Reconfigurable Computing

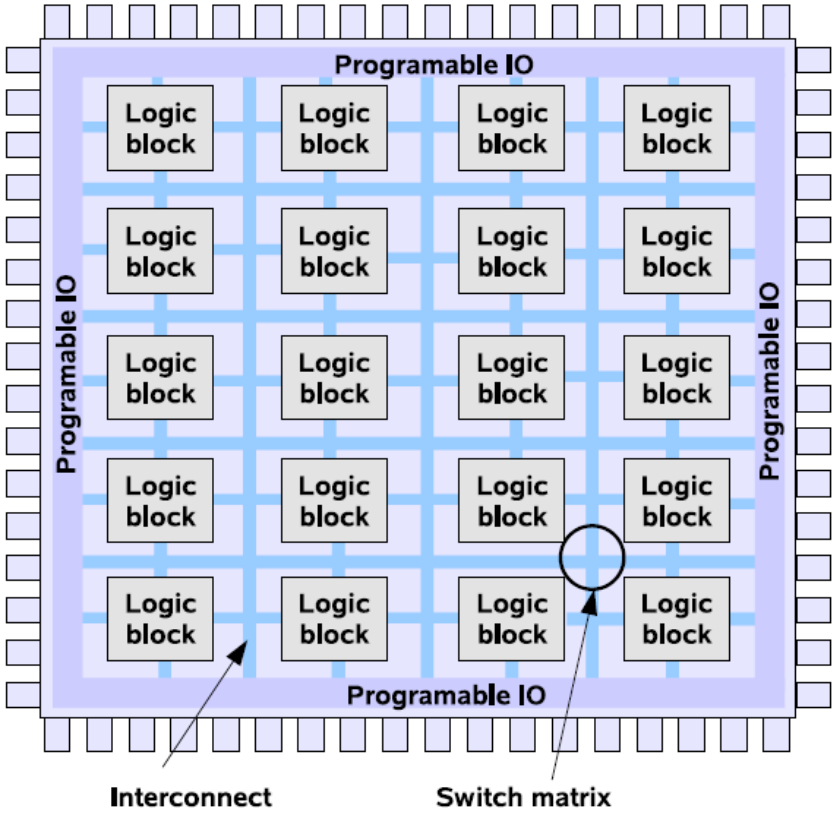
FPGA

° Introduced in 1985 by Xilinx

° Similar to CPLDs

° A function to be implemented in FPGA

- Partitioned into modules , each implemented in a logic block.
- Logic blocks connected with the programmable interconnection.



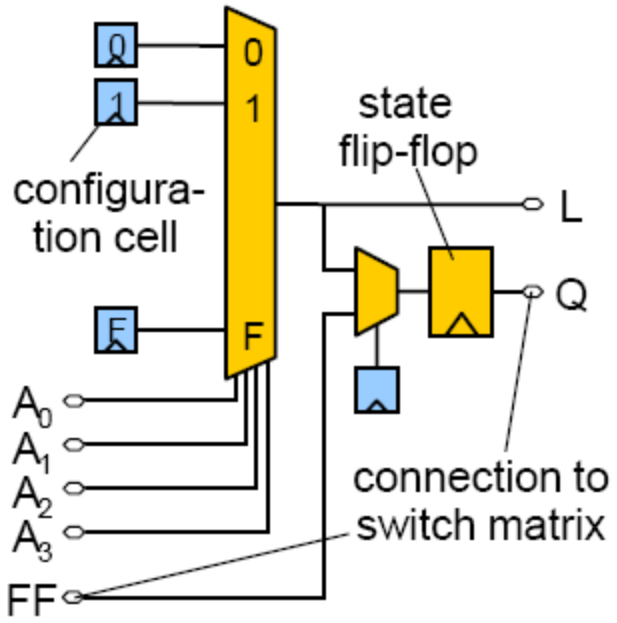
FPGA Components

° Problem: How to handle sequential logic
  - Truth tables don't work

° Possible solution:
  - Add a flip-flop to the output of LUT

° BLEs: the basic logic element
  - **Circuit can now use output from LUT or from FF**
  - **Where does select come from?**

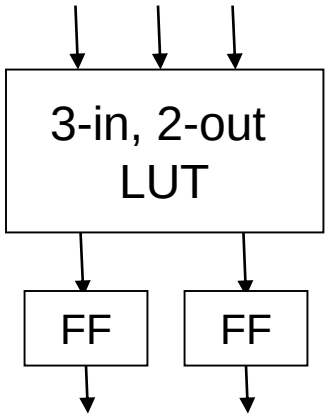
F(a3,a2,a1,a0)=….



FPGA Components

° Example: 8-bit register using 3-input, 2-output LUTs

- Input: x, Output: y

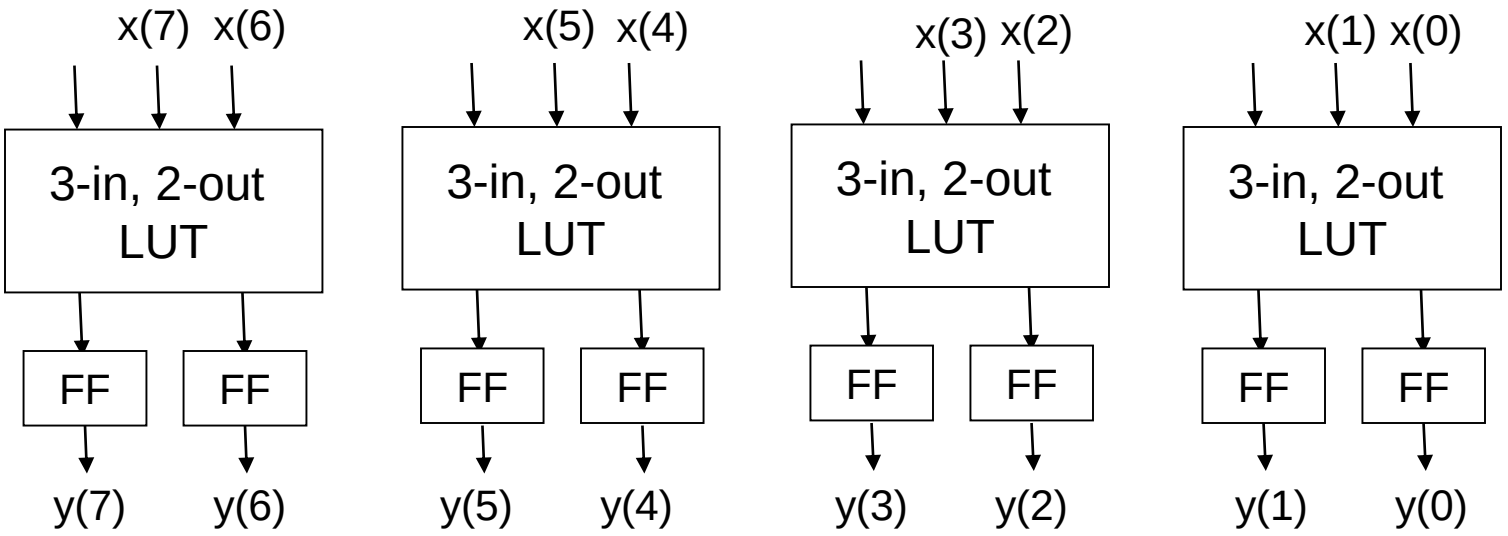


FPGA Components

° Example: 8-bit register using 3-input, 2-output LUTs

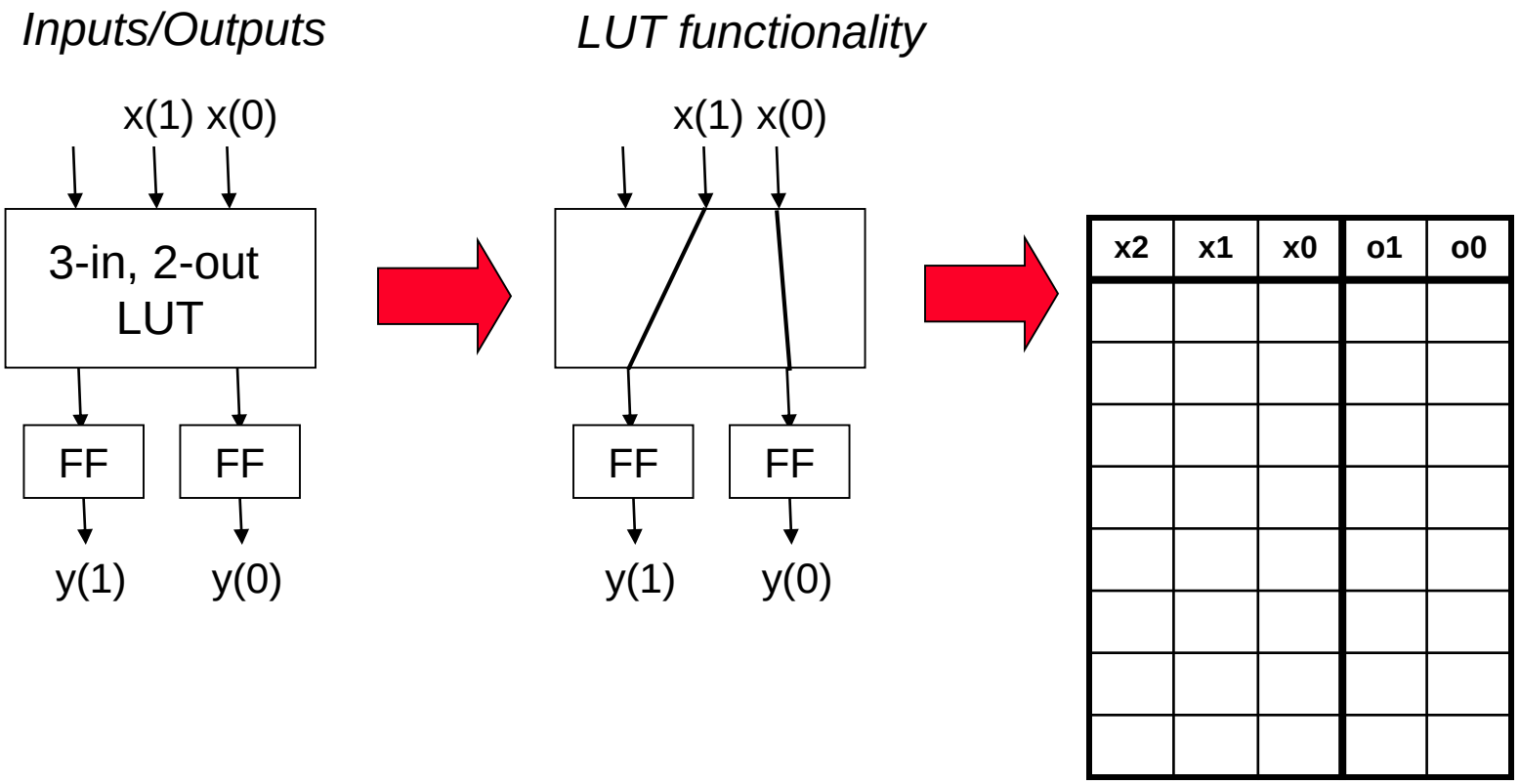
- Input: x, Output: y

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



FPGA Components

- **LUT simply passes inputs to appropriate output**

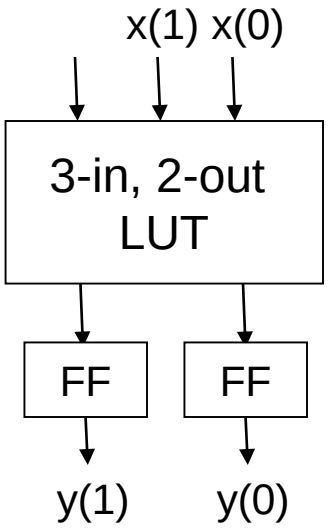


x2	x1	x0	o1	o0

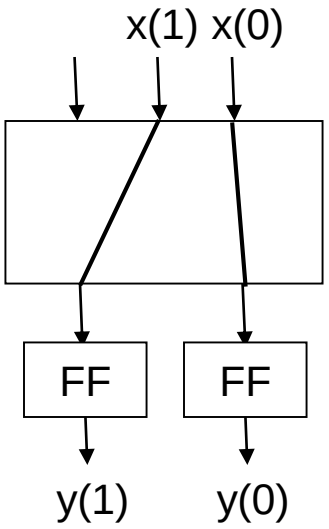
FPGA Components

- **LUT simply passes inputs to appropriate output**

Inputs/Outputs



LUT functionality



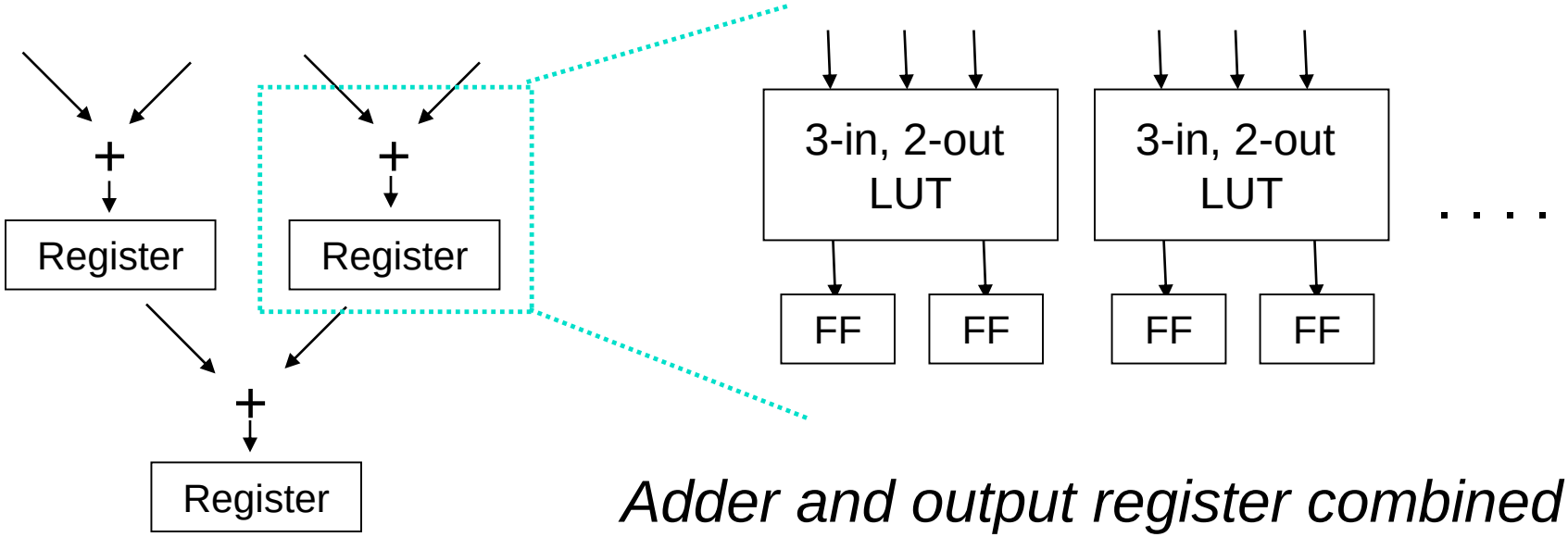
x2	x1	x0	o1	o0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

FPGA Components

° 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

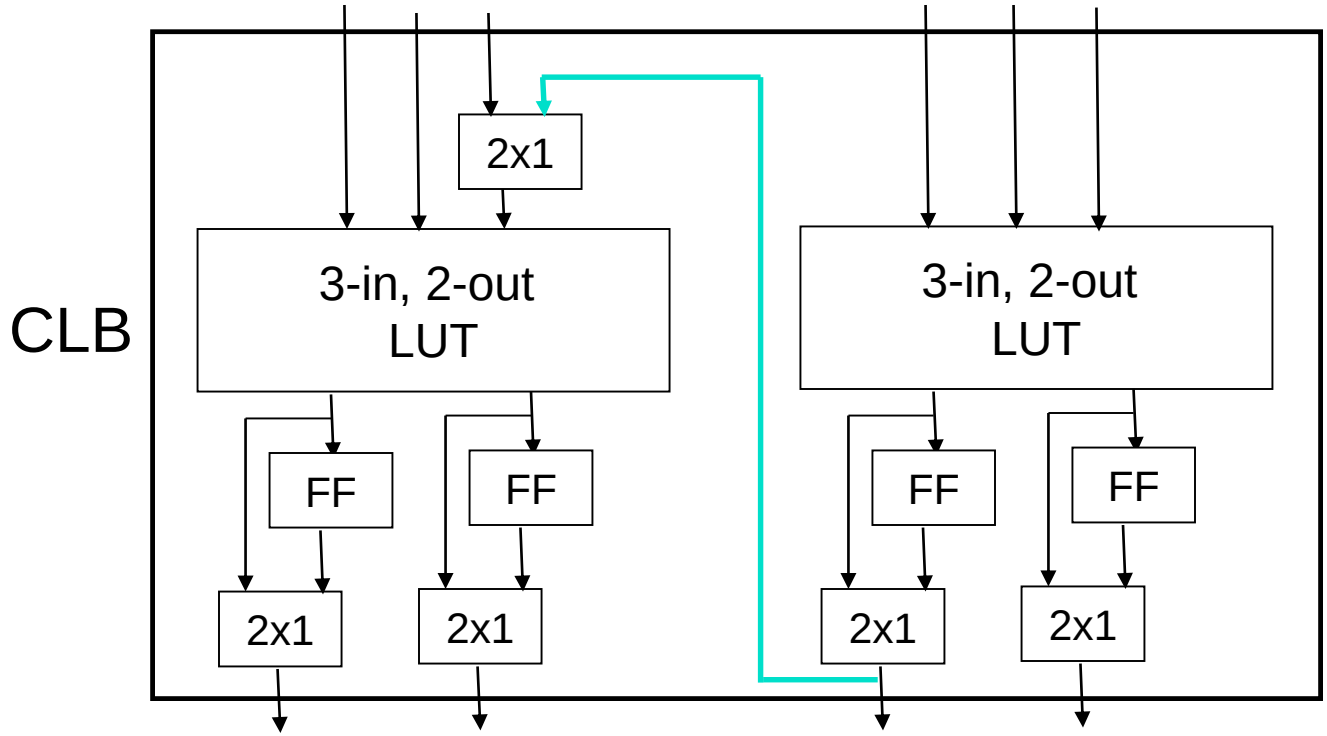


Adder and output register combined – not a separate LUT for each

FPGA Components

° Configurable Logic Blocks (CLBs) usually contain more than 1 BLE

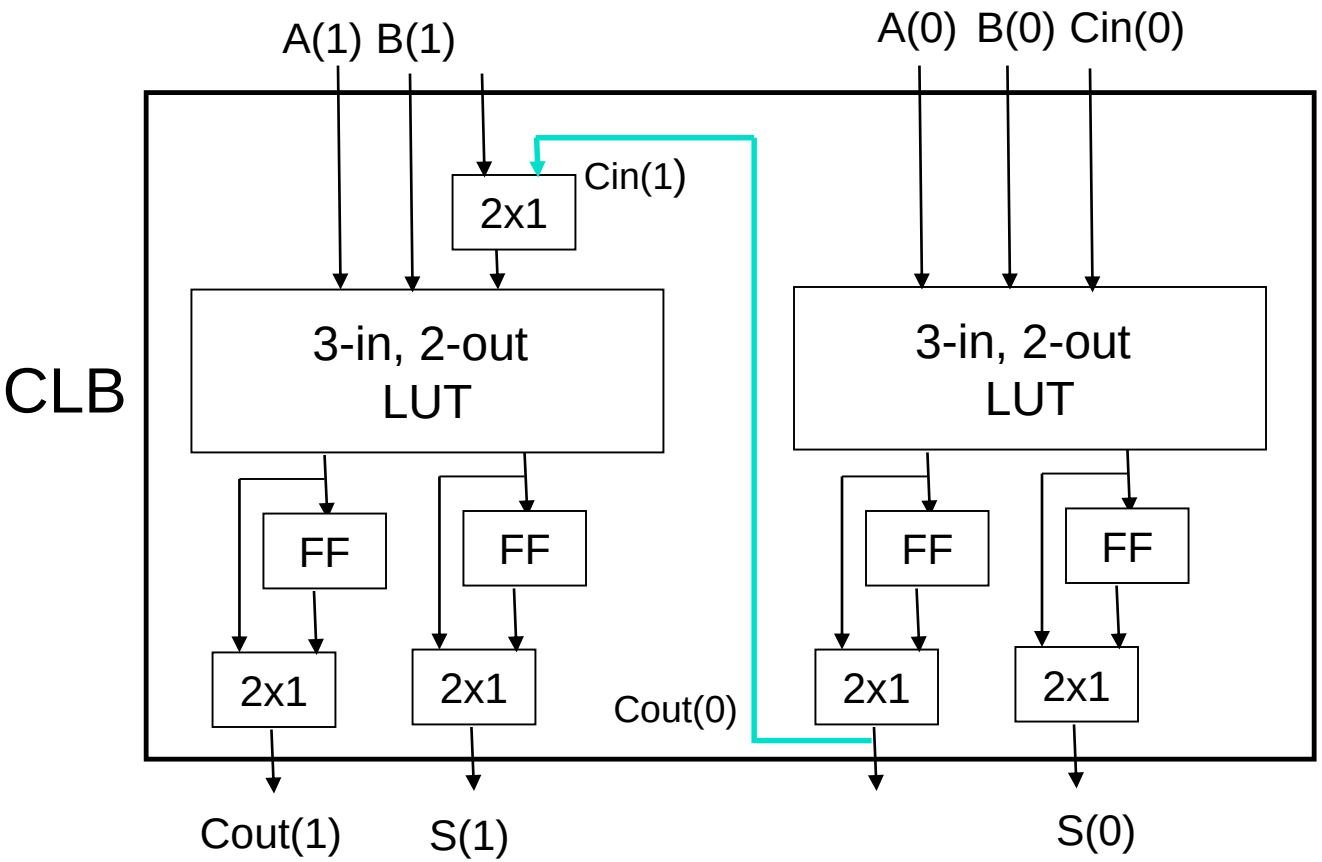
- Why?
  - Efficient way of handling common I/O between adjacent LUTs
  - Saves routing resources



