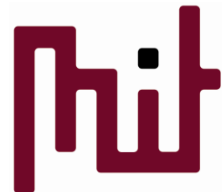


Embedded and Ambient Systems

2020.12.08.

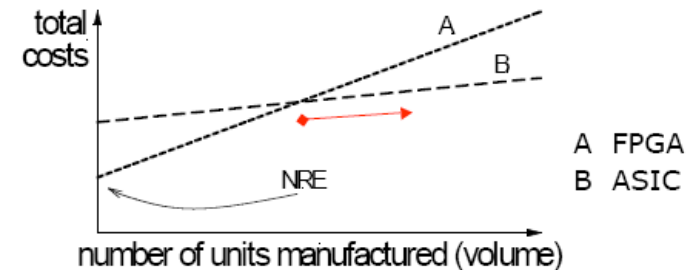
FPGA (Field Programmable Gate Array)



Méréstechnika és
Információs Rendszerek
Tanszék

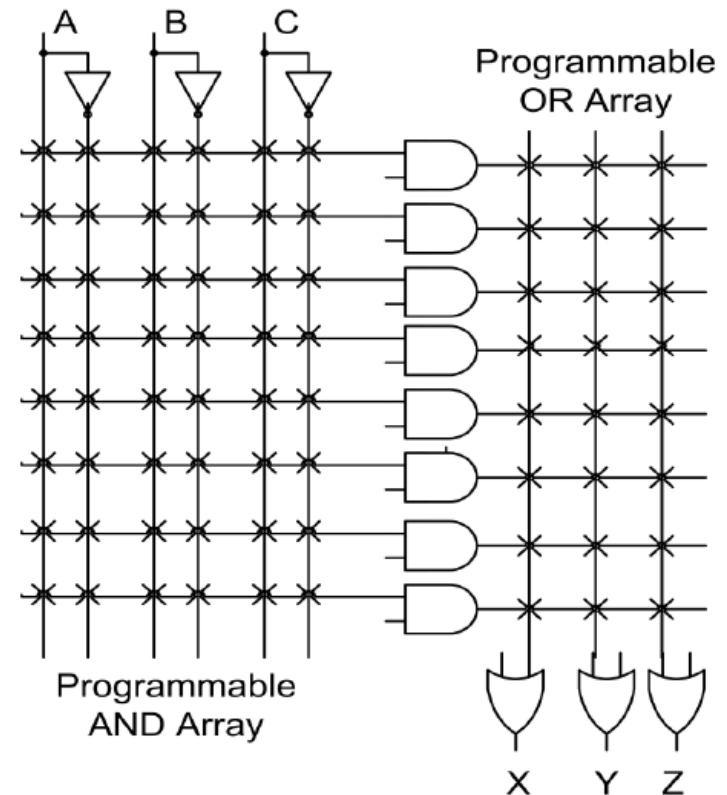
FPGA: evolution

- Dynamically configurable devices
- When applying FPGA a system is built up based on basic digital circuit elements
- Motivation:
 - Solve problems based on digital HW:
 - Rapid operation
 - Slow development
 - Production time is long
 - » In the past (mainly): printed circuit board + discrete logic gates
 - » Today: ASIC (Application Specific Integrated Circuits)
 - » NRE: Non-recurring engineering
 - Testing and re-design take a long time: slow iterations
 - Time to market is important, therefore the development process needs to be accelerated
- A device is needed that is suitable for the implementation of low level functions but the production and development time is shorter



FPGA: evolution

- PLA: Programmable Logic Array
 - In 1970s
 - Programmable AND and OR gates
 - Implementation of logical functions in canonical form
 - Advancement: special circuit for the implementation of complex logic functions
 - Drawback: PLSs can be configured during production process and cannot be reconfigured later



