ECE4810J System-on-Chip Design

Topic 6

Reconfigurable SoC

Xinfei Guo
xinfei.guo@sjtu.edu.cn

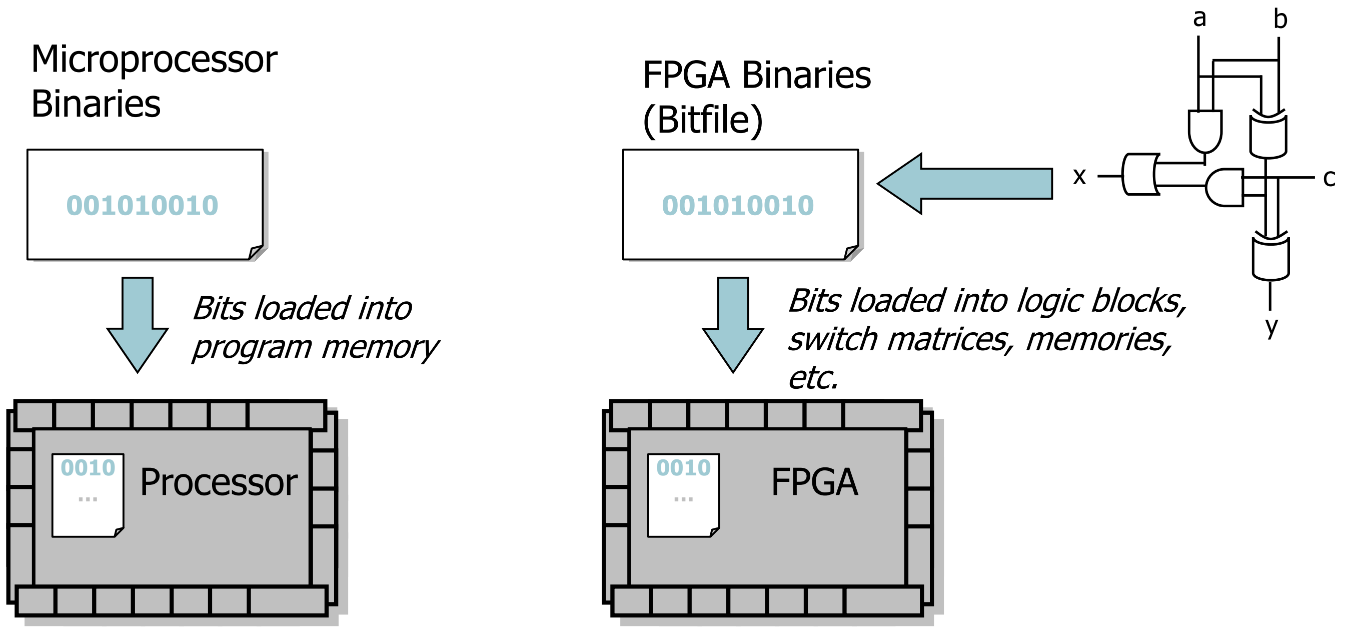
November 11th, 2024

T7 learning goals

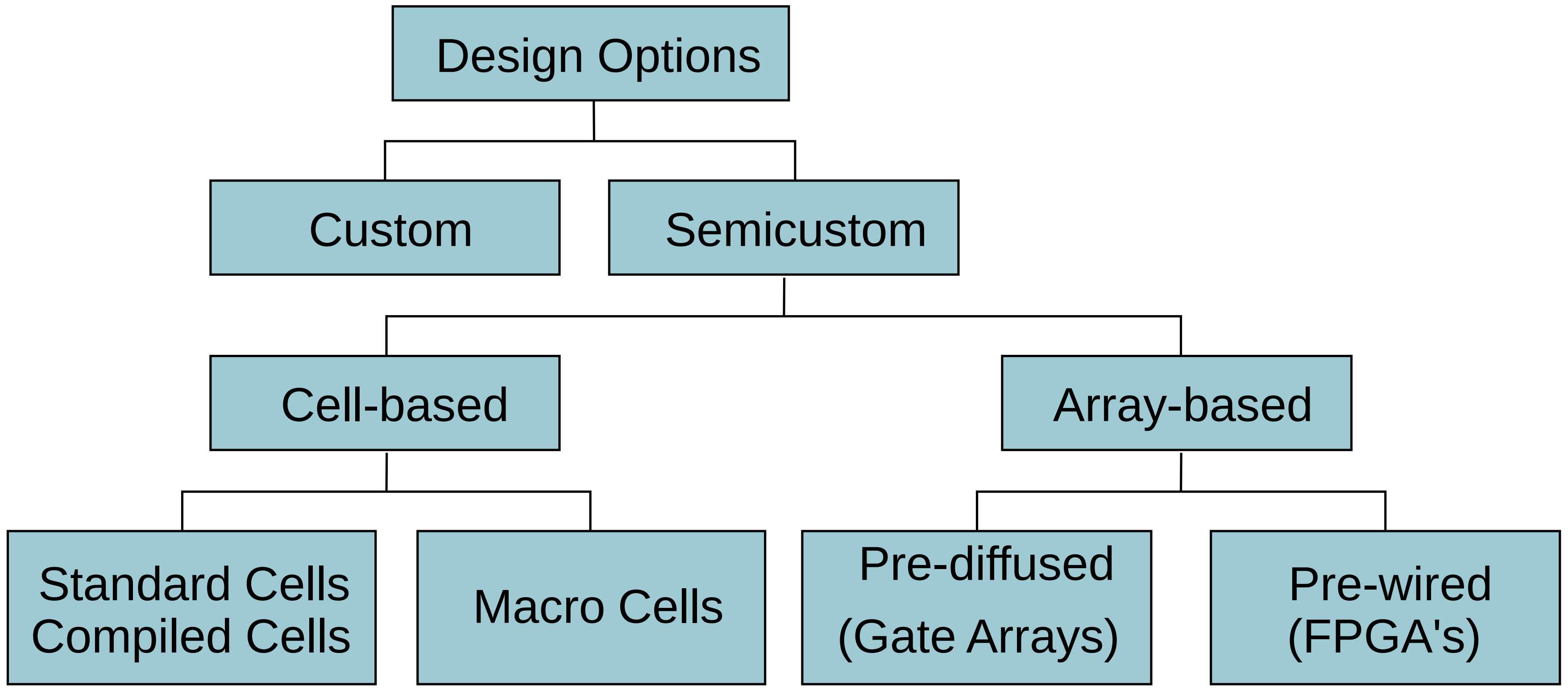
- Reconfigurable SoC
  - Why?
  - How?
  - What?

What is Reconfigurable Computing?

- Reconfigurable computing (RC) is study of SoC architectures that can adapt (after fabrication) to a specific application or application domain
  - Involves architecture, tools, CAD, design automation, algorithms, languages, etc.



Recall: Design Options



Recall: Design Approaches

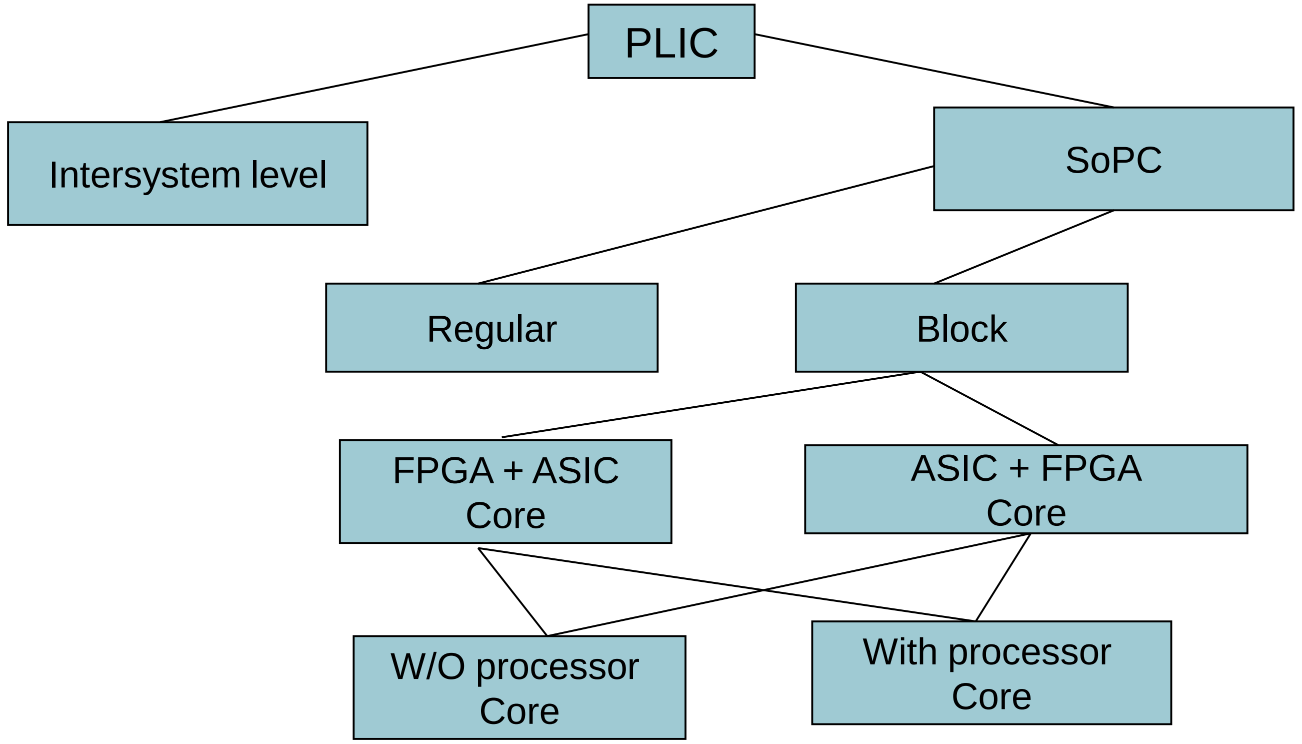
	Custom	Cell-based	Pre-diffused	Pre-wired
Density	Very High	High	High	Medium
Performance	Very High	High	High	Medium
Flexibility	Very High	High	Medium	Low
Design time	Very High	Short	Short	Very Short
Manufacturing time	Medium	Medium	Short	Very Short
Cost (low volume)	Very High	High	High	Low
Cost (high volume)	Low	Low	Low	High

Why is RC important?

- Performance
  - Often orders of magnitude faster than microprocessors
- Low power consumption
  - A few RC devices can provide similar performance as large cluster at a fraction of the power
    - Also smaller, cheaper, etc.
- Motivating example: Novo-G
  - FPGA-based supercomputer
  - 100s of Altera/Intel FPGAs
  - 10s of Linux nodes
  - Speedups of 100,000x to 550,000x for computation biology apps (compared to 2.4 GHz Opteron)
  - Performance similar to top supercomputers
  - However, power consumption is only 8 kilowatts compared to 2-7 megawatts

RC Classification

According to Integration Level

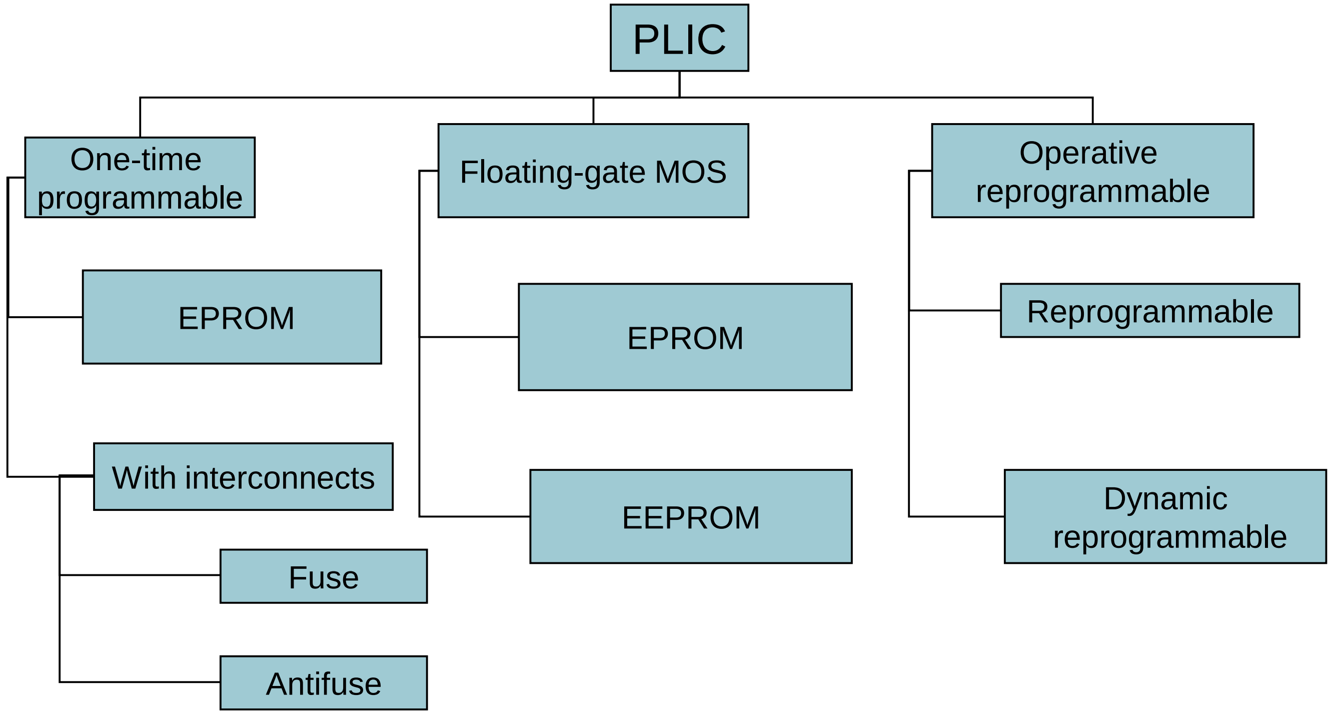


Here PLIC – Programmable Logic IC

- SoPC – System-on-a-Programmable Chip

RC Classification

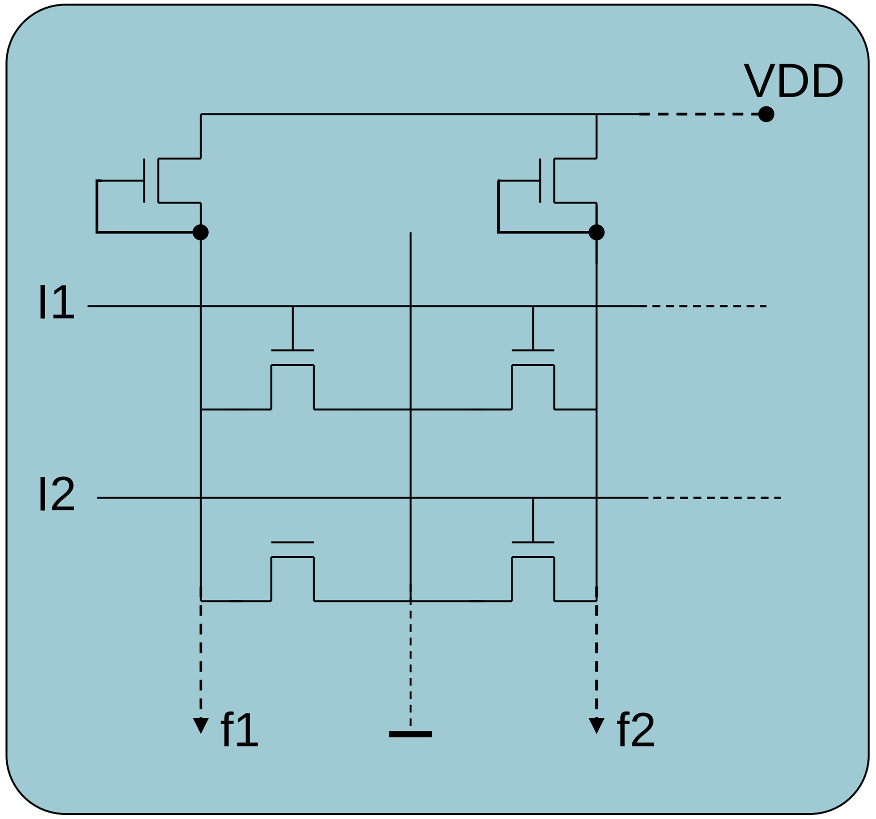
According to Reprogrammable Capacity



Implementation of Programmable Interconnects (1)

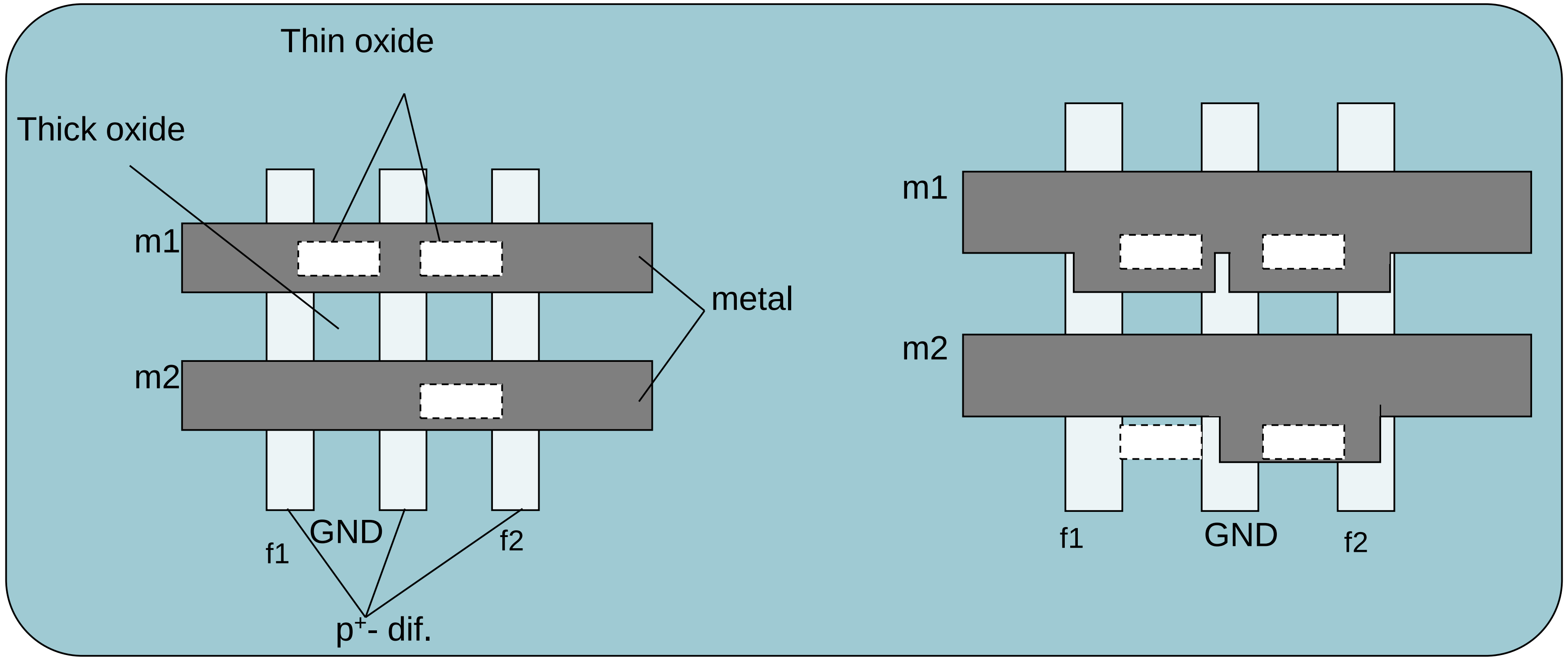
Mask Programming Interconnect

- Oxide thickness change (oxide mask changes)
- Bus width change (oxide mask changes)



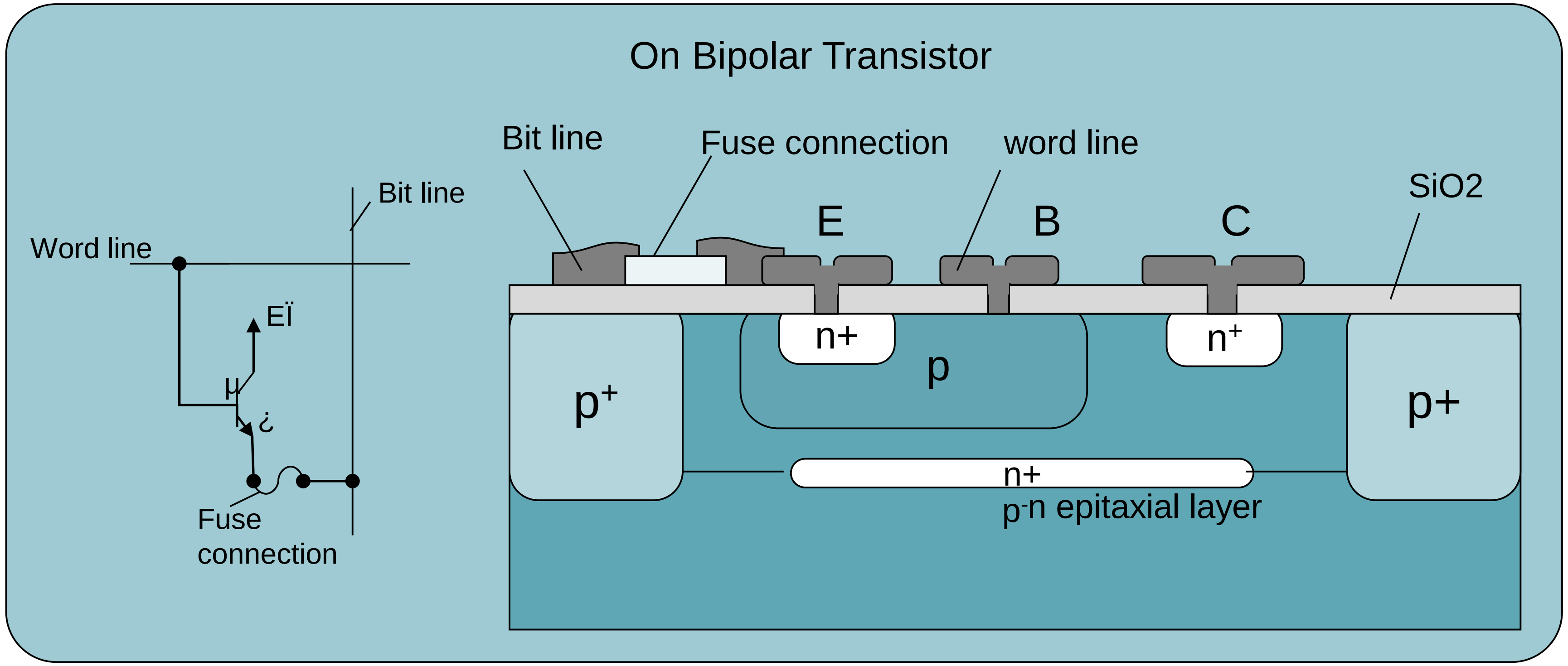
Implementation of Programmable Interconnects (2)

- Oxide Thickness Change
- Line Width Change



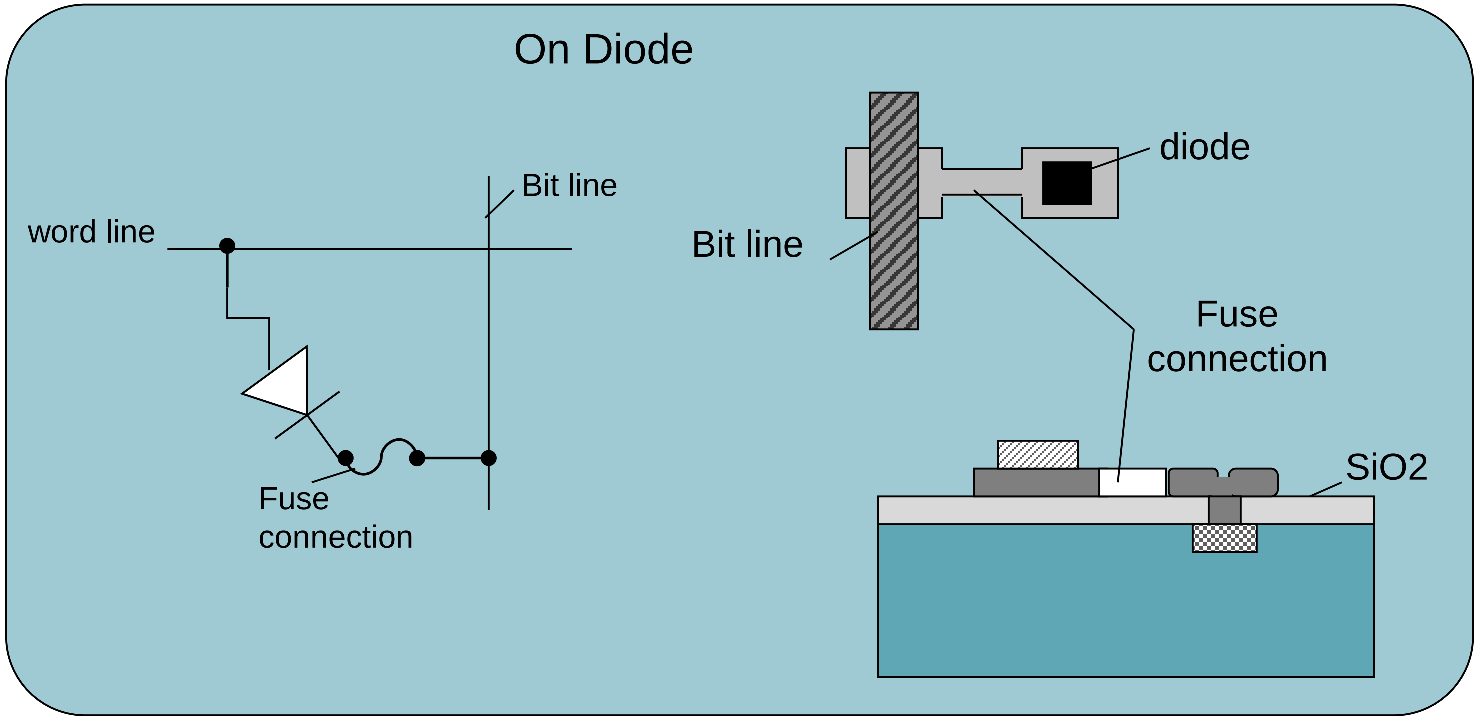
Implementation of Programmable Interconnects (3)

One Time Programmable Interconnect by the User



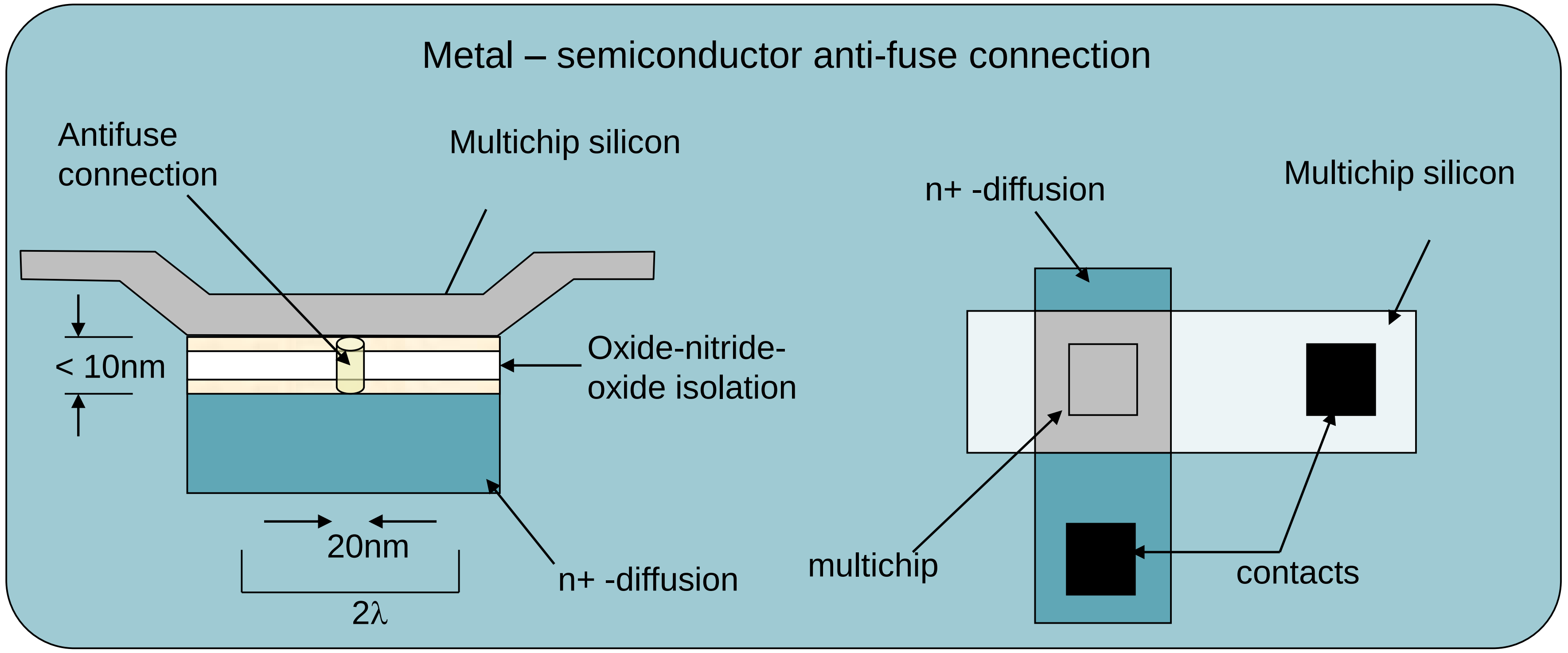
Implementation of Programmable Interconnects (4)