FPGA Components

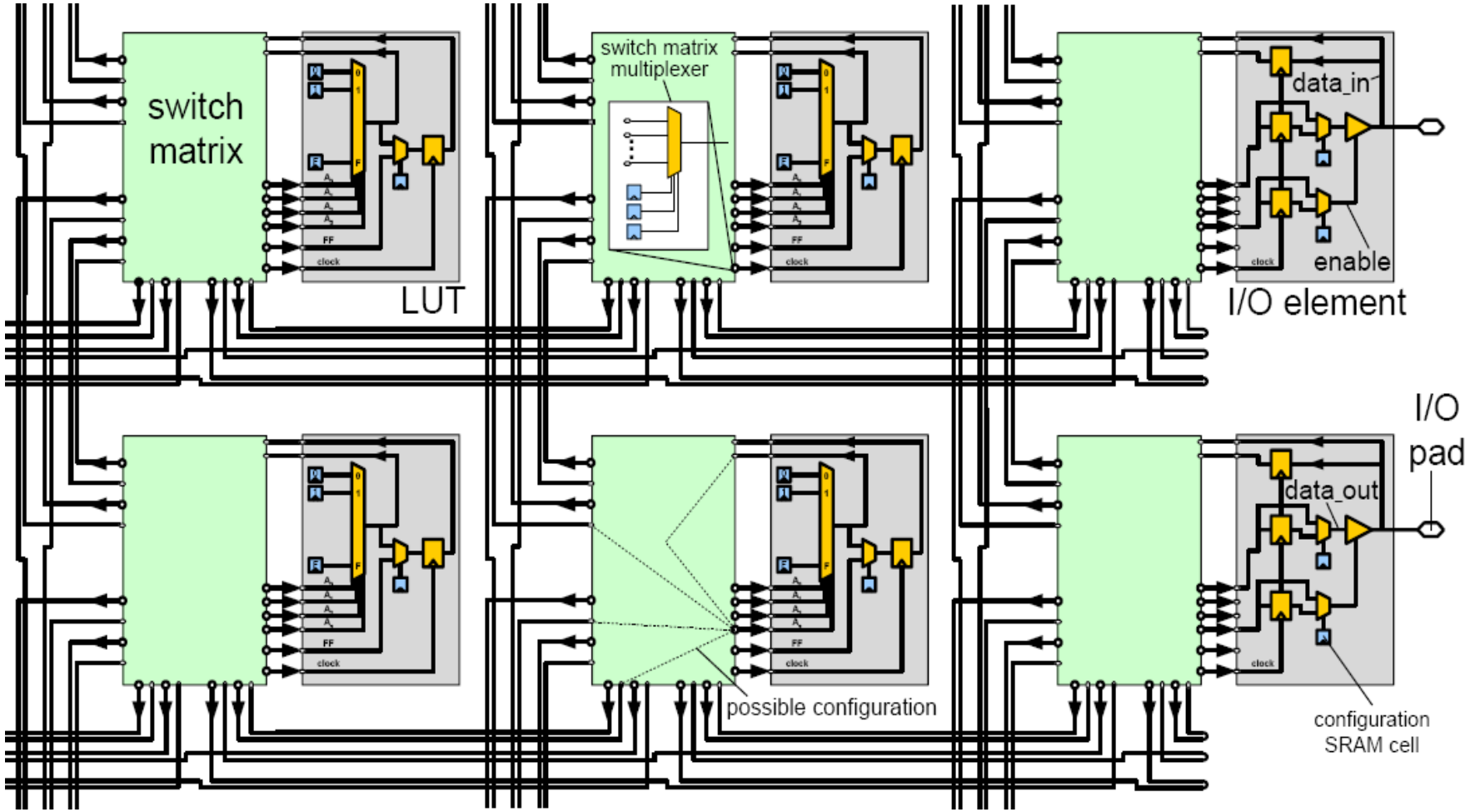
° Example: Ripple-carry adder

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



FPGA Components

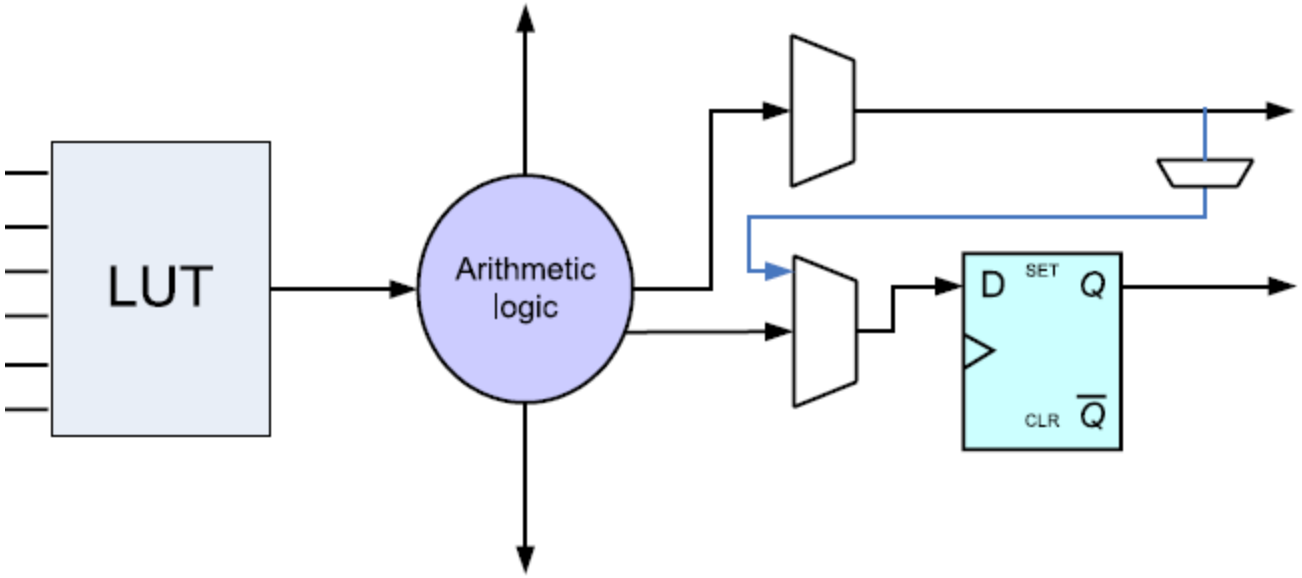
° On real FPGAs: a cluster of LUTs per switch matrix (e.g., eight LUTs and switch matrix form a configurable logic



Typical CLB

° The arithmetic logic provides

- XOR-gate and faster carry chain to build faster adder without wasting too much LUT-resources.



Xilinx CLBs

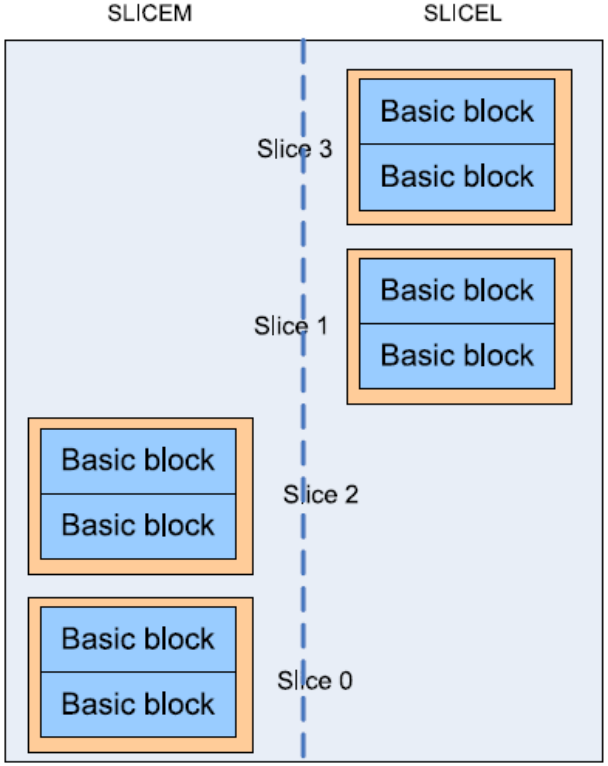
° Macro cells are CLBs.

° The number of basic blocks in a CLB varies from device to device.

- 4000, Virtex, Virtex E, Spartan: 1 slice, 2 basic blocks

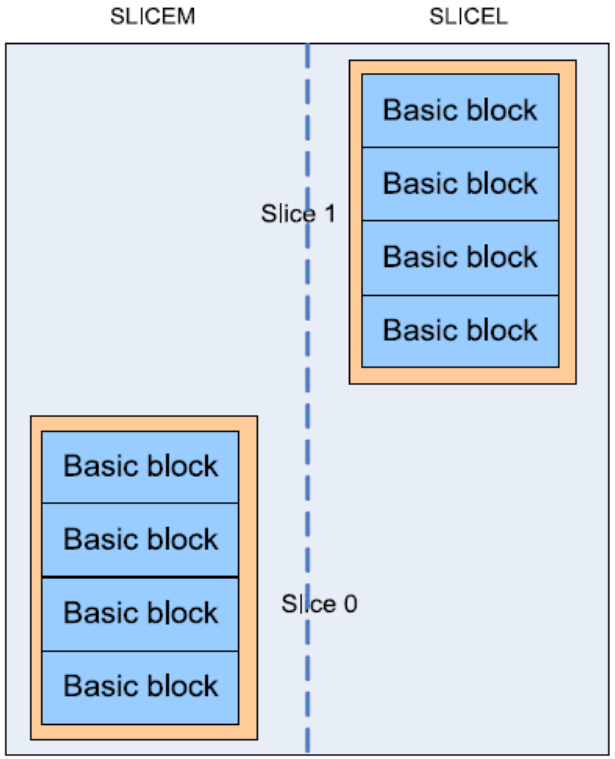
Spartan 3, VirtexII, Virtex II-Pro Virtex 4:

4 slices, 2 basic blocks/slice



Virtex 5:

2 slices, 4 basic blocks/slice



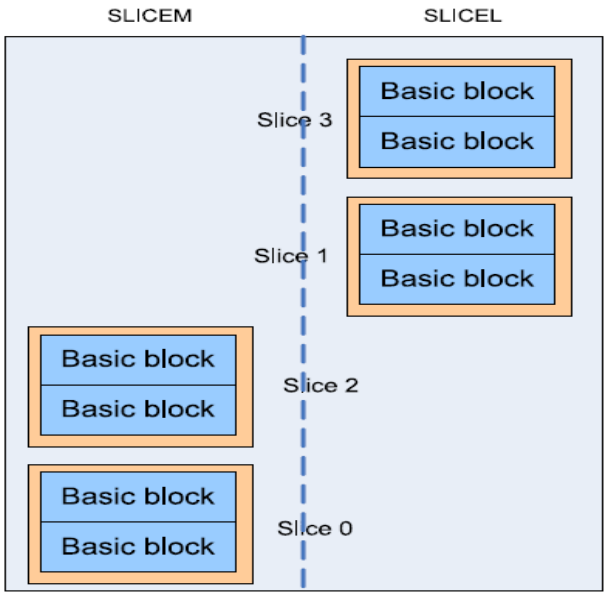
Xilinx CLBs

° Left part slices of a CLB (SLICEM)
  - configured either as combinatorial logic or SRAM or shift register

° SLICEL
  - only to be configured as combinatorial logic.

° Each BLE
  - 4 inputs and 1 output

Spartan 3, VirtexII, Virtex II-Pro Virtex 4:
4 slices, 2 basic blocks/slice



Xilinx CLBs

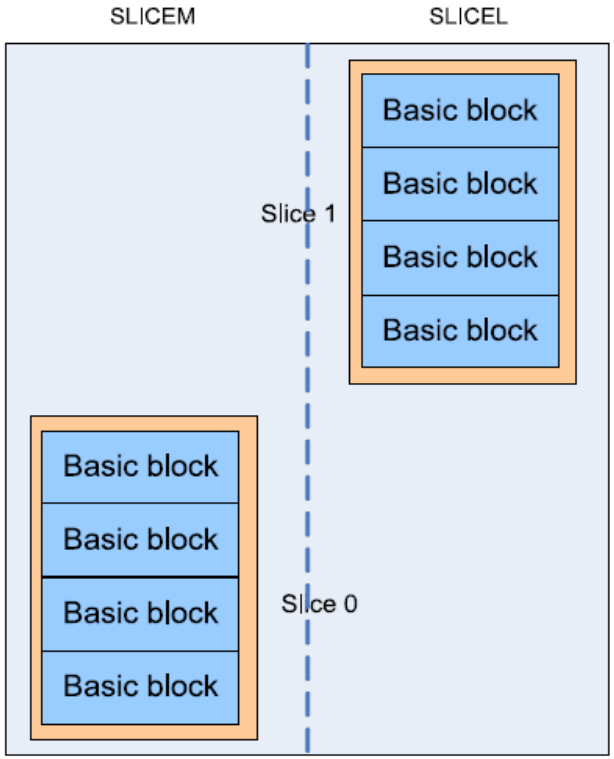
° LUT has 6 inputs and 2 outputs.

° LUT can be configured either as

- a 6-input LUT, in which case only one output can be used,
- or as two 5-input LUTs, two outputs used

Virtex 5:

2 slices, 4 basic blocks/slice



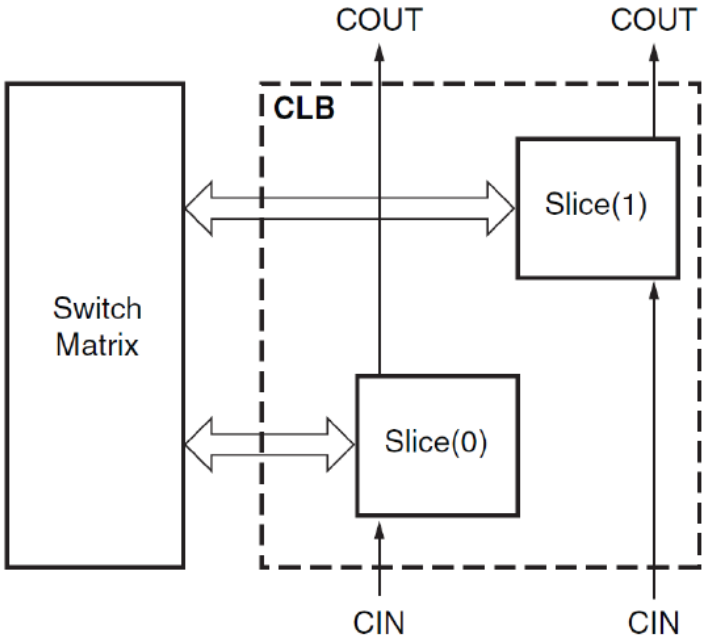
Xilinx Virtex6 CLB

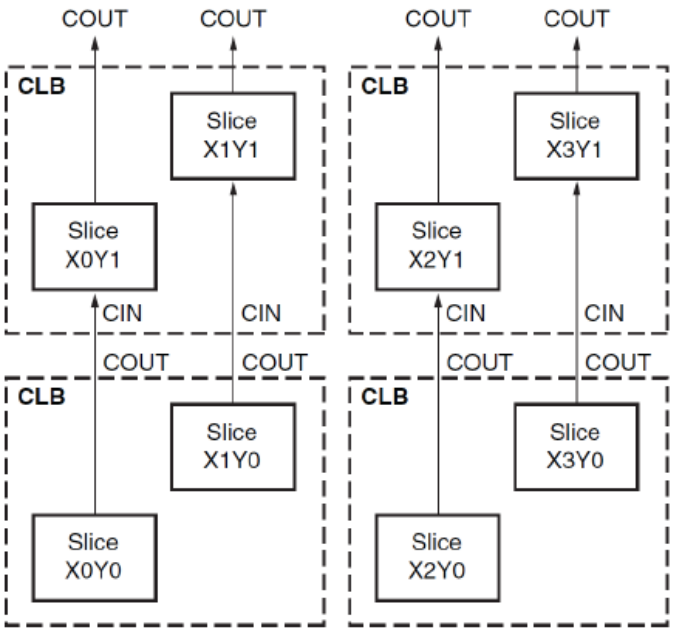
° A CLB contains 2 identical slices on Virtex 6