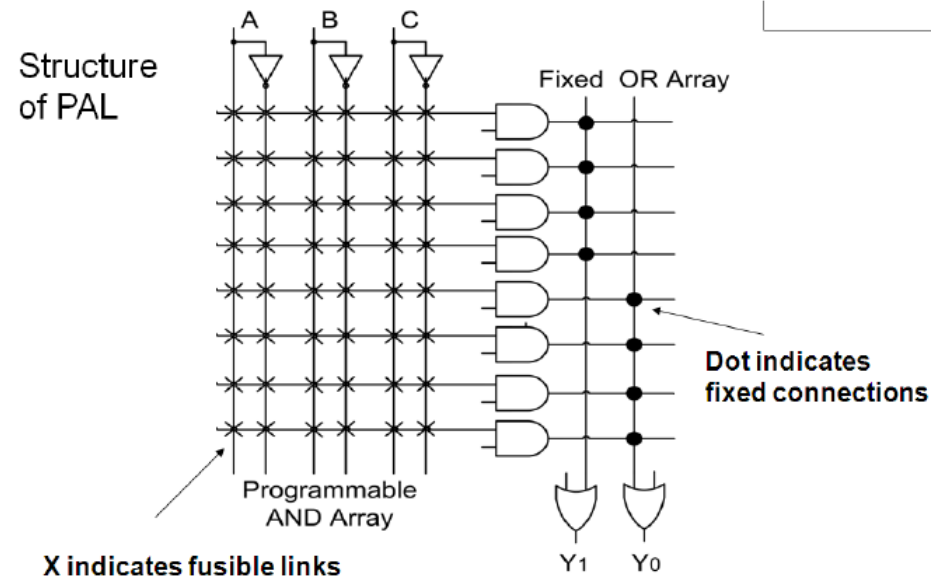
FPGA: evolution

■ PAL: Programmable Array Logic

- End of 1970s
- Canonical form
- Programmable input, fixed output
 - Less programmed connections: faster signal propagation
- Method of programming
 - OTP: one time programmable
 - Erasable: using UV light
 - Flash kofiguration
- Advancement: PAL can be configured not only during production but also by developer

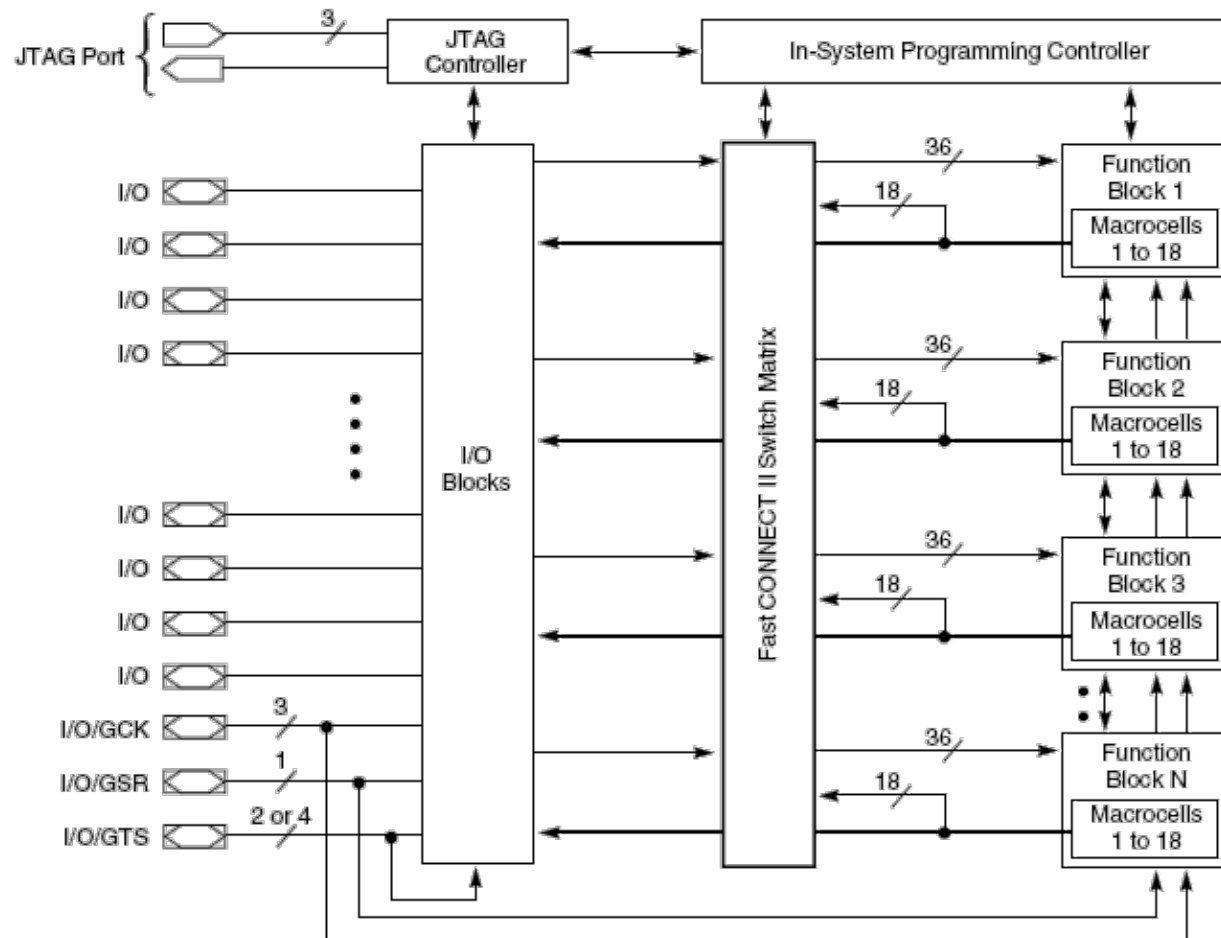
■ Advanced version: GAL (Generic Array Logic)

- Larger complexity, can substitute more PAL device
- Reconfigurable
- Adequate for prototyping



