Partial reconfiguration
    - With shift registers, entire FPGA has to be reconfigured
- Solutions?
  - Virtex II allows columns to be reconfigured
  - Virtex IV allows custom regions to be reconfigured
  - Requires a lot of user effort
    - Better tools needed

FPGA Architecture Tradeoffs

- LUTs with many inputs can implement large circuits efficiently
  - Why not just use LUTs with many inputs?
- High flexibility in routing resources improves routability
  - Why not just allow all possible connections?
- Answer: architectural tradeoffs
  - Anytime one component is increased/improved, there is less area for other components
    - Larger LUTs => less total LUTs, less routing resources
    - More Block RAM => less LUTs, less DSPs
    - More DSPs => less LUTs, less Block RAM
    - Etc.

FPGA Architecture Tradeoffs

- Example:
  - Determine best LUTs for following circuit
    - Choices
      - 4-input, 2-output LUT (delay = 2 ns)
      - 5-input, 2-output LUT (delay = 3 ns)
    - Assume each SRAM cell is 6 transistors
      - 4-input LUT = 6 * 2^4 * 2 = 192 transistors
      - 5-input LUT = 6 * 2^5 * 2 = 384 transistors

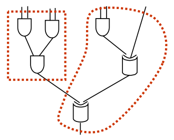


Reconfigurable Computing

FPGA Architecture Tradeoffs

- Example:
  - Determine best LUTs for following circuit
    - Choices
      - 4-input, 2-output LUT (delay = 2 ns)
      - 5-input, 2-output LUT (delay = 3 ns)
    - Assume each SRAM cell is 6 transistors
      - 4-input LUT = 6 * 2^4 * 2 = 192 transistors
      - 5-input LUT = 6 * 2^5 * 2 = 384 transistors

5-input LUT



Propagation delay = 6 ns

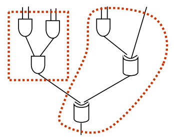
Total transistors = 384 * 2 = 768

Reconfigurable Computing

FPGA Architecture Tradeoffs

- Example:
  - Determine best LUTs for following circuit
    - Choices
      - 4-input, 2-output LUT (delay = 2 ns)
      - 5-input, 2-output LUT (delay = 3 ns)
    - Assume each SRAM cell is 6 transistors
      - 4-input LUT = 6 * 2^4 * 2 = 192 transistors
      - 5-input LUT = 6 * 2^5 * 2 = 384 transistors

4-input LUT



Propagation delay = 4 ns

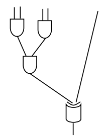
Total transistors = 192 * 2 = 384

4-input LUTs are 1.5x faster and use 1/2 the area

Reconfigurable Computing

FPGA Architecture Tradeoffs

- Example 2
  - Determine best LUTs for following circuit
    - Choices
      - 4-input, 2-output LUT (delay = 2 ns)
      - 5-input, 2-output LUT (delay = 3 ns)
    - Assume each SRAM cell is 6 transistors
      - 4-input LUT = 6 * 2^4 * 2 = 192 transistors
      - 5-input LUT = 6 * 2^5 * 2 = 384 transistors

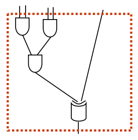


Reconfigurable Computing

FPGA Architecture Tradeoffs

- Example 2
  - Determine best LUTs for following circuit
    - Choices
      - 4-input, 2-output LUT (delay = 2 ns)
      - 5-input, 2-output LUT (delay = 3 ns)
    - Assume each SRAM cell is 6 transistors
      - 4-input LUT = 6 * 2^4 * 2 = 192 transistors
      - 5-input LUT = 6 * 2^5 * 2 = 384 transistors

5-input LUT



Propagation delay = 3 ns

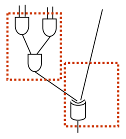
Total transistors = 384

Reconfigurable Computing

FPGA Architecture Tradeoffs

- Example 2
  - Determine best LUTs for following circuit
    - Choices
      - 4-input, 2-output LUT (delay = 2 ns)
      - 5-input, 2-output LUT (delay = 3 ns)
    - Assume each SRAM cell is 6 transistors
      - 4-input LUT = 6 * 2^4 * 2 = 192 transistors
      - 5-input LUT = 6 * 2^5 * 2 = 384 transistors

4-input LUT

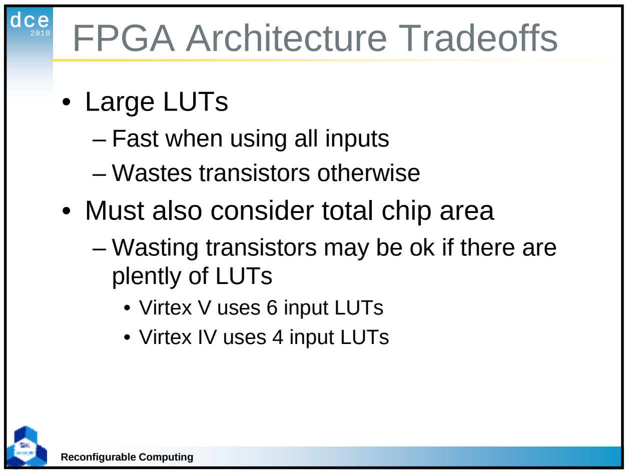


Propagation delay = 4 ns

Total transistors = 384 transistors

5-input LUTs are 1.3x faster and use same area

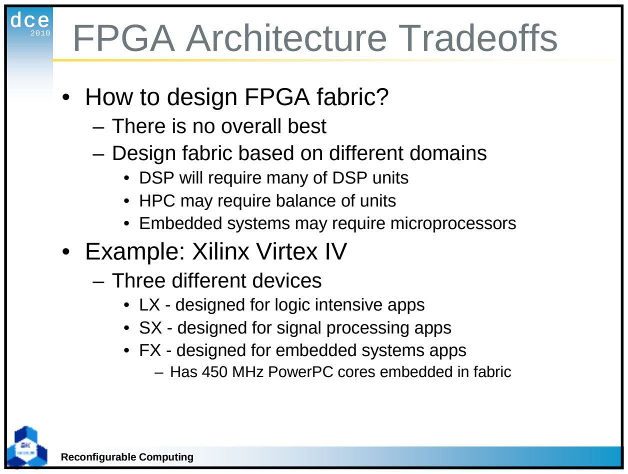
Reconfigurable Computing



FPGA Architecture Tradeoffs

- Large LUTs
  - Fast when using all inputs
  - Wastes transistors otherwise
- Must also consider total chip area
  - Wasting transistors may be ok if there are plently of LUTs
    - Virtex V uses 6 input LUTs
    - Virtex IV uses 4 input LUTs

Reconfigurable Computing



FPGA Architecture Tradeoffs

- How to design FPGA fabric?
  - There is no overall best
  - Design fabric based on different domains
    - DSP will require many of DSP units
    - HPC may require balance of units
    - Embedded systems may require microprocessors
- Example: Xilinx Virtex IV
  - Three different devices
    - LX - designed for logic intensive apps
    - SX - designed for signal processing apps
    - FX - designed for embedded systems apps
      - Has 450 MHz PowerPC cores embedded in fabric

Reconfigurable Computing