- 2 slices are split in two columns of 1 slices each

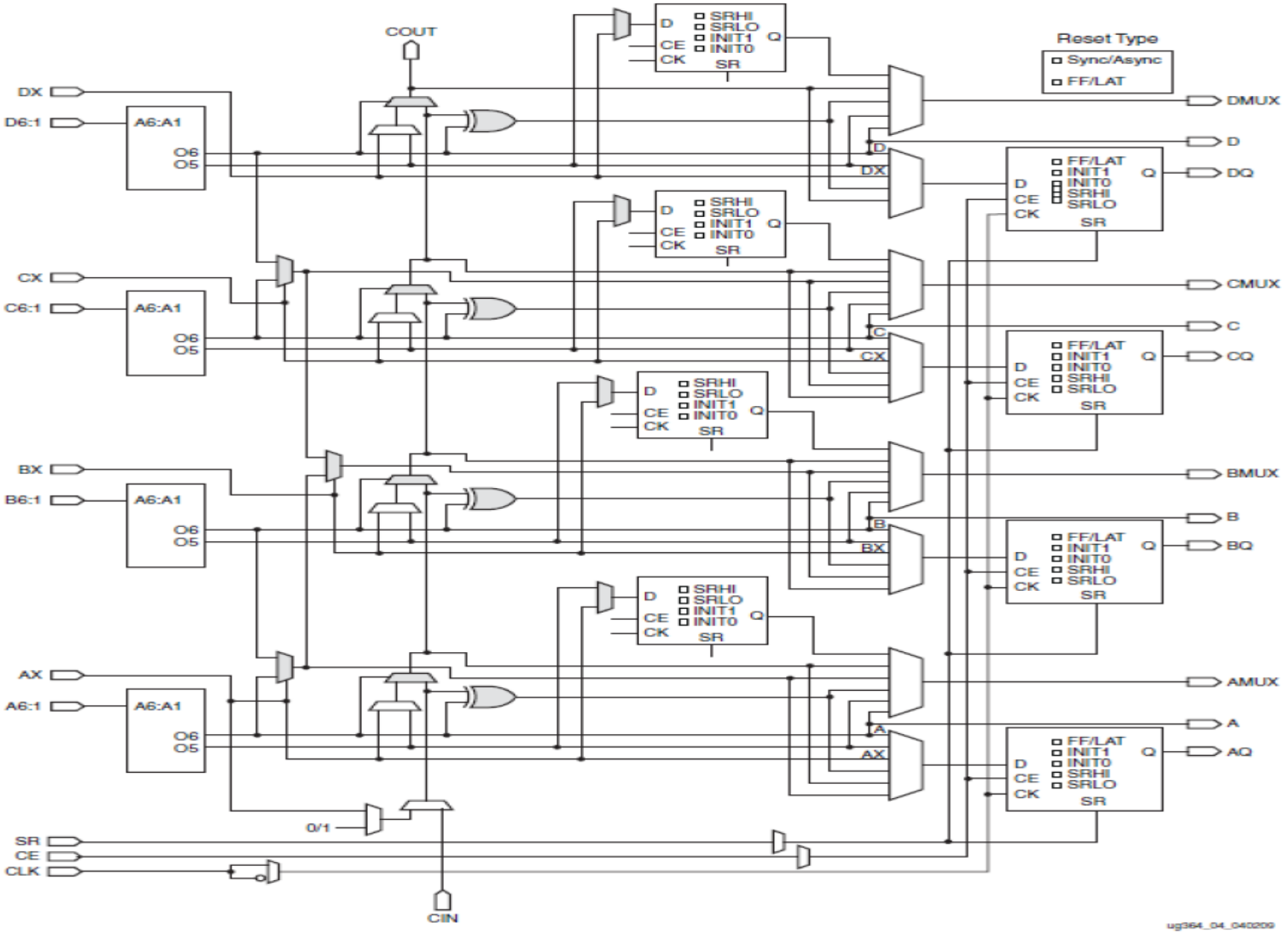
° 1 slice contains:

- 4x 6-inputs LUT
- 8x FF for storing LUT results
- MUX to feed LUT either to a FF or the output
- Carry in and carry out to construct fast adder using neighbor CLBs





Xilinx Virtex6 CLB



Altera FPGA Basic Blocks

° Altera's FPGAs (Cyclone, FLEX)

- basic unit of logic is the ***logic element*** **(LE)**
- also LUT-based
- a 4-LUT, a flip flop, a multiplexer and additional logic for carry chain
- LEs can operate in different modes each of which defines different usage of the LUT inputs.

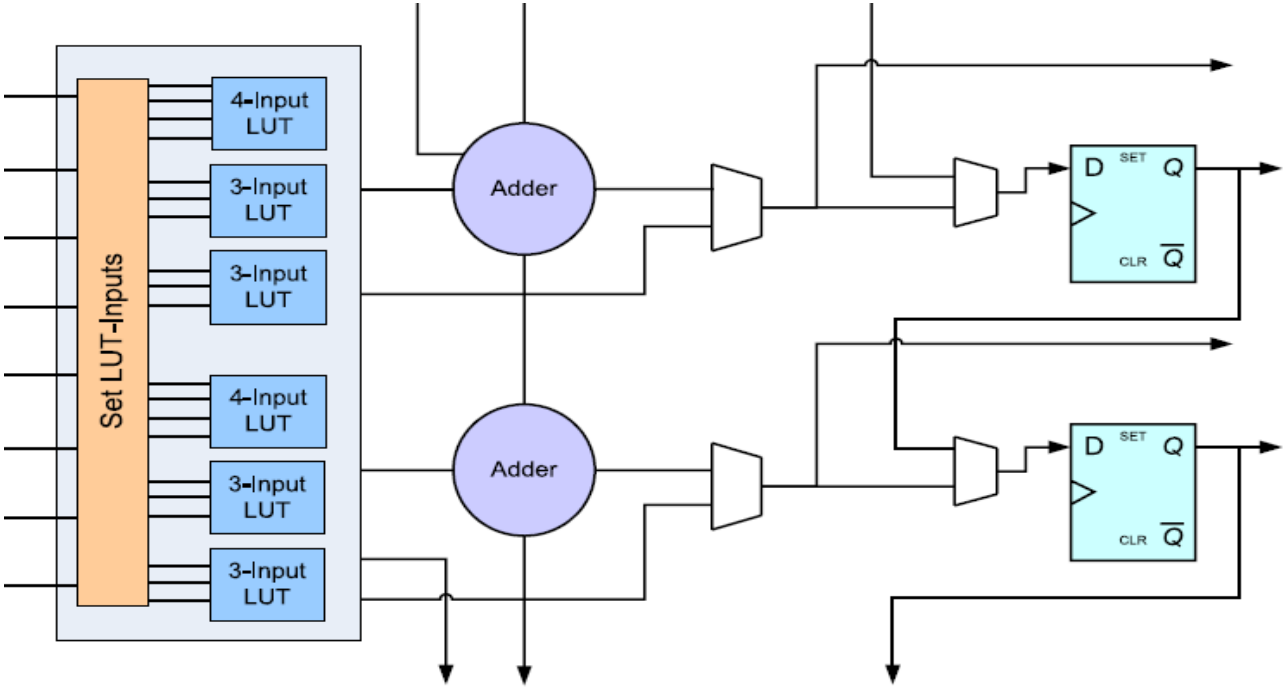
° Altera LEs

- grouped into ***logic array blocks*** **(LAB).**
- Flex 6000 LAB contains 10 LEs
- FLEX 8000 LAB contains 8 LEs.
- Cyclone II LAB contains 16 LEs

Altera FPGA Basic Blocks

° Stratix II

- basic computing unit is called ***adaptive logic module*** **(ALM)**
- **Each LAB contains 8 ALMs**
- ALM can be used to implement functions with variable number of inputs.
- Ensures a backward compatibility to 4-input-based designs,
- Possible to implement module with up to 8 inputs.
- Additional modules: including flip flops, adders and carry logic



FPGA Components

° CLBs often have specialized connections between adjacent CLBs
  - Further improves carry chains
  - Avoids routing resources

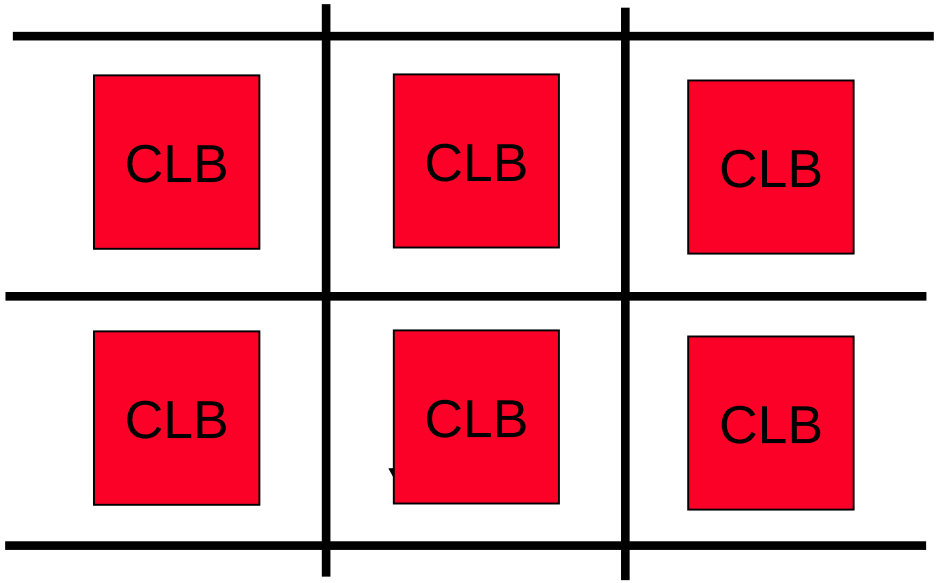
° Basic building block is CLB
  - Can implement combinational+sequential logic
  - All circuits consist of combinational and sequential logic

° So what else is needed?
  - FPGAs need some way of connecting CLBs together
    - Reconfigurable interconnect
    - But, we can only put fixed wires on a chip

FPGA Components

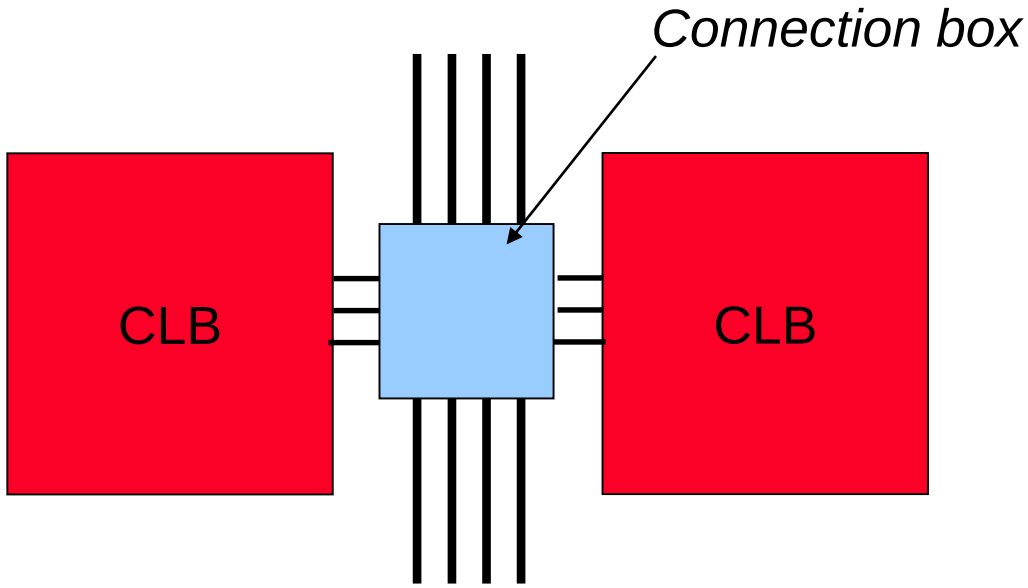
° Problem: If FPGA doesn't know which CLBs will be connected, where does it put wires?

- 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



FPGA Components

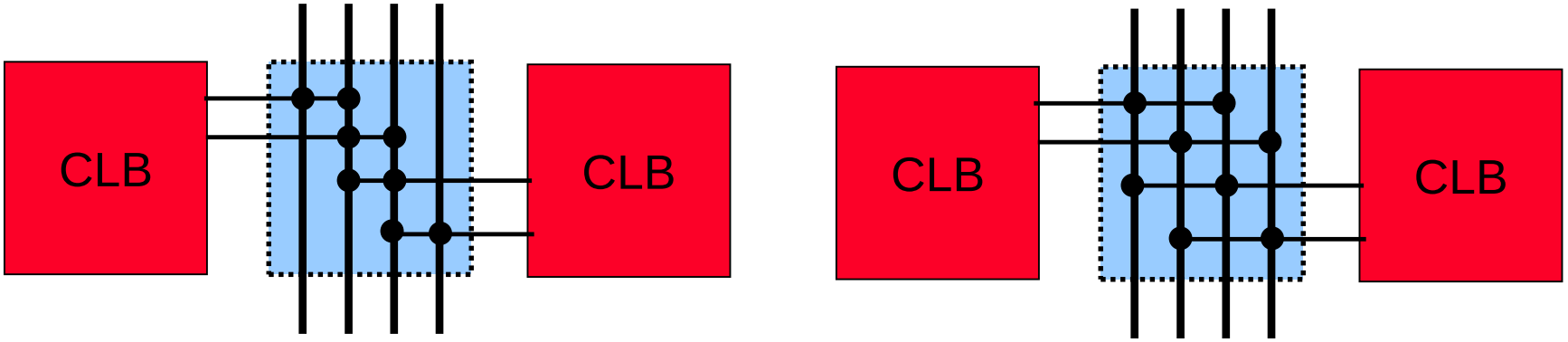
- How to connect CLB to wires?
- Solution: Connection box
  - Device that allows inputs and outputs of CLB to connect to different wires



FPGA Components

° Connection box characteristics

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



FPGA Components

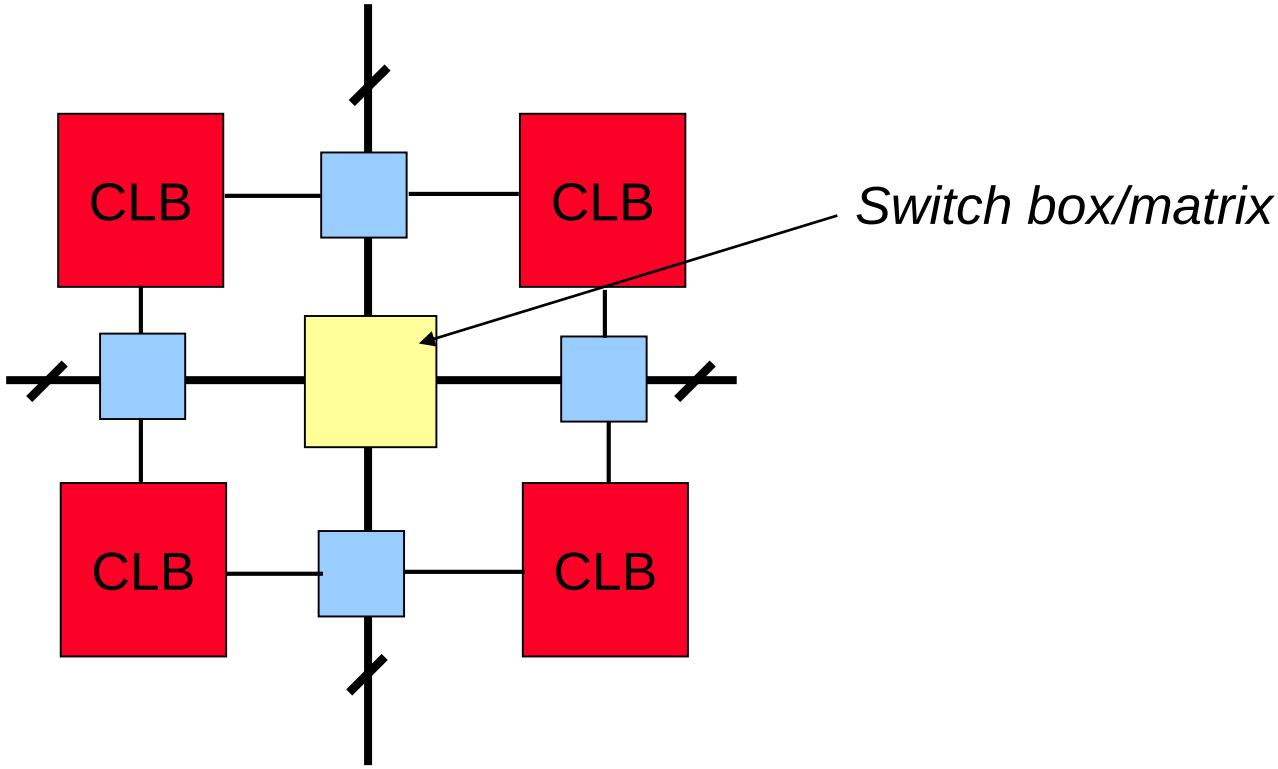
° Connection boxes allow CLBs to connect to routing wires
  - But, that only allows us to move signals along a single wire
  - Not very useful

° How do FPGAs connect wires together?

FPGA Components

° Solution: Switch boxes, switch matrices

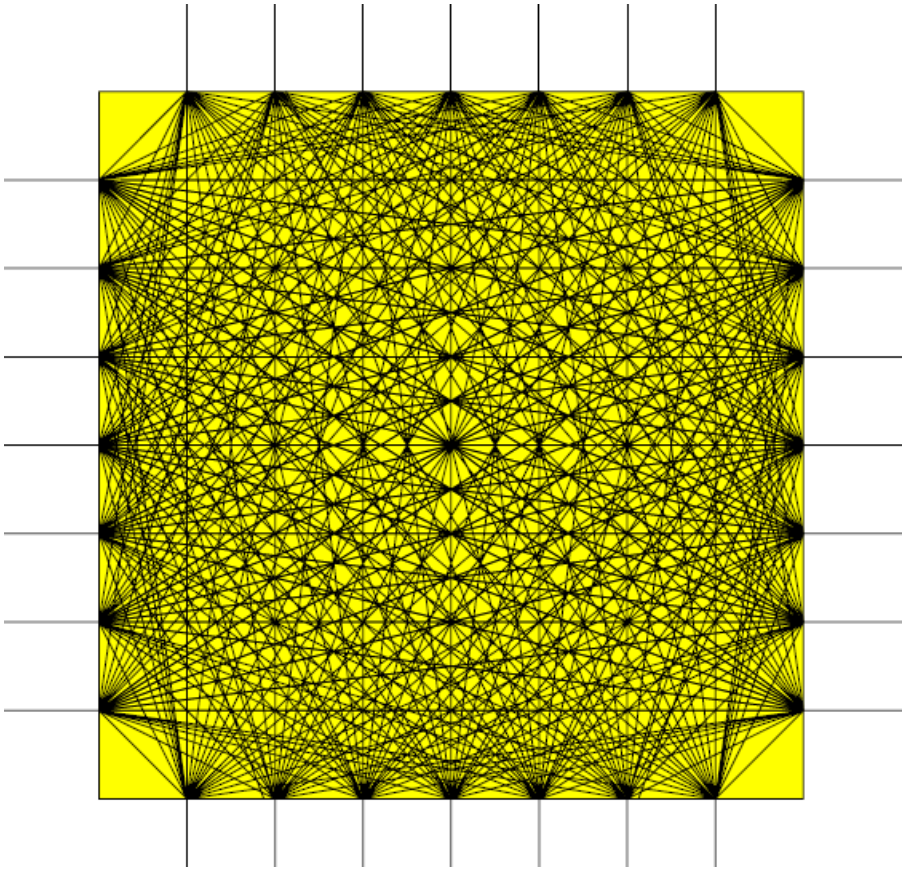
- Connects horizontal and vertical routing channels
  - But, we can only put fixed wires on a chip



FPGA Components

° Switch boxes
  - Flexibility - defines how many wires a single wire can connect to