One Time Programmable Interconnect by the User



Implementation of Programmable Interconnects (5)

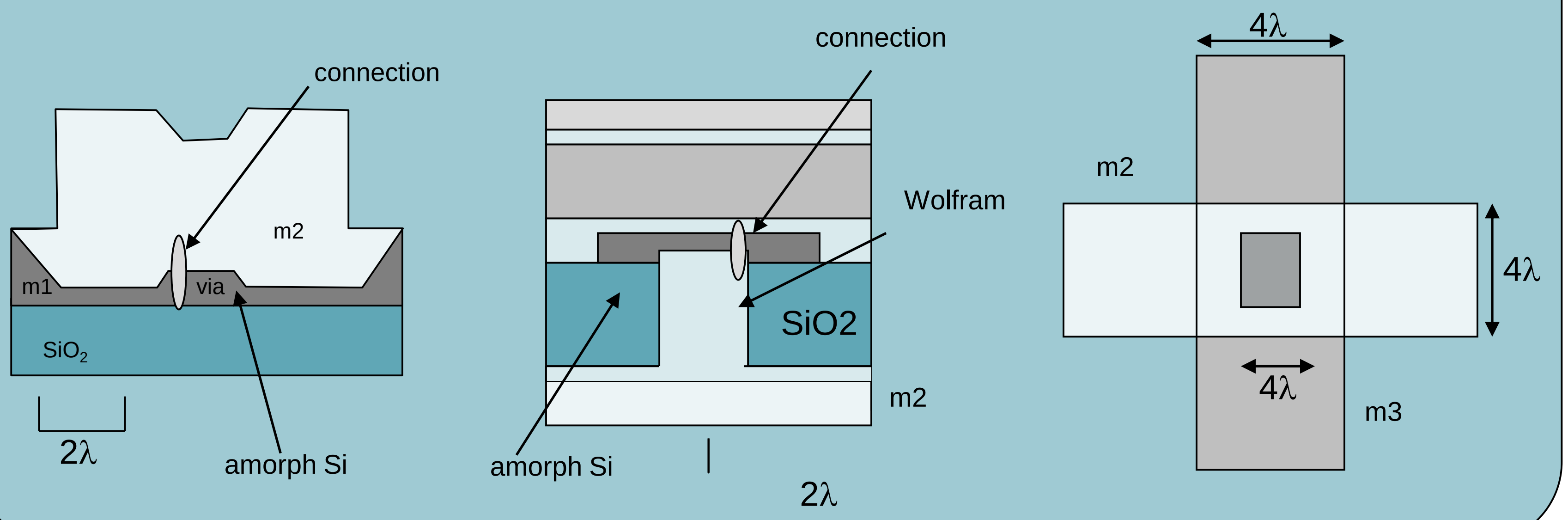
One Time Programmable Interconnect by the User



Implementation of Programmable Interconnects (6)

One Time Programmable Interconnect by the User

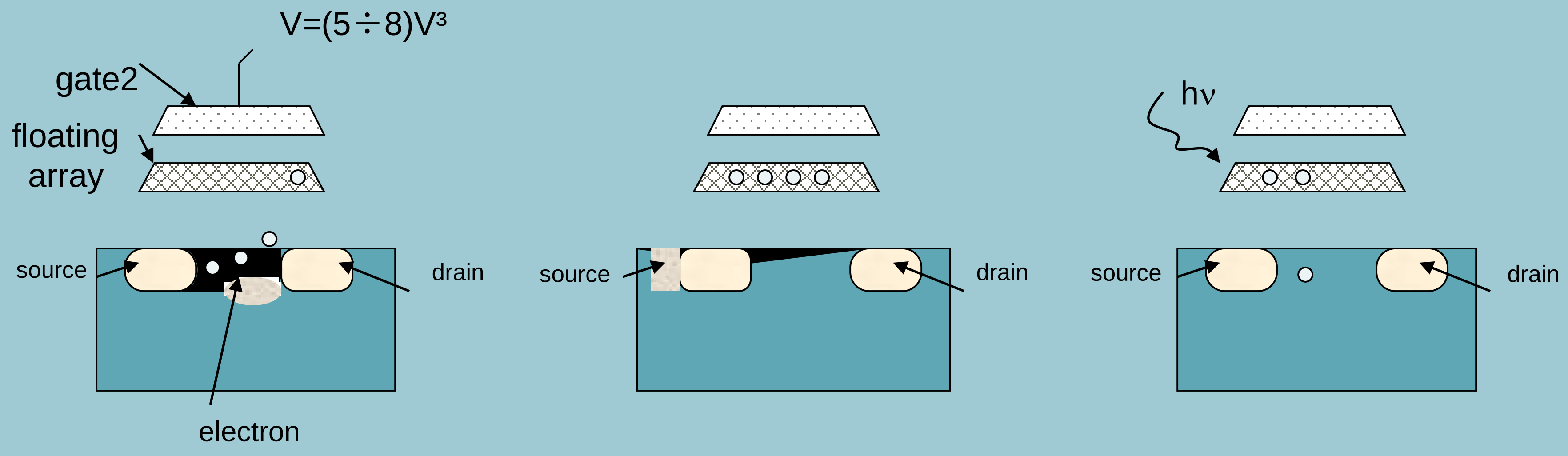
Metal – metal anti-fuse connection



Implementation of Programmable Interconnects (7)

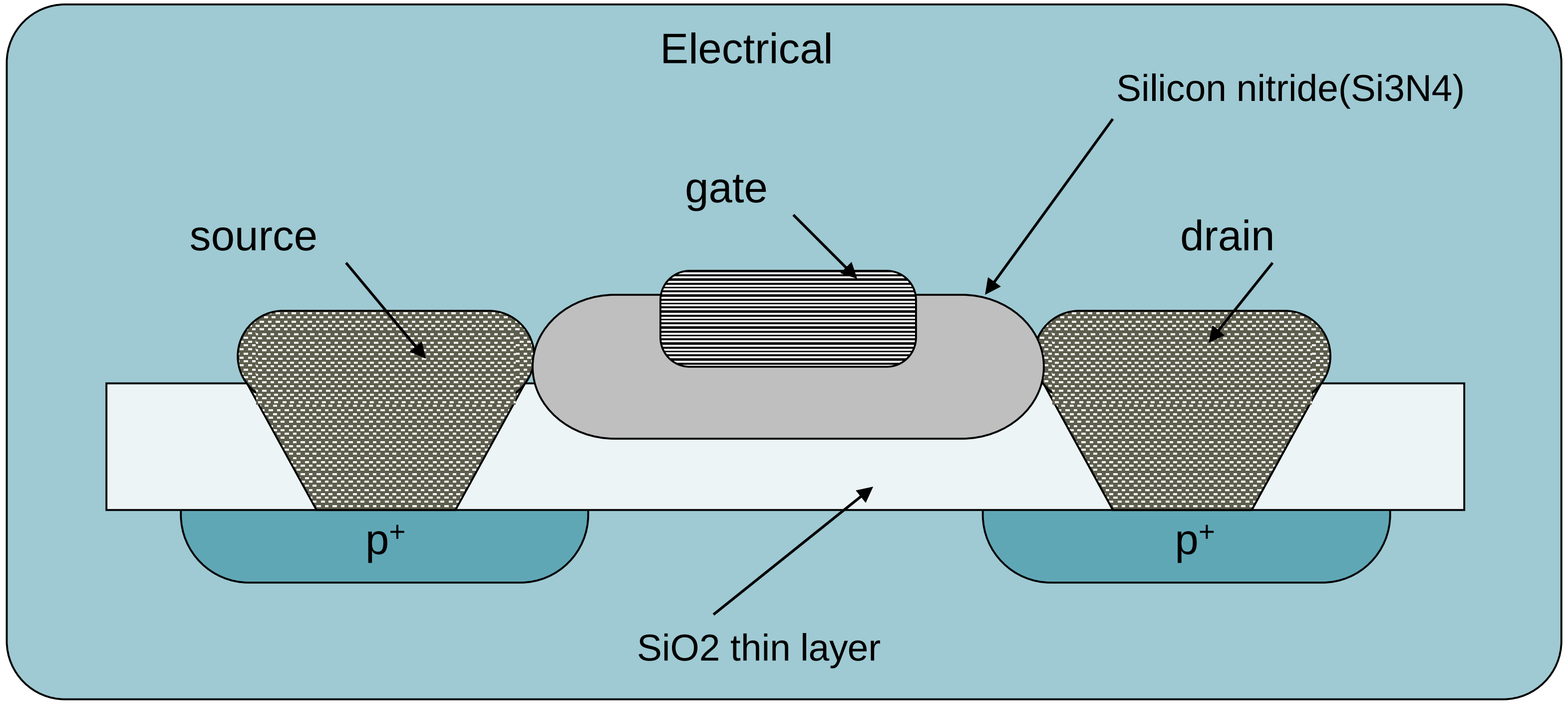
Reprogrammable Interconnect

By Ultraviolet Light

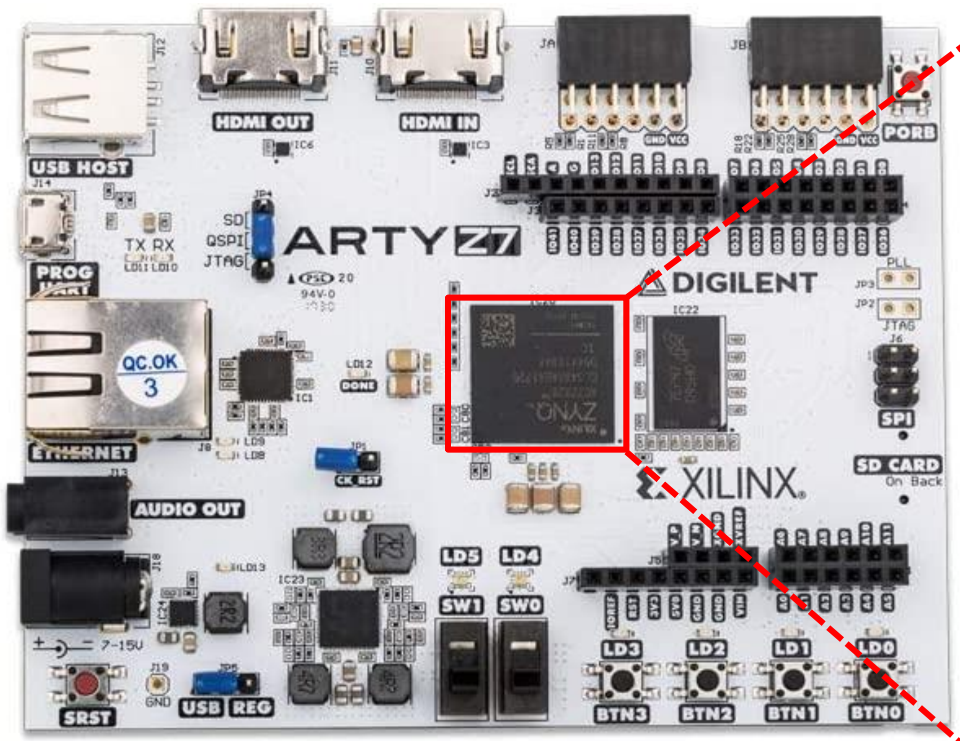


Implementation of Programmable Interconnects (8)

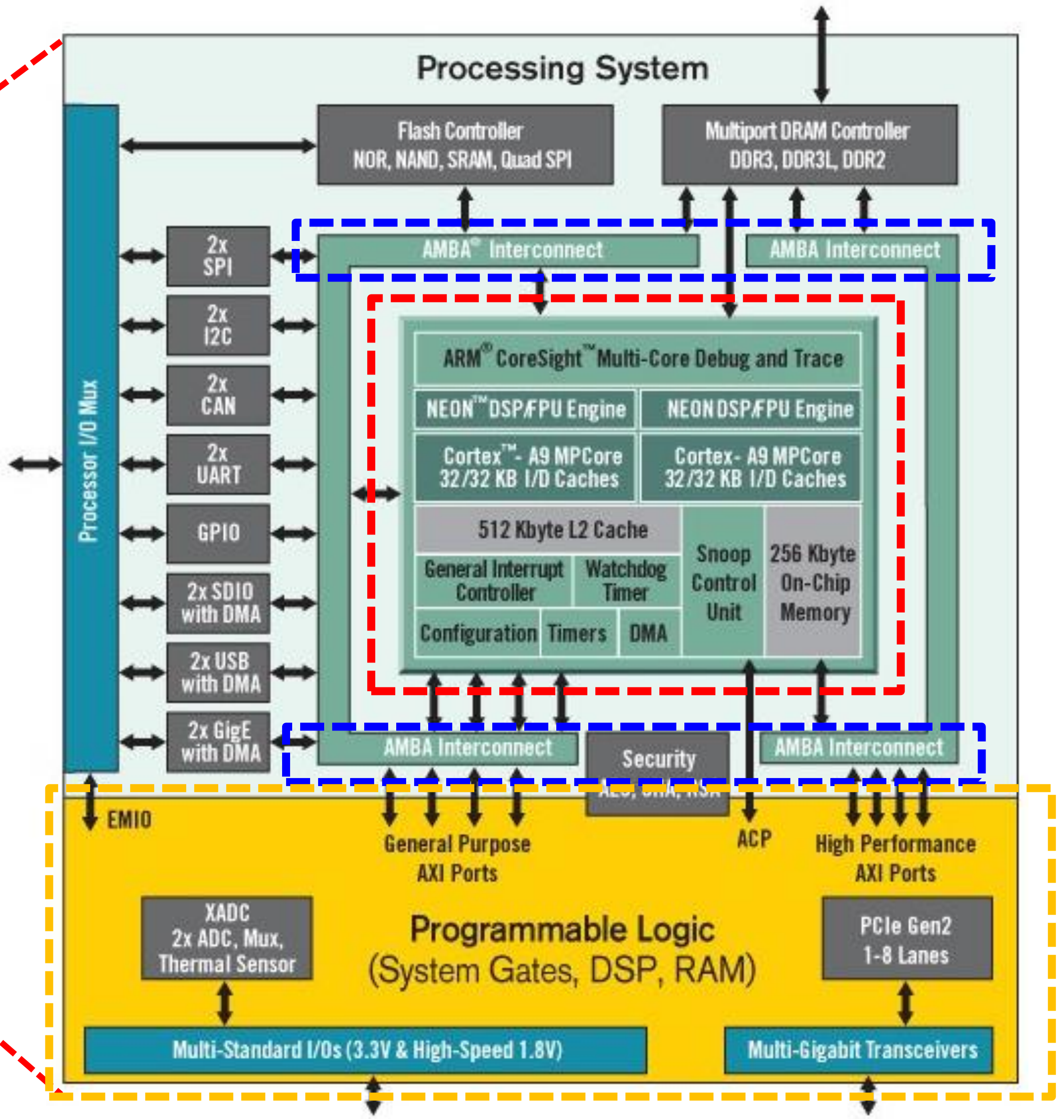
Reprogrammable Interconnect



Zynq Reconfigurable SoC



Zynq®-7000 SoC



Processing System
Flash Controller NOR, NAND, SRAM, Quad SPI
Multiport DRAM Controller DDR3, DDR3L, DDR2
Processor I/O Mux
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
AMBA® Interconnect
AMBA Interconnect
ARM® CoreSight™ Multi-Core Debug and Trace
NEON™ DSP/FPU Engine
NEON DSP/FPU Engine
Cortex™- A9 MPCore 32/32 KB I/D Caches
Cortex- A9 MPCore 32/32 KB I/D Caches
512 Kbyte L2 Cache
General Interrupt Controller
Watchdog Timer
Configuration
Timers
DMA
Snoop Control Unit
256 Kbyte On-Chip Memory
AMBA Interconnect
Security
AES, SHA, RSA
AMBA Interconnect
EMIO
General Purpose AXI Ports
ACP
High Performance AXI Ports
XADC 2x ADC, Mux, Thermal Sensor
Programmable Logic (System Gates, DSP, RAM)
PCIe Gen2 1-8 Lanes
Multi-Standard I/Os (3.3V & High-Speed 1.8V)
Multi-Gigabit Transceivers

JOINT INSTITUTE 交大密西根学院

Image: Xilinx

Major Players

Share over 60% of the market

AMD
XILINX

ALTERA®
now part of Intel

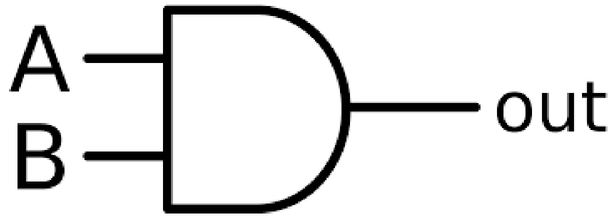
LATTICE
SEMICONDUCTOR®

ATMEL

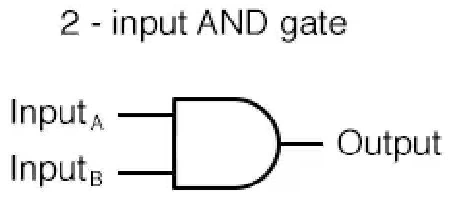
上海安路信息科技股份有限公司
SHANGHAI ANLOGIC INFOTECH CO., LTD.

FPGA Architectures

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

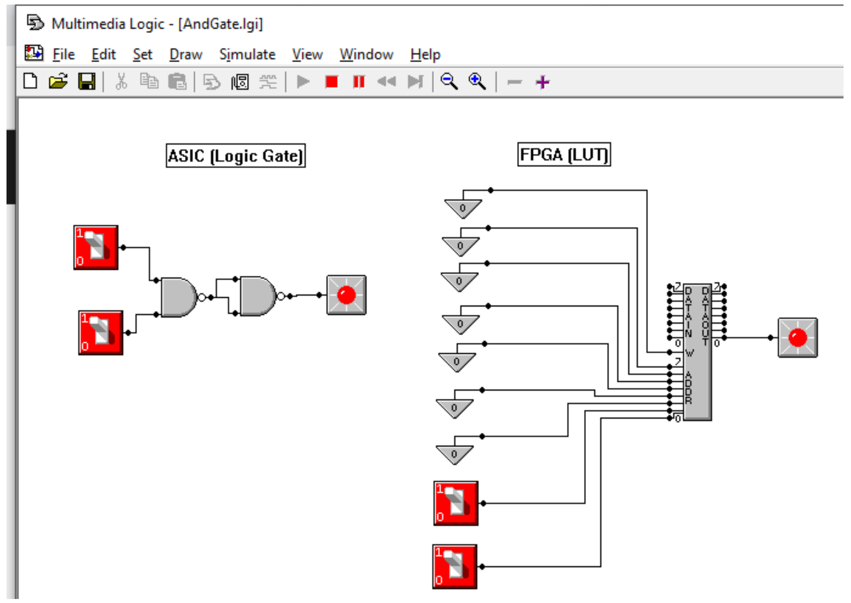


ASIC



A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

LUT

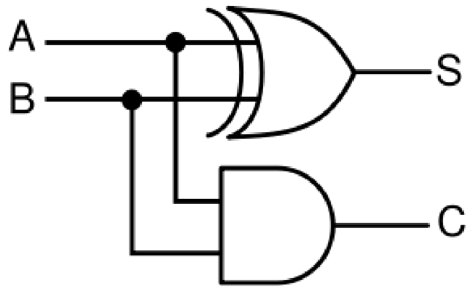


Multimedia Logic - [AndGate.lgi]
File Edit Set Draw Simulate View Window Help
ASIC (Logic Gate)
FPGA (LUT)

Implementation

FPGA Architectures

- Example: Half adder
  - Combinational logic represented by truth table
  - What kind of hardware can implement a truth table?

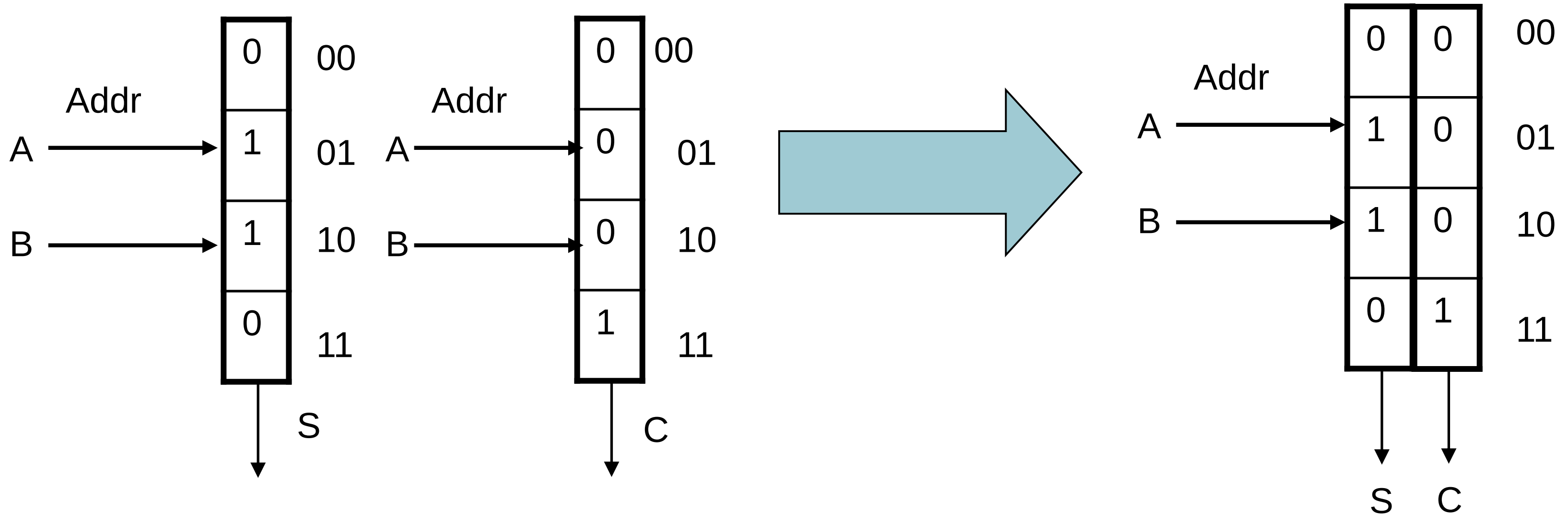


Input		Out
A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

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

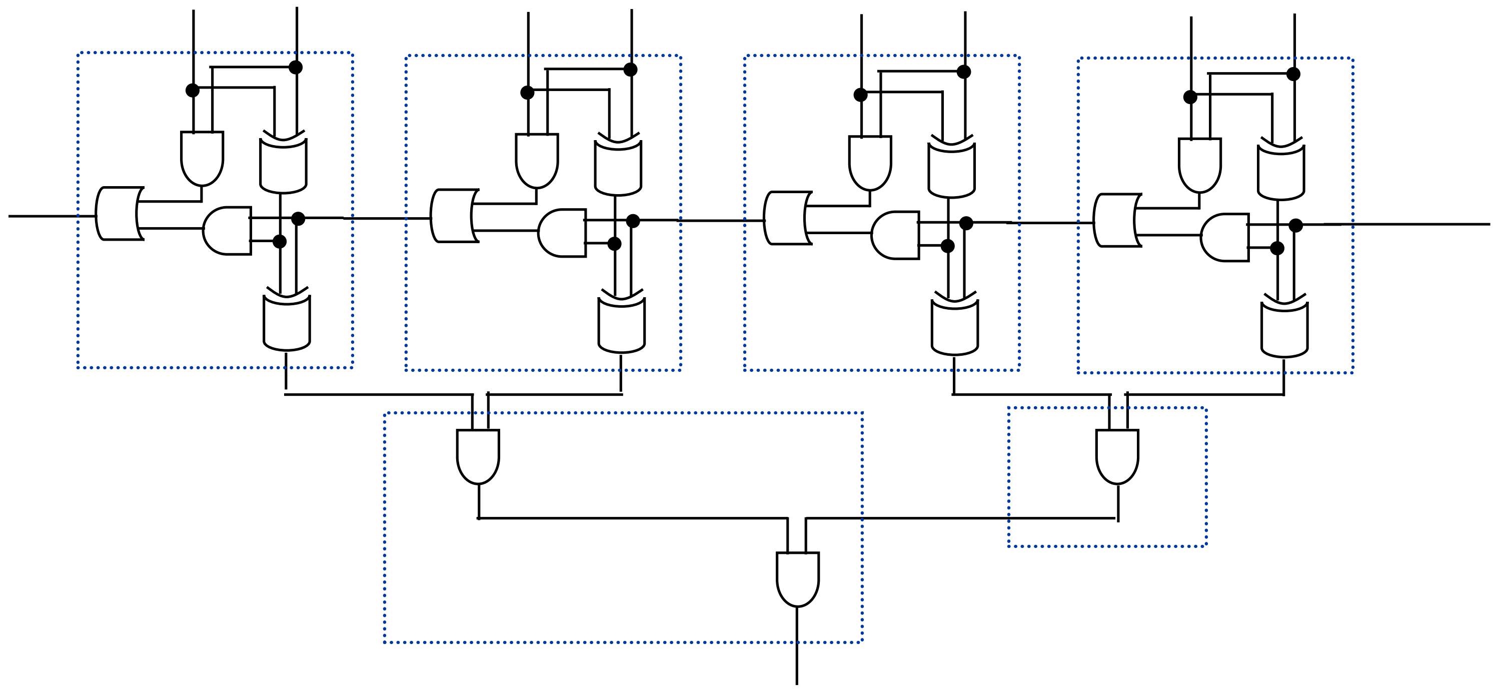
Look-up-tables (LUTs)

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

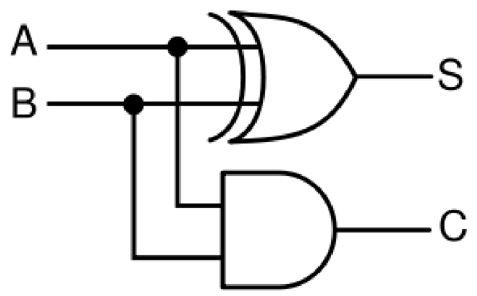


Look-up-tables (LUTs)

- 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)

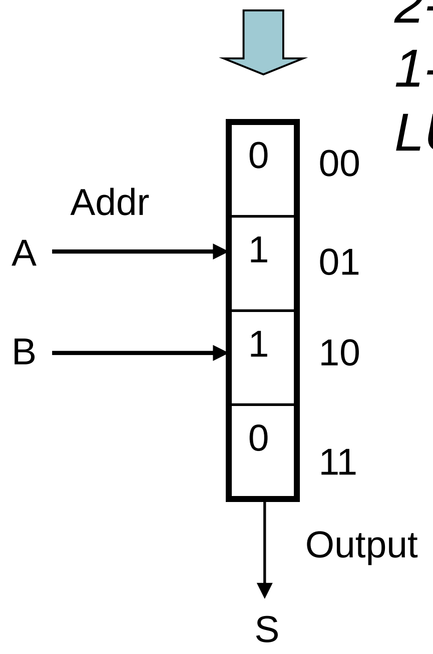
- Implement truth table in small memories (LUTs)
  - Usually SRAM