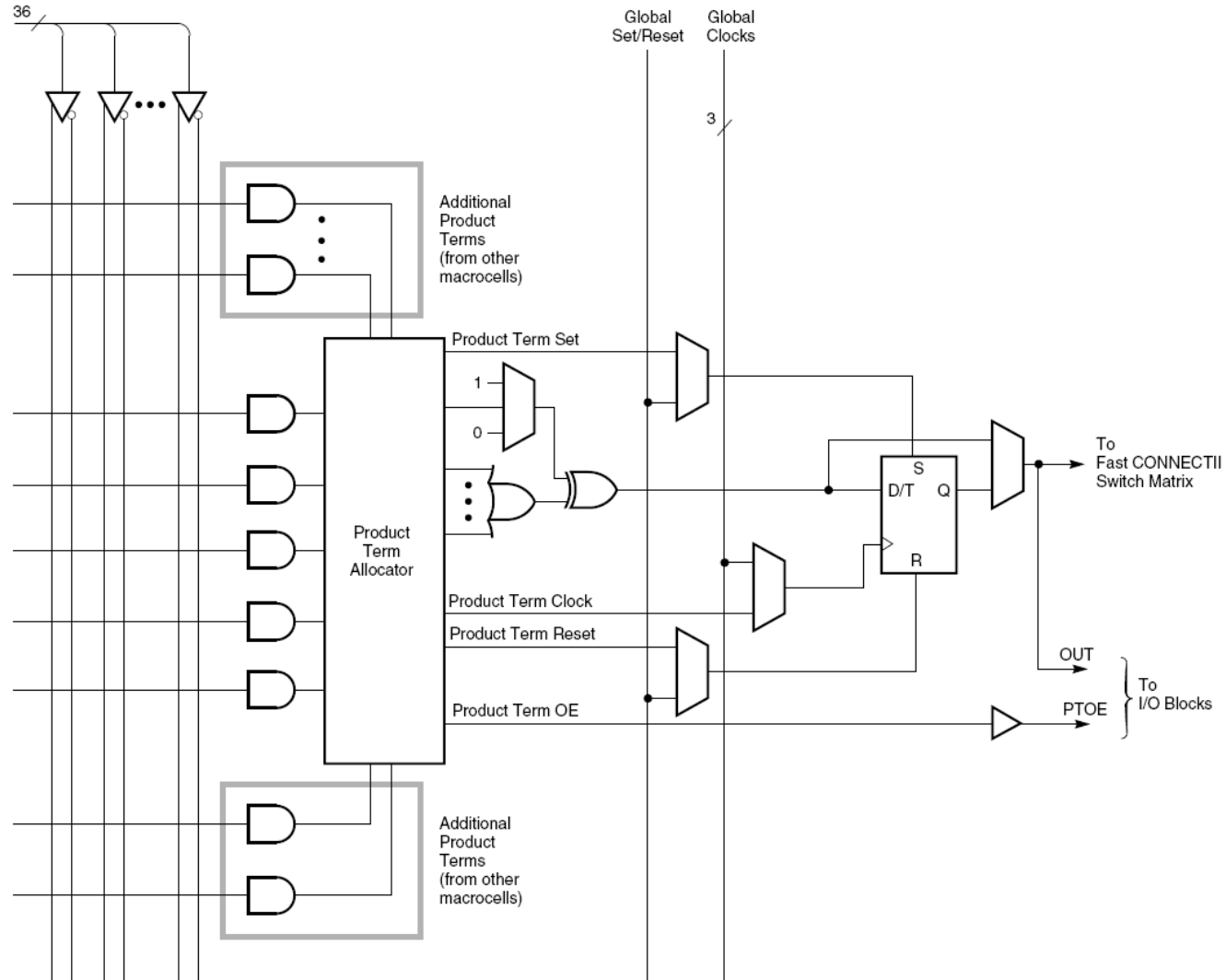
FPGA: evolution

- CPLD (Complex Programmable Logic Devices)
- Complexity: between PAL and GAL
- Architecture:
 - Function block
 - Macrocell
 - Wiring matrix
- Function block: contains macrocell
- Macrocell: multiple-input single-output logic function (combinational or register output)
 - Architecture is similar to PAL