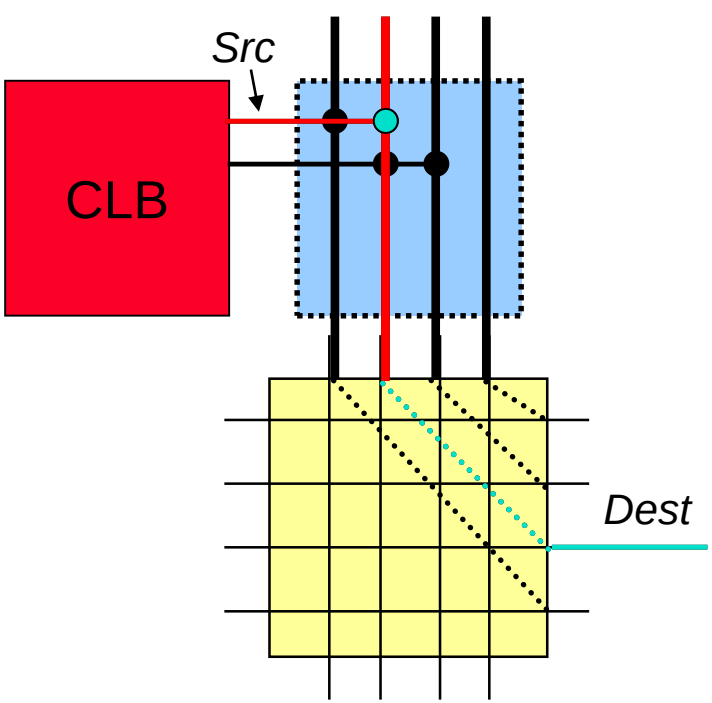
° Every possible connection?
  - Too big
  - Too slow

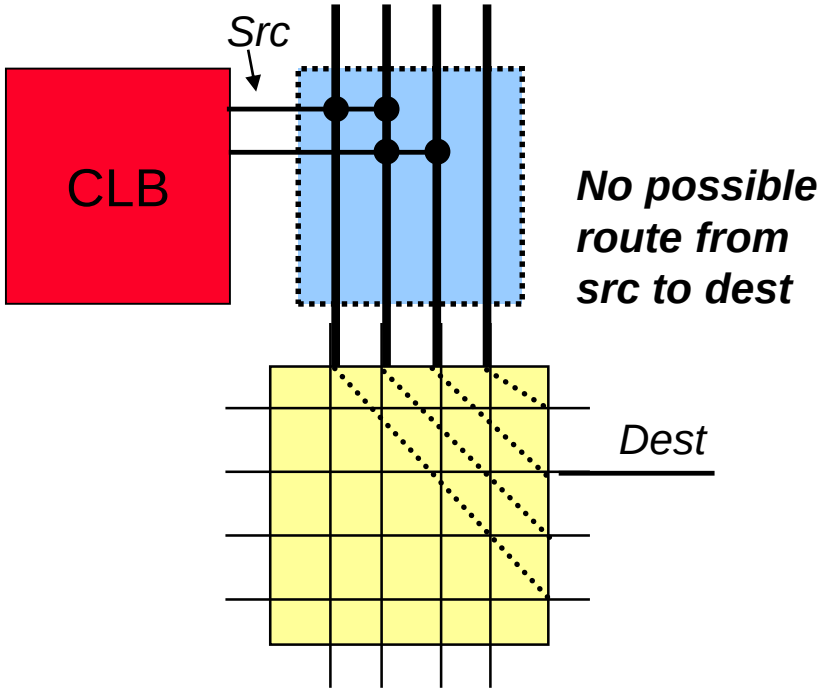


FPGA Components

° 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

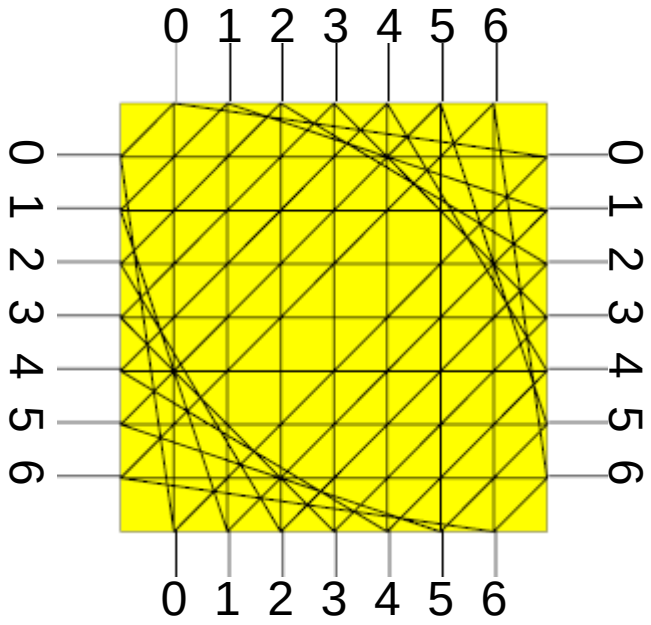




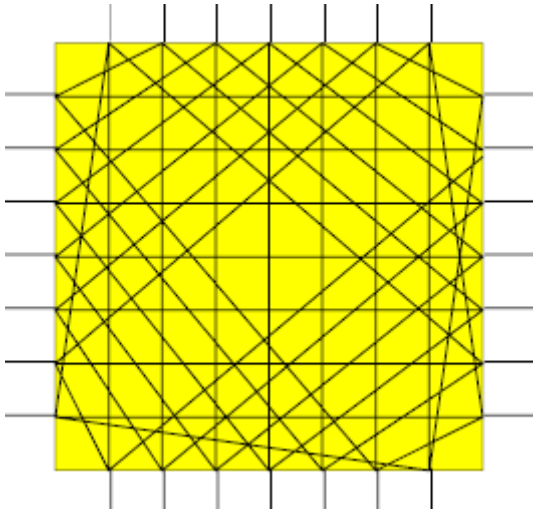
FPGA Components

° Many Topologies possible

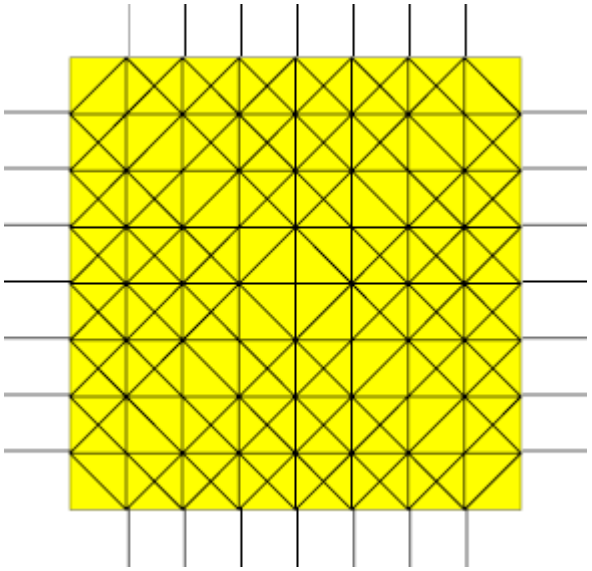
- Fs = 3 is common



Disjoint



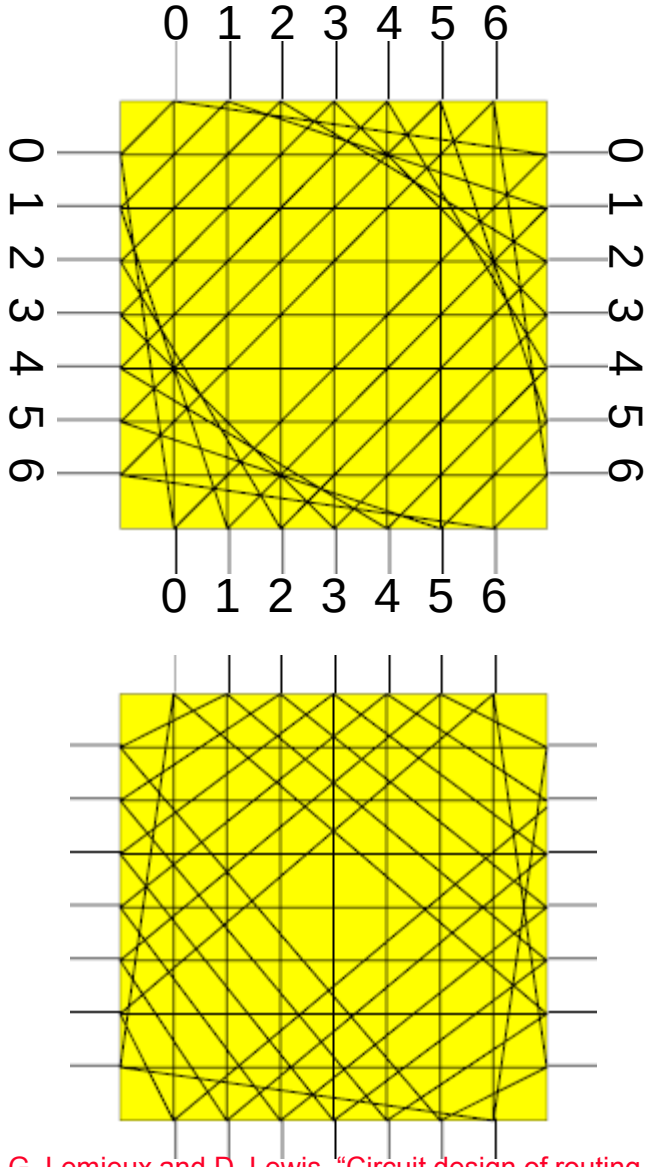
Wilton



Universal

° Topology - defines which wires can be connected

FPGA Components



° Disjoint: a wire entering can only connect to other wires with the same numerical designation.

- potential source–destination routes in the FPGA are isolated into distinct routing domains, limiting routing flexibility.

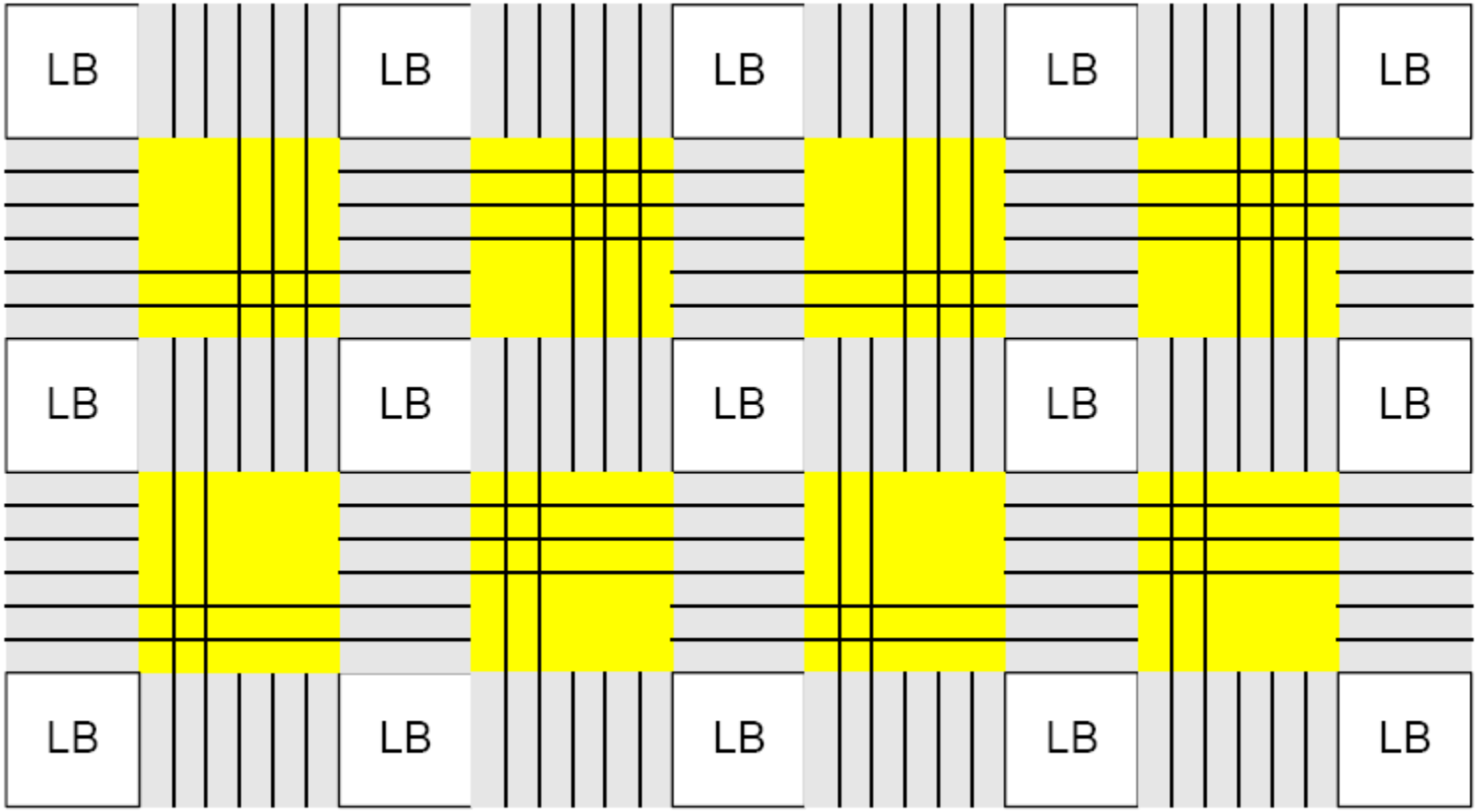
° Wilton: uses same number of routing switches but overcomes the domain issue

- By allowing for a change in domain assignment on connections that turn.
- ability to change domains in at least one direction facilitates routing as a greater diversity of routing paths from a net source to a destination is possible.

G. Lemieux and D. Lewis, “Circuit design of routing switches,” in *Proceedings: ACM/SIGDA International Symposium on Field Programmable Gate Array*, pp. 19–28, February 2002.
G. Lemieux and D. Lewis, *Design and Interconnection Networks for Programmable Logic*. Boston, MA: Kluwer Academic Publishers, 2004.

FPGA Components

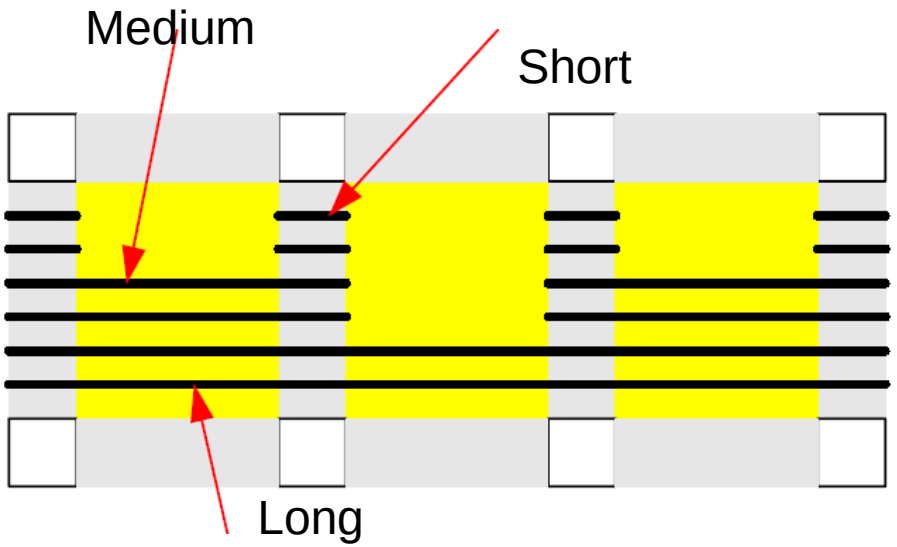
° At each switch block: some tracks end some tracks pass right through



FPGA Components

° Switch boxes

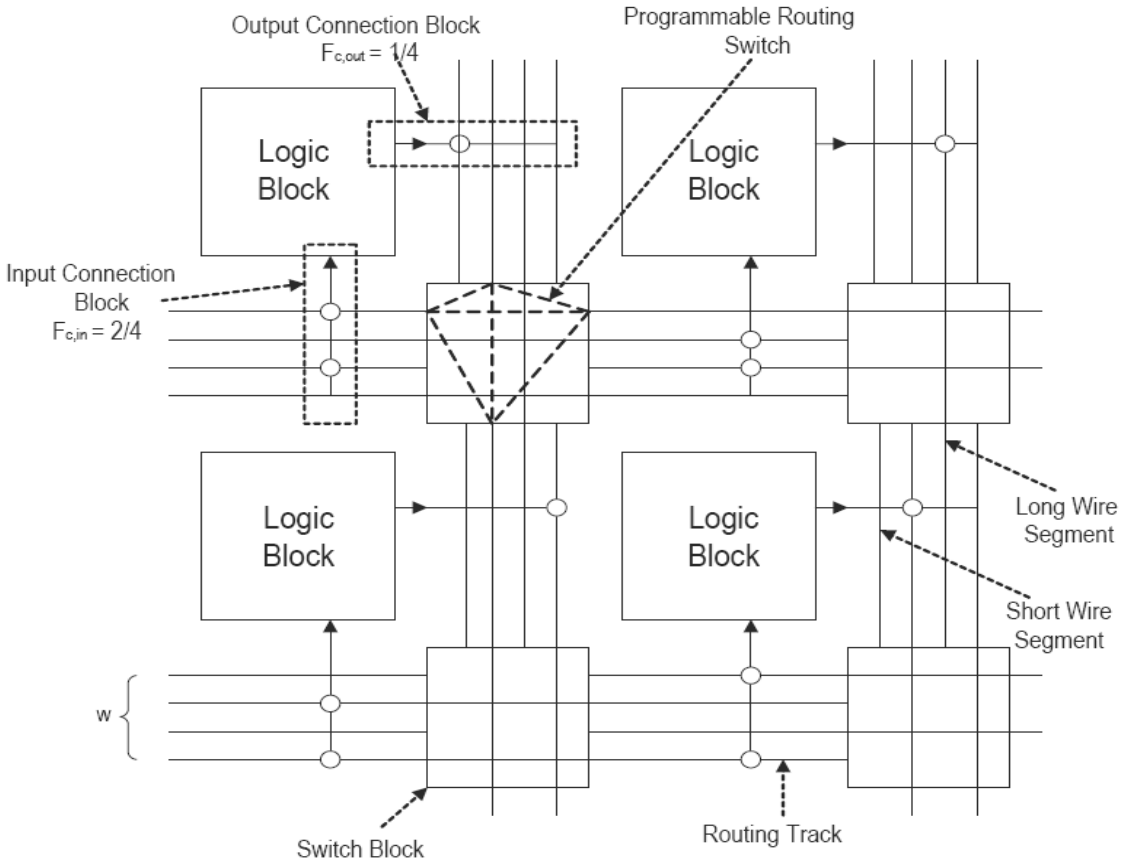
- Short channels
  - Useful for connecting adjacent CLBs
- Long channels
  - Useful for connecting CLBs that are separated
  - Allows for reduced routing delay for non-adjacent CLBs



FPGA Components

° FPGA layout called a “fabric”

- 2-dimensional array of CLBs and programmable interconnect
- Sometimes referred to as an “island style” architecture