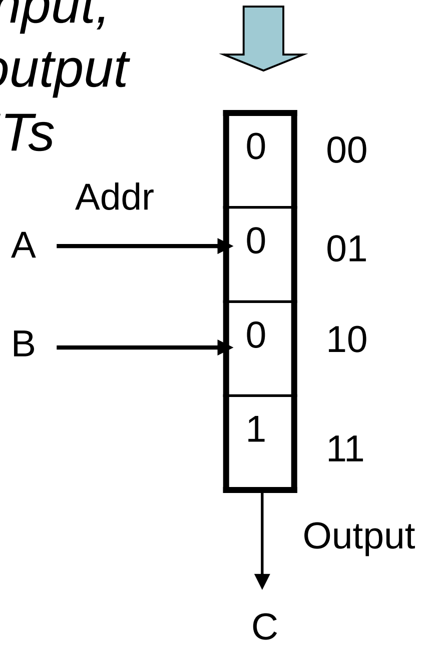


A	B	S
0	0	0
0	1	1
1	0	1
1	1	0

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

2-input, 1-output LUTs

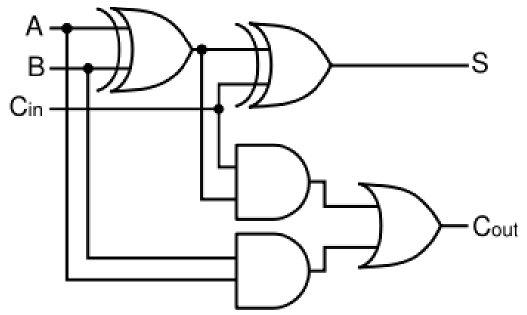




Logic inputs connect to address inputs, logic output is memory output

Look-up-tables (LUTs)

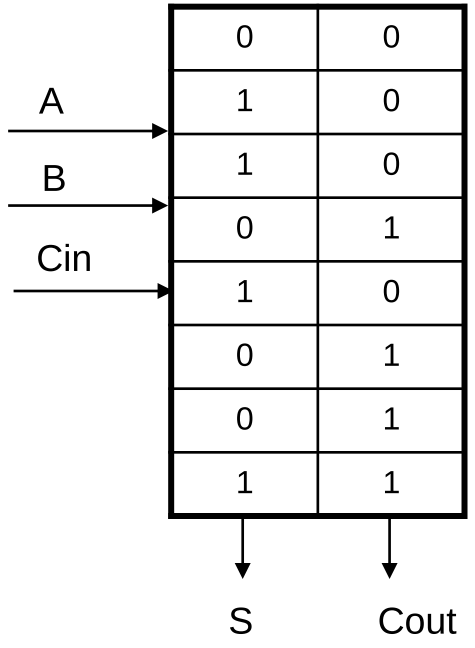
- Slightly bigger example: Full adder
  - ***Combinational logic can be implemented in a LUT with same number of inputs and outputs***
    - 3-input, 2-ouput LUT



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



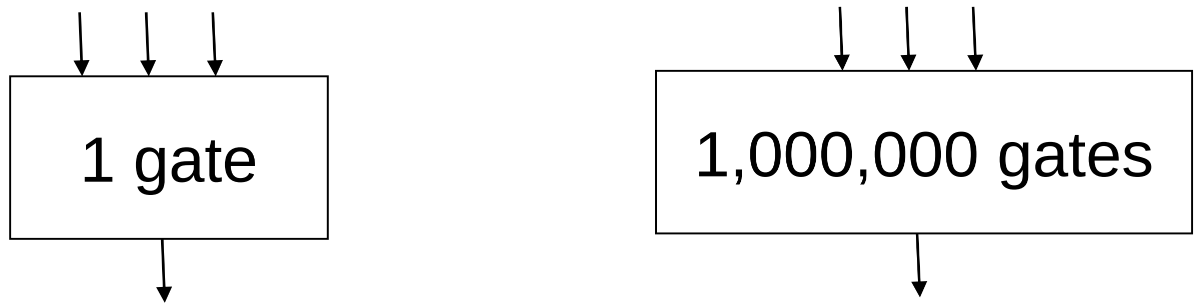
0	0
1	0
1	0
0	1
1	0
0	1
0	1
1	1

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)

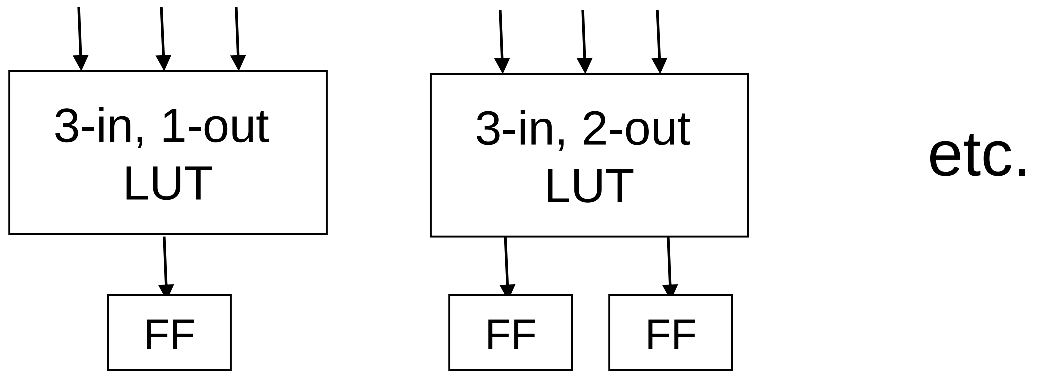
- Important Point
  - The number of gates in a circuit has no effect on the mapping into a LUT
    - All that matters is the number of inputs and outputs
    - Unfortunately, it isn't common to see large circuits with a few inputs



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

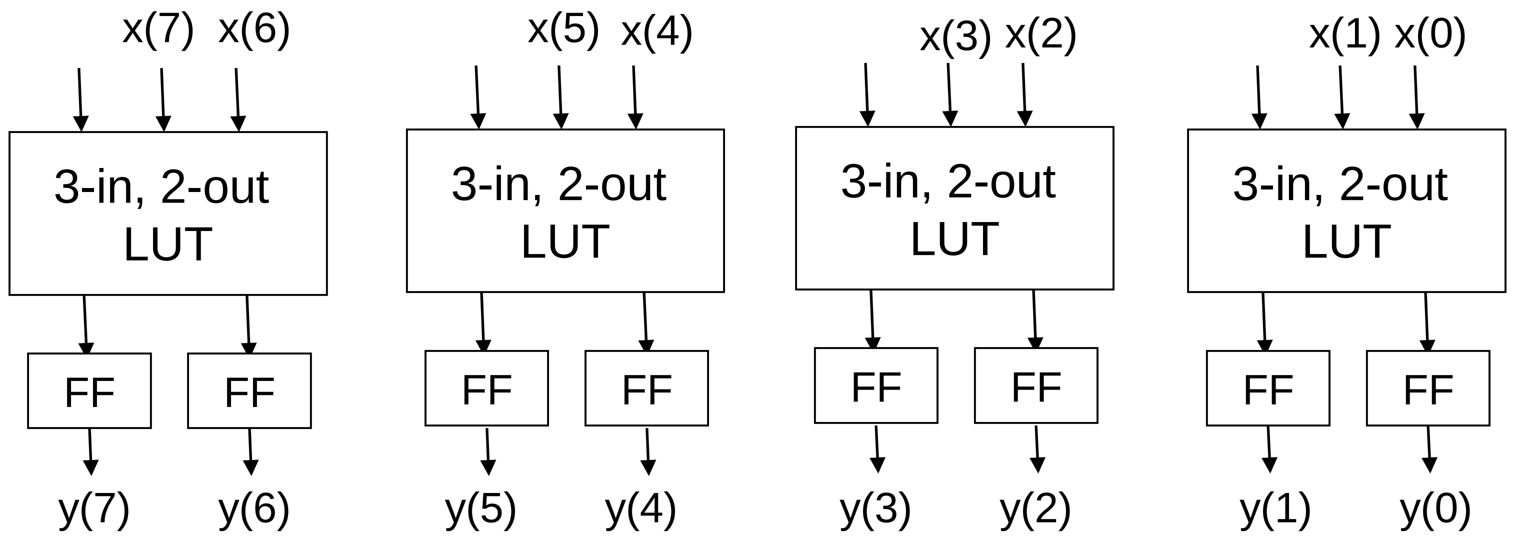
Sequential Logic

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



Sequential Logic

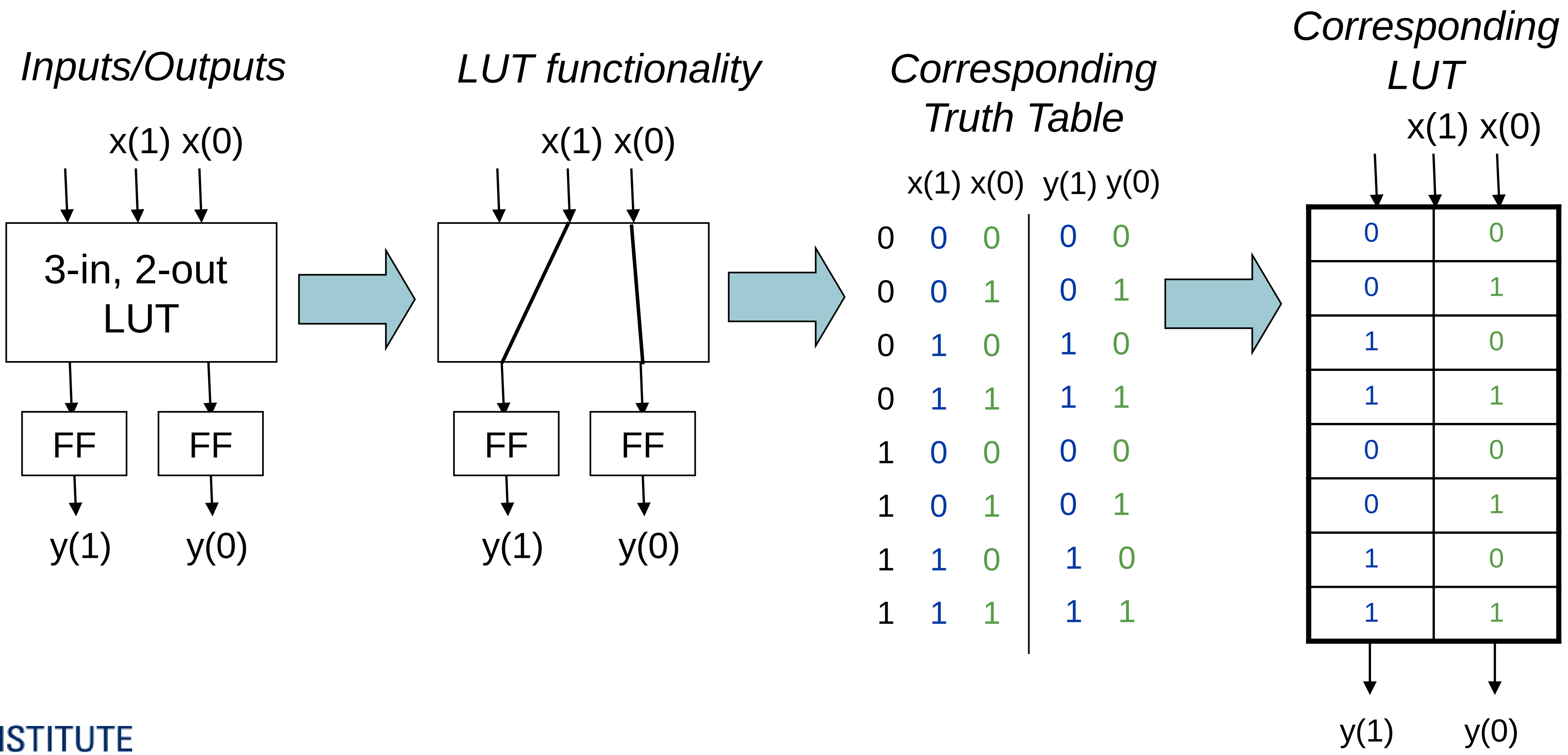
- Example: 8-bit register using 3-input, 2-output LUTs
  - Input: x, Output: y



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

Sequential Logic

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

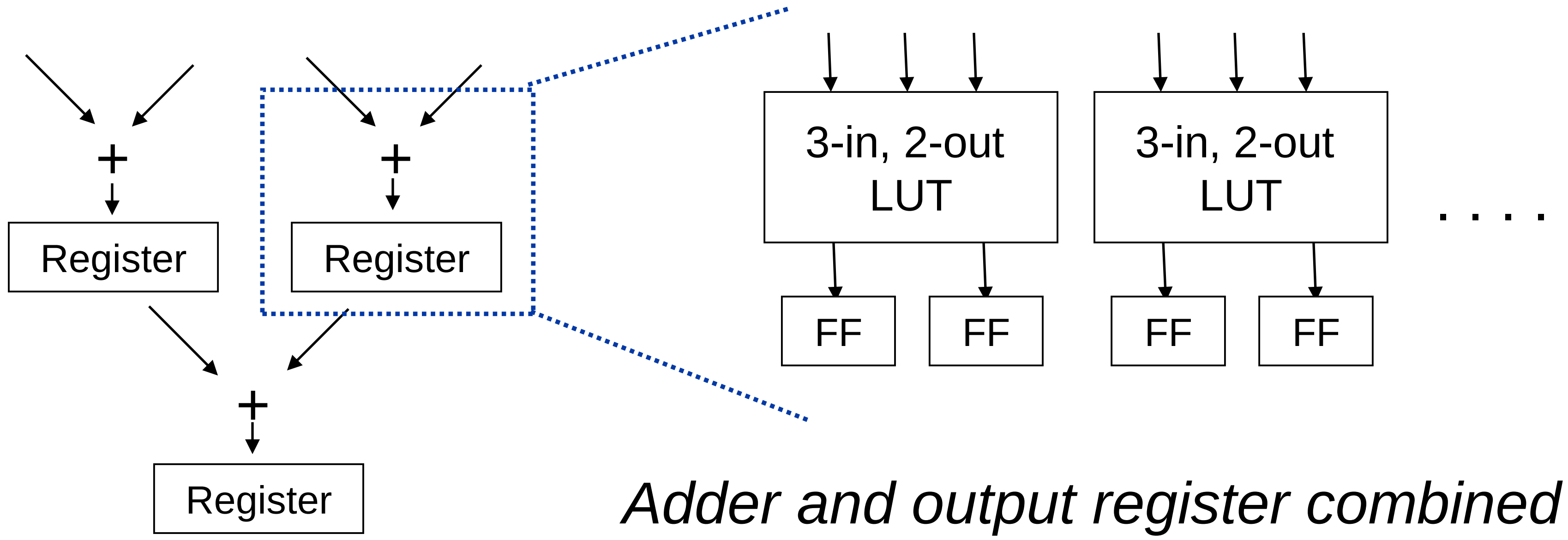


	x(1)	x(0)	y(1)	y(0)
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

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



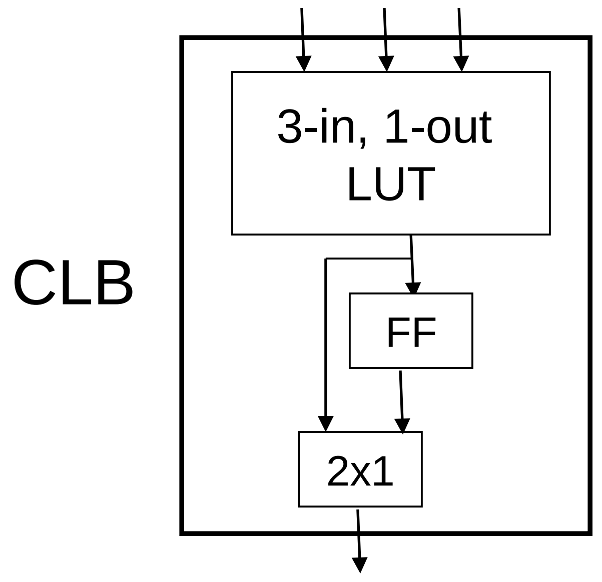
Adder and output register combined – not a separate LUT for each

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)

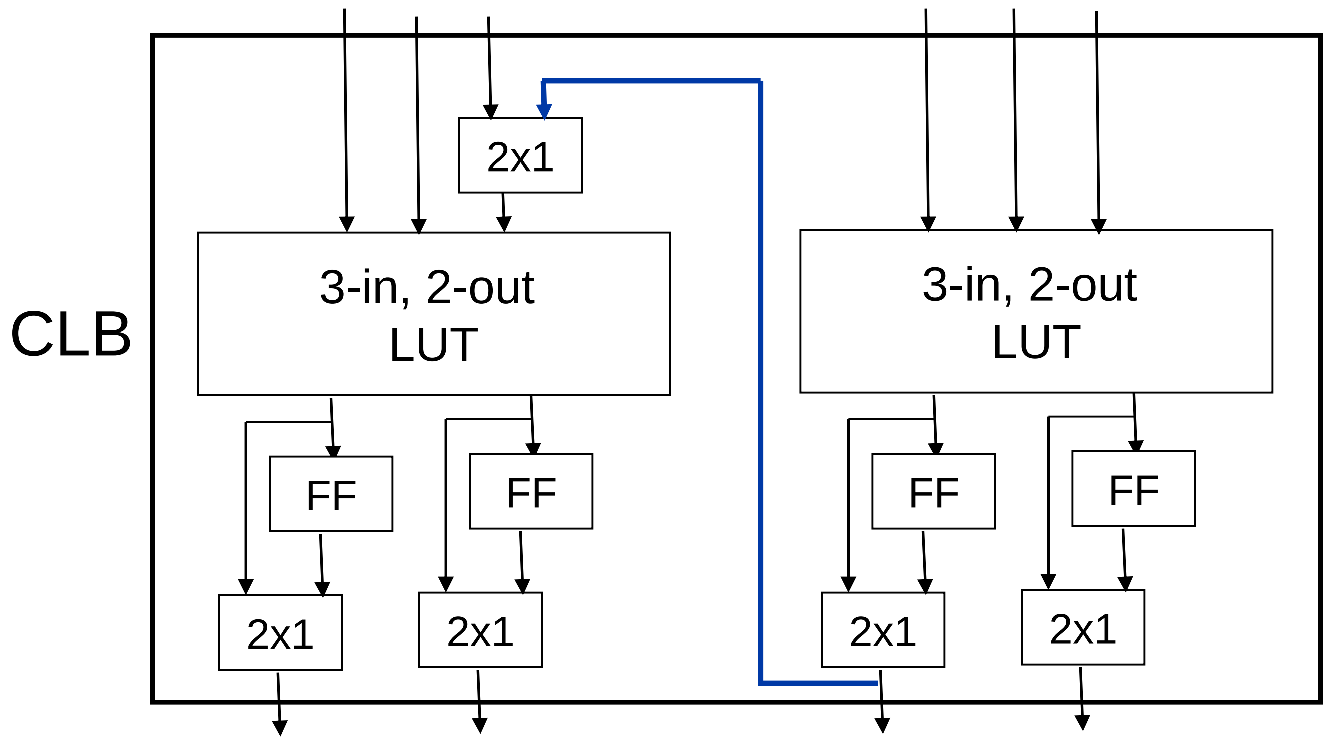
Configurable Logic Blocks (CLBs)

- 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)



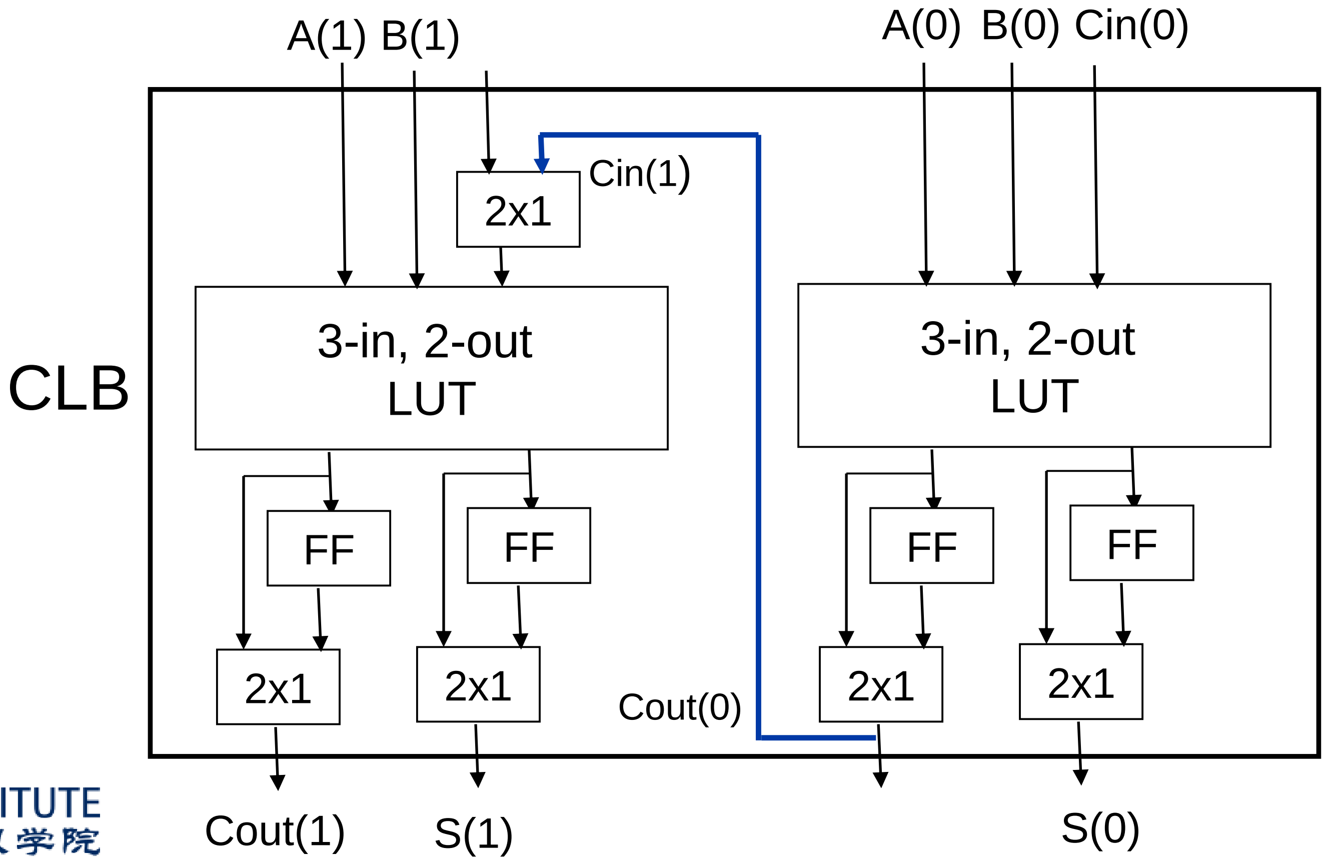
Configurable Logic Blocks (CLBs)

- 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

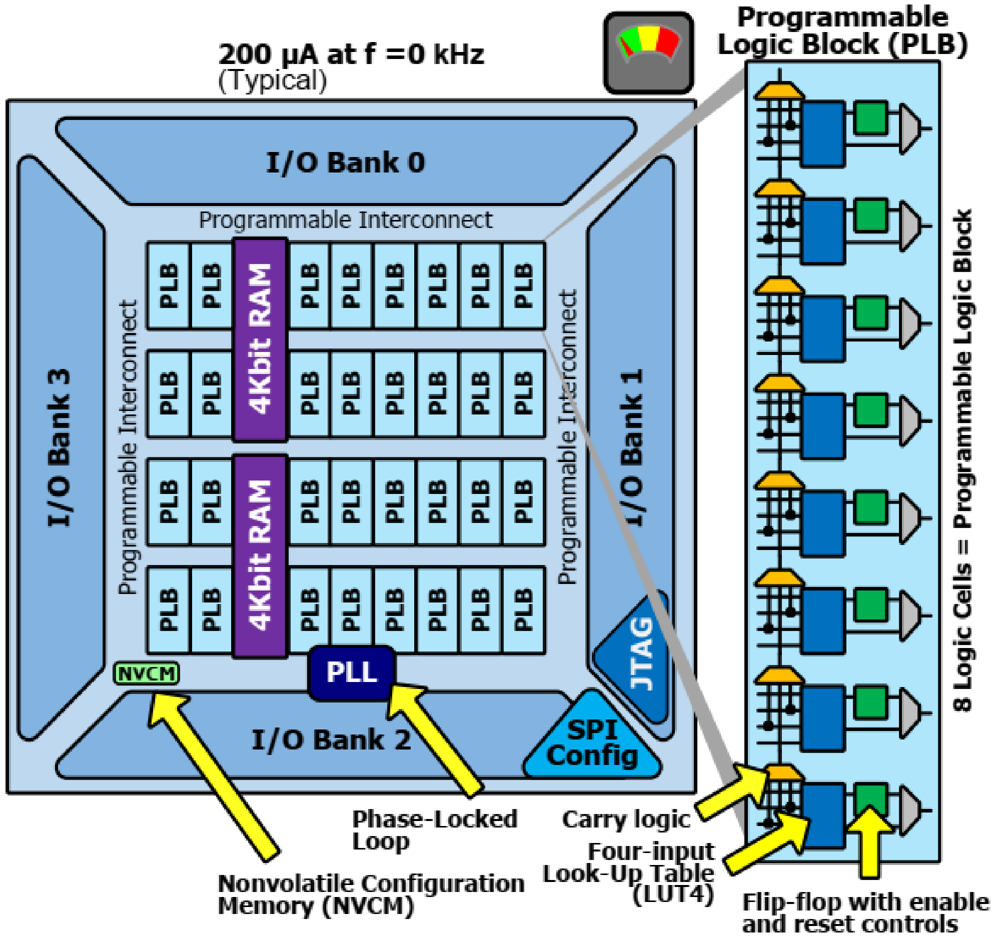
CLB Examples

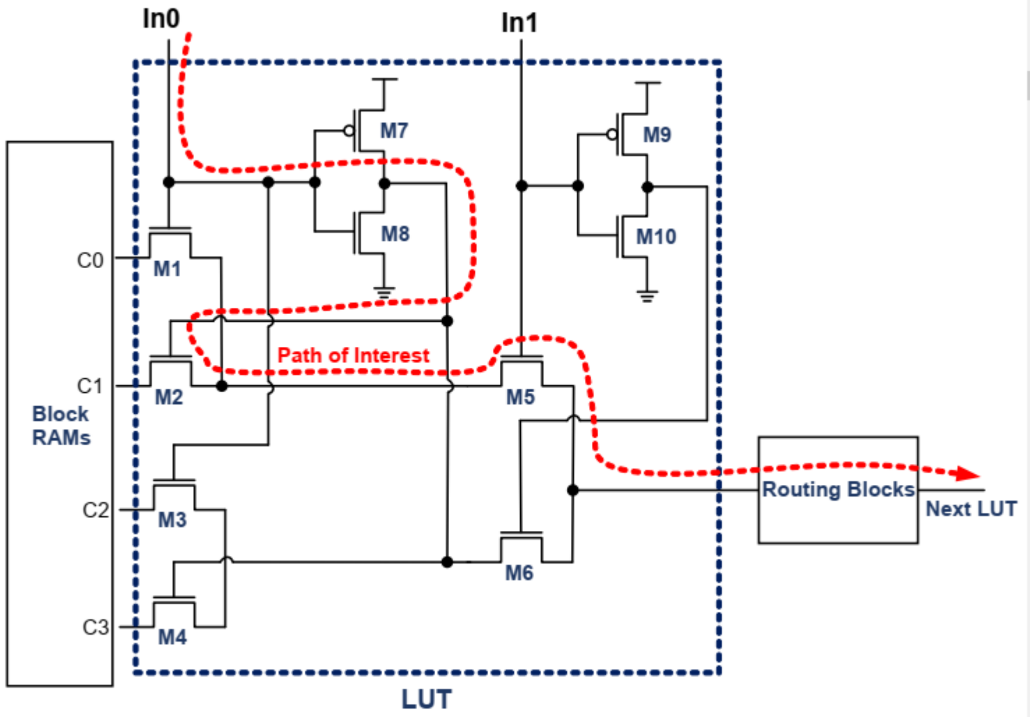
- Virtex 4 CLB
  - http://www.xilinx.com/support/documentation/user_guides/ug070.pdf (pg. 183)
- Altera Stratix 5
  - http://www.altera.com/literature/hb/stratix-v/stratix5_handbook.pdf (pg. 10)
- Virtex 7 CLB
  - http://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf (pg. 13)
  - http://www.xilinx.com/csi/training/7_series_CLB_architecture.htm

CLB Examples

- Intel/Altera Arria 10 CLB (ALM)
  - https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/arria-10/a10_overview.pdf (pg. 18)
- Intel/Altera Stratix 10
  - https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/s10-overview.pdf (pg. 24)
- Xilinx UltraSacle
  - https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf (pg.36)

An example





2input, 1output LUT

courtesy of Lattice Semiconductor

What Else?