CPLD macrocell

Architecture of a macrocell:
canonical
implementation
of logic
functions



DS063_03_110501

FPGA: evolution

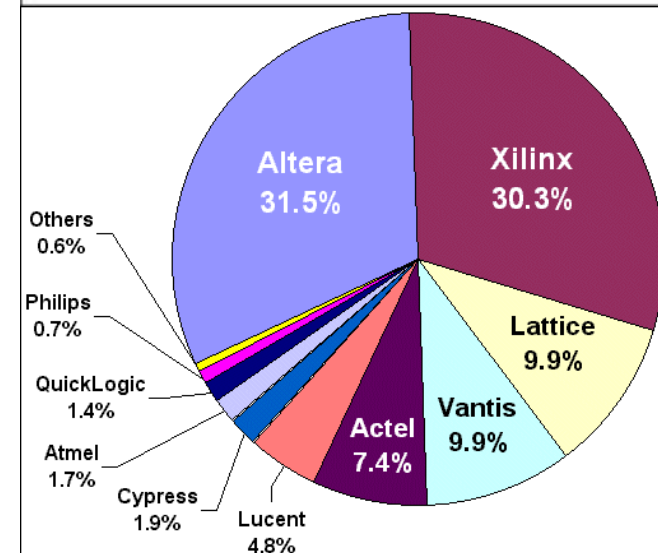
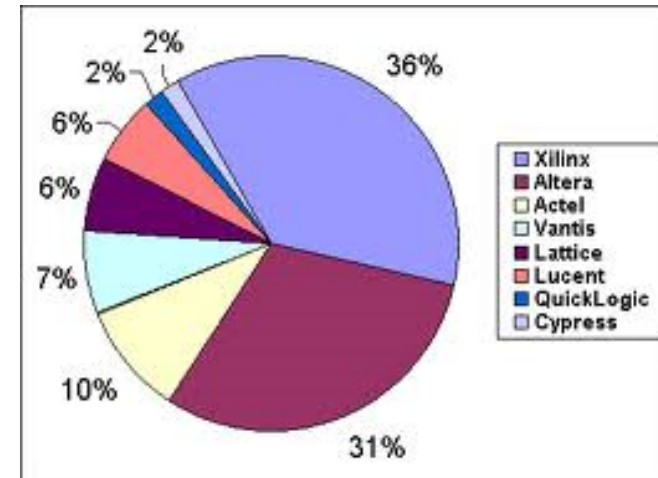
- FPGA: Field Programmable Gate Array
- High-complexity device
- Not necessarily follows the canonical structure
- Several auxiliary components are found
 - Clock-management
 - Flexible configurable IO block
 - Embedded RAM
 - Multiplier

FPGA manufacturers

- Some larger manufacturer:

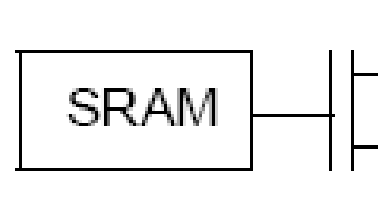
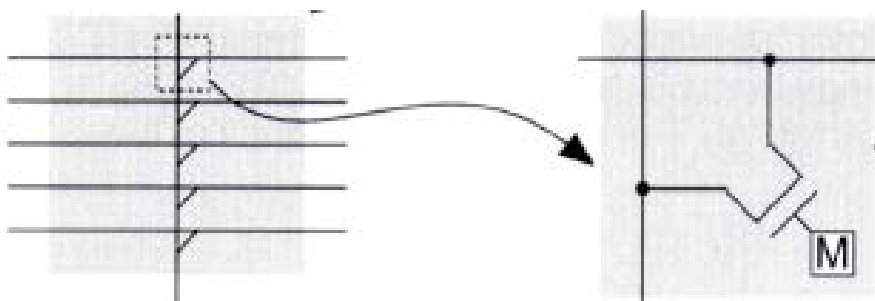
- Xilinx
- Altera
- Actel
- Vantis
- Lattice
- Lucent
- QuickLogic
- Cypress
- Atmel

- In the followings Xilinx products are used to learn about FPGAs



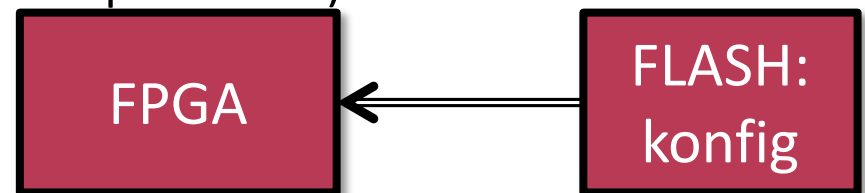
Storing the configuration

- Configuration file: contains the internal connections
- NOT A PROGRAM
 - Word 'programming' mainly refers to downloading the program but in case of FPGA not a program is written instead it is configured how the HW should work (behave)
 - The configuration is quite complex and really seems to be a program but it is not a program
- Configuration: making connections between data lines