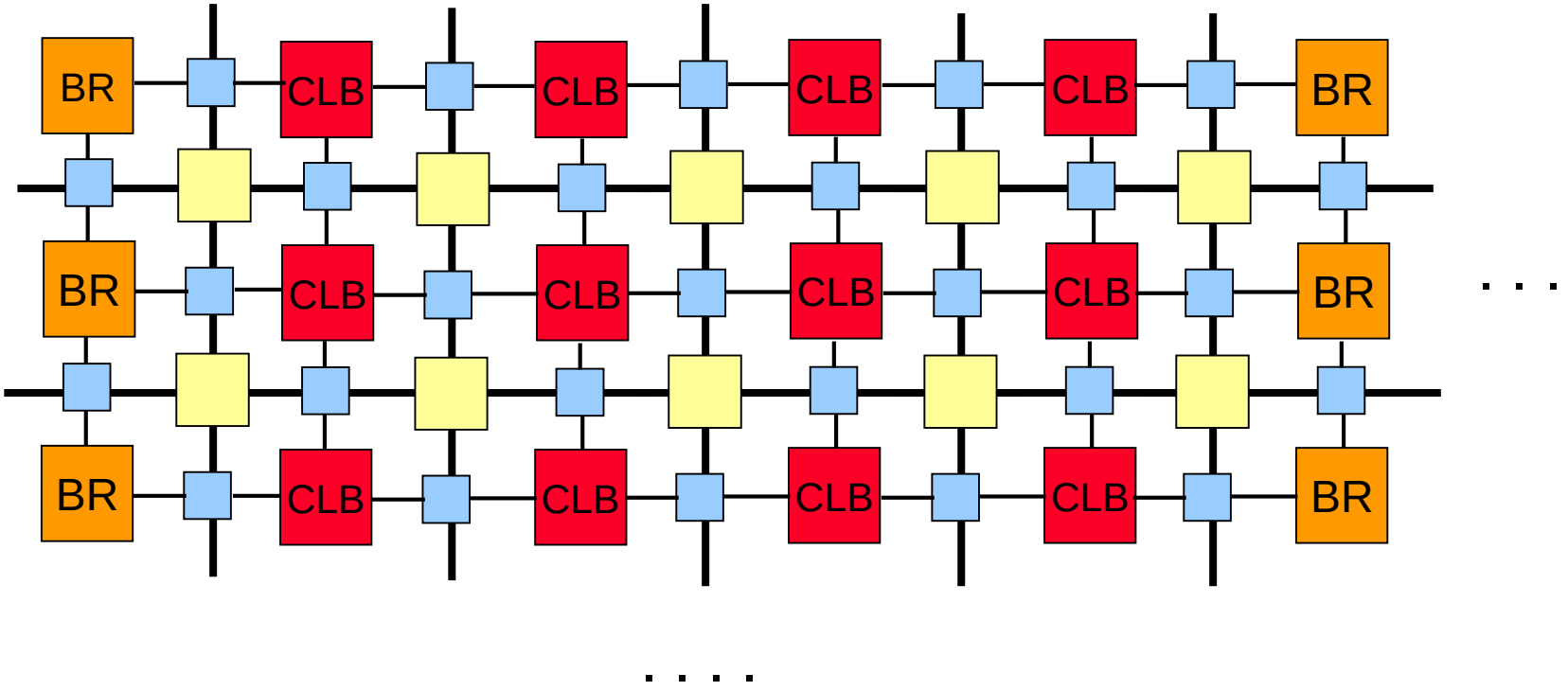
FPGA and Data Storage

- Solution 1: Use LUTs for logic or memory
  - LUTs are just an SRAM
  - 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

FPGA Components

° Fabric with Block RAM

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



FPGA Components

° FPGAs commonly used for DSP apps

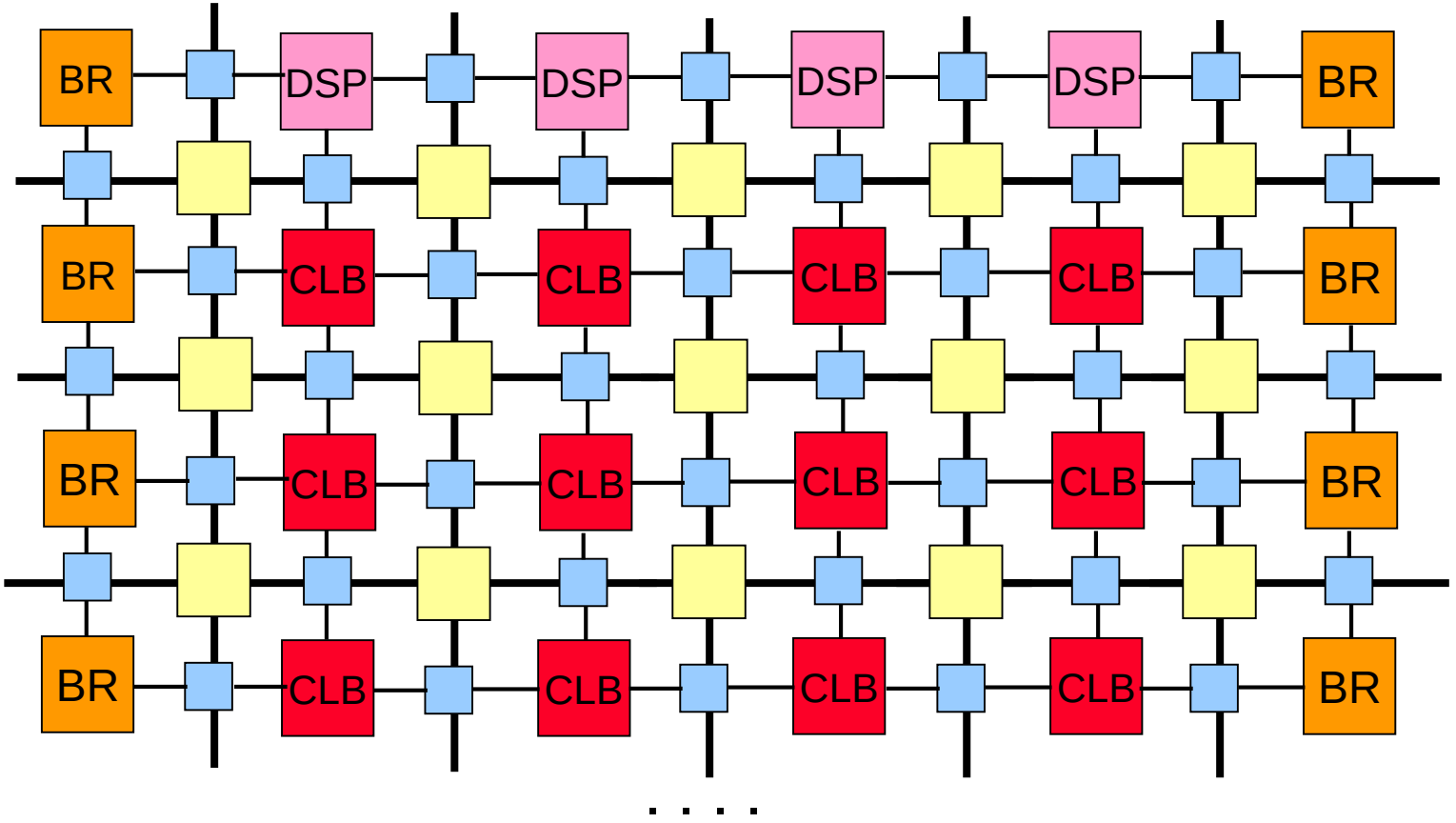
- Makes sense to include custom DSP units instead of mapping onto LUTs
  - Custom unit = faster/smaller

° Example: Xilinx DSP48

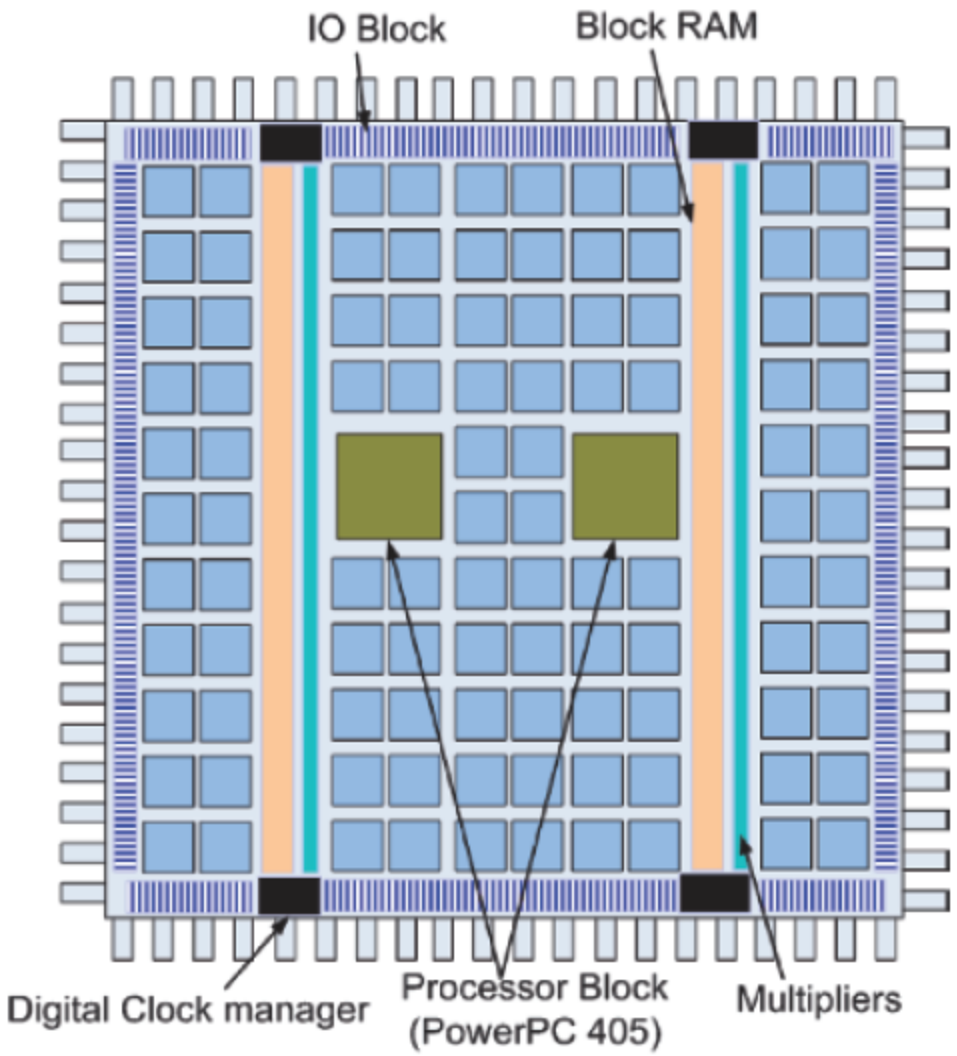
- Starting with Virtex 4 family, Xilinx introduced DSP48 block for high-speed DSP on FPGAs
- Essentially a multiply-accumulate core with many other features
- Provides efficient way of implementing
  - Add/subtract/multiply
  - MAC (Multiply-accumulate)
  - Barrel shifter
  - FIR Filter
  - Square root

FPGA Components

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

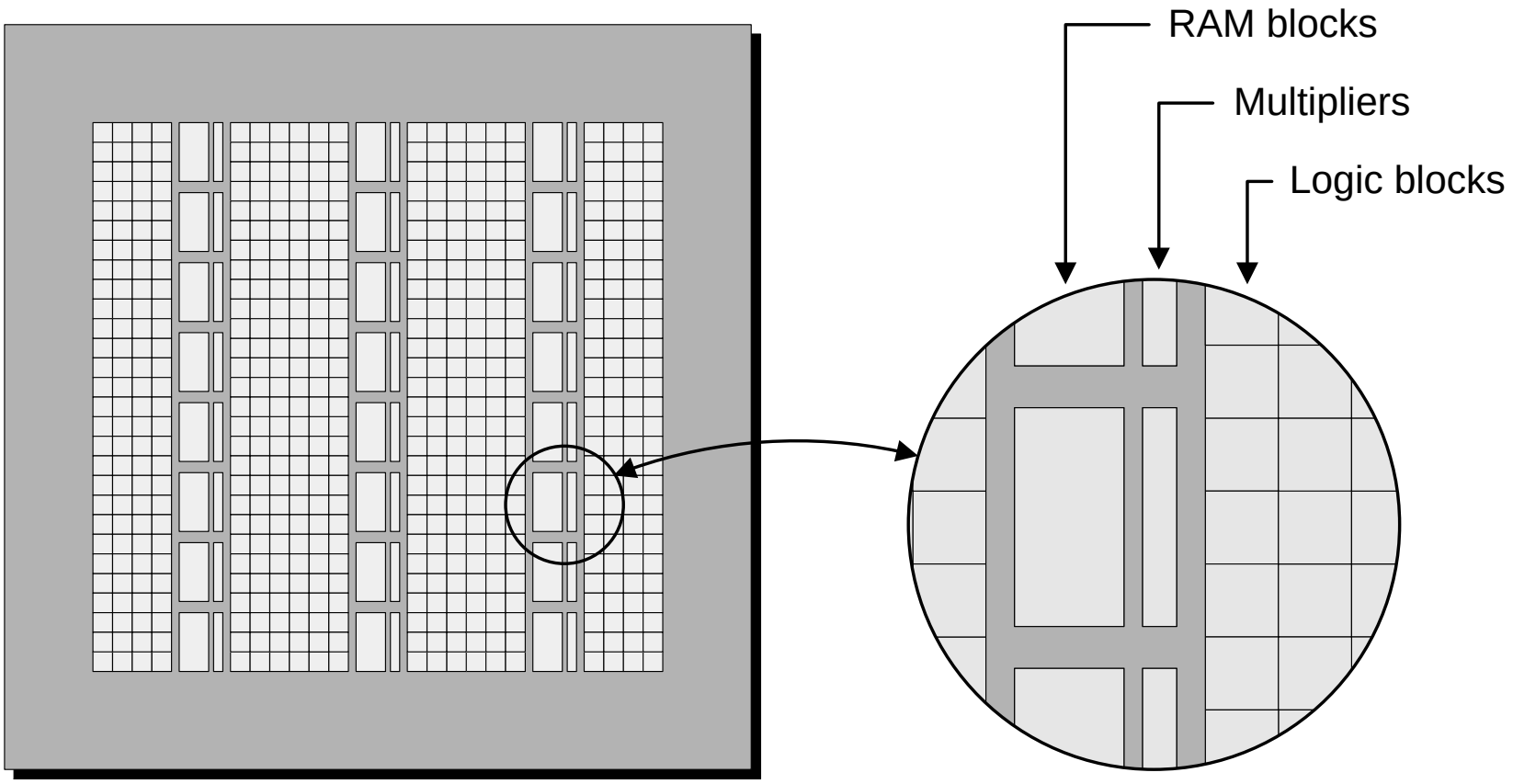


FPGA Components



Xilinx Virtex II Pro FPGA

Spartan3 Components

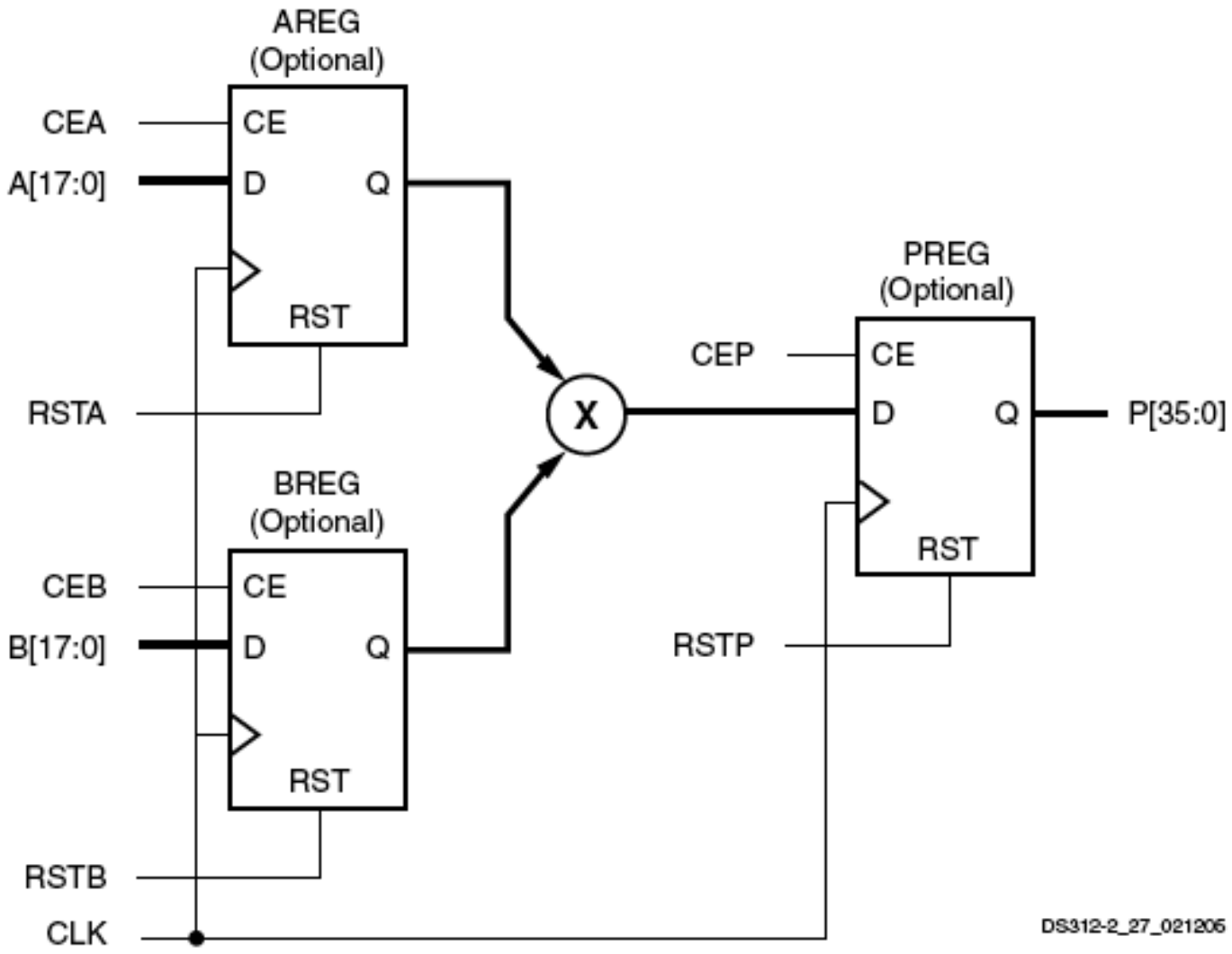


Spartan 3

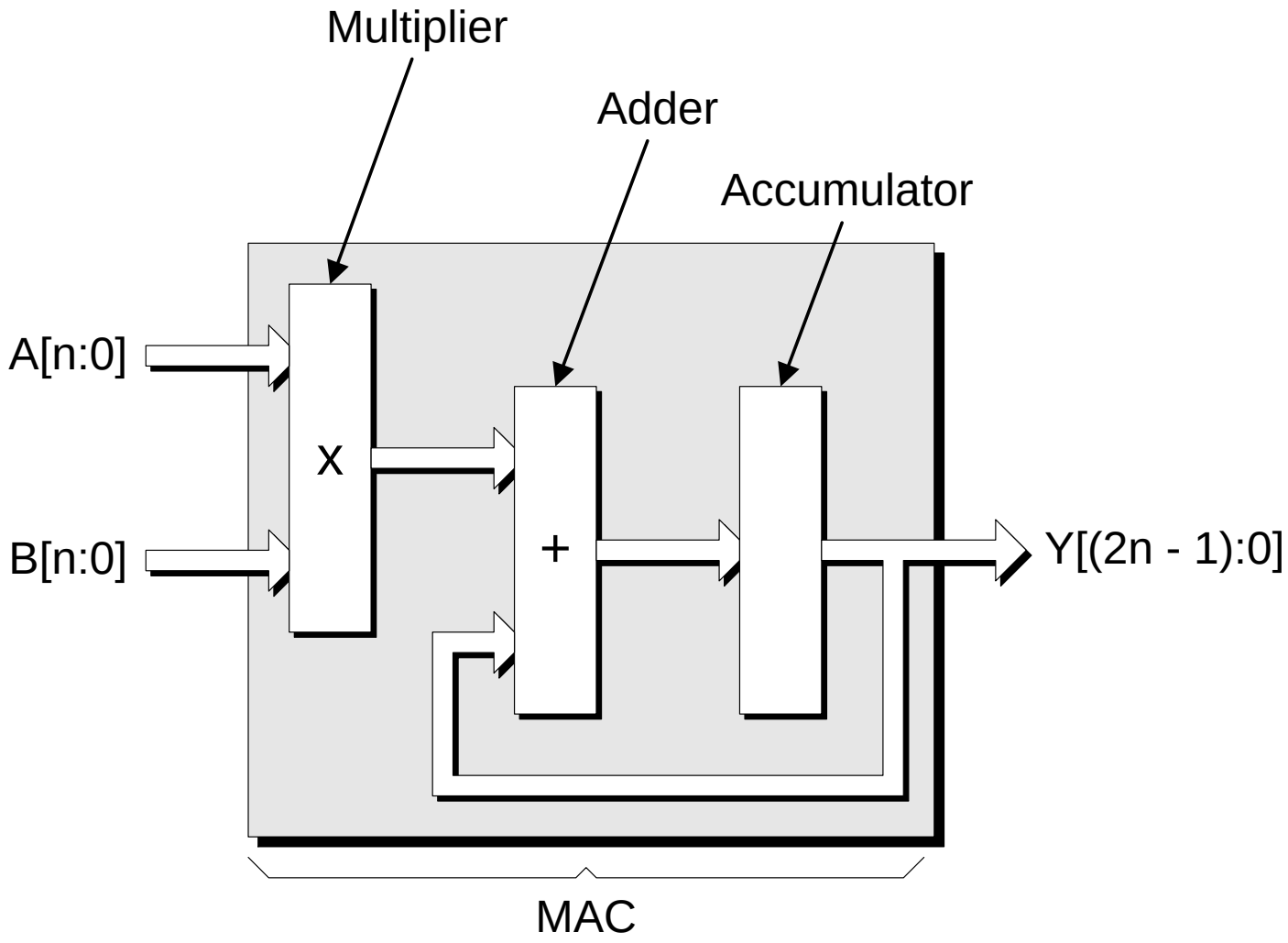
Spartan3 Components

Device	Multiplier Columns	Multipliers
XC3S50	1	4
XC3S200	2	12
XC3S400	2	16
XC3S1000	2	24
XC3S1500	2	32
XC3S2000	2	40
XC3S4000	4	96
XC3S5000	4	104