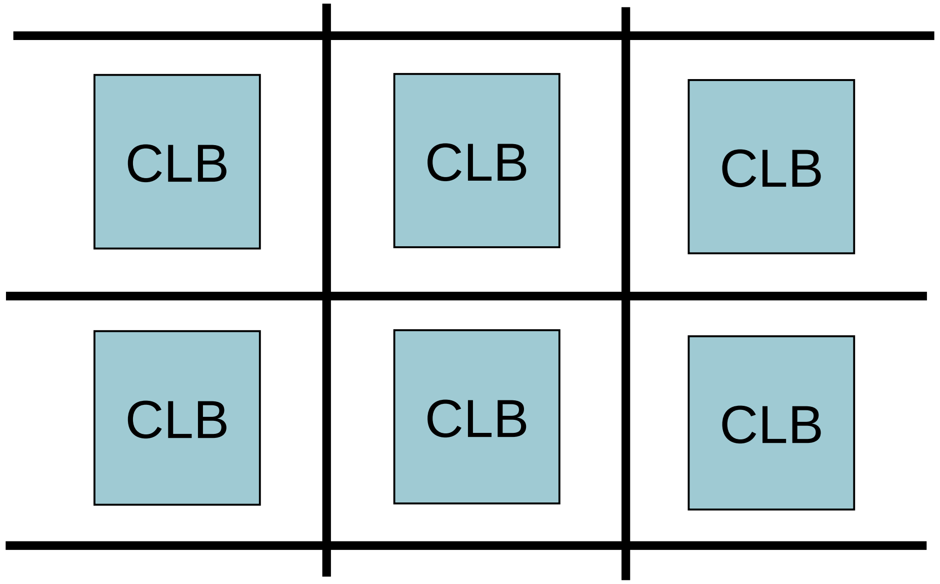
- Basic building block is CLB
  - Can implement combinational+sequential logic
  - All circuits consist of combinational and sequential logic
- So what else is needed?

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

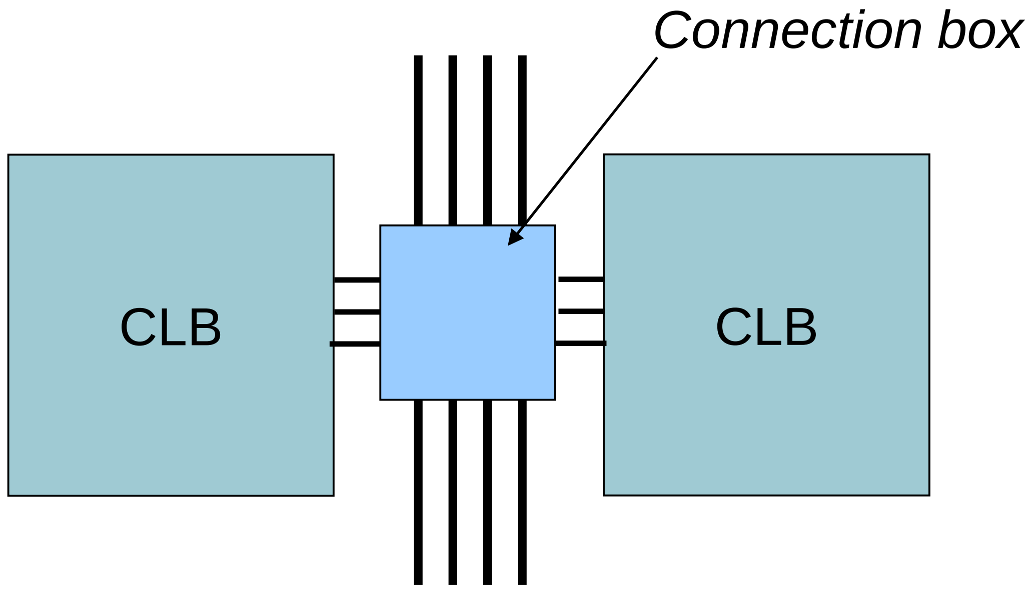
Reconfigurable Interconnect

- 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
  - Remind of NoC!



Reconfigurable Interconnect

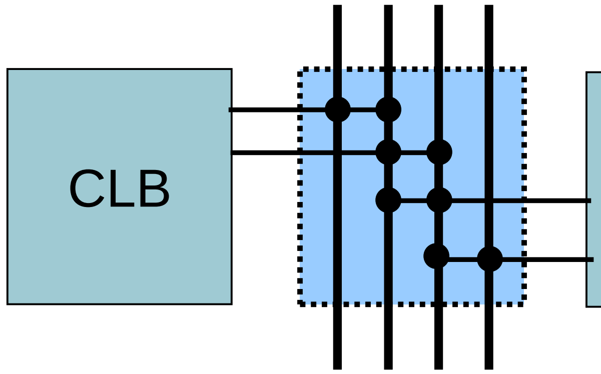
- Problem 3: How to connect CLB to wires?
- Solution: Connection box
  - Device that allows inputs and outputs of CLB to connect to different wires



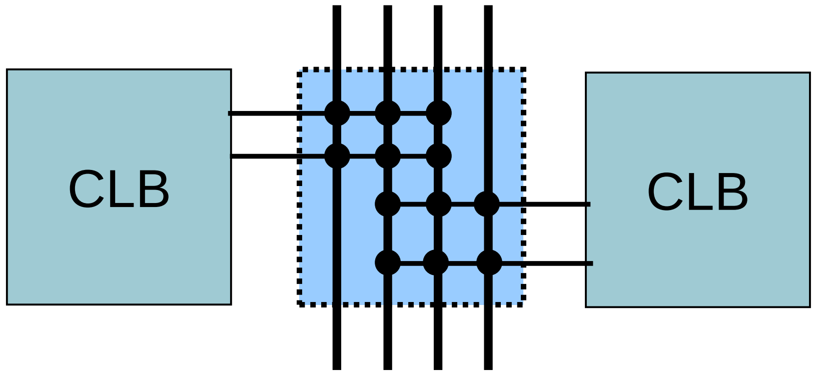
Reconfigurable Interconnect

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

Flexibility = 2



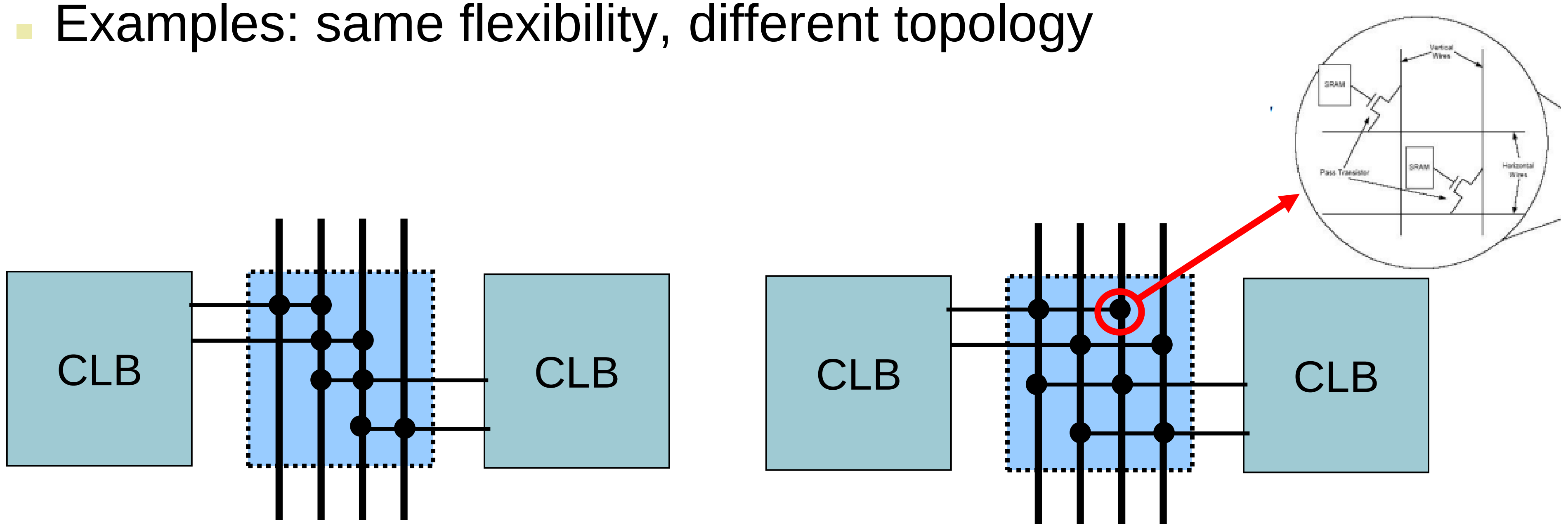
Flexibility = 3



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



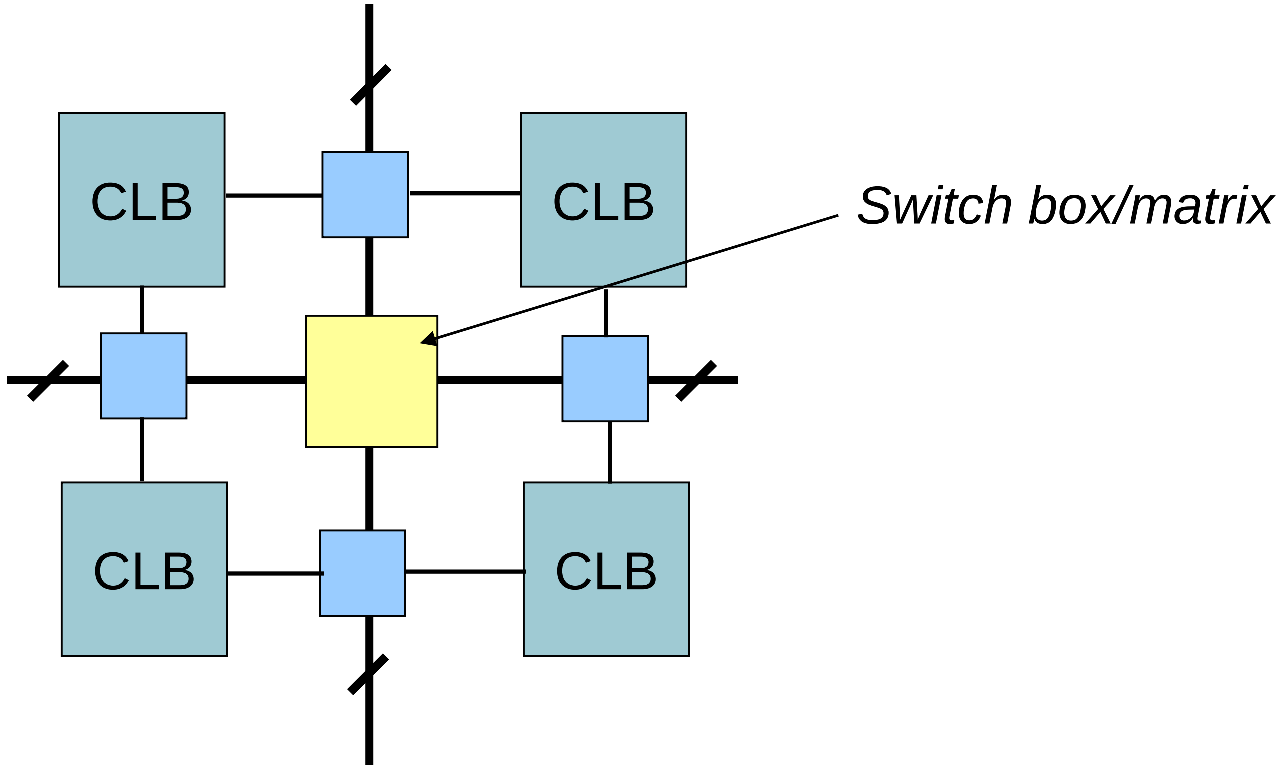
*Dots represent **possible** connections

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?

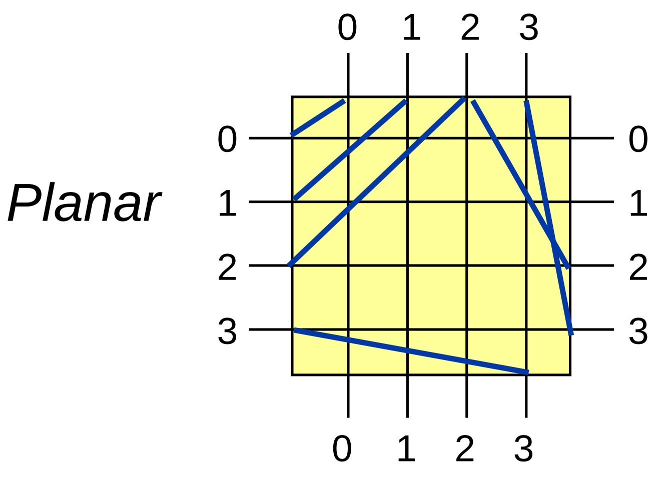
Reconfigurable Interconnect

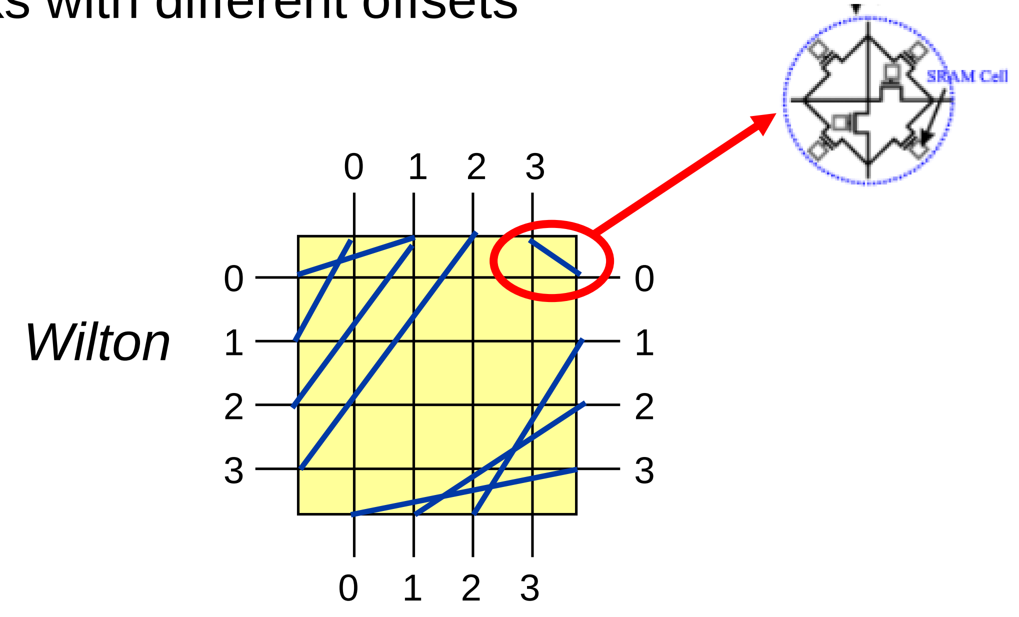
- Solution: Switch boxes, switch matrices
  - Connects horizontal and vertical routing channels



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 tracks with same id/offset (e.g. 0 to 0, 1 to 1, etc.)
    - Wilton switch box: connects tracks with different offsets

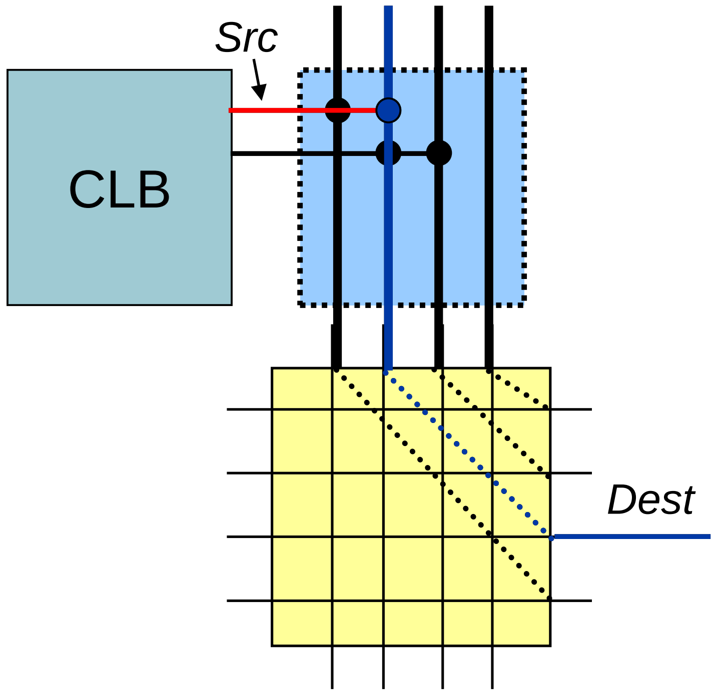


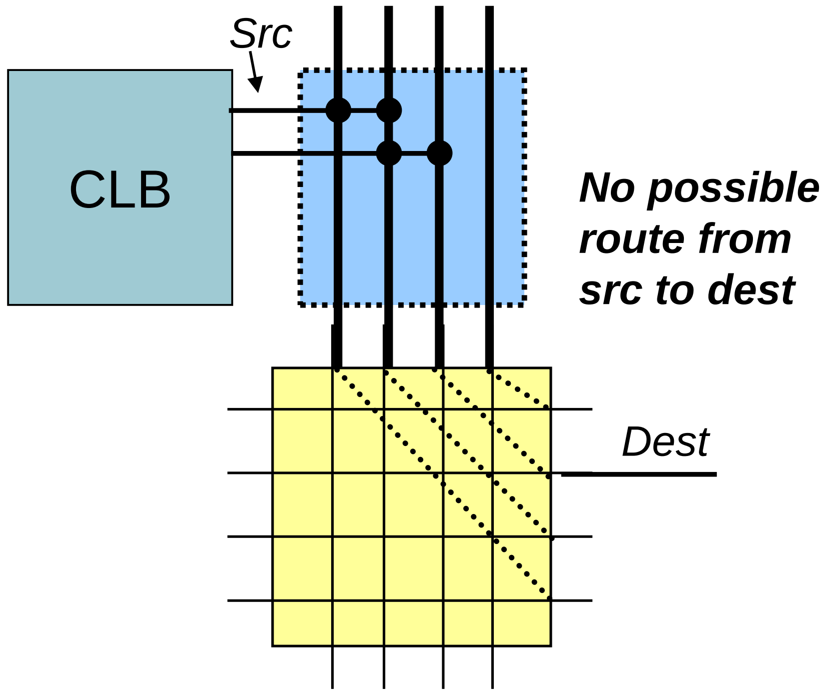


**Not all possible connections shown*

Reconfigurable Interconnect

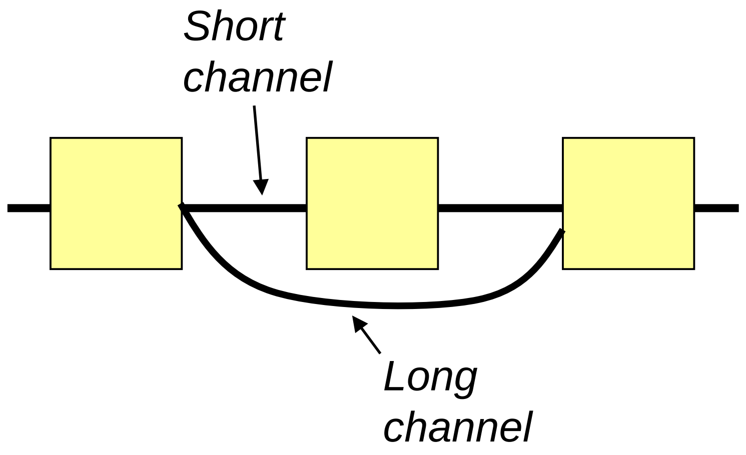
- Why do flexiblity and topology matter?
  - Routability: a measure of the number of circuits that can be routed
    - Higher flexibility = better routability
    - Wilton switch box topology = better routability





Reconfigurable Interconnect

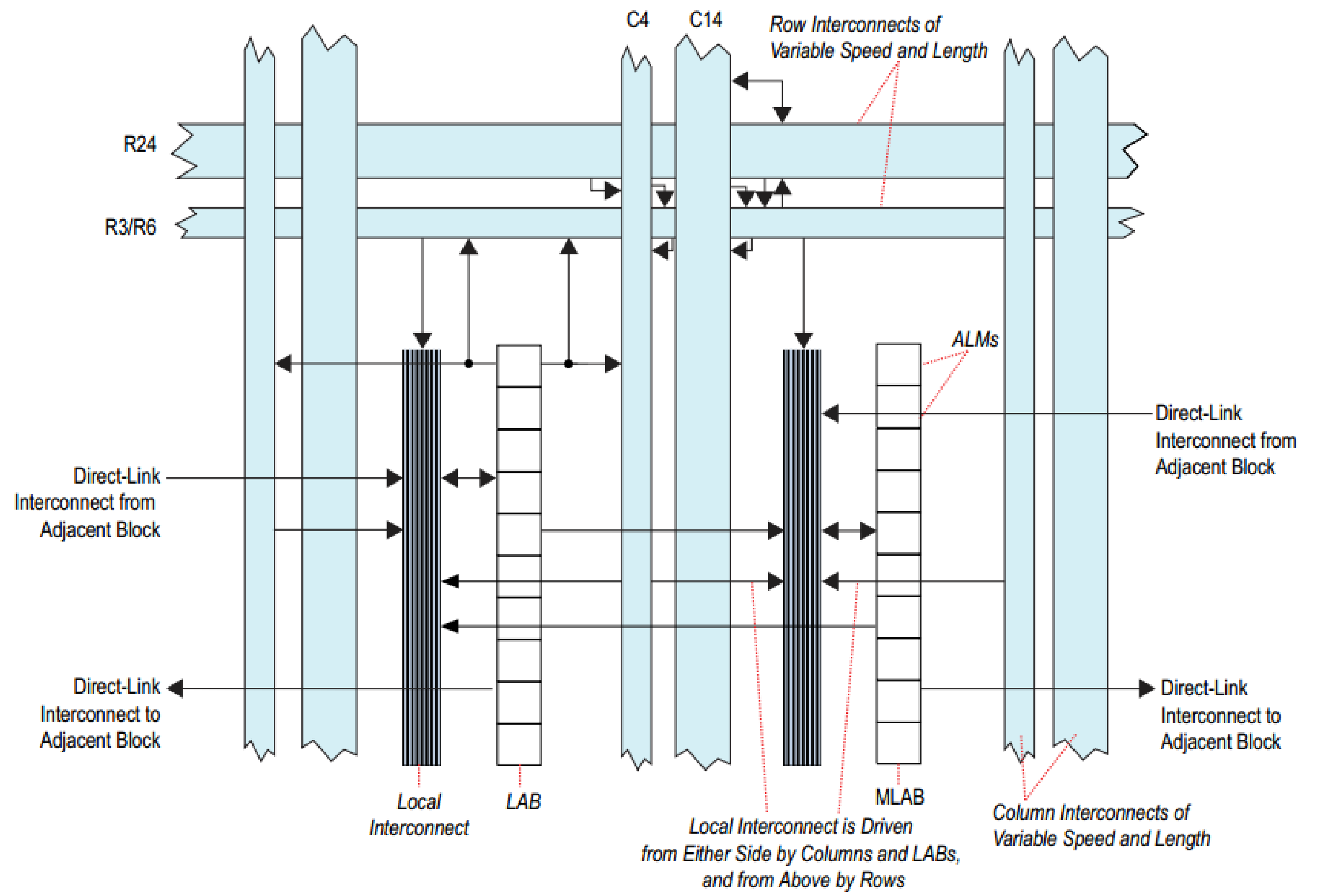
- Switch boxes
  - Short channels
    - Useful for connecting adjacent CLBs
  - Long channels
    - Useful for connecting CLBs that are separated
    - Allows for reduced routing delay for non-adjacent CLBs



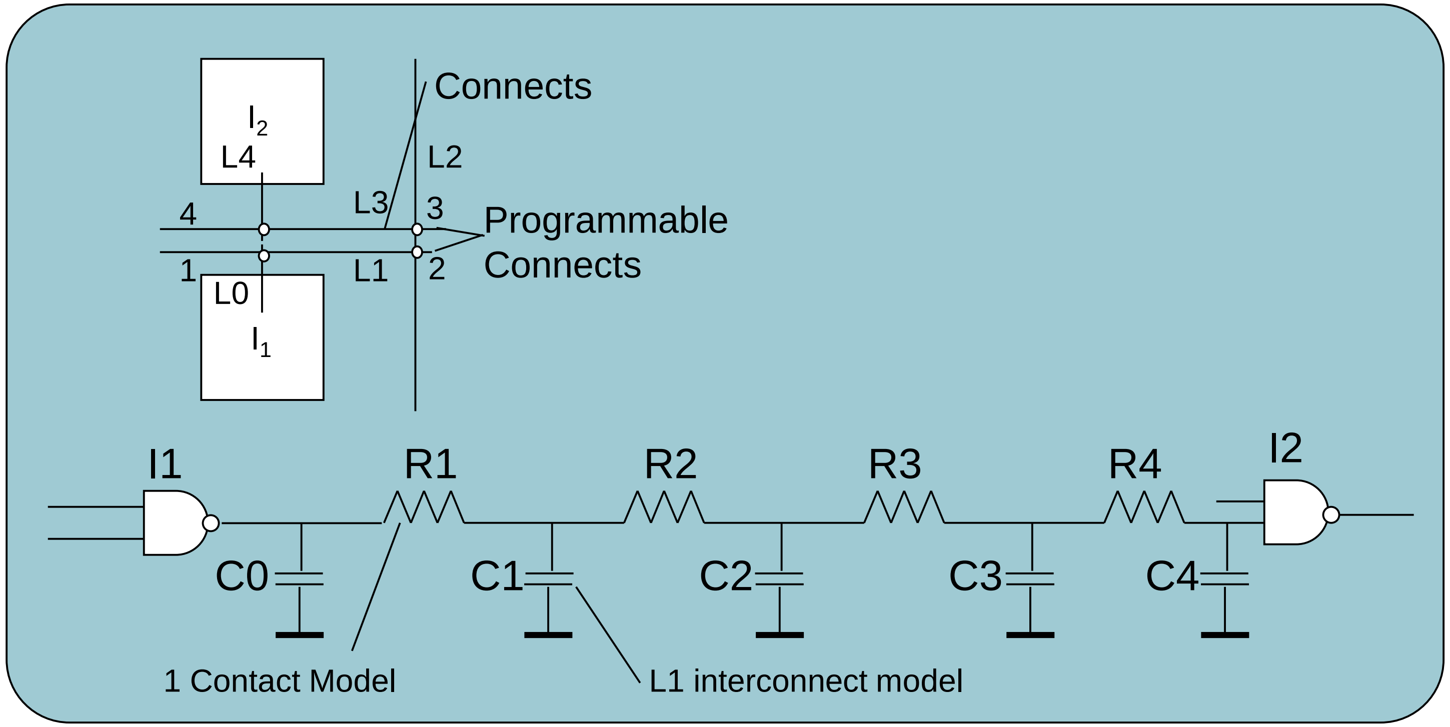
Interconnect Example

- Altera provides long tracks of length 3, 4, 6, 14, 24 along with local interconnect (short tracks)
- Image from Stratix V handbook.   LAB = CLB, ALM = LUT

This figure shows an overview of the Stratix V LAB and MLAB structure with the LAB interconnects.