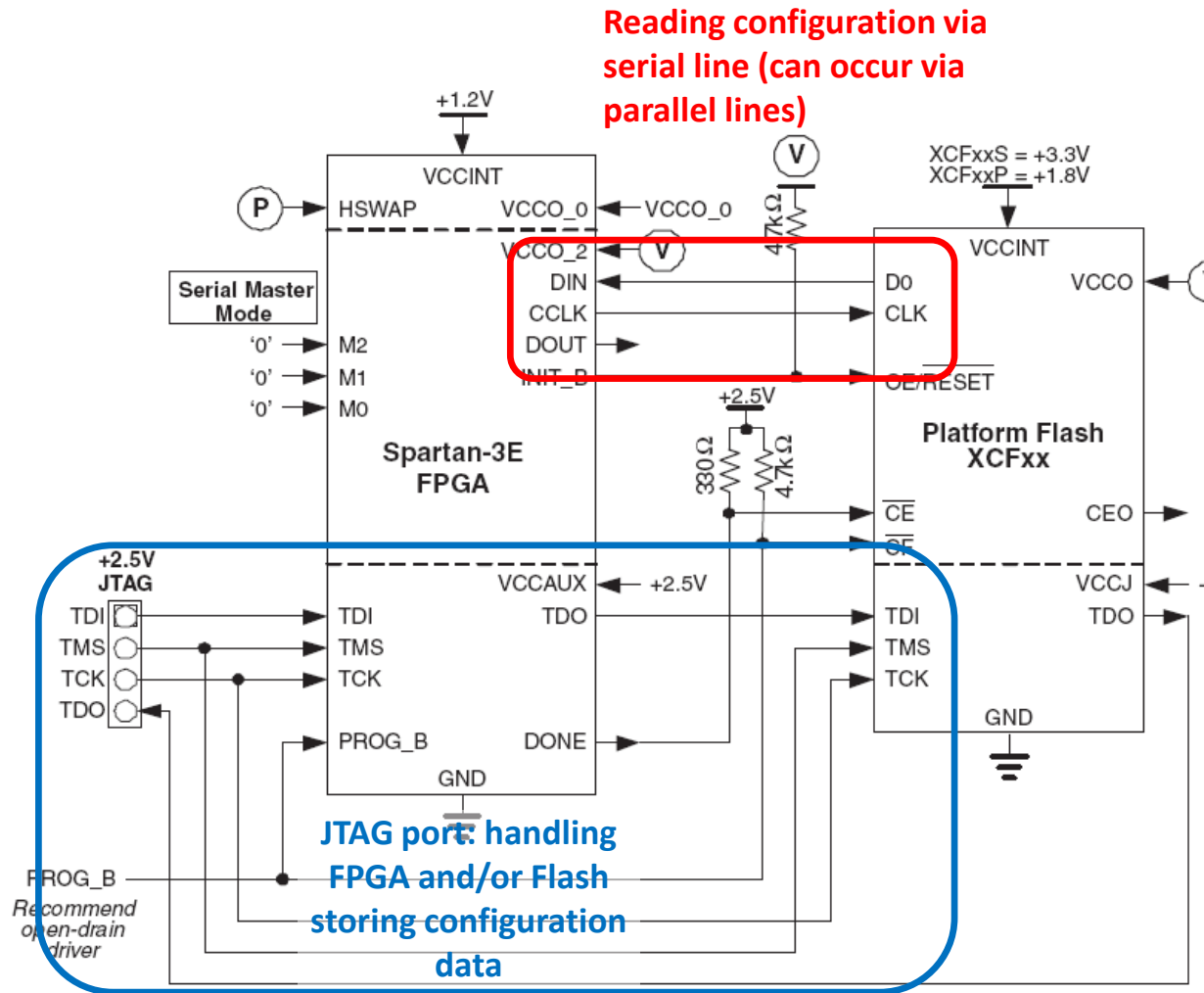
Storing the configuration

- Reading method of configuration (serial, parallel, JTAG) can be set using external wires
- In FPGAs the configuration data (e.g. connections in a switching matrix) is loaded into an internal SRAM
- Flash-based FPGAs are quite rare now
 - SRAM-based configuration allows larger component density 😊
 - During operation can be reconfigured even partly 😊
 - Booting delay of the system is larger: configuration must be loaded into the internal SRAM cells 😞
- Loading configuration: during booting it occurs from external Flash memory or a host PC
- There exist FPGAs containing internal Flash but even in this case booting delay of configuration is inevitable. Their advantage is the smaller area requirement compared to FPGA+Flash and more secure since the configuration process cannot be seen (reverse engineering can be prevented)



Storing the configuration

- FPGAs are reconfigurable during development via a standardized JTAG interface
- JTAG port can be used in a daisy chain toward other devices
- JTAG can be used for writing the external Flash that stores the configuration
- The configuration is stored in the Flash and read from Flash later at FPGA booting