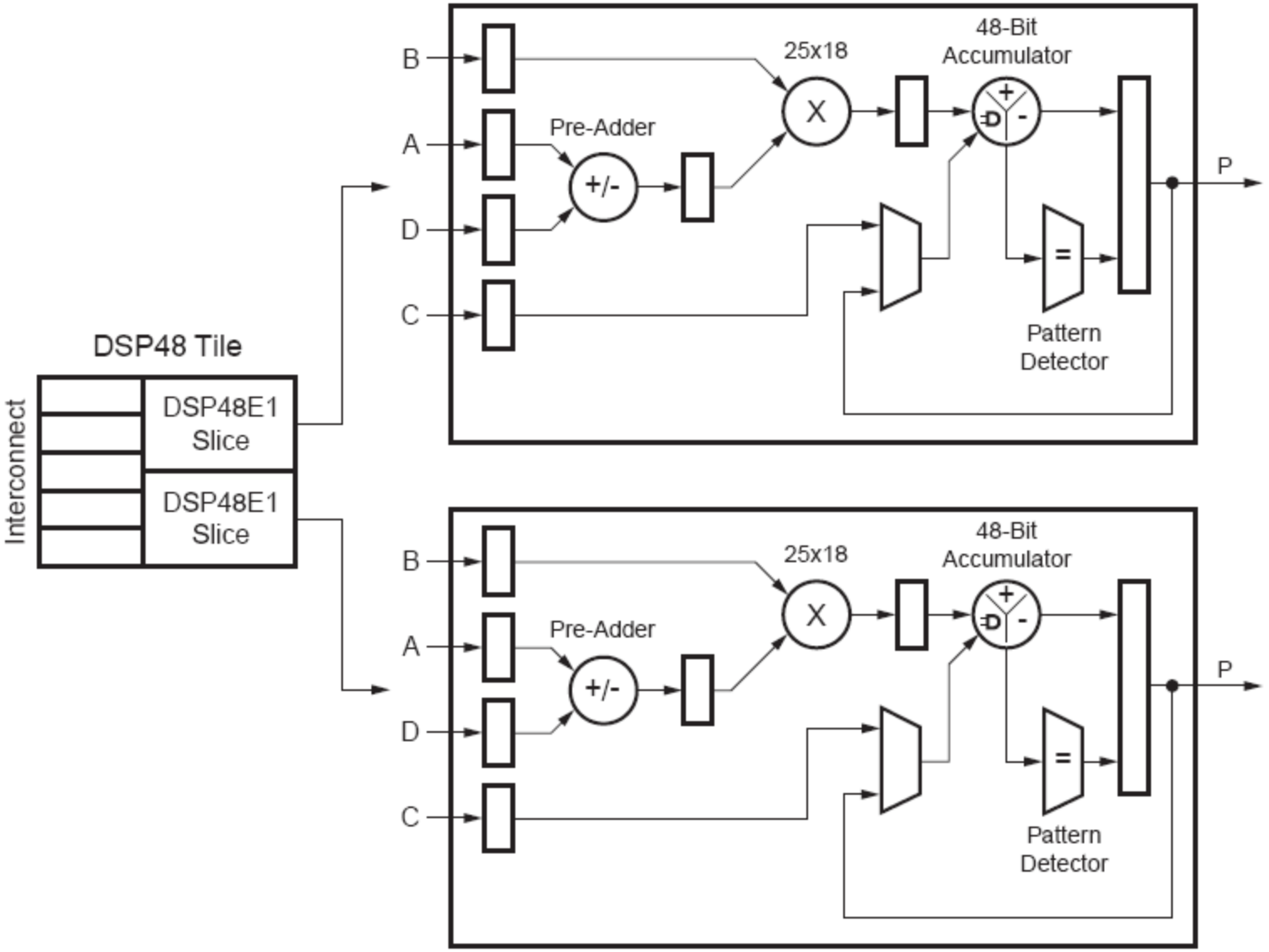
Spartan3 Components



FPGA Components



Virtex7 FPGA DSP48

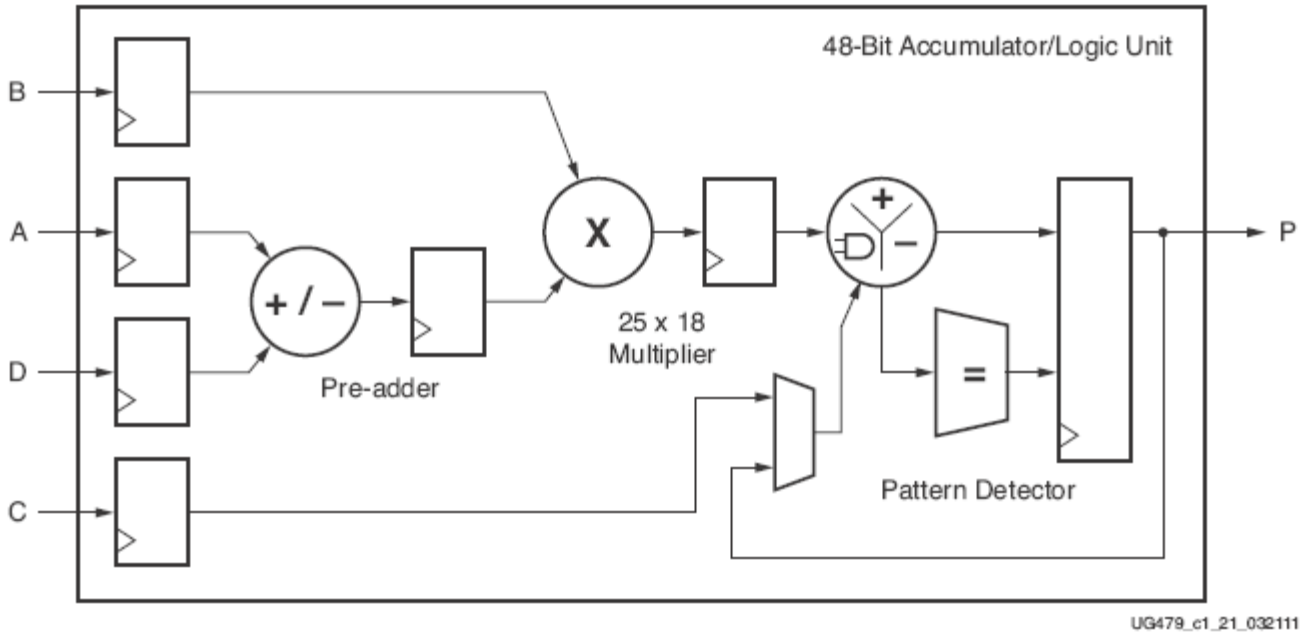


DSP48E1 Tile (Two DSP48E1 Slices and Interconnects)

Virtex7 FPGA DSP48

° Single-instruction-multiple-data (SIMD) arithmetic unit:
  - Dual 24-bit or quad 12-bit add/subtract/accumulate

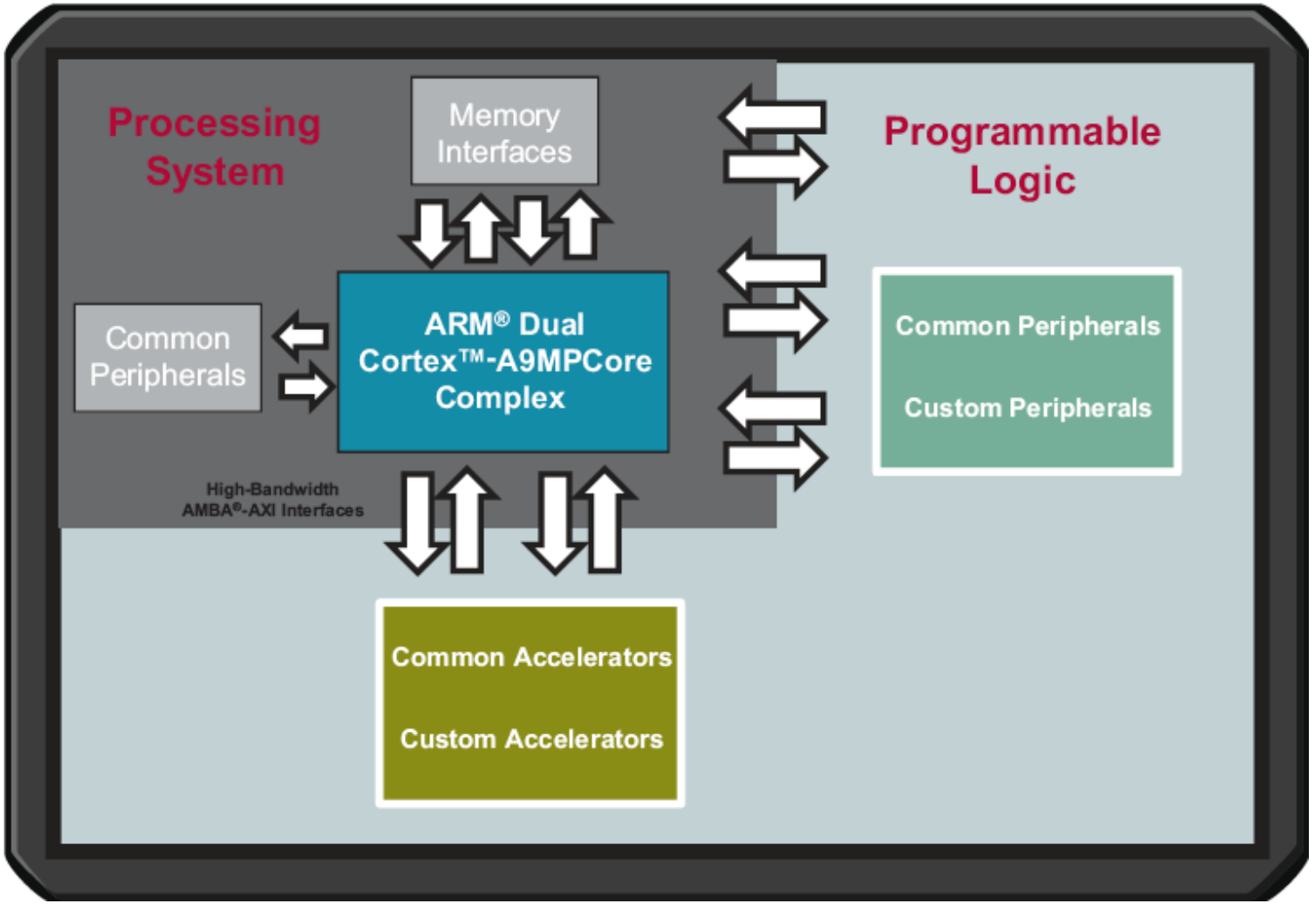
° Cascading capability on both pipeline paths for larger multipliers and larger post-adders



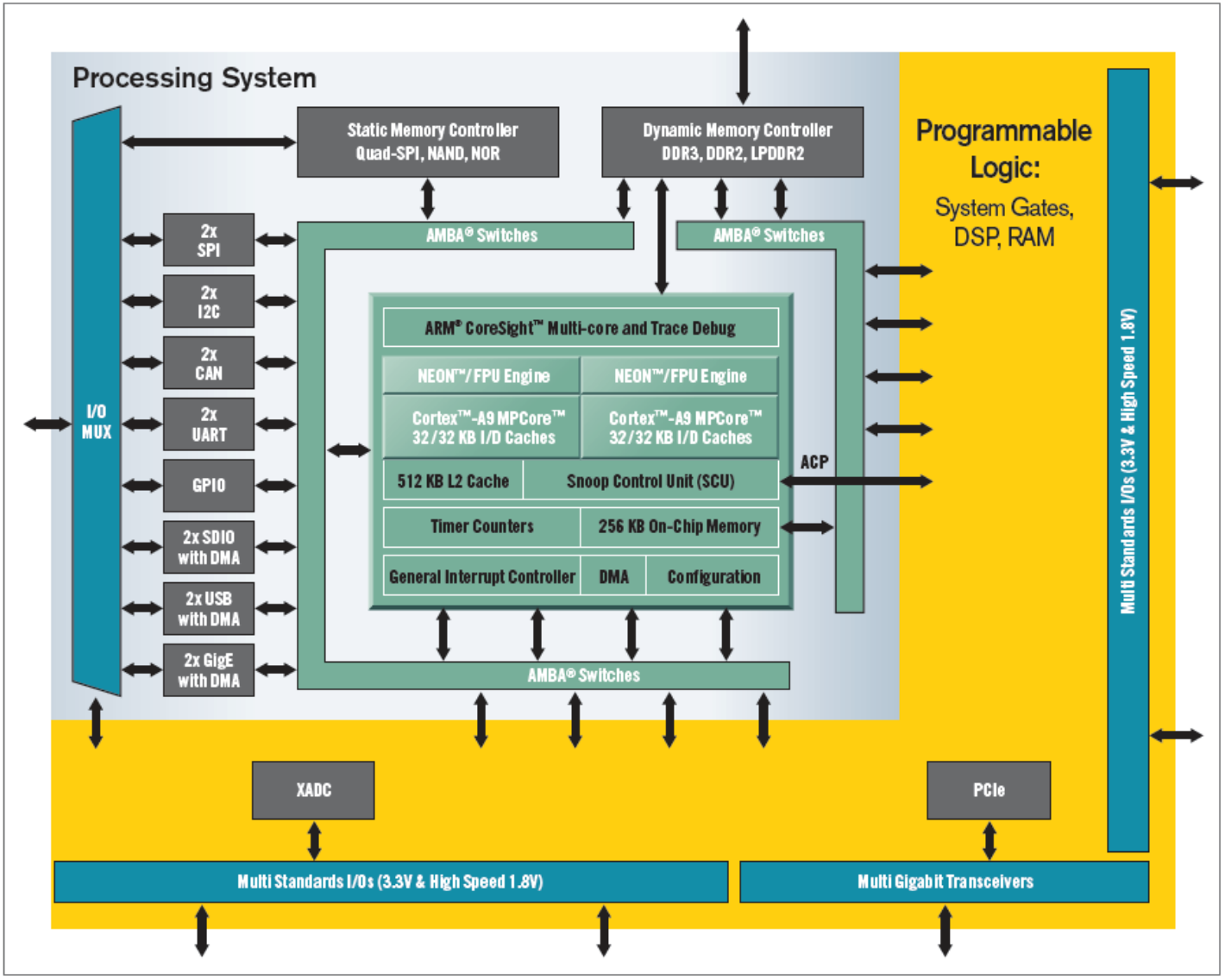
Virtex7 FPGA DSP48

Device	Total DSP48E1 Slices per Device	Number of DSP48E1 Columns per Device	Number of DSP48E1 Slices per Column
7A35T	90	2	60(2)
7A50T	120	2	60
7A75T	180	3	80(2)
7A100T	240	3	80
7A200T	740	9	100(1)
7K70T	240	3	80
7K160T	600	6	100
7K325T	840	6	140
7K355T	1,440	12	120
7VX485T	2,800	20	140
7VX550T	2,880	18	200(2)
7VX690T	3,600	18	200

Zynq FPGA Components



Zynq FPGA Components



Zynq FPGA Components

Zynq-7000 Product Table (Software View)				
Device Name	Z-7010	Z-7020	Z-7030	Z-7045
Part Number	XC7Z010	XC7Z020	XC7Z030	XC7Z045
Processor Core	Dual ARM® Cortex™-A9 MPCore™ with CoreSight™			
Processor Extensions	NEON™ and Single/Double Precision Floating Point			
Maximum Frequency	800 MHz			
L1 Cache	32 KB Instruction, 32 KB Data per processor			
L2 Cache	512 KB			
On-Chip Memory	256 KB			
External Memory Support	DDR3, DDR2, LPDDR2			
External Static Memory Support	2x QSPI-SPI, NAND, NOR			
DMA Channels	8 (4 dedicated to Programmable Logic)			
Peripherals	2x USB 2.0 (OTG) w/DMA, 2x Tri-mode Gigabit Ethernet w/DMA, 2x SD/SDIO w/DMA, 2x UART (2), 2x CAN2.0B, 2x I2C, 2x SPI, 4x 32b GPIO			
Security	AES and SHA 256b for secure boot			
Peripherals and Static Memory Multiplexed I/O[1]	54			
Processing System to Programmable Logic Interface Ports (Primary Interfaces & Interrupts Only)	2x AXI 32b Master, 2x AXI 32b Slave, 4x AXI 64b/32b Memory AXI 64b ACP 16 Interrupts			
Xilinx 7 Series Programmable Logic Equivalent	Artix™-7 FPGA	Artix™-7 FPGA	Kintex™-7 FPGA	Kintex™-7 FPGA
Programmable Logic Cells (Approximate ASIC Gates[3])	28K Logic Cells (~430K)	85K Logic Cells (~1.3M)	125K Logic Cells (~1.9M)	350K Logic Cells (~5.2M)
Extensible Block RAM (# 36 Kb Blocks)	240KB (60)	560KB (140)	1,060KB (265)	2,180KB (545)
Programmable DSP Slices (18x25 MACCs)	80	220	400	900
Peak DSP Performance (Symmetric FIR)	58 GMACS	158 GMACS	480 GMACS	1080 GMACS
PCI Express® (Root Complex or Endpoint)	—	—	Gen2 x4	Gen2 x8
Agile Mixed Signal (AMS)/XADC	2x 12 bit, 1 MSPS ADCs with up to 17 Differential Inputs			
Security	AES and SHA 256b for secure configuration			
Multi-Standards 3.3V I/O[2]	100	200	250	350
Serial Transceivers[2]	—	—	4	16

Programming FPGAs

- Mapping 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 connecting wires in switch boxes?
      - Etc.