C4
C14
Row Interconnects of Variable Speed and Length
R24
R3/R6
ALMs
Direct-Link Interconnect from Adjacent Block
Direct-Link Interconnect from Adjacent Block
Direct-Link Interconnect to Adjacent Block
Direct-Link Interconnect to Adjacent Block
Local Interconnect
LAB
MLAB
Local Interconnect is Driven from Either Side by Columns and LABs, and from Above by Rows
Column Interconnects of Variable Speed and Length

Delays in Reconfigurable Interconnects (1)

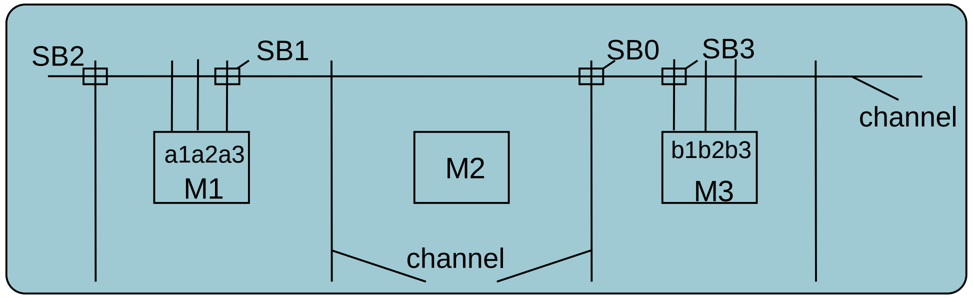


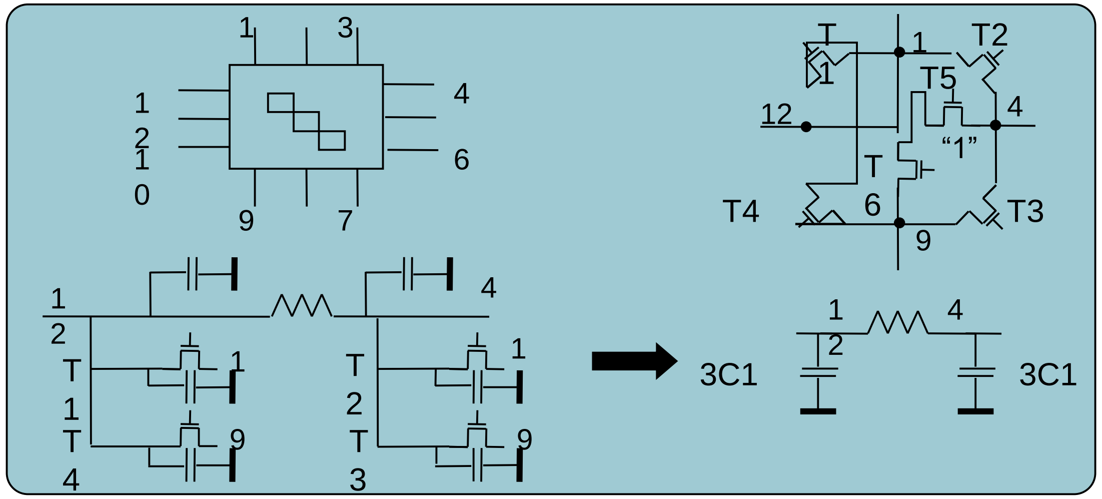
From the above figure:

τ=R1·C1+(R1+R2) · C2+(R1+R2+R3) · C3+(R1+R2+R3+R4) · C4

If R=R1=R2=R3=R4 & C=C1=C2=C3=C4, then τ=R · C+2R · C+3R · C+4R · C=10R · C

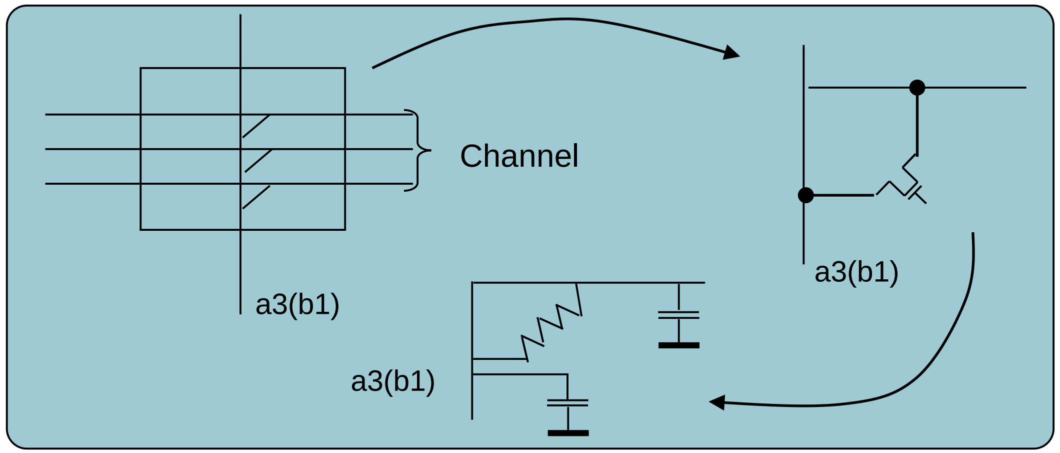
Delays in Reconfigurable Interconnects (2)



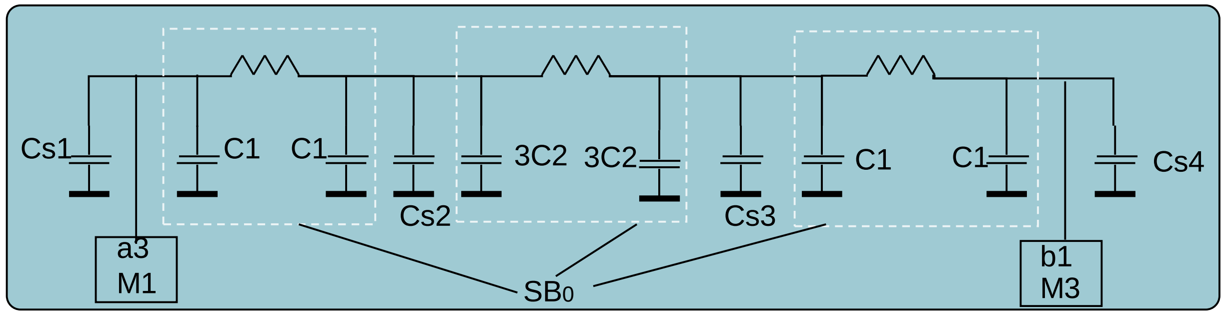


Delays in Reconfigurable Interconnects (3)

Switching block SB1, SB2



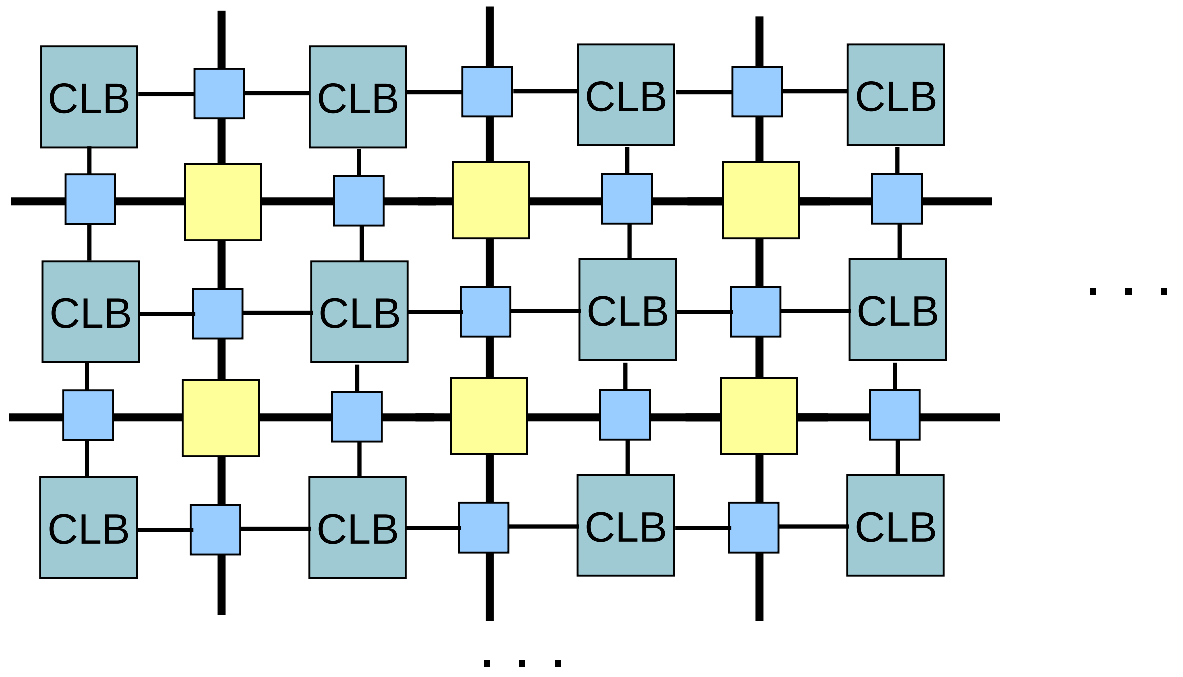
General equivalent circuit



From the above figure: τ=R5C+2R5C+3R2C=21RC

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?

FPGA Fabrics

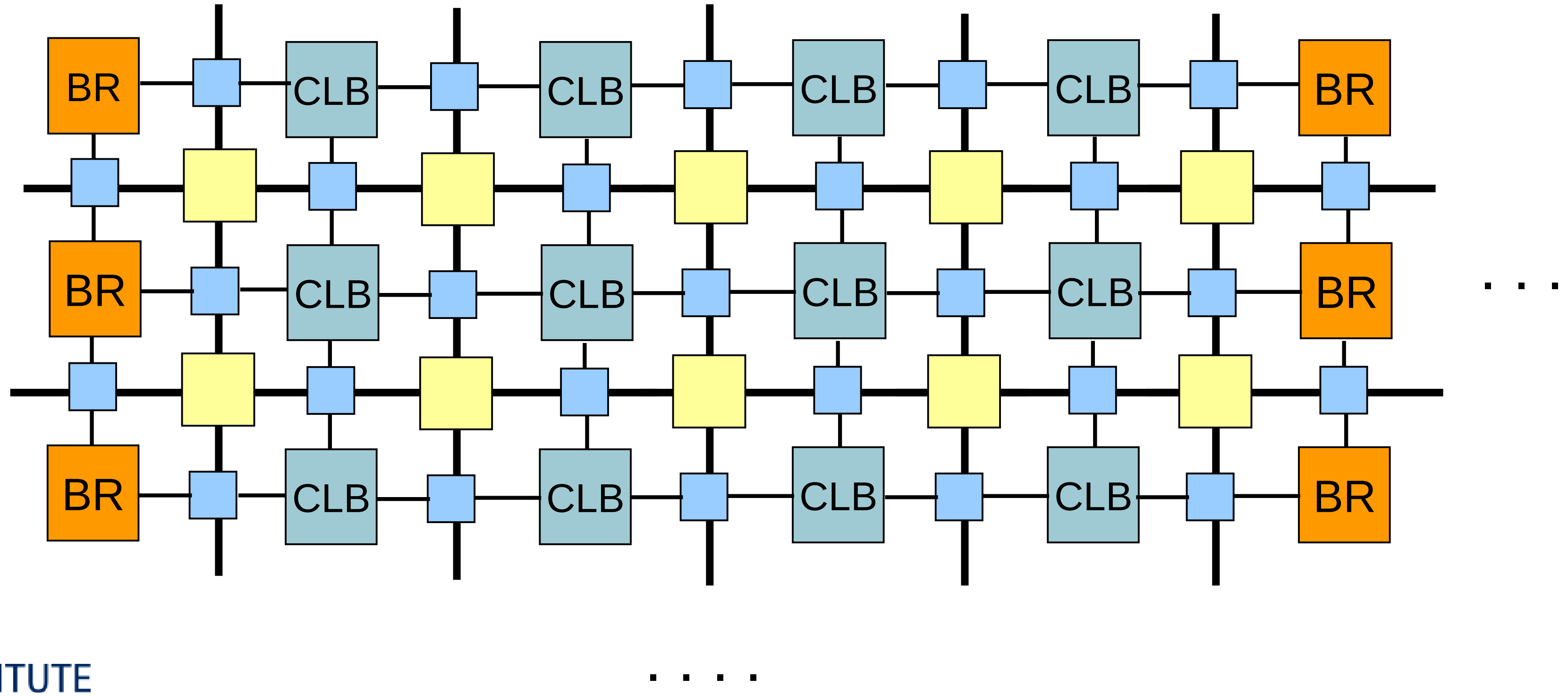
- What about memory?
  - Could use FF’s in CLBs to create a memory
    - Example: Create a 1 MB memory with:
      - CLB with a single 3-input, 2-output LUT
    - Each CLB = 2 bits of memory (because of 2 outputs)
    - Total CLBs = (1 MB * 8 bits/byte) / 2 bits/CLB
      - 4 million CLBs!!!!
      - 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!

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
    - Intel/Altera refers to as MLABs
- 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

FPGA Memory Components

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

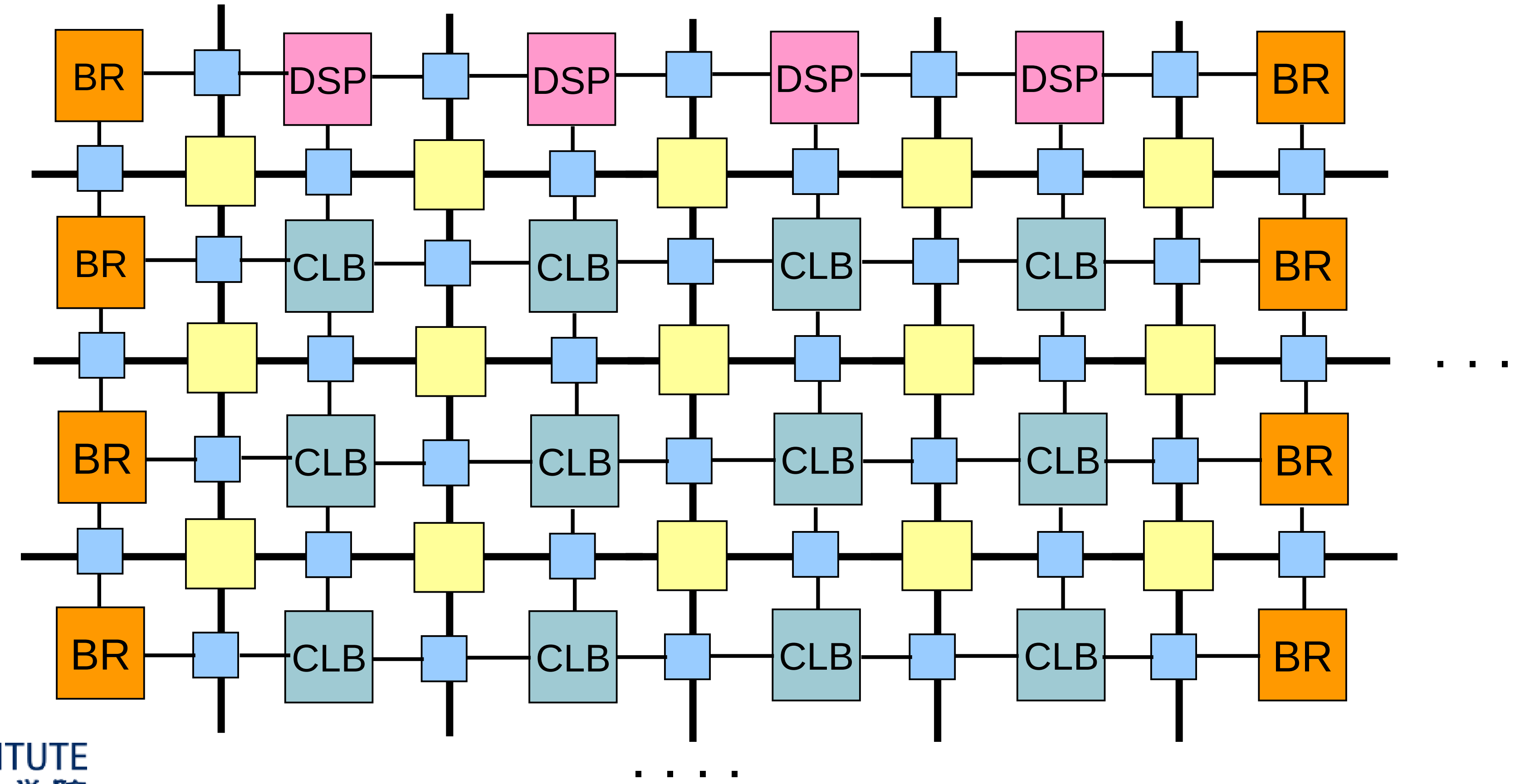


DSP Components

- 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.
- Older Altera devices have multiplier blocks
  - Can be configured as 18x18 or 2 separate 9x9 multipliers
- Intel Arria/Stratix 10 have floating-point DSPs
  - https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug-a10dsp.pdf

Example Fabric

- Existing FPGAs are 2-dimensional arrays of CLBs, DSP, Block RAM, and programmable interconnect
  - Actual layout/placement differs for different FPGAs
  - Specialized resources tend to be in columns not rows

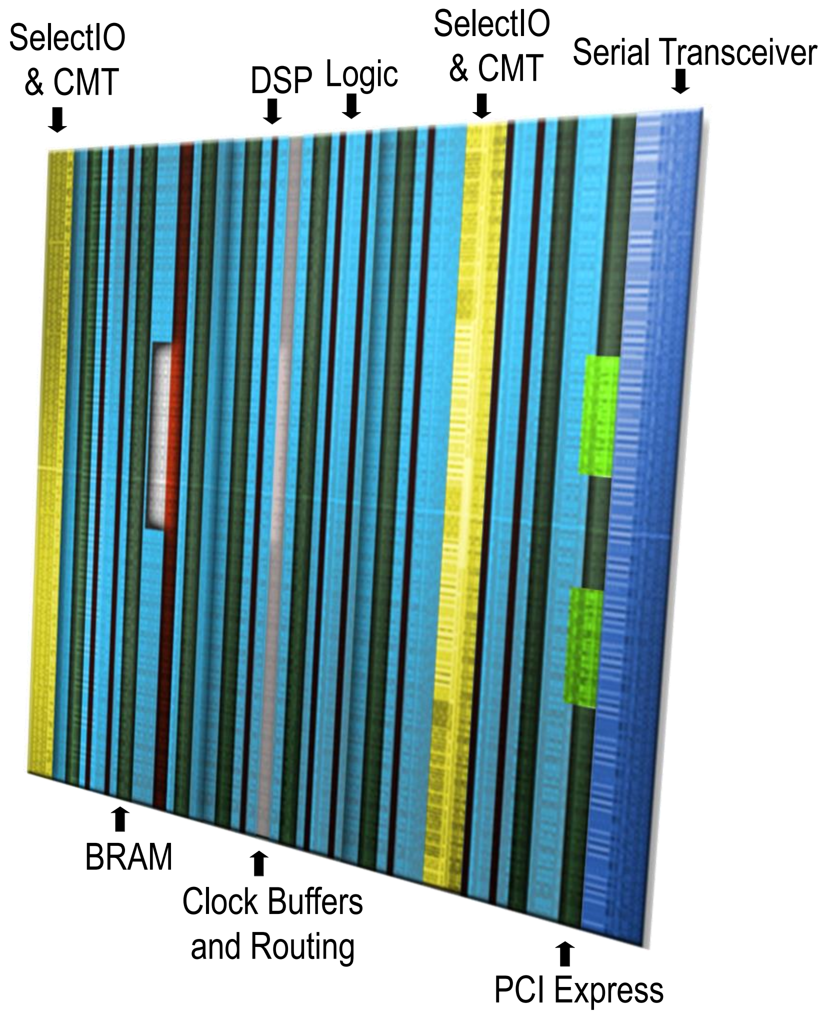


Other resources

- I/O
  - Virtex 7 has 1,200 pins
  - Communication is still often a bottleneck
    - Pins don’t increase with new FPGAs, but logic does
  - Trend: High-speed serial transceivers
- Clock resources
  - Using reconfigurable interconnect for clock introduces timing problems
    - Skew, jitter
  - FPGAs often provided clock trees, both globally and locally
  - e.g. Virtex 7 http://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf

Example Fabrics

- Virtex 7 (image from Xilinx 7-series overview)



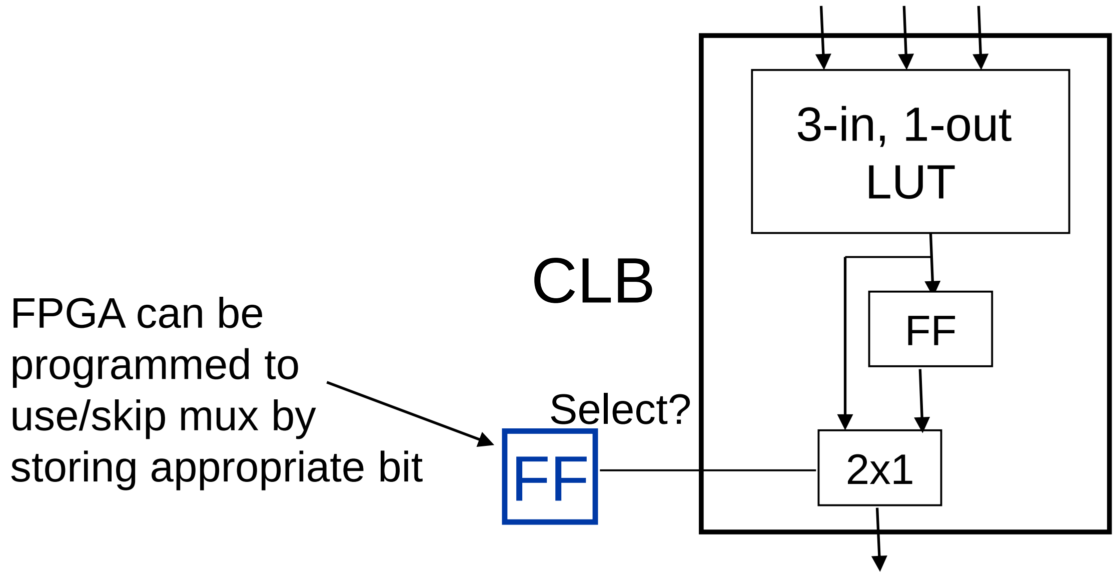
CMT: clock management tiles

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.

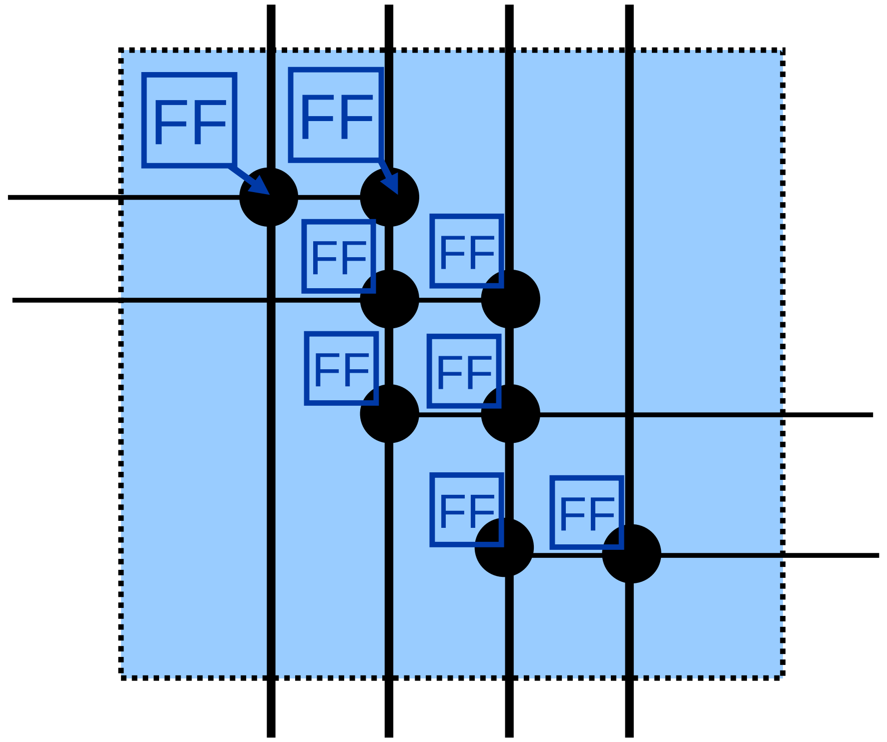
Programming FPGAs

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


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

Programming FPGAs

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

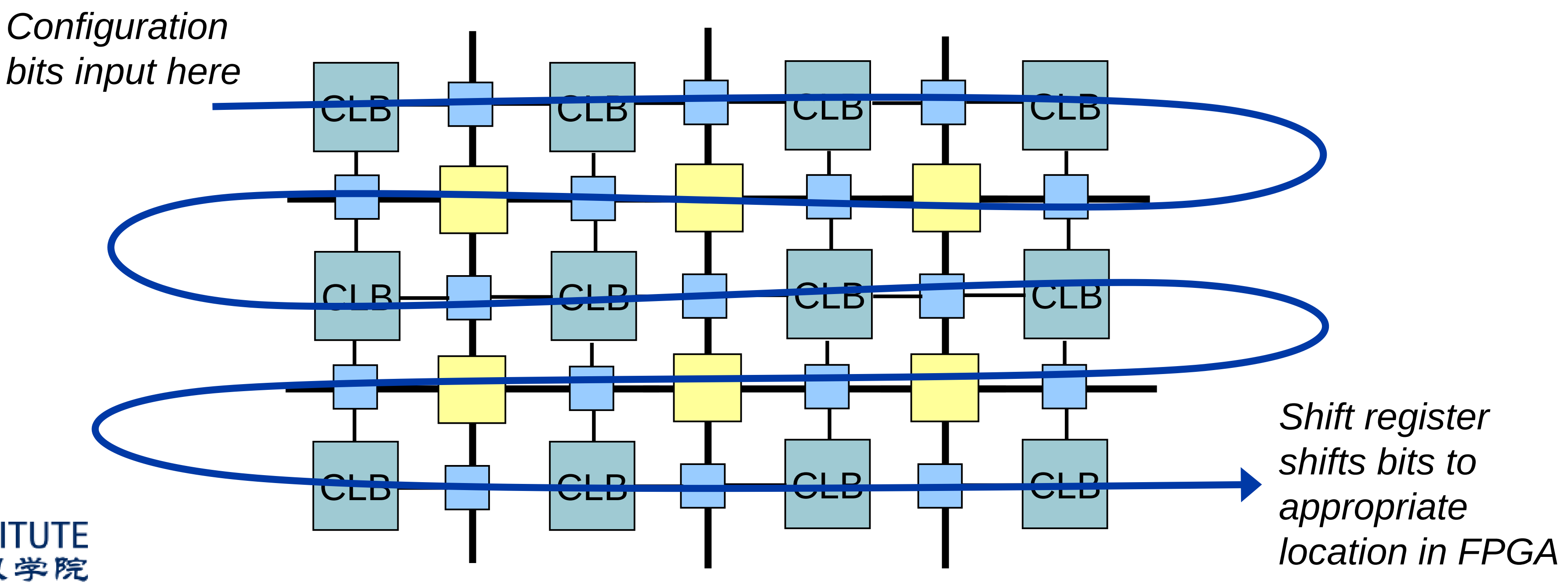


Programming FPGAs

- 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

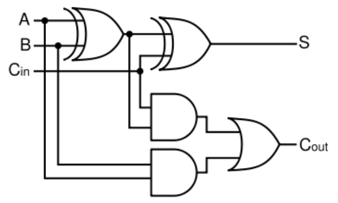
Programming FPGAs

- 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

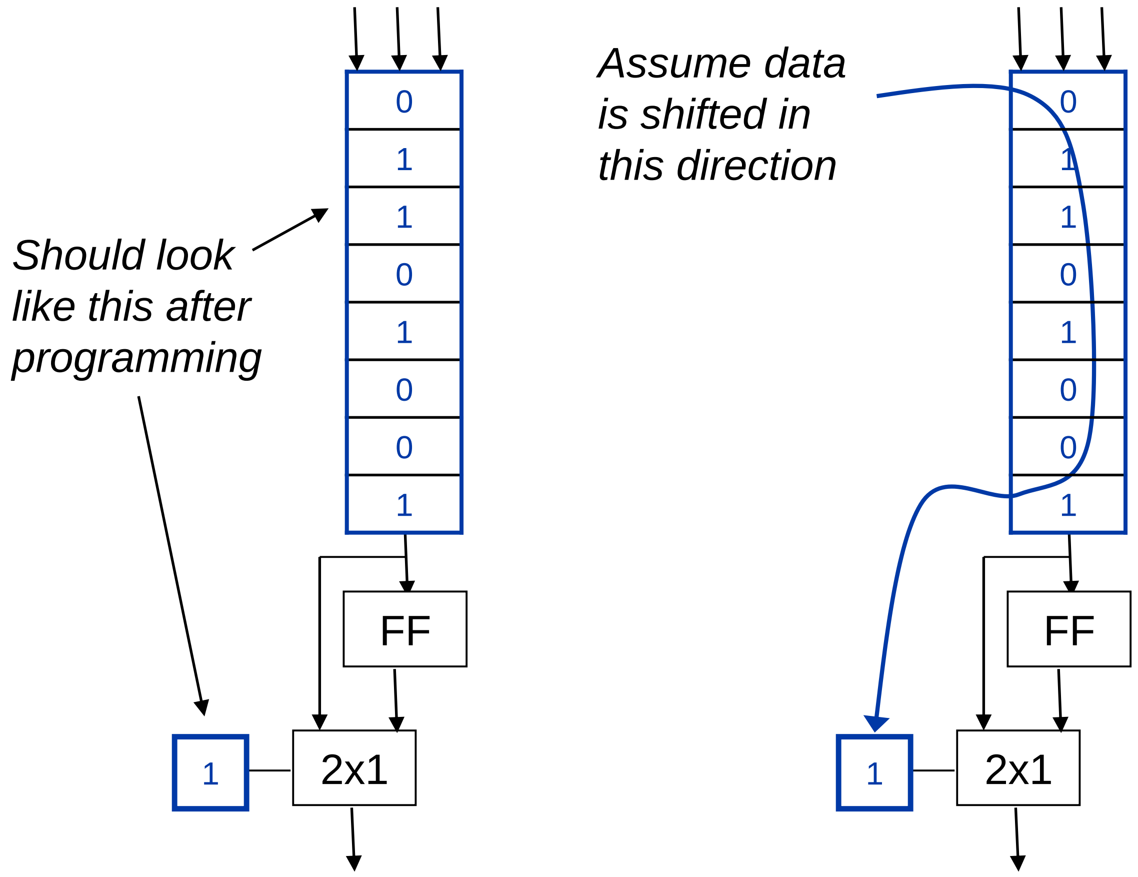

Configuration bits input here
CLB
Shift register shifts bits to appropriate location in FPGA