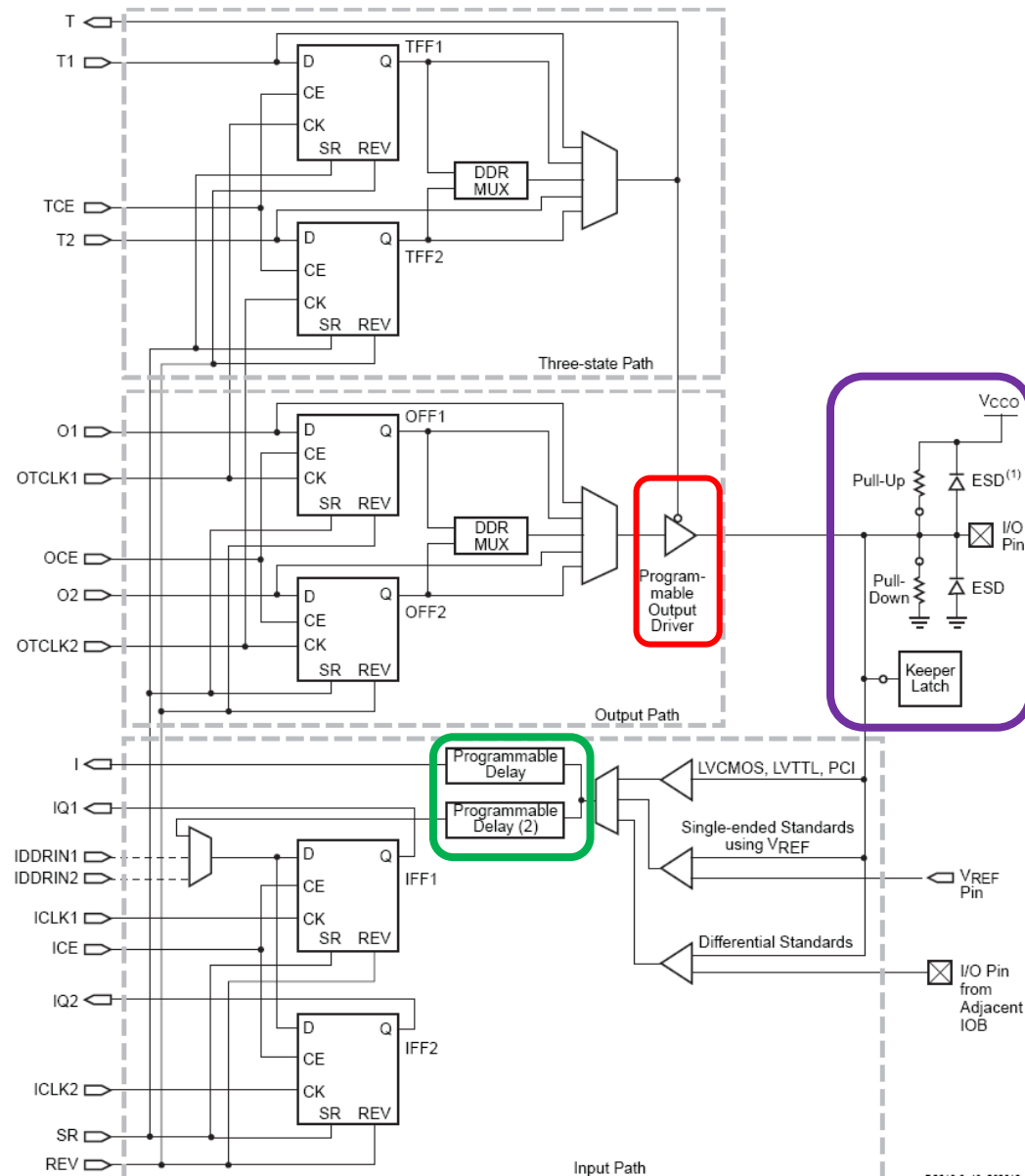


Input-Output blocks in FPGA

- Flexible configurable Input-Output (IO) block
- Three main signal lines:
 - Output drive
 - Input lines
 - Output lines