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

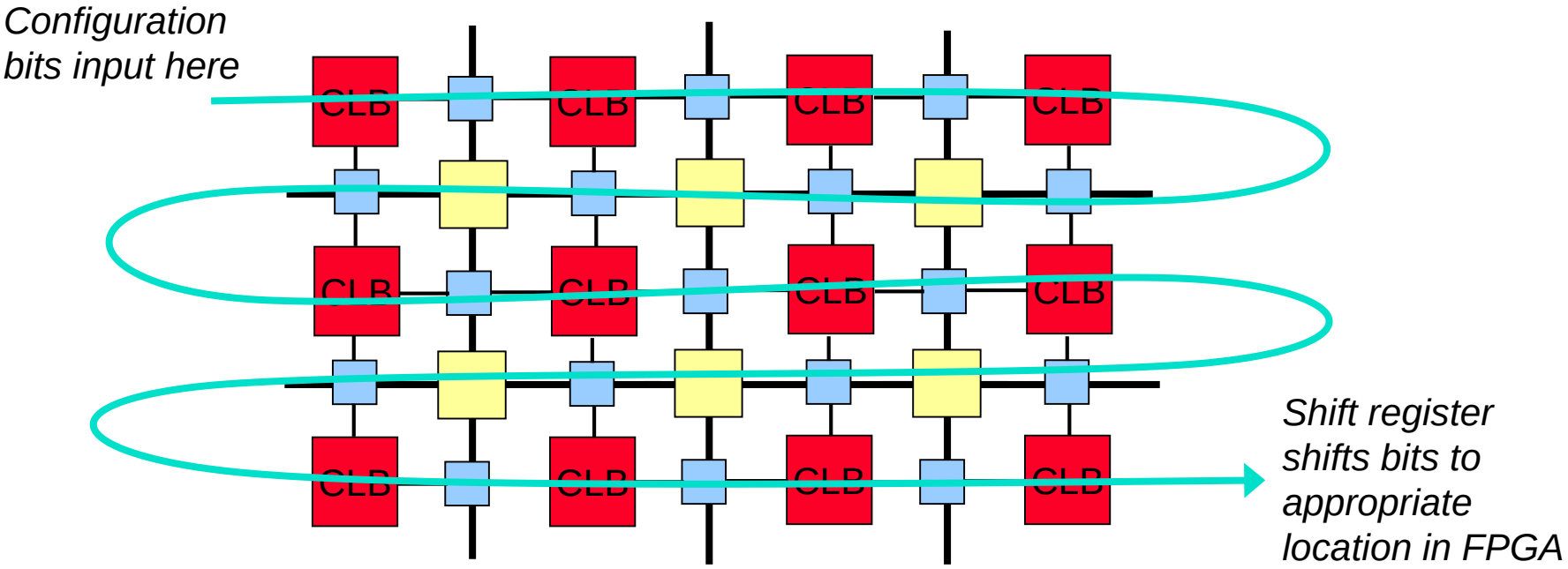
° But, how do you get the bitfile into the FPGA?

- > 10k LUTs
- Small number of pins

Programming FPGAs

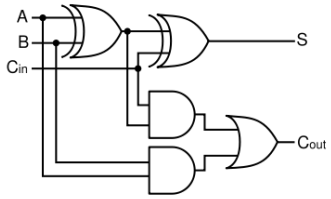
° Solution: Shift Registers

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

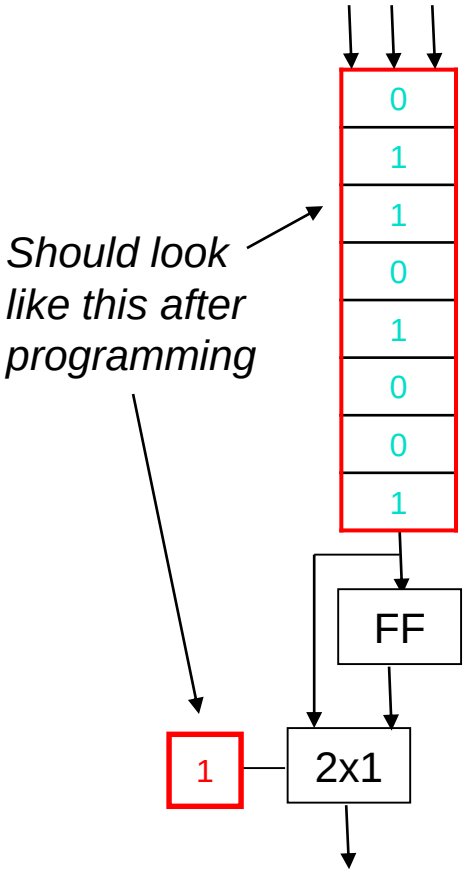


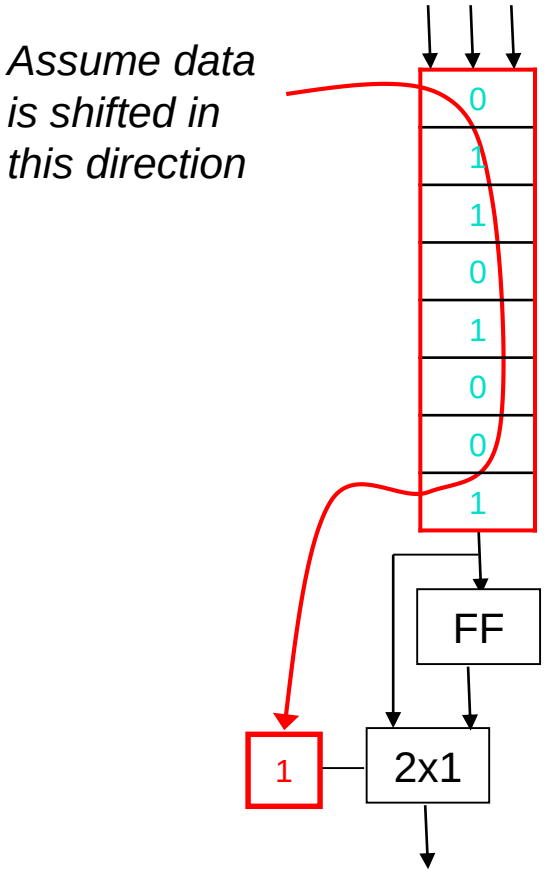
Programming FPGAs

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



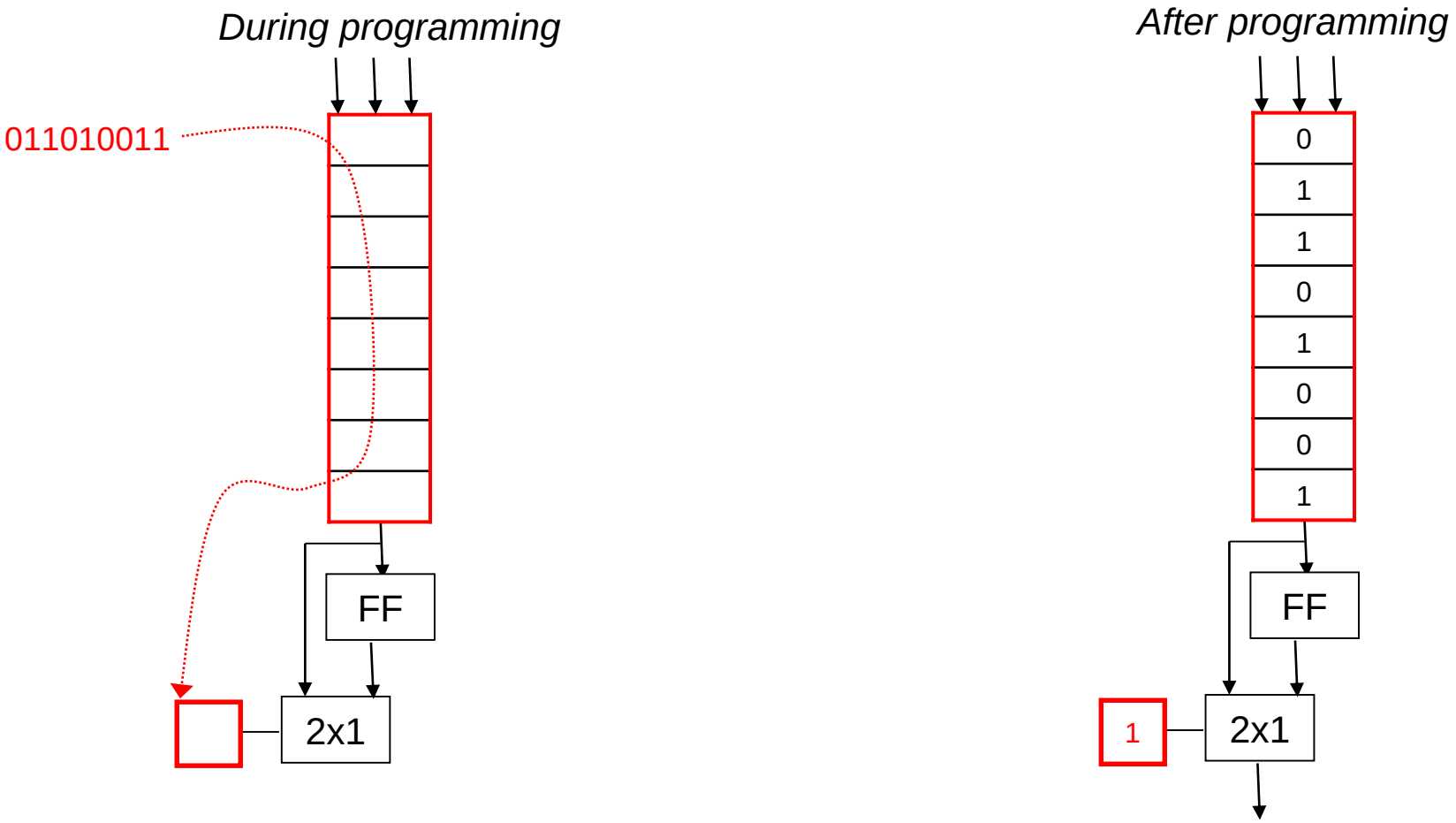
In			Out
A	B	Cin	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