Programming FPGAs

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

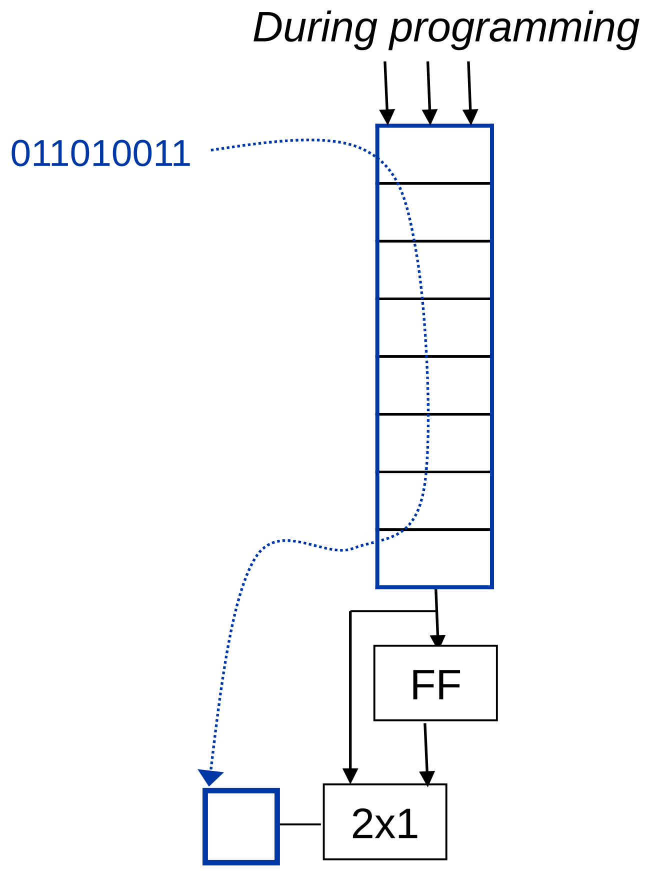


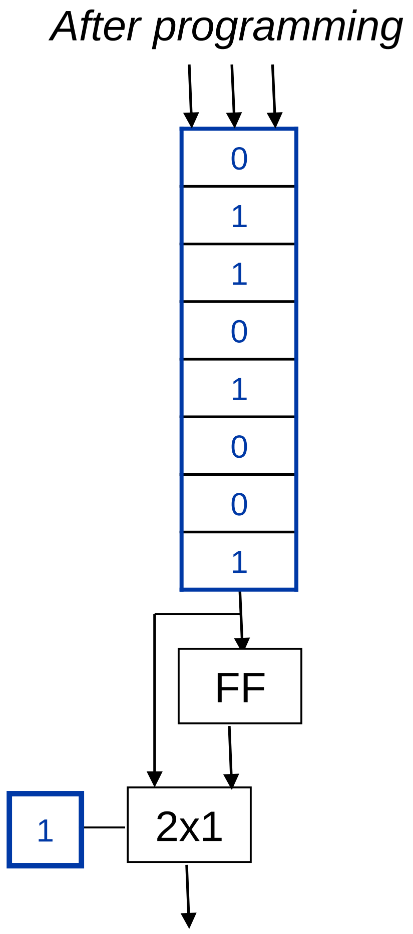
In			Out
A	B	Cin	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1


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

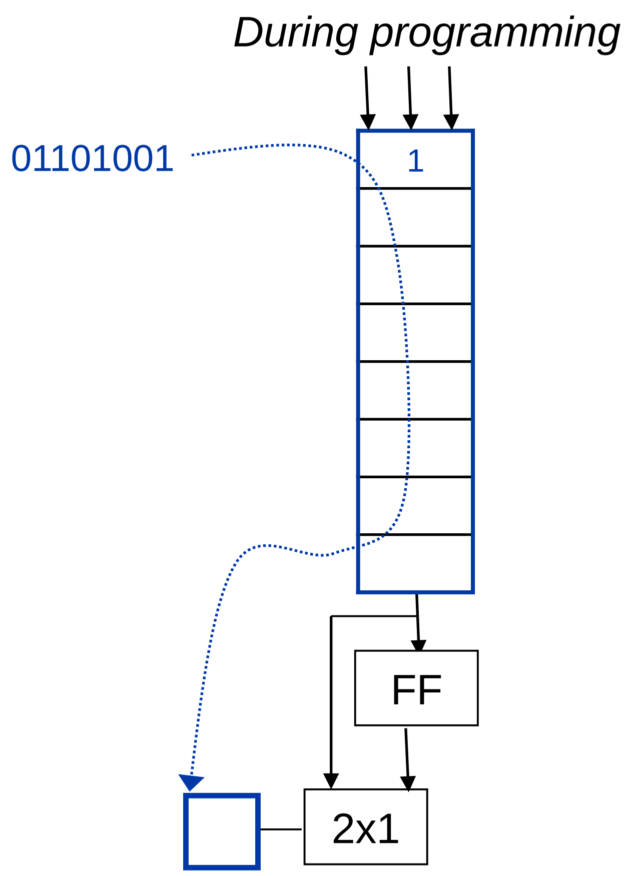
- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register

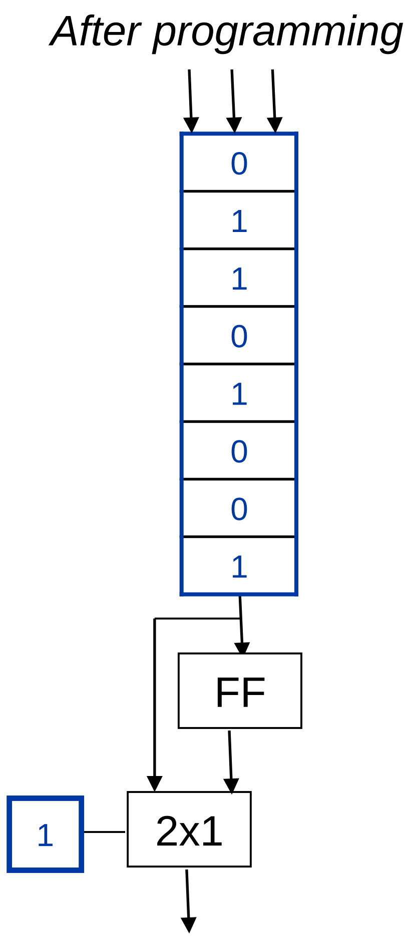




Programming FPGAs

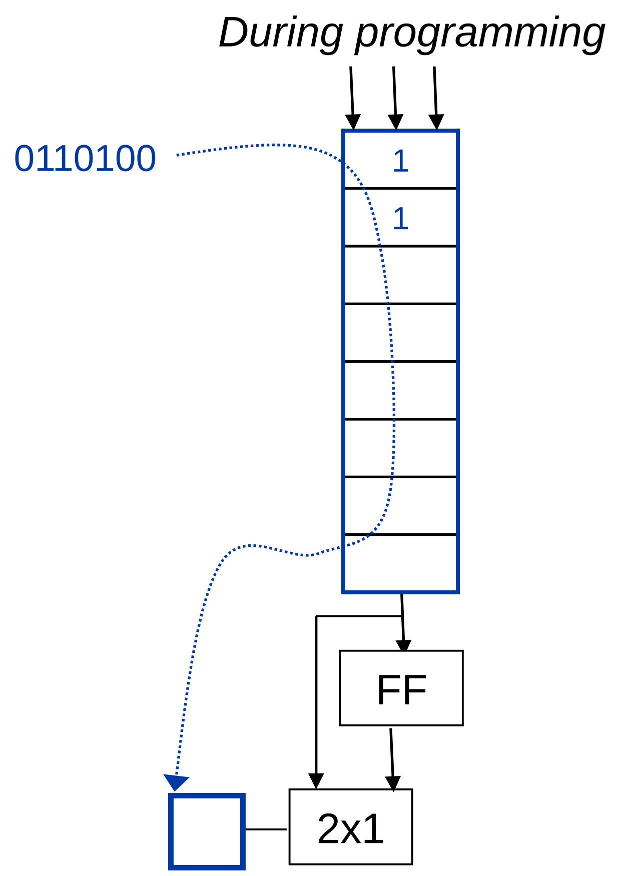
- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register

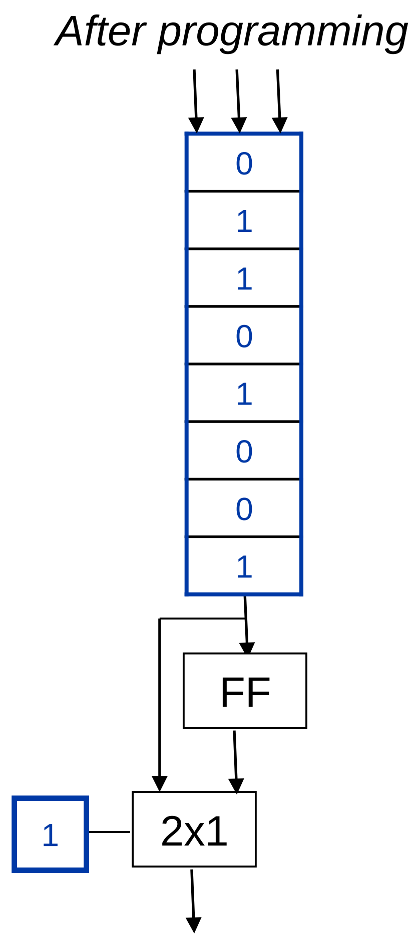




Programming FPGAs

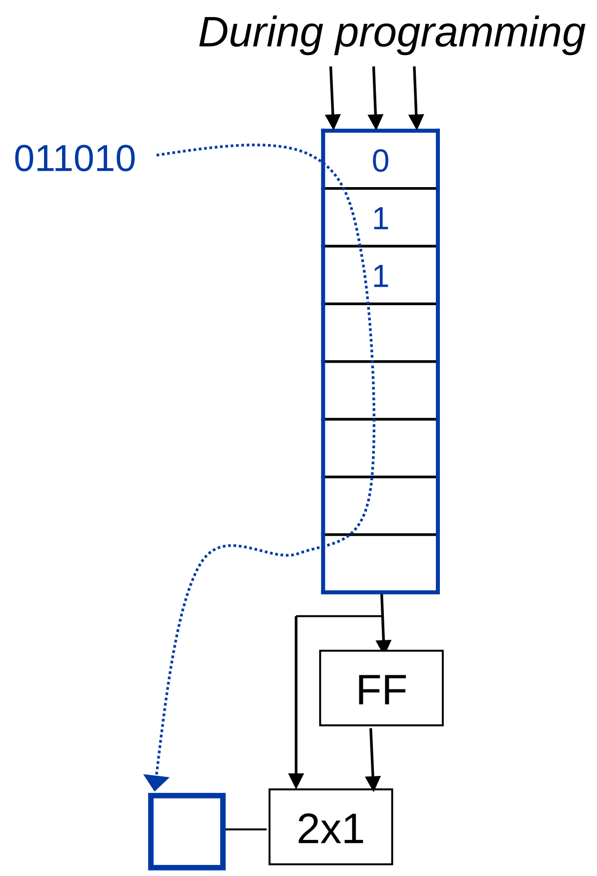
- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register

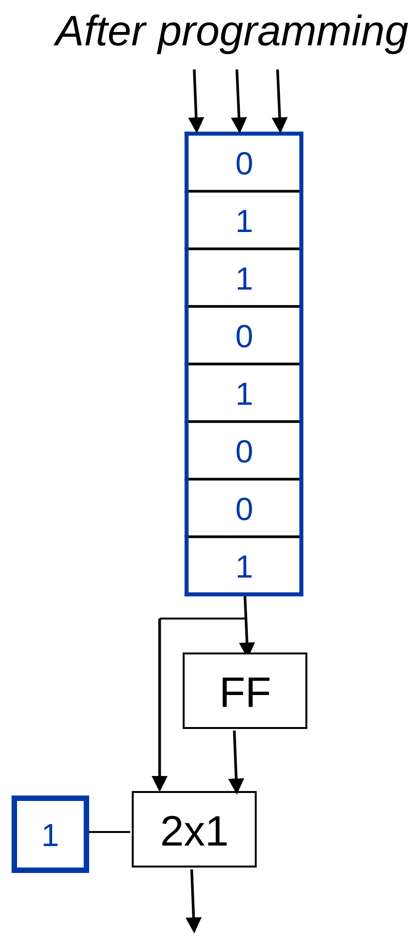




Programming FPGAs

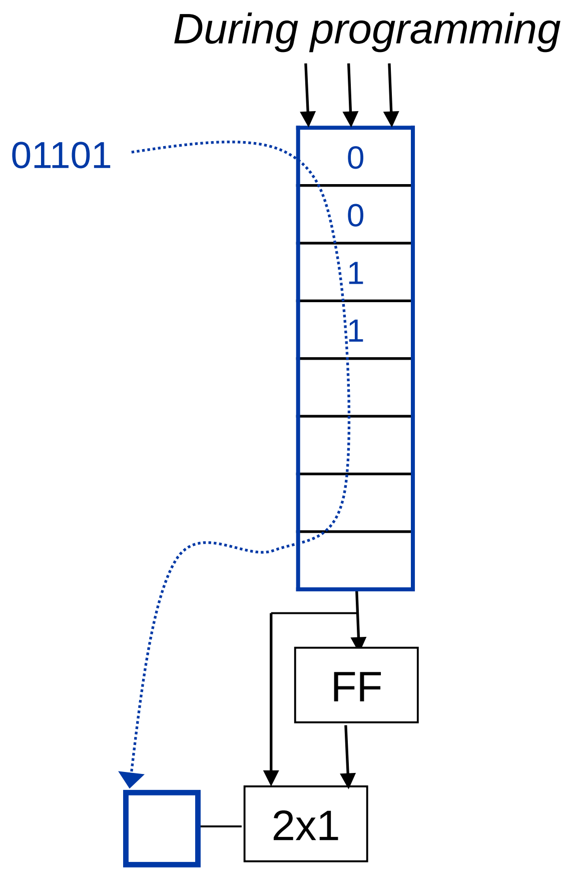
- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register

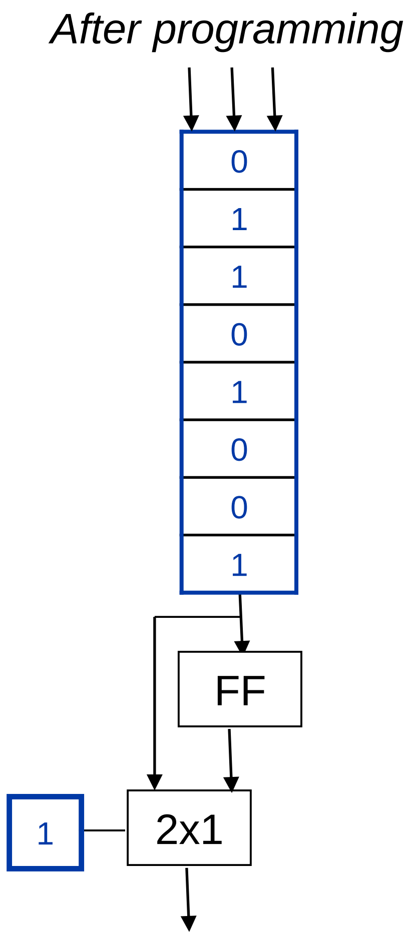




Programming FPGAs

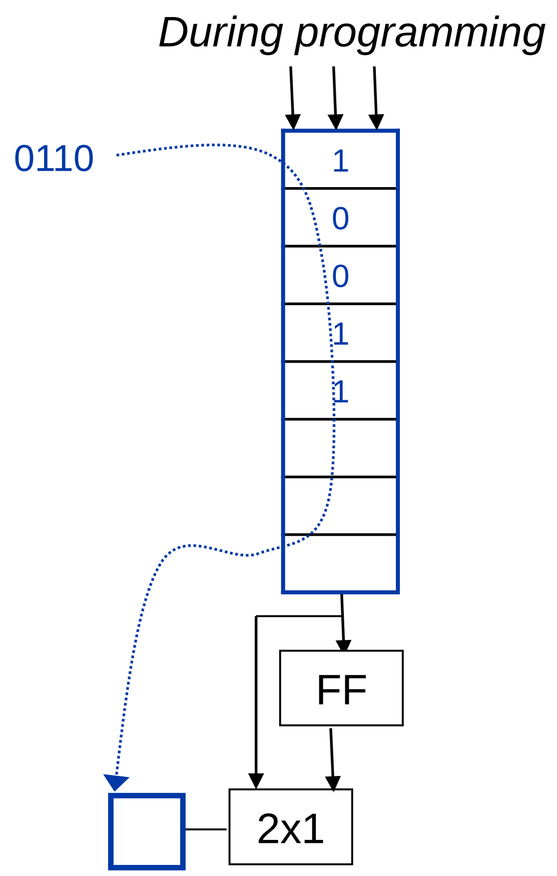
- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register

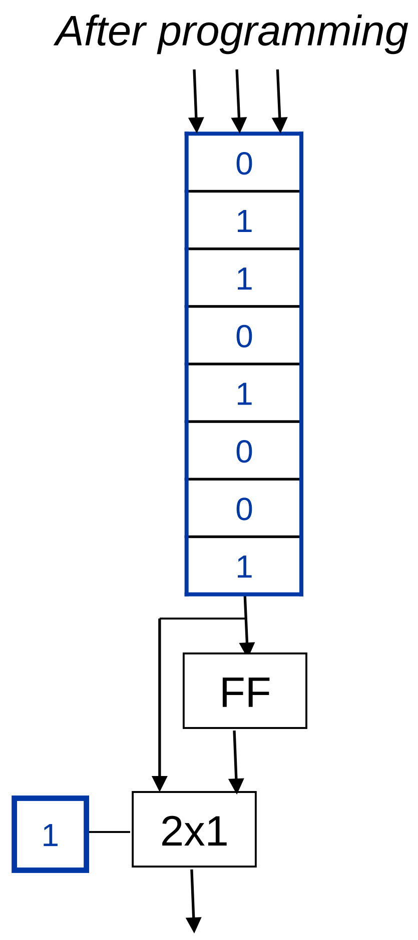




Programming FPGAs

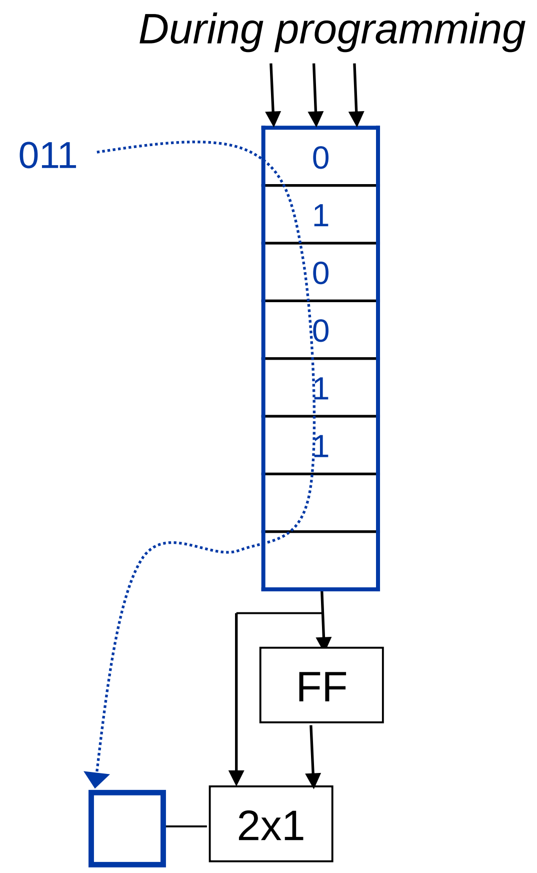
- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register

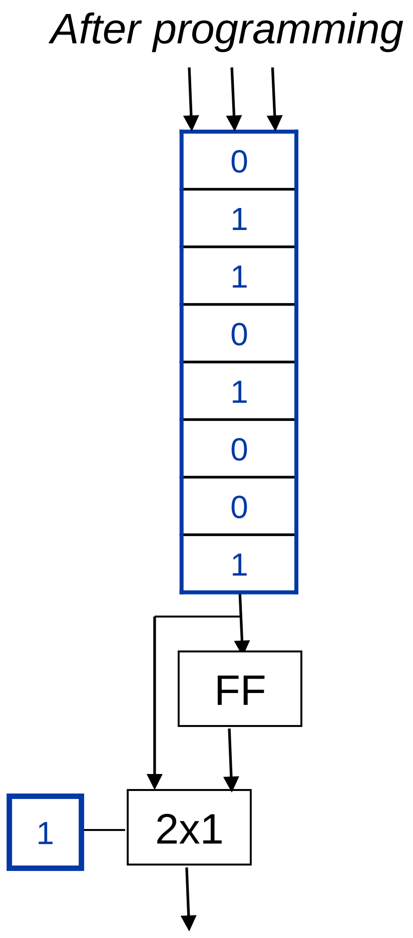




Programming FPGAs

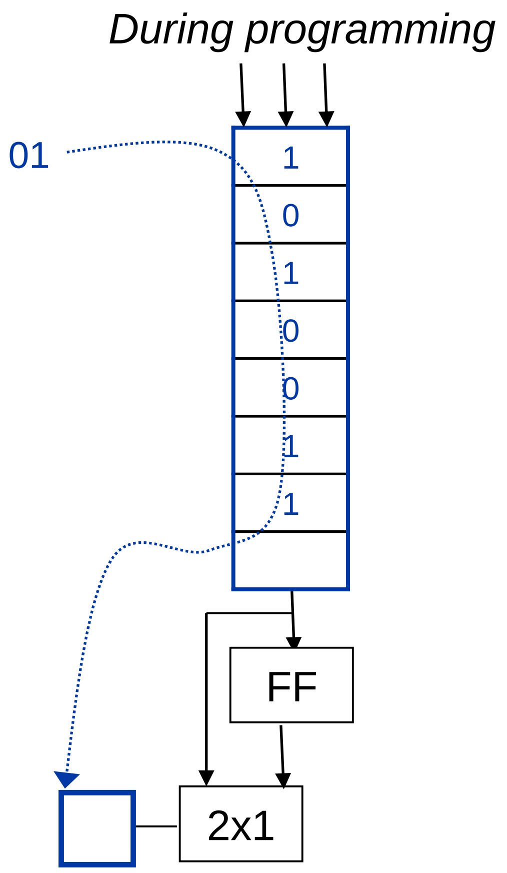
- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register

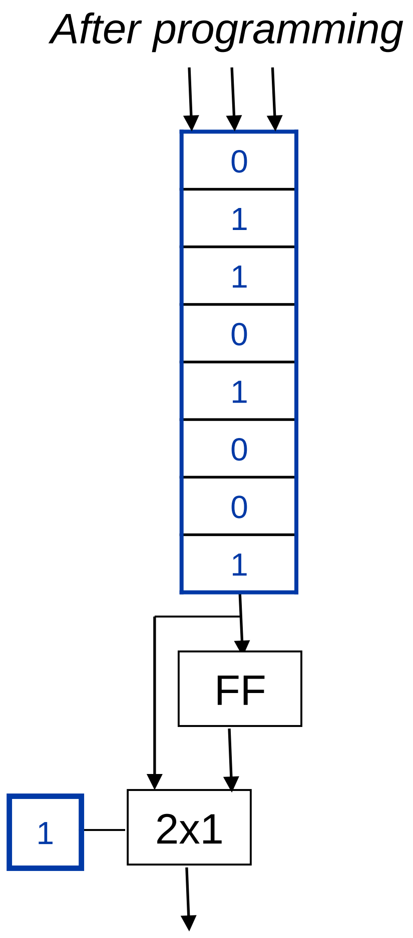




Programming FPGAs

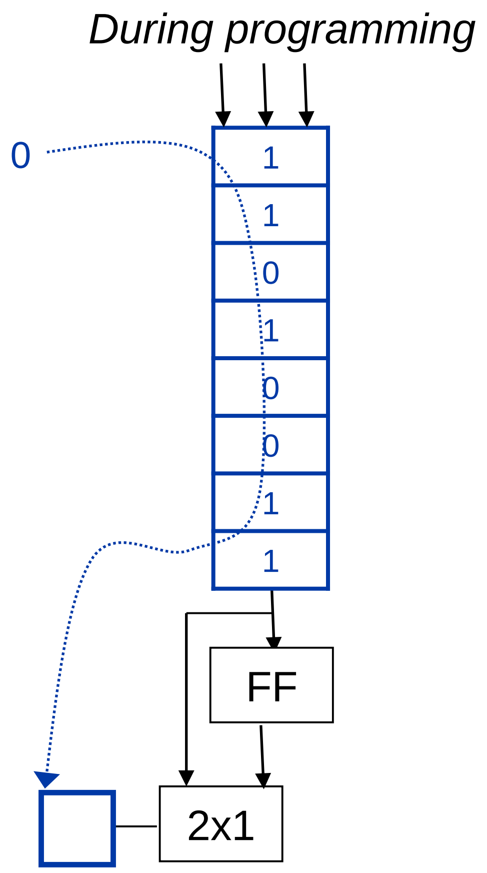
- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register