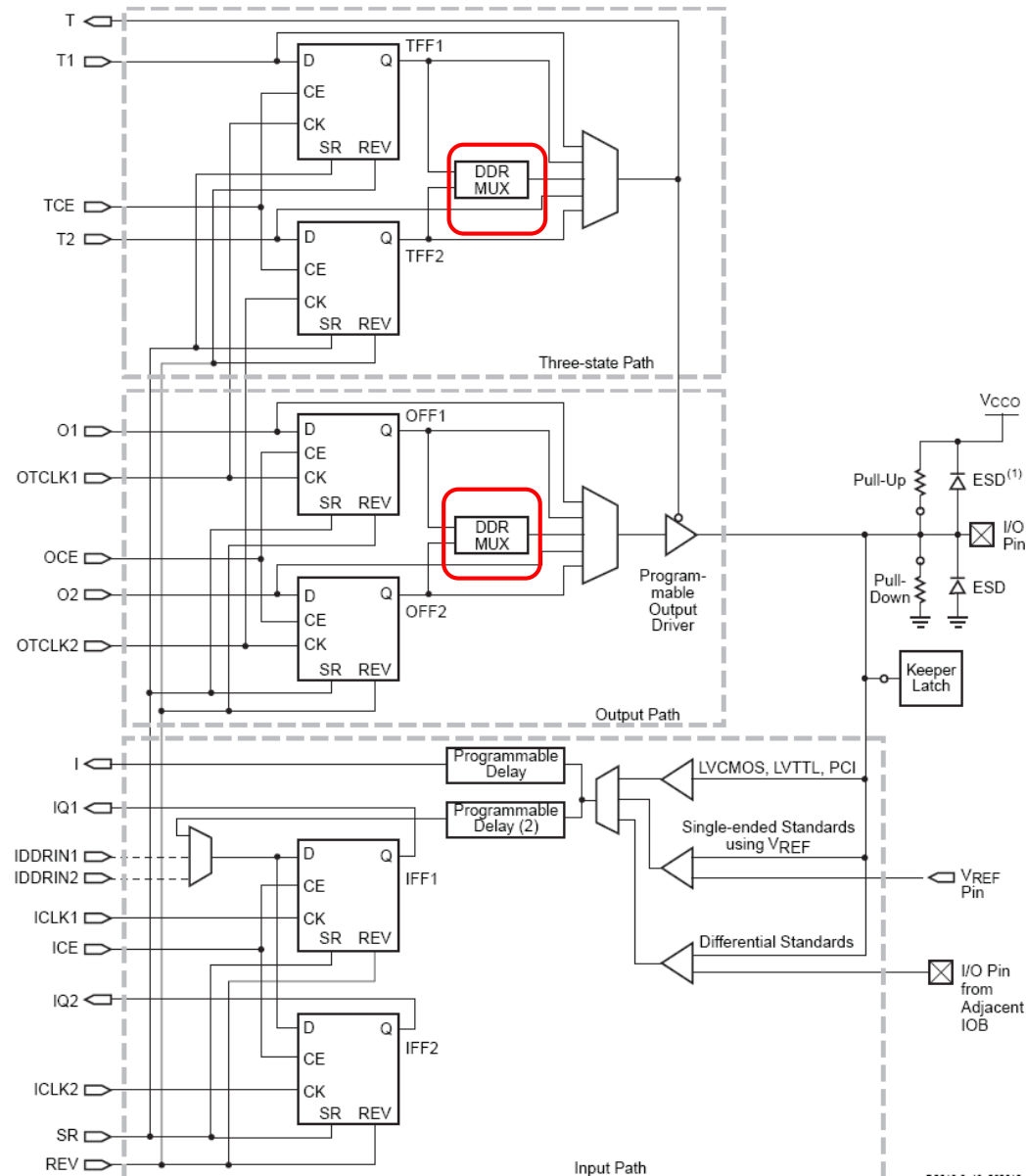
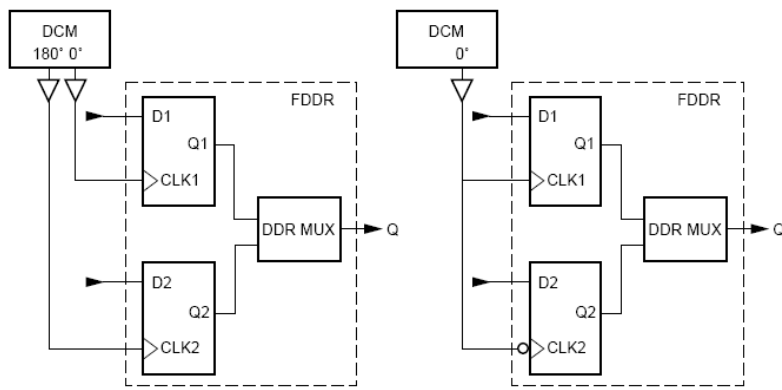
Input-Output blocks in FPGA

- Programmable input/output setting (see: **Programmable Output Driver**)
- Programmable pull-up or pull-down resistances
- **Keeper latch**: can be set that the last value is kept in high-impedance mode and not allow the level floating
- **Internal delay lines**: input signals are fed directly into the FPGA, the clock signal however get to the IO block via the internal clock division unit and suffers from delay: forcing delay to the data input it can be synchronized with the CLK
 - Delay is dynamical, can be configured even during operation