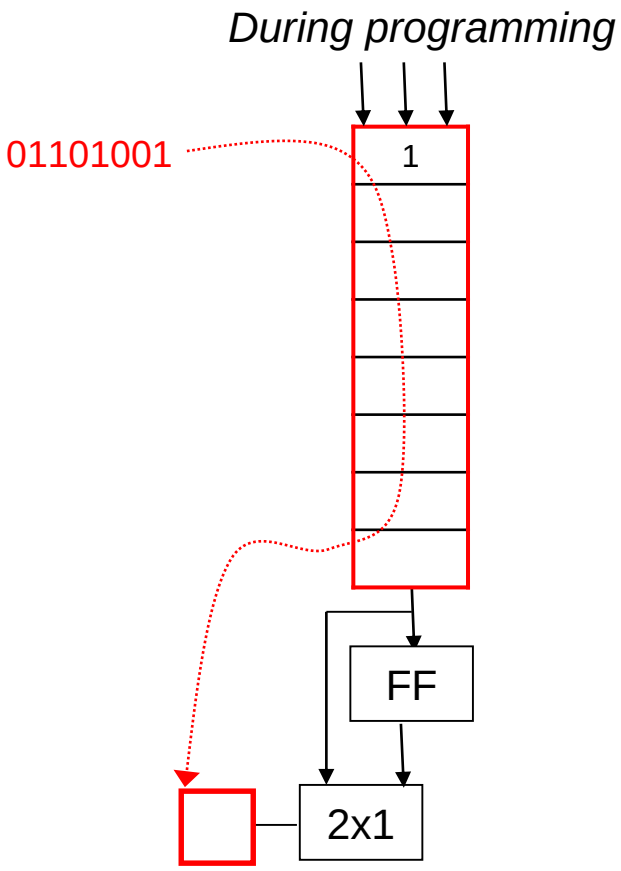


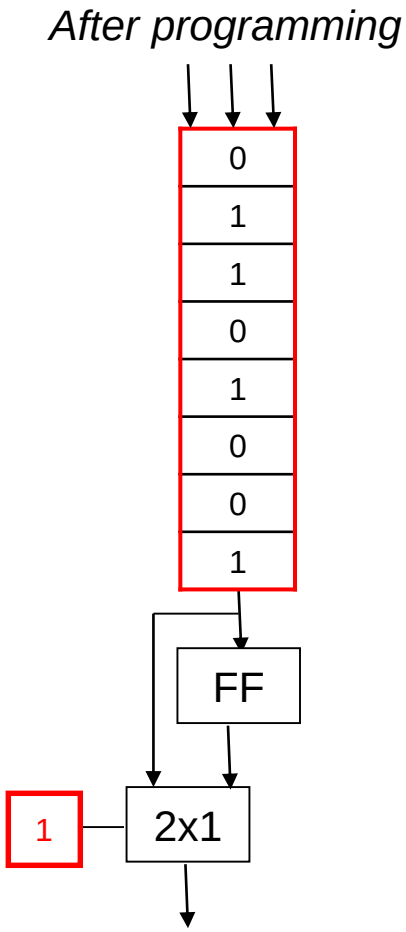
Programming FPGAs

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

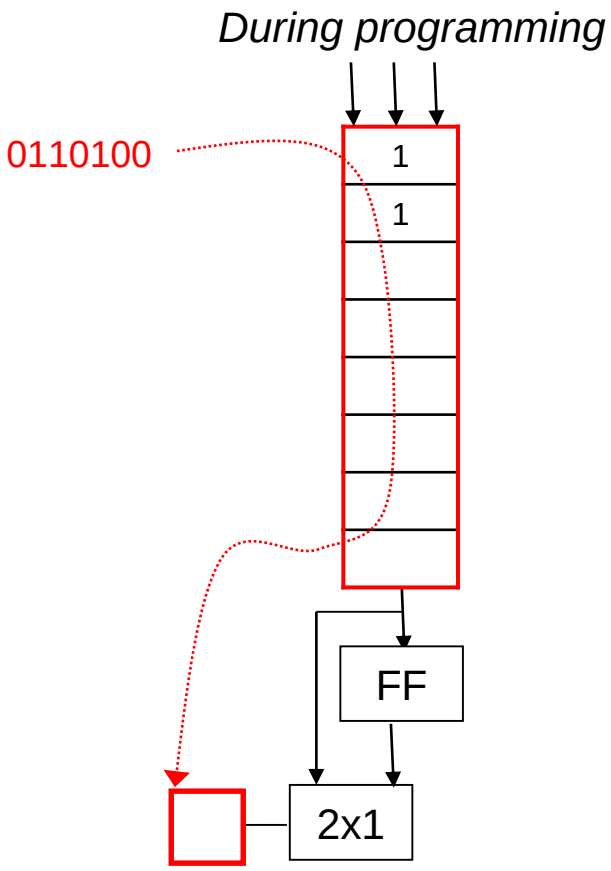


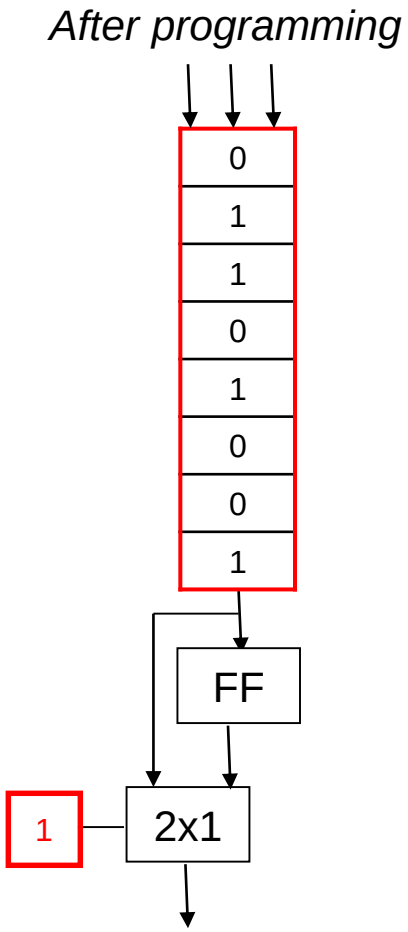
Programming FPGAs



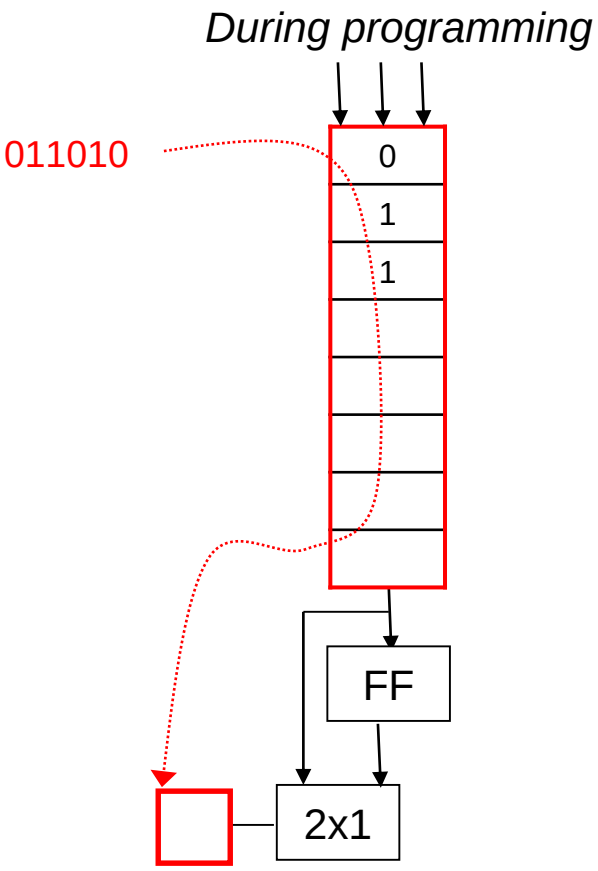


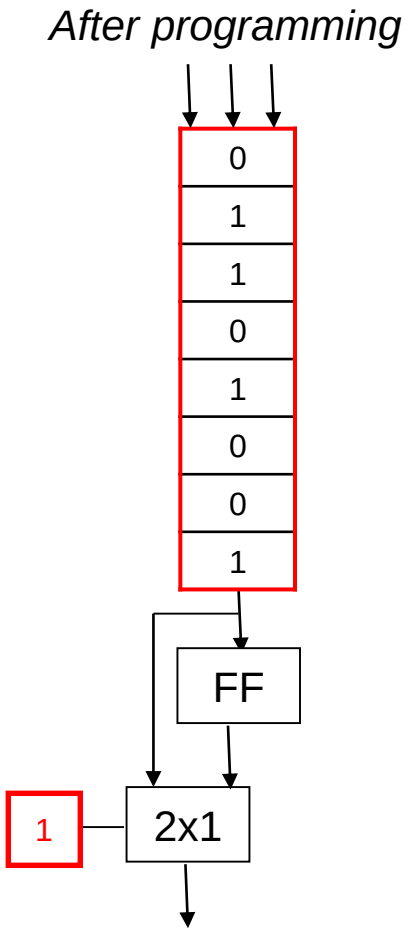
Programming FPGAs



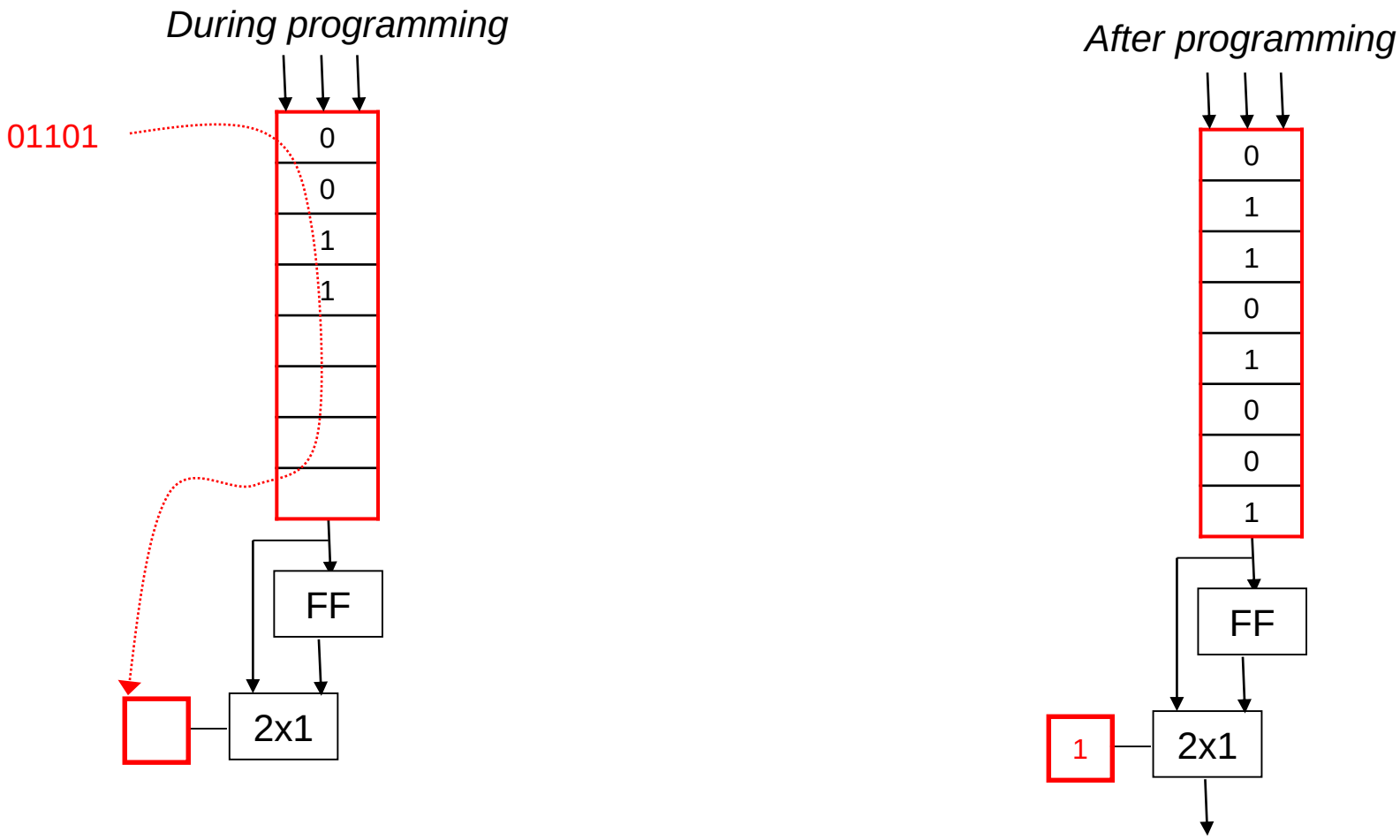


Programming FPGAs

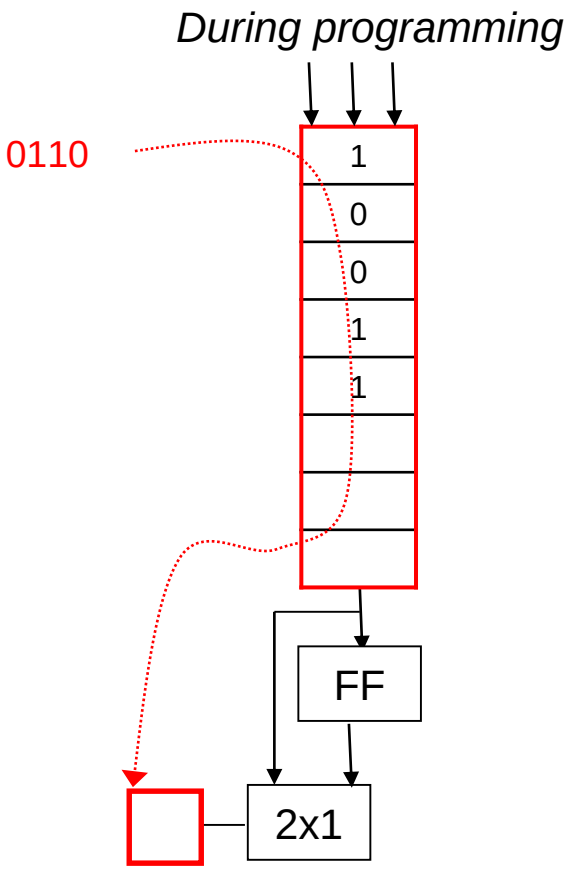


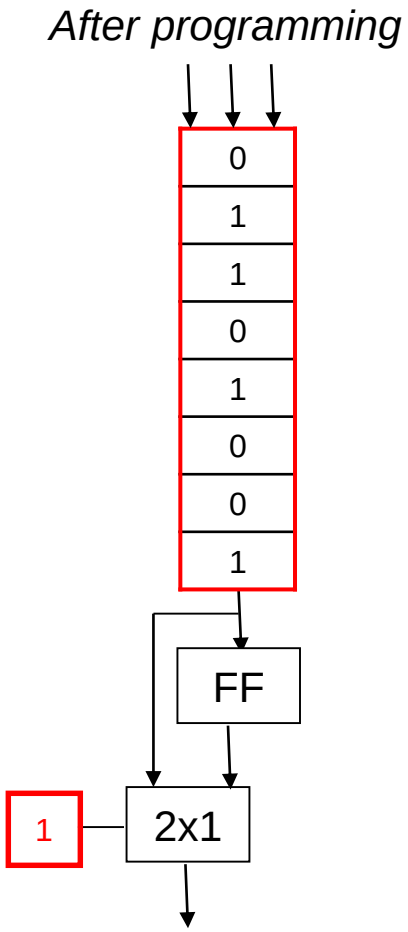


Programming FPGAs

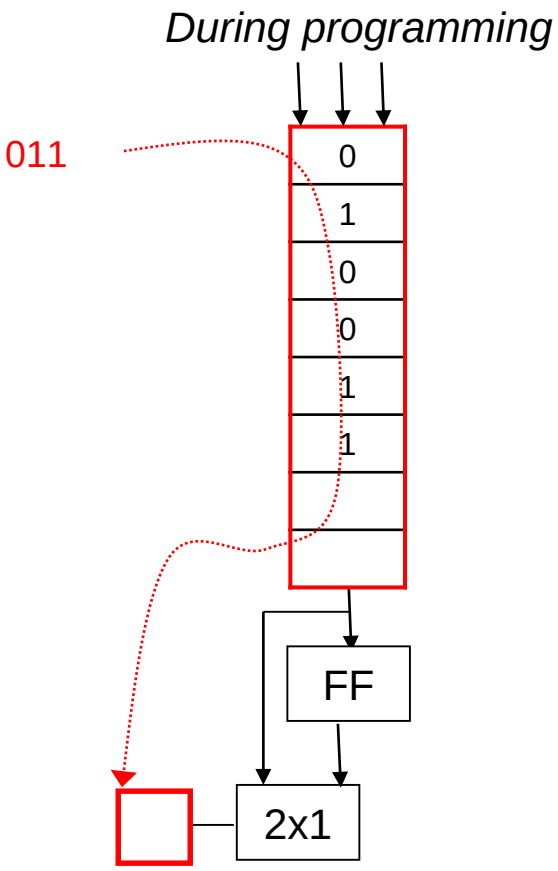


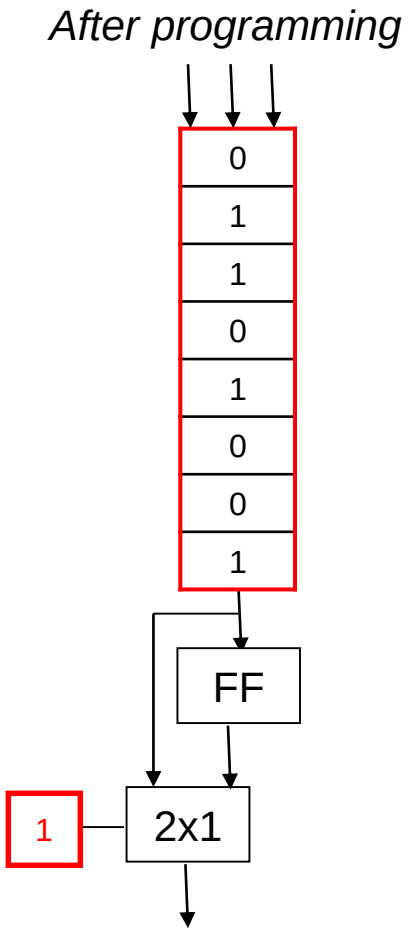
Programming FPGAs



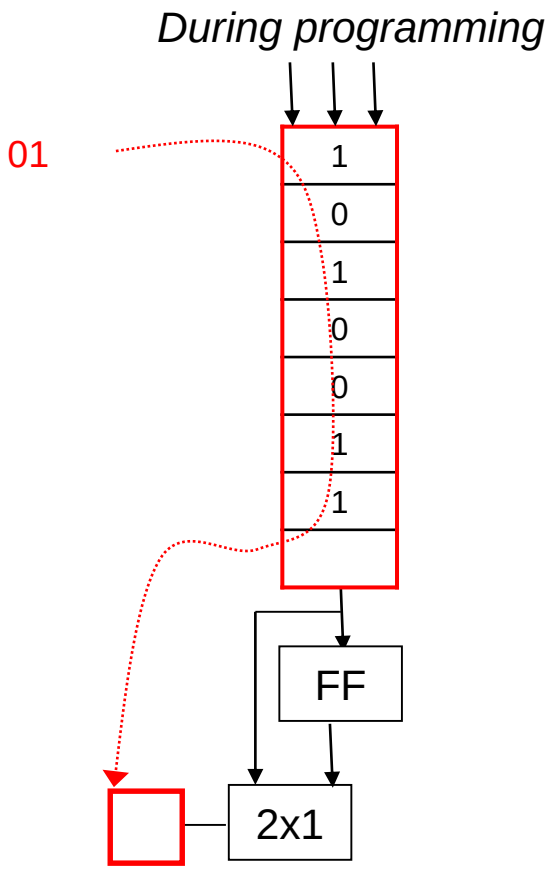


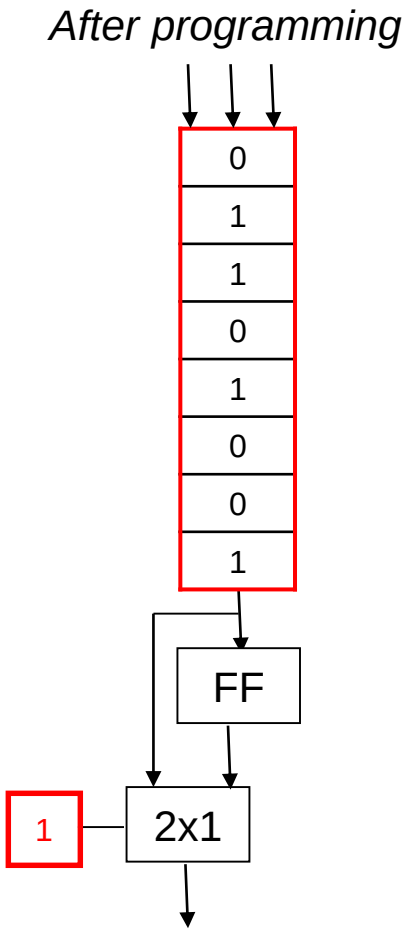
Programming FPGAs



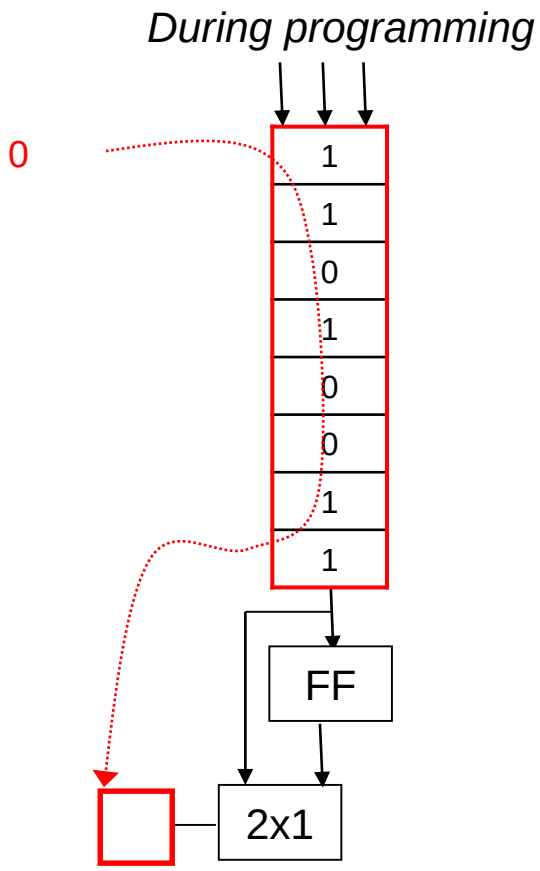


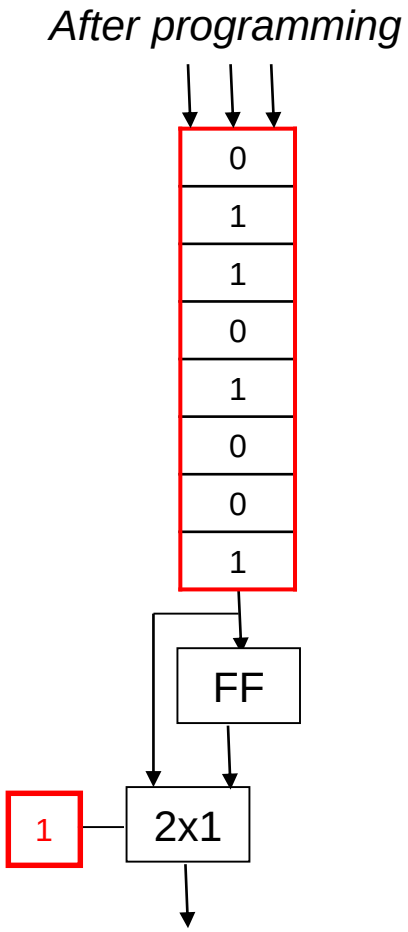
Programming FPGAs



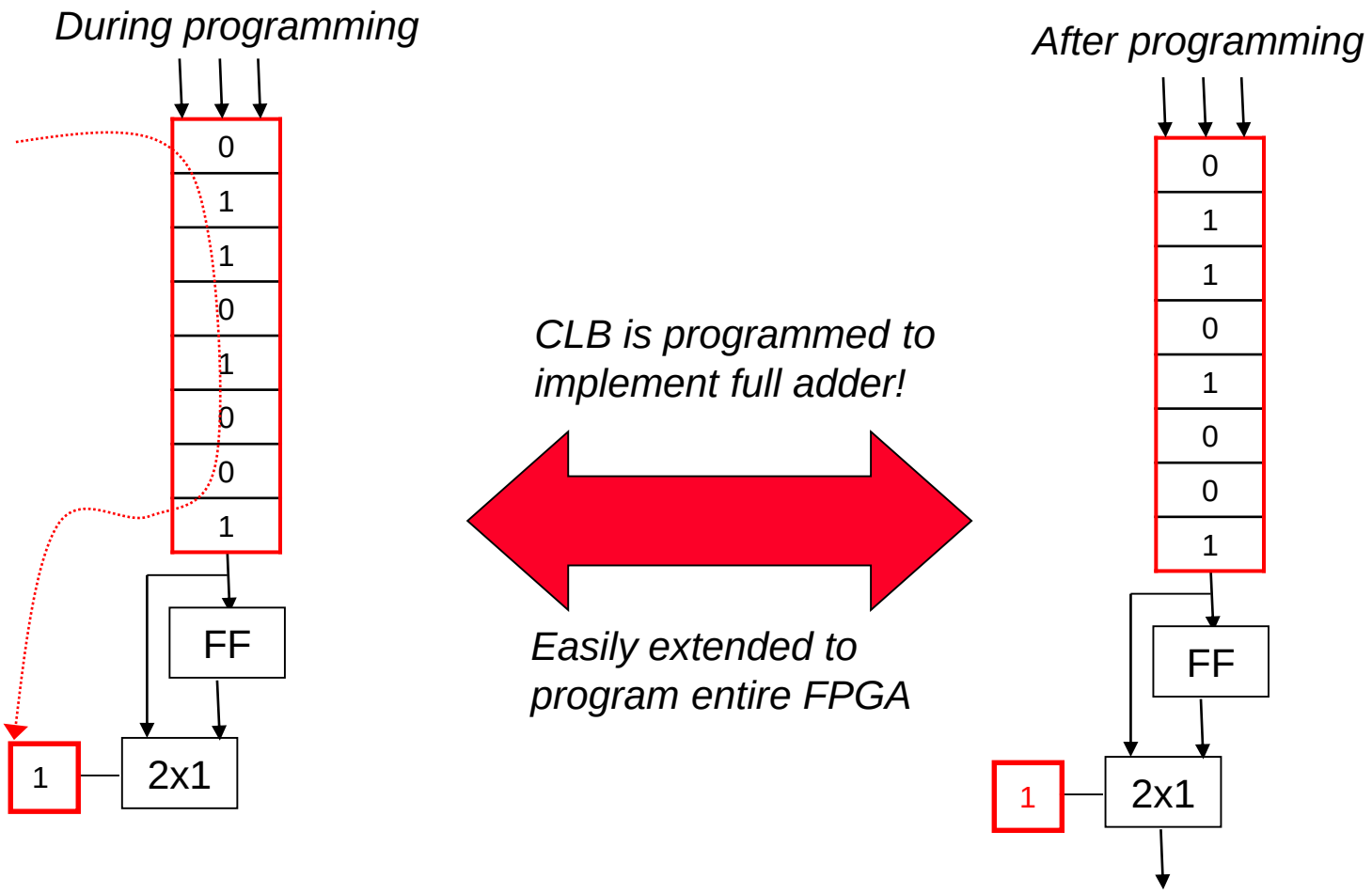


Programming FPGAs





Programming FPGAs



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

FPGA Components

- **High-performance families**
  - **Virtex (220 nm)**
  - **Virtex-E, Virtex-EM (180 nm)**
  - **Virtex-II (130 nm)**
  - **Virtex-II PRO (130 nm)**
  - **Virtex-4 (90 nm)**
  - **Virtex-5 (65 nm)**
  - **Virtex-6 (40 nm)**
  - **Virtex-7 (28 nm)**
- **Low Cost Family**
  - **Spartan/XL – derived from XC4000**
  - **Spartan-II – derived from Virtex**
  - **Spartan-IIE – derived from Virtex-E**
  - **Spartan-3 (90 nm)**
  - **Spartan-3E (90 nm) – logic optimized**
  - **Spartan-3A (90 nm) – I/O optimized**
  - **Spartan-3AN (90 nm) – non-volatile,**
  - **Spartan-3A DSP (90 nm) – DSP optimized**
  - **Spartan-6 (45 nm)**
  - **Artix-7 (28 nm)**

CPLD vs FPGA

° Simpler interconnect structure
  - Timing performance more predictable than FPGAs.

° Density is less than most FPGAs
  - CPLDs feature logic resources with a wide number of inputs (AND planes)

° Performance is usually better than FPGAs

° A single FPGA can replace tens of normal PLDs
  - Primitive FPGA 'logic cells' are more complex than PLD cells.
  - Can program the routing between FPGA logic cells in addition to programming the logic cells themselves.
  - Many FPGAs now offer embedded memory blocks in addition to logic blocks or other special features such as fast carry logic chains.

° FPGAs offer a higher ratio of flip-flops to logic resources than do CPLDs.