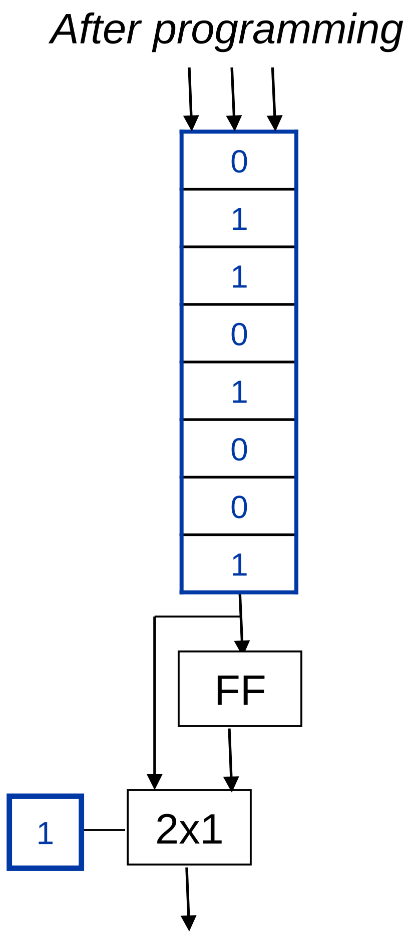




Programming FPGAs

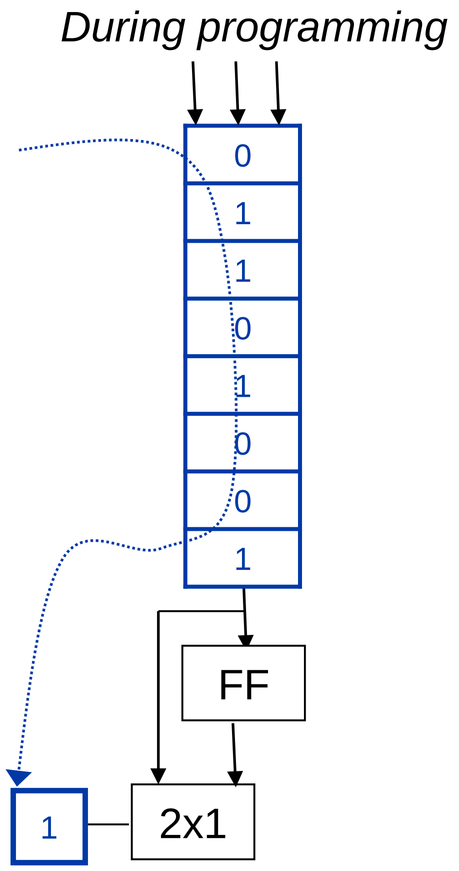
- Example, Cont:
  - Bitfile is just a sequence of bits based on order of shift register





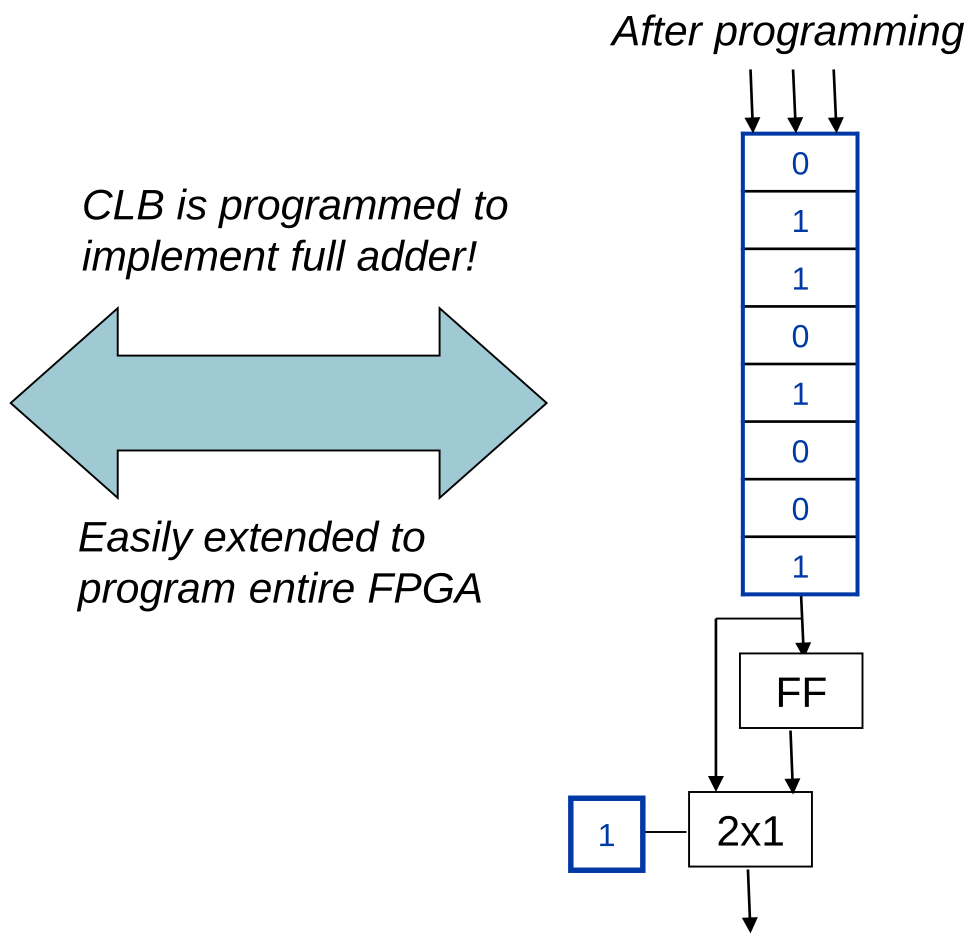
Programming FPGAs

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



CLB is programmed to implement full adder!

Easily extended to program entire FPGA



Programming FPGAs

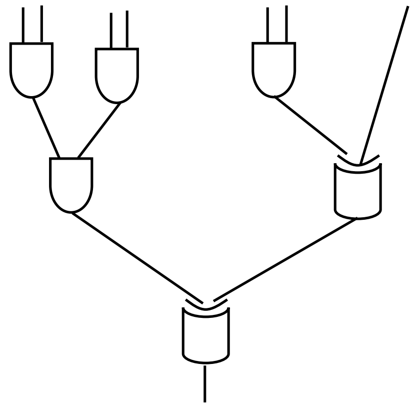
- Problem: Reconfiguring FPGA is slow
  - Shifting in 1 bit at a time not efficient
  - Bitfiles can be greater than 1 MB
  - Eliminates one of the main advantages of RC
    - Partial reconfiguration
    - With shift registers, entire FPGA has to be reconfigured
- Solutions?
  - Virtex II allowed columns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

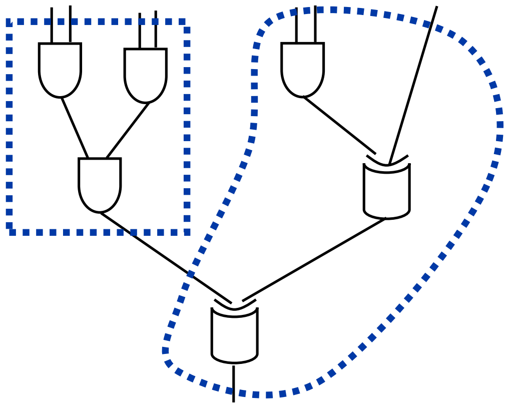
- Example:
  - Determine best LUTs for following circuit
    - Choices
      - 4-input, 2-output LUT (delay = 2 ns)
      - 5-input, 2-output LUT (delay = 3 ns)
    - Assume each SRAM cell is 6 transistors
      - 4-input LUT = 6 * 2^4 * 2 = 192 transistors
      - 5-input LUT = 6 * 2^5 * 2 = 384 transistors



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



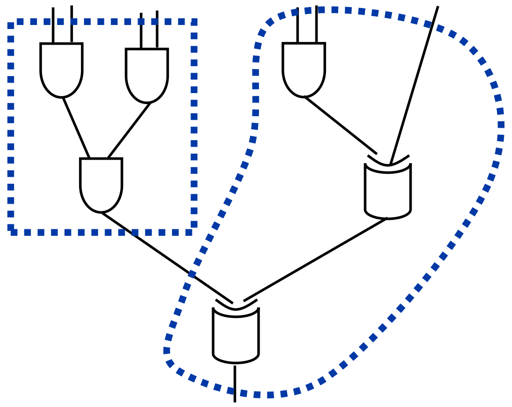
Propagation delay = 6 ns

Total transistors = 384 * 2 = 768

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



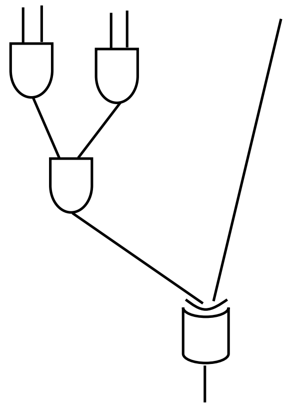
Propagation delay = 4 ns

Total transistors = 192 * 2 = 384

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

FPGA Architecture Tradeoffs

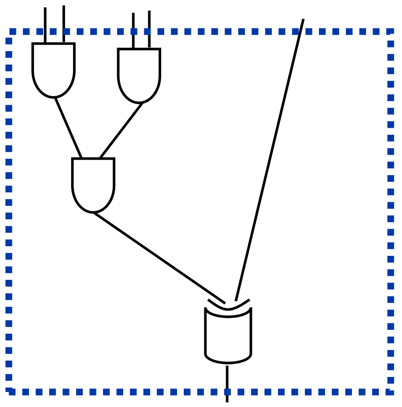
- Example 2
  - Determine best LUTs for following circuit
    - Choices
      - 4-input, 2-output LUT (delay = 2 ns)
      - 5-input, 2-output LUT (delay = 3 ns)
    - Assume each SRAM cell is 6 transistors
      - 4-input LUT = 6 * 2^4 * 2 = 192 transistors
      - 5-input LUT = 6 * 2^5 * 2 = 384 transistors



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



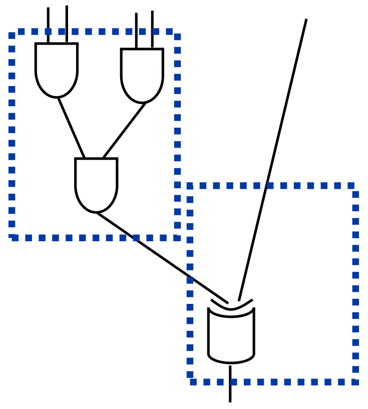
Propagation delay = 3 ns

Total transistors = 384

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

FPGA Architecture Tradeoffs

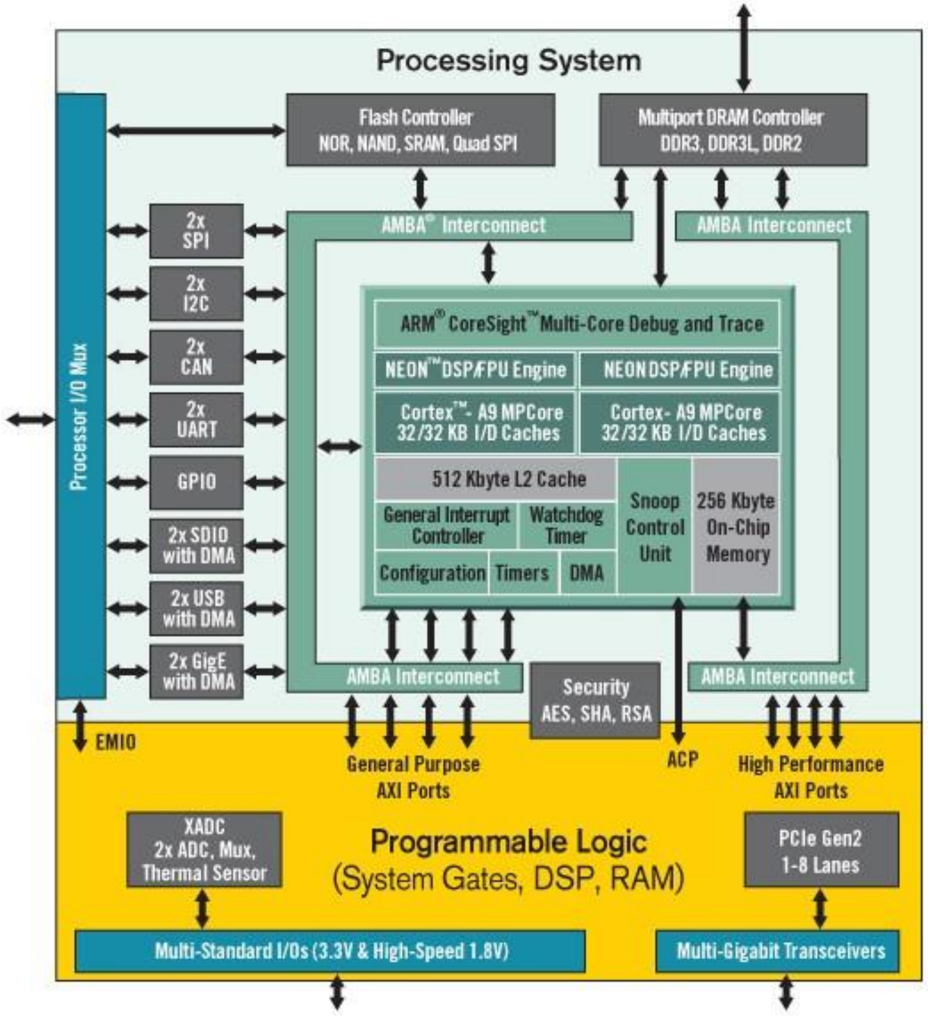
- Large LUTs
  - Fast when using all inputs
  - Wastes transistors otherwise
- Must also consider total chip area
  - Wasting transistors may be ok if there are plently of LUTs
    - Virtex V uses 6 input LUTs
    - Virtex IV uses 4 input LUTs

FPGA Architecture Tradeoffs

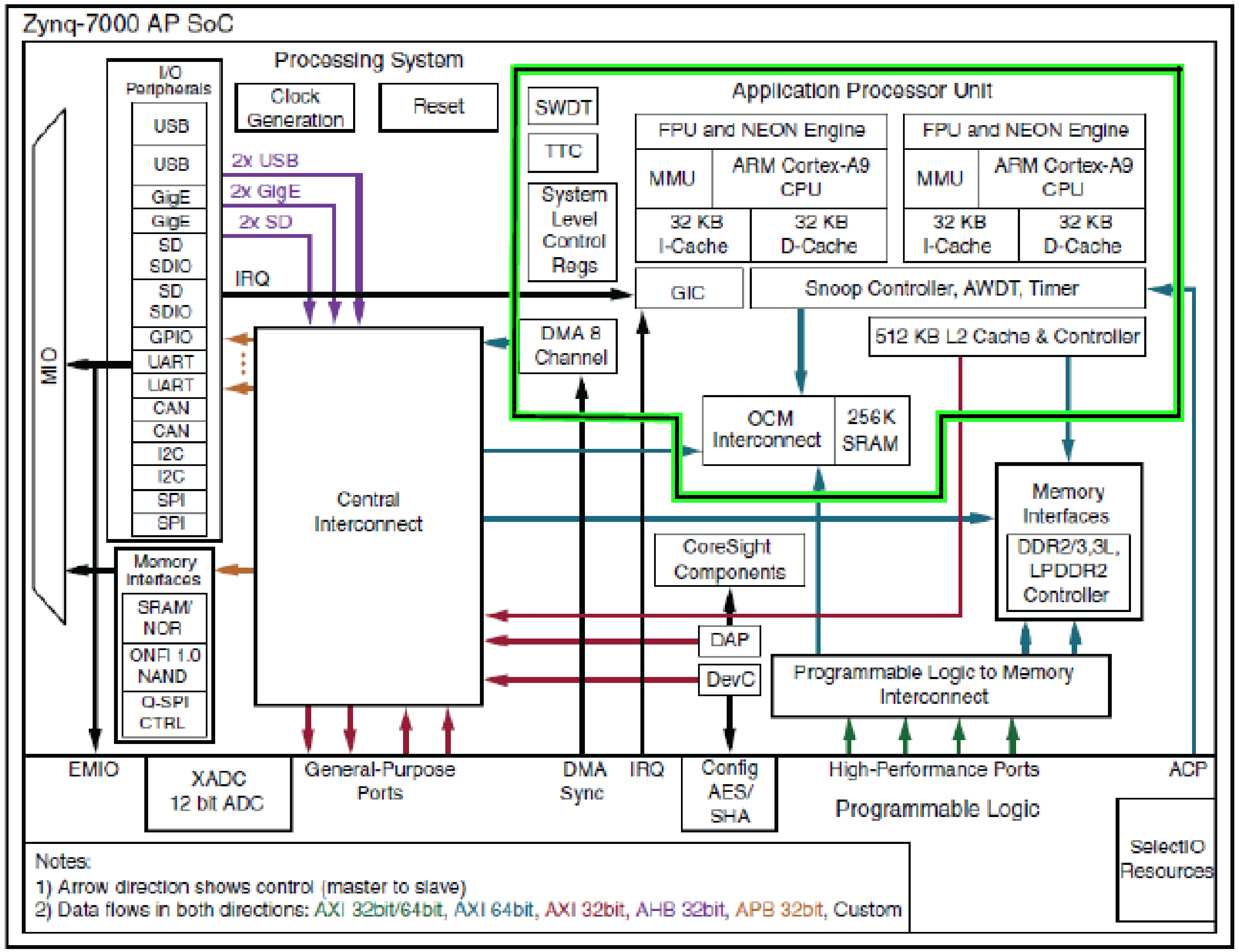
- How to design FPGA fabric?
  - There is no overall best
  - Design fabric based on different domains
    - DSP will require many of DSP units
    - HPC may require balance of units
    - SoCs may require microprocessors
- Examples:
  - Xilinx Virtex IV
    - LX - designed for logic intensive apps
    - SX - designed for signal processing apps
    - FX - designed for embedded systems apps
      - Has 450 MHz PowerPC cores embedded in fabric
  - Xilinx 7 Series
    - Artix, Kintex, Virtex

Come back to Zynq

- Combines ARM processor with programmable logic (PL)
  - Artix FPGA
  - DRAM controller
  - PCIe controller
  - Other peripherals



The Zynq Processing System

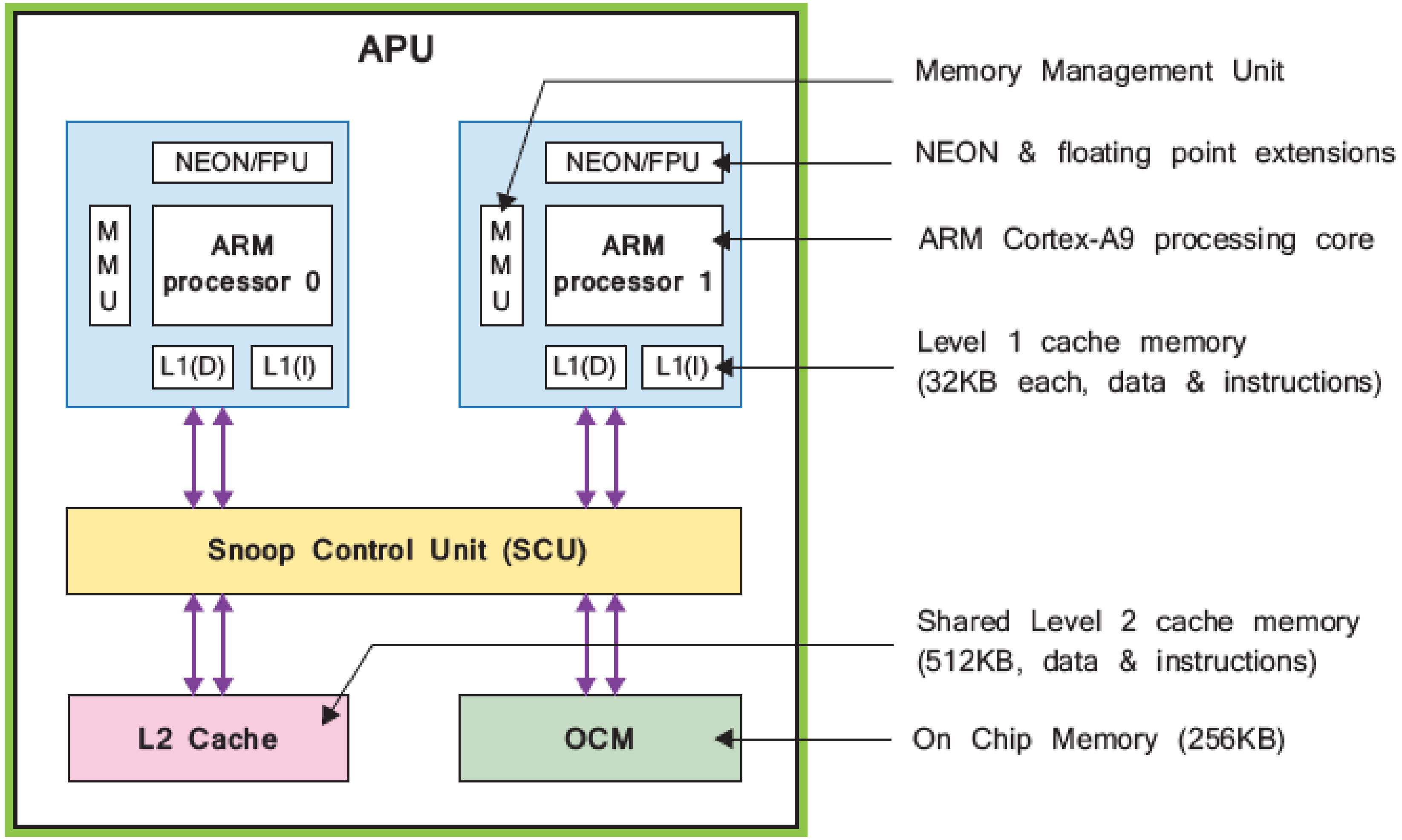


© Xilinx

Source: The Zynq Book

JOINT INSTITUTE 交大密西根学院

Application Processing Unit (APU)

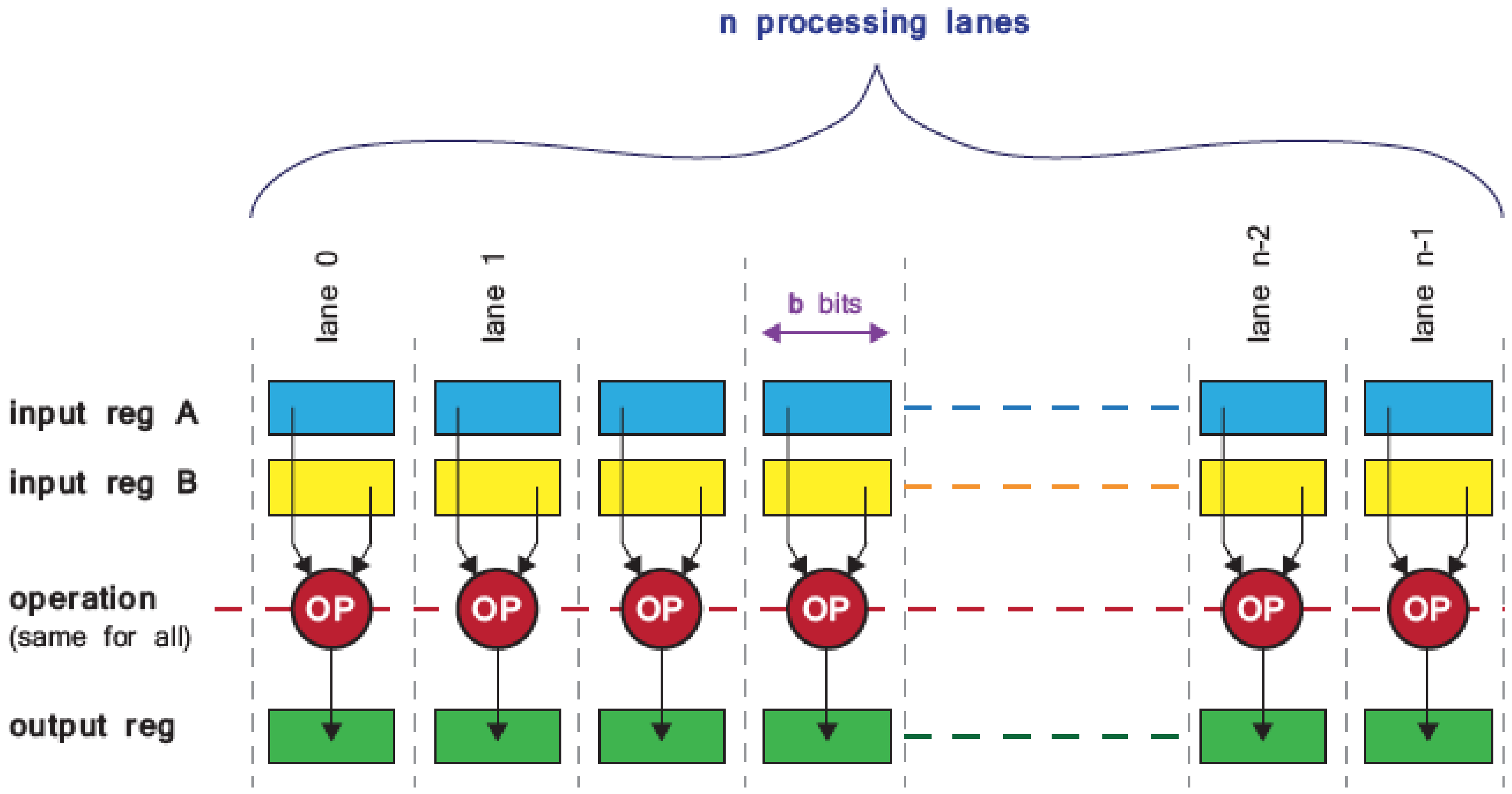


APU
NEON/FPU
M M U
ARM processor 0
L1(D)
L1(I)
NEON/FPU
M M U
ARM processor 1
L1(D)
L1(I)
Snoop Control Unit (SCU)
L2 Cache
OCM
Memory Management Unit
NEON & floating point extensions
ARM Cortex-A9 processing core
Level 1 cache memory (32KB each, data & instructions)
Shared Level 2 cache memory (512KB, data & instructions)
On Chip Memory (256KB)

APU programming is through Xlinx SDK

Source: The Zynq Book

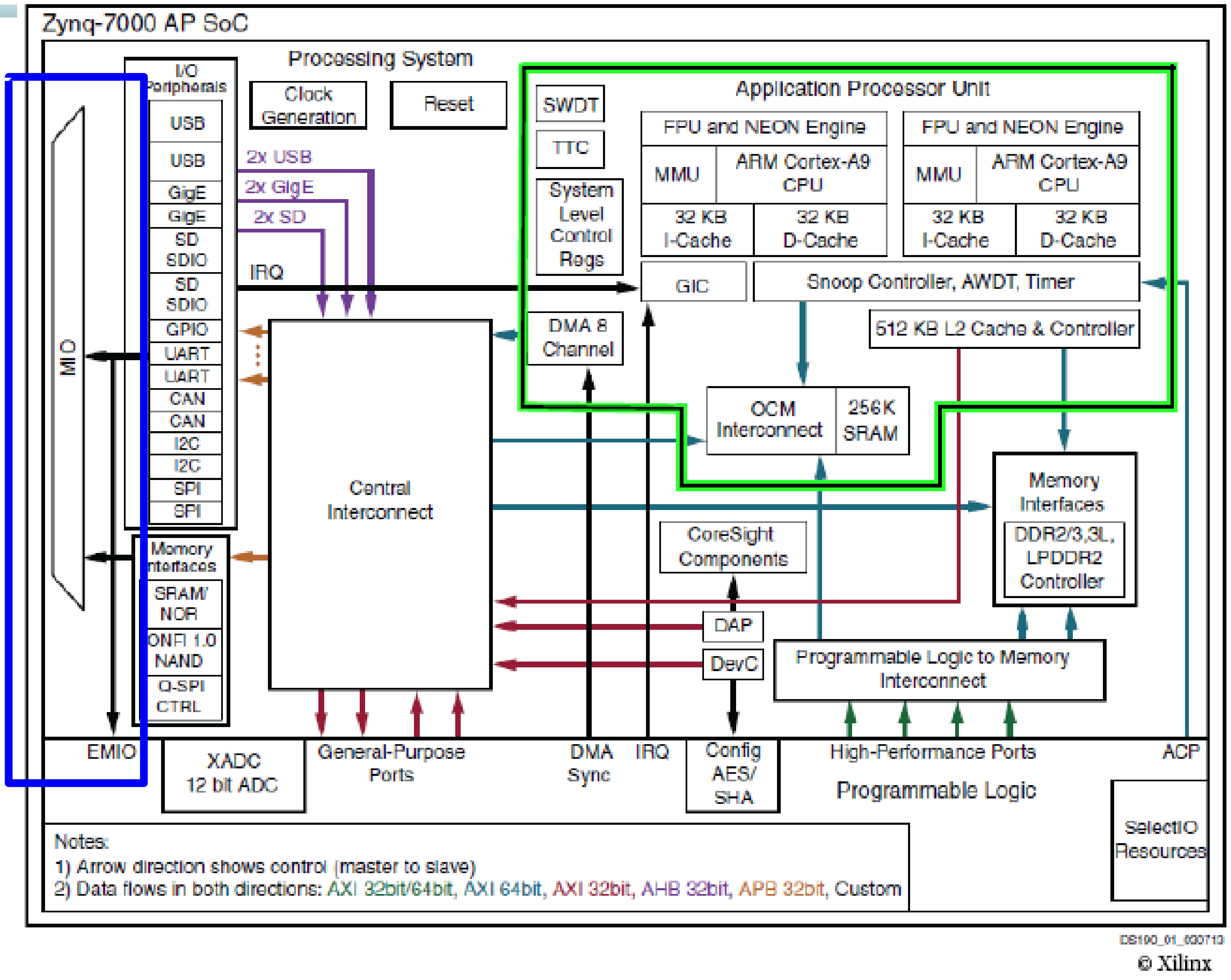
NEON Co-Processor



SIMD Execution for media and DSP applications

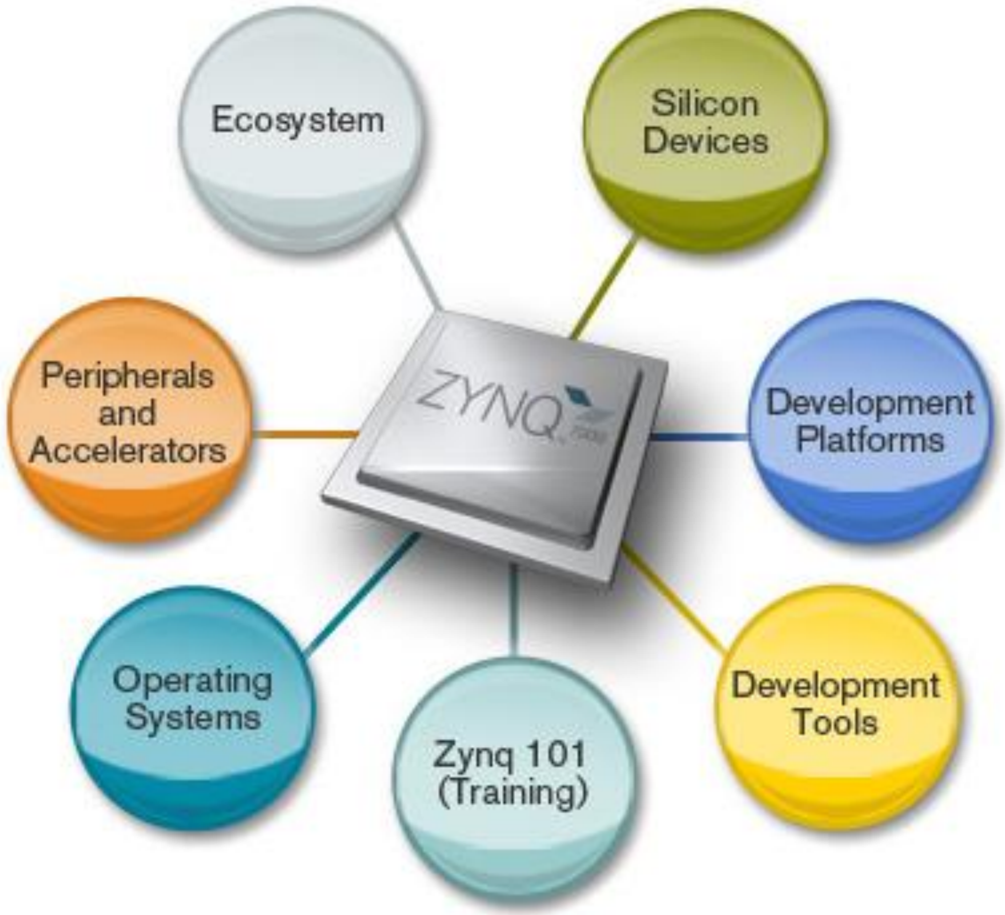
Source: The Zynq Book

PS External Interfaces: MIO

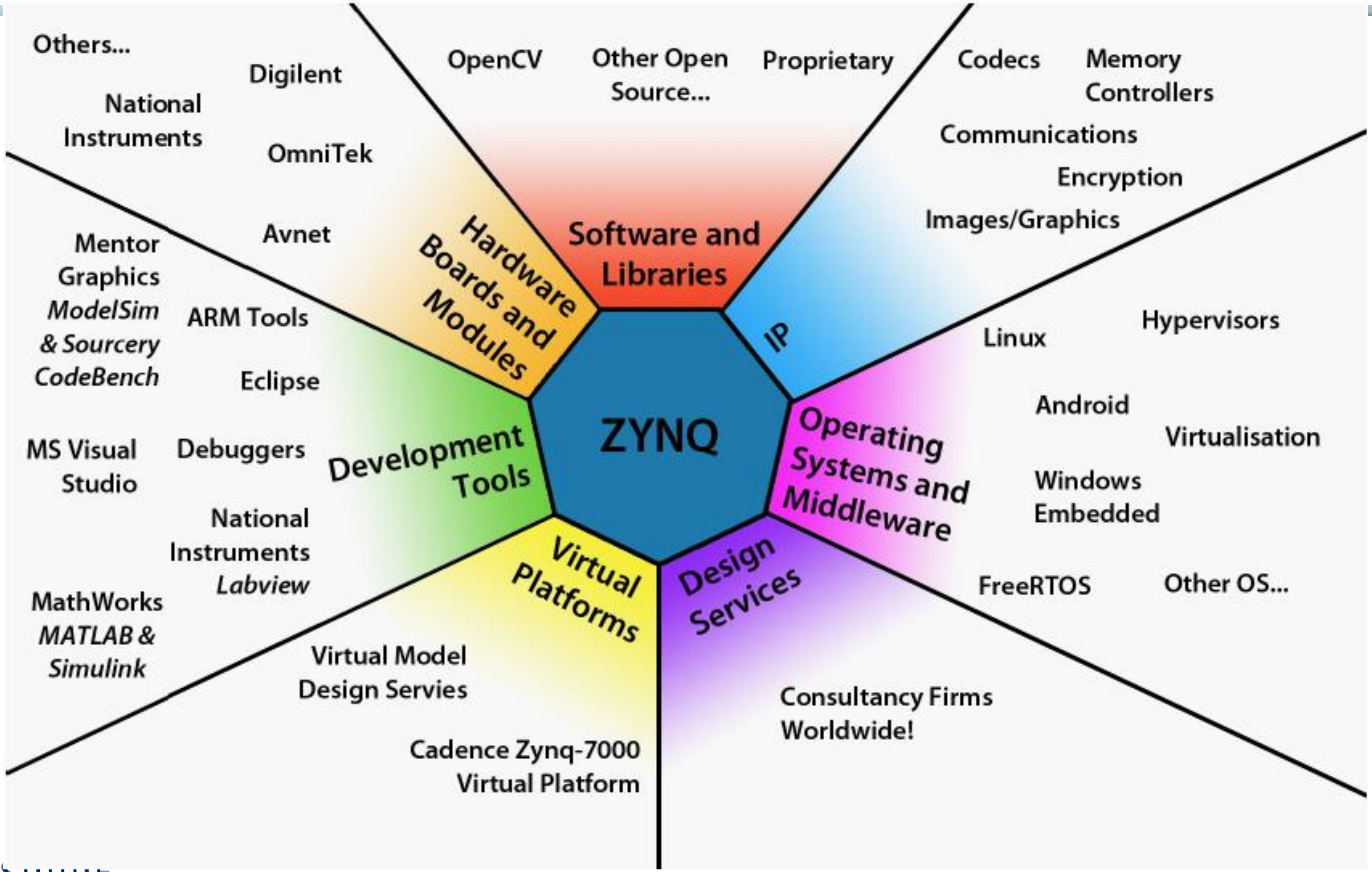


Source: The Zynq Book

Zynq SoC Ecosystem

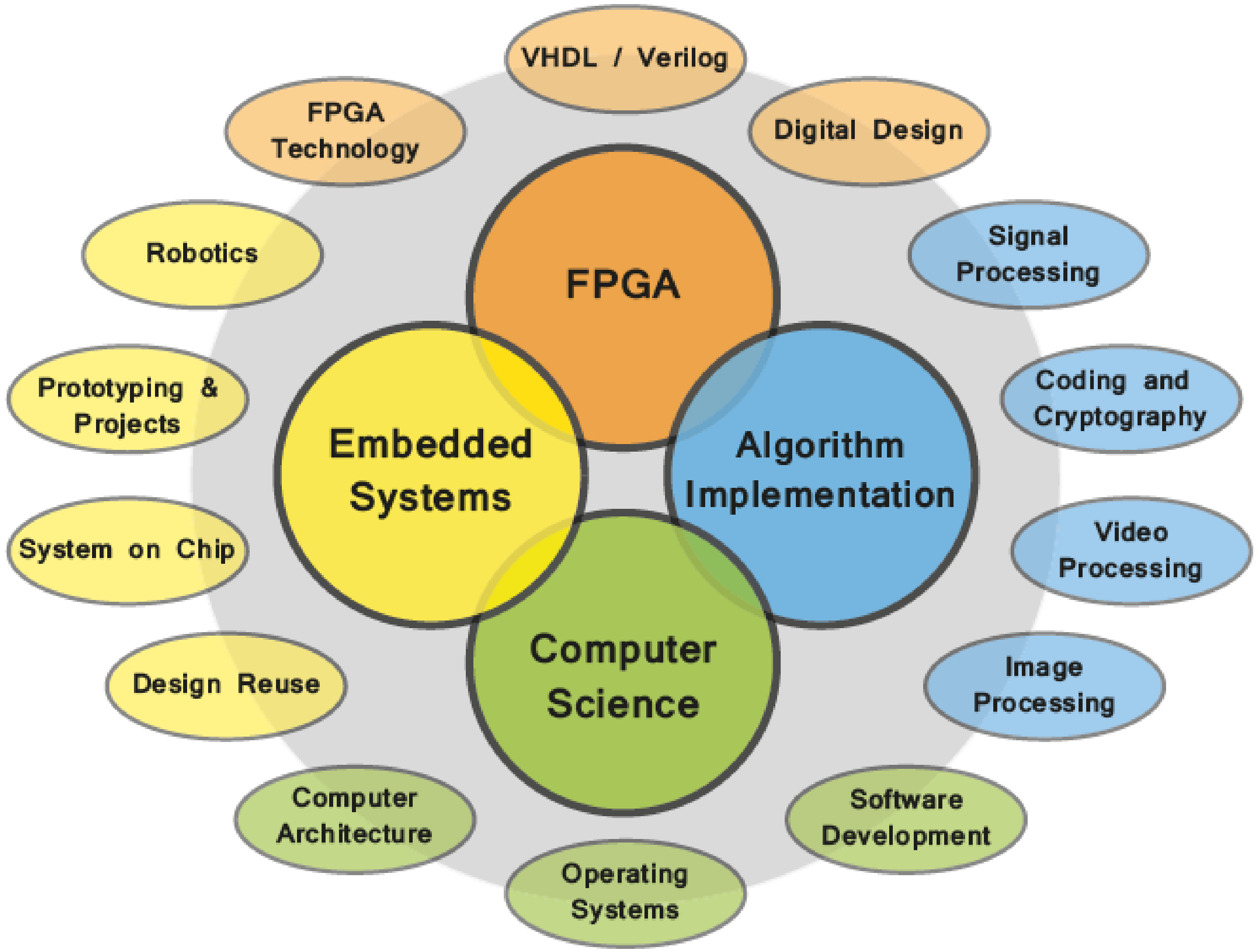


Zynq SoC Ecosystem



Source: The Zynq Book

Academic Subjects to which Zynq is Relevant

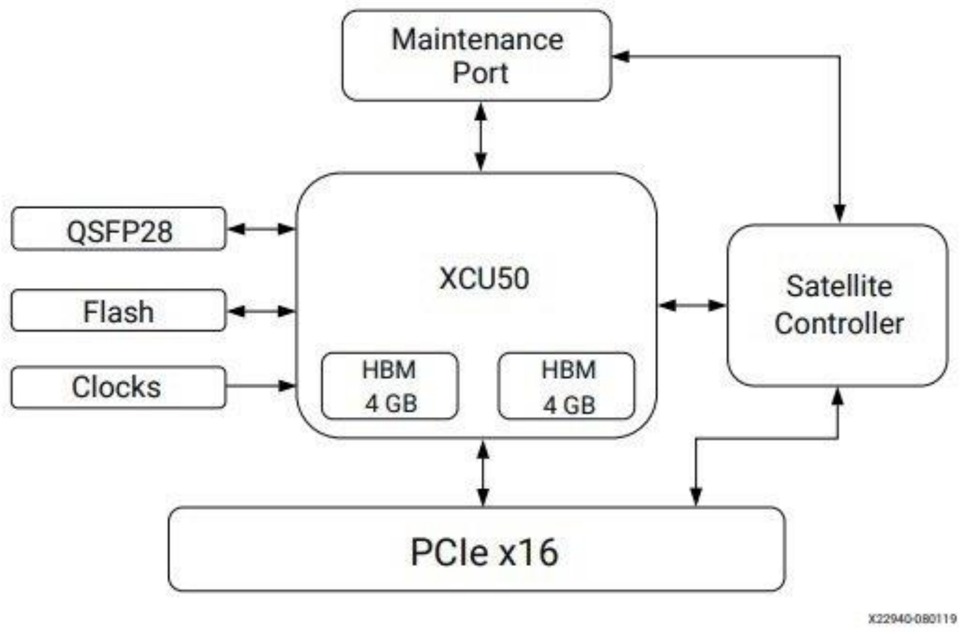


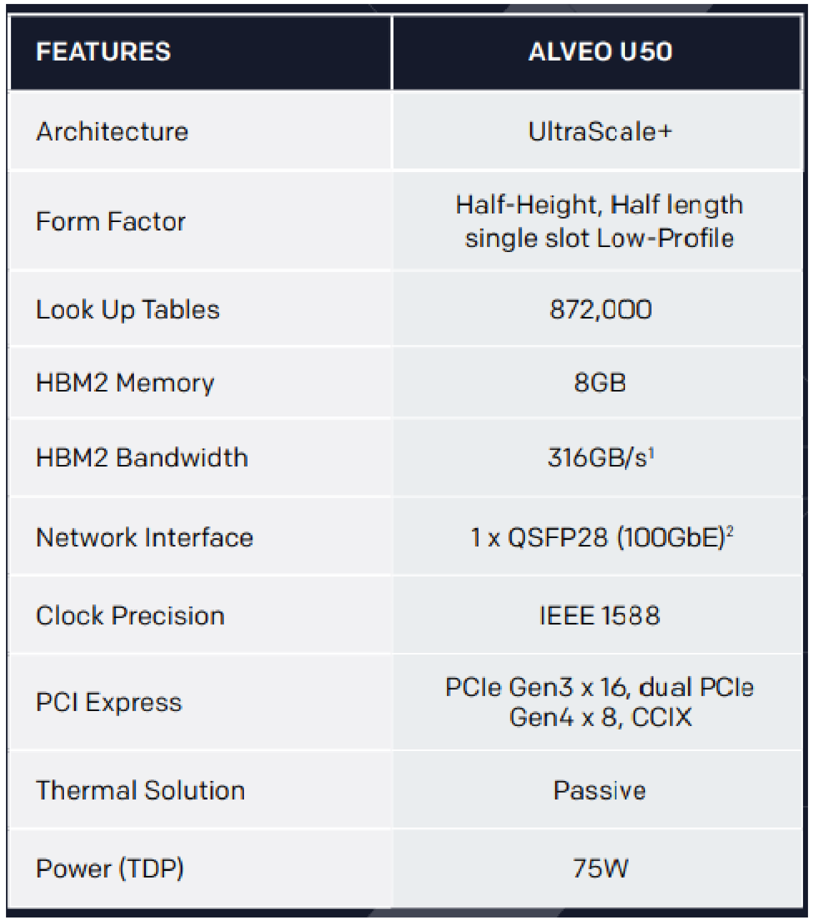
Source: The Zynq Book

Altera Stratix/Arria 10

- > 1 million LUTs
- Floating-point cores in fabric
- HyperFlex Interconnect
  - Embeds flip flops into reconfigurable interconnect
  - Enables significantly faster clock speeds
- https://www.altera.com/products/fpga/stratix-series/stratix-10/overview.html

Xilinx Alveo U50 Data Center Accelerator Card

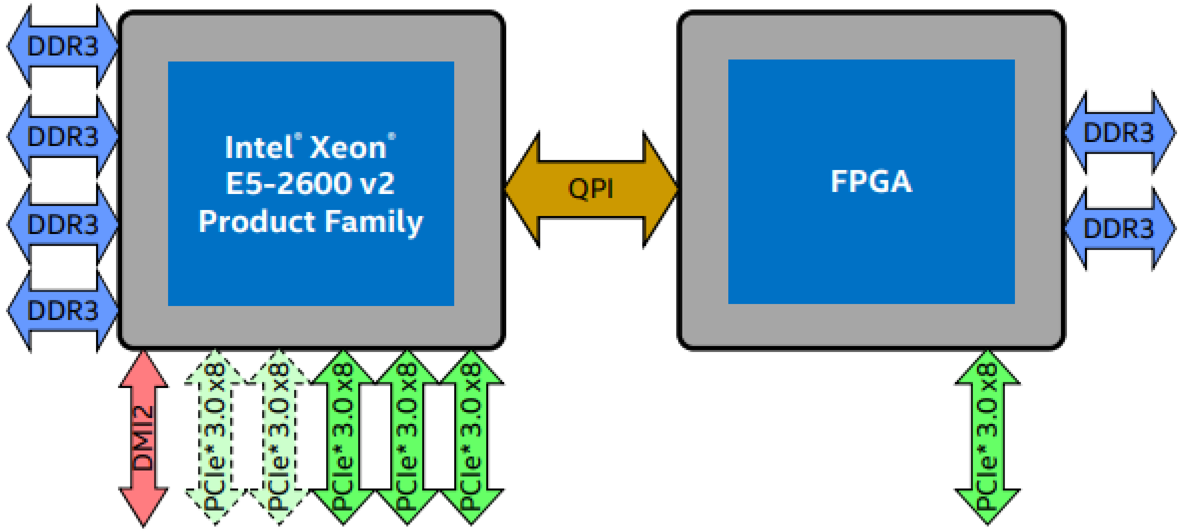




FEATURES	ALVEO U50
Architecture	UltraScale+
Form Factor	Half-Height, Half length single slot Low-Profile
Look Up Tables	872,000
HBM2 Memory	8GB
HBM2 Bandwidth	316GB/s[1]
Network Interface	1 x QSFP28 (100GbE)[2]
Clock Precision	IEEE 1588
PCI Express	PCIe Gen3 x 16, dual PCIe Gen4 x 8, CCIX
Thermal Solution	Passive
Power (TDP)	75W

courtesy of Xilinx

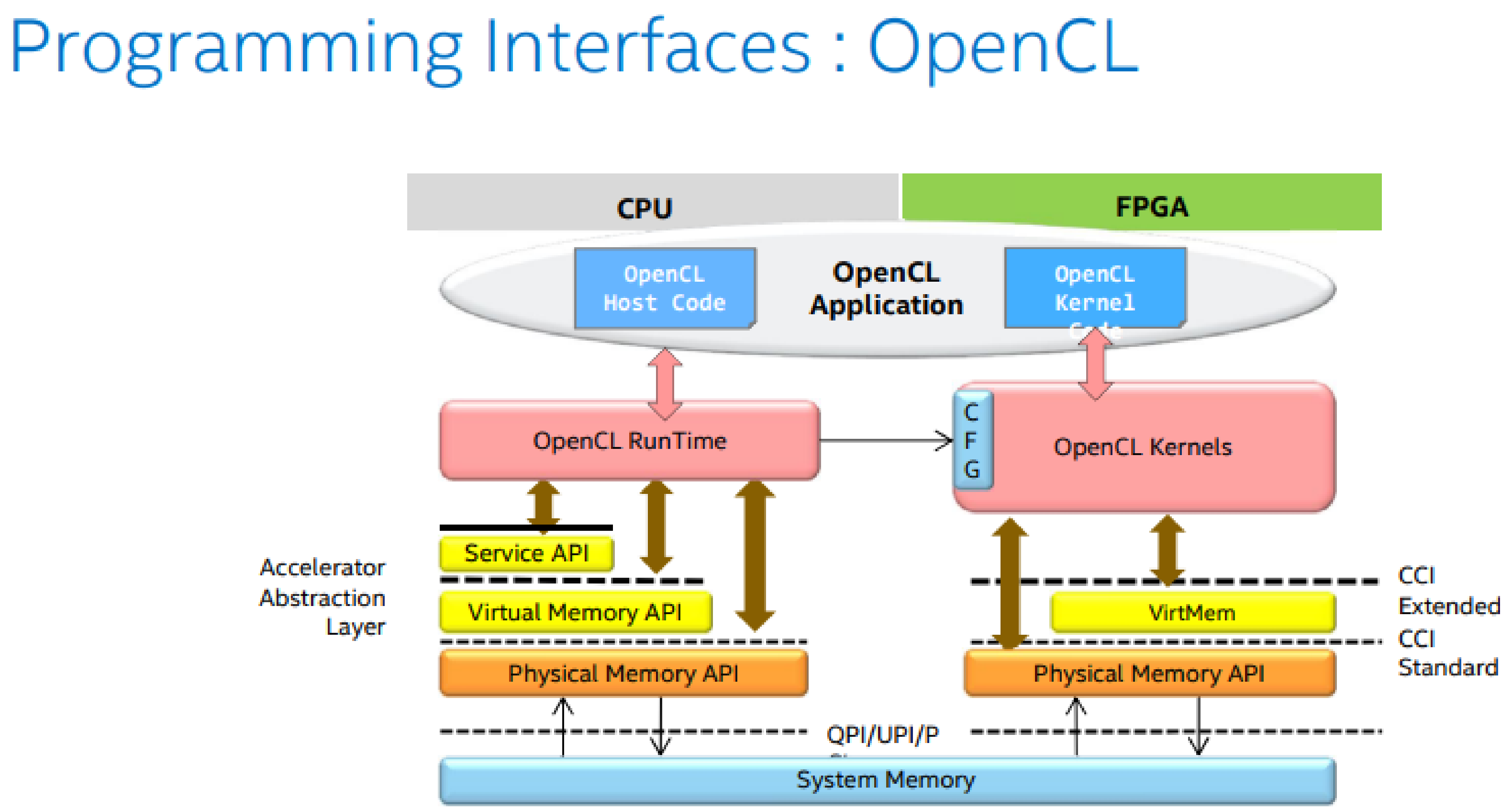
INTEL XEON-FPGA HYBRID CHIP



Processor	Intel® Xeon® E5-26xx v2 Processor
FPGA Module	Altera Stratix V
QPI Speed	6.4 GT/s full width (target 8.0 GT/s at full width)
Memory to FPGA Module	2 channels of DDR3 (up to 64 GB)
Expansion connector to FPGA Module	PCIe 3.0 x8 lanes - maybe used for direct I/O e.g. Ethernet
Features	Configuration Agent, Caching Agent,, (optional) Memory Controller
Software	Accelerator Abstraction Layer (AAL) runtime, drivers, sample applications

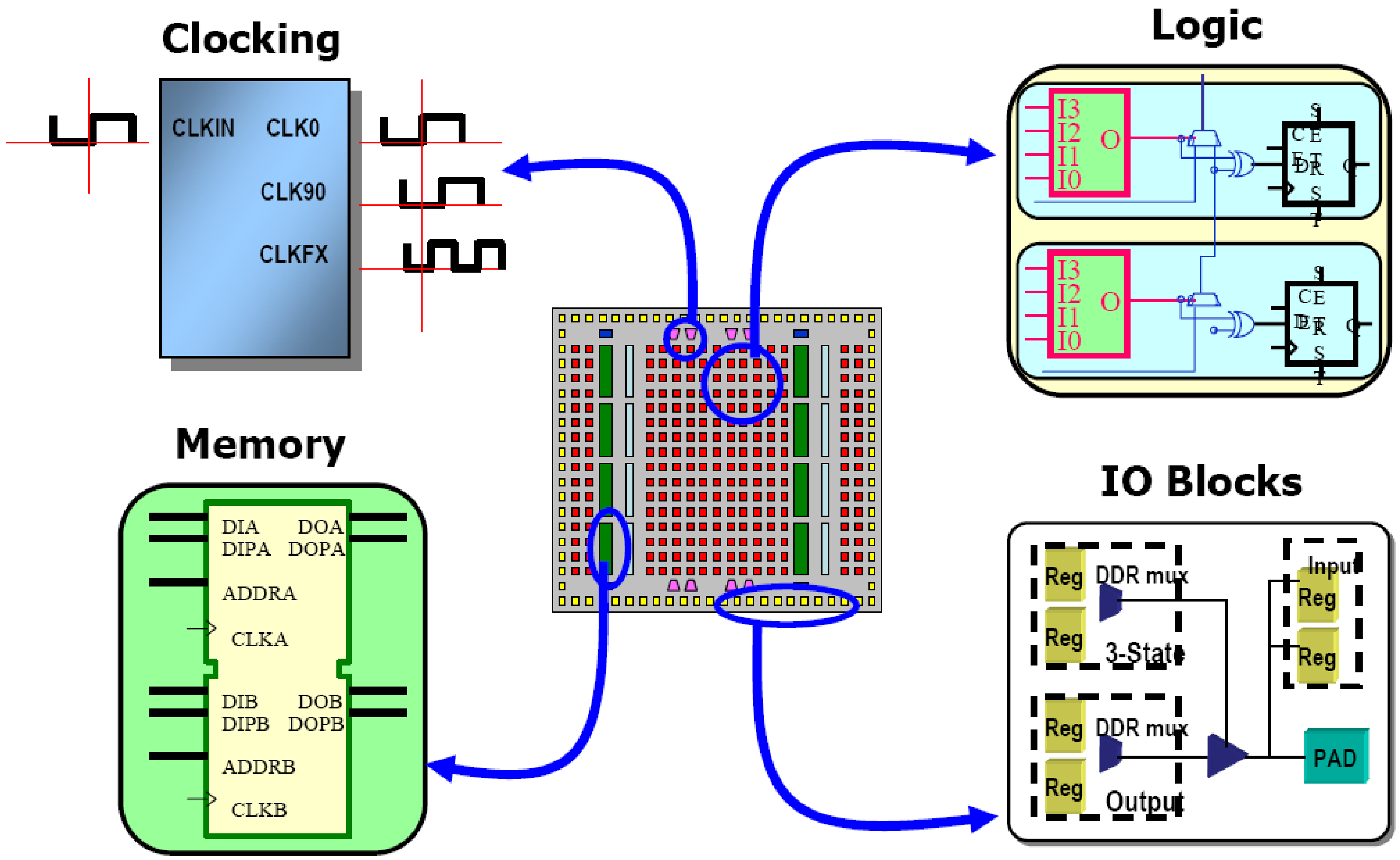
courtesy of Intel

INTEL XEON-FPGA HYBRID CHIP



courtesy of Intel

Summary



Clocking
CLKIN
CLK0
CLK90
CLKFX
Logic
I3
I2
I1
I0
O
CE
Memory
DIA
DIPA
DOA
DOPA
ADDRA
CLKA
DIB
DIPB
DOB
DOPB
ADDRB
CLKB
IO Blocks
Reg
DDR mux
3-State
Input
Reg
Reg
DDR mux
Output
PAD

Summary

- Reconfigurable SoC is trending, FPGAs are dominant
- FPGAs
  - Better for very rapid design and redesign.
  - Good for prototyping but also end design.
  - Better for small numbers of products
  - More expensive, and less in performance
  - more efficient than GPU
  - Spatial and temporal parallelism
  - Programmable SoCs can be used as accelerators, especially in data centers

Where are we Heading?

- Advanced Topics: Advanced Packaging for SoCs

Action Items

- Final projects – Start ASAP
- Reading Assignment
  - Ch. 6.5, Slides

Acknowledgement

Slides in this topic are inspired in part by material developed and copyright by:

- Synopsys Courseware
- Prof. Greg Stitt (U of Florida)