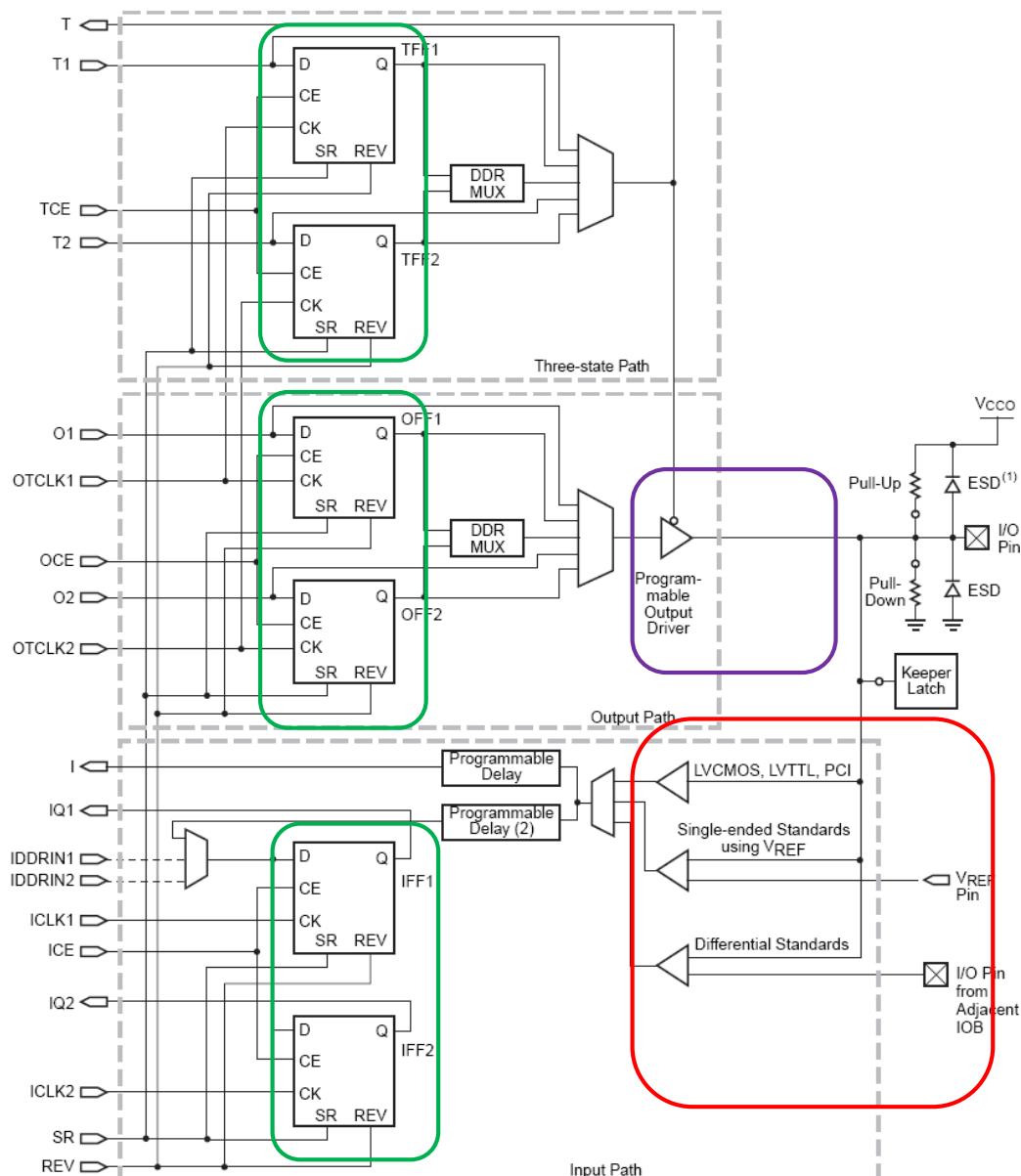
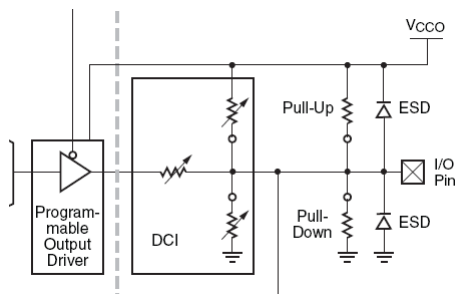
Input-Output blocks in FPGA

- **DDR: Dual-Data-Rate Transmission:** data rate is doubled when data transmission occurs for both the rising- and falling edge of the CLK
 - D flip-flops used in the conventional manner (triggered for only rising CLK edge) but their CLK signal is designed in a special way:
 - 180° phase shift
 - inverting
 - Writing is done alternately: once into one FF, then into the other FF



Input-Output blocks in FPGA

- Input and output lines can be synchronized using **flipflops** to the internal CLK
- Several **standardized logic signal level** can be applied on the inuts
 - TTL levels
 - CMOS levels
 - Differential signal inputsdifferenciális jelbemenetek
 - Switching threshold voltage via external pin
- Driving can be set:
 - Fast raise up: larger speed with larger noise
 - Slow raise up: slower with less noise
- DCI (**Digitally Controlled Impedance**): to eliminate reflexion: a certain impedance value can be set for the output buffer



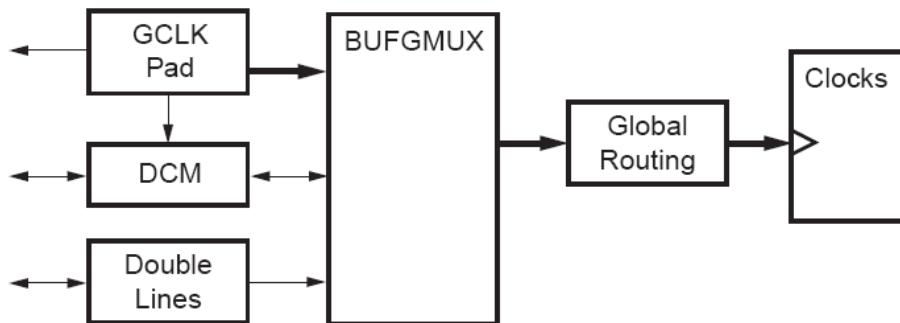
Input-Output blocks in FPGA

- Configuration of IO pins: in Verilog language find it in user constraints file (ucf)
 - Examples:
 - NET "name" IOSTANDARD = "LVTTTL";
 - NET "net_name" LOC= " P6 " | PULLUP;
- In Verilog code they can be given by primitives
 - Example:

```
PULLUP PULLUP_inst (  
    .O(signal_name) // Pullup output (connect directly to top-level  
    port)  
);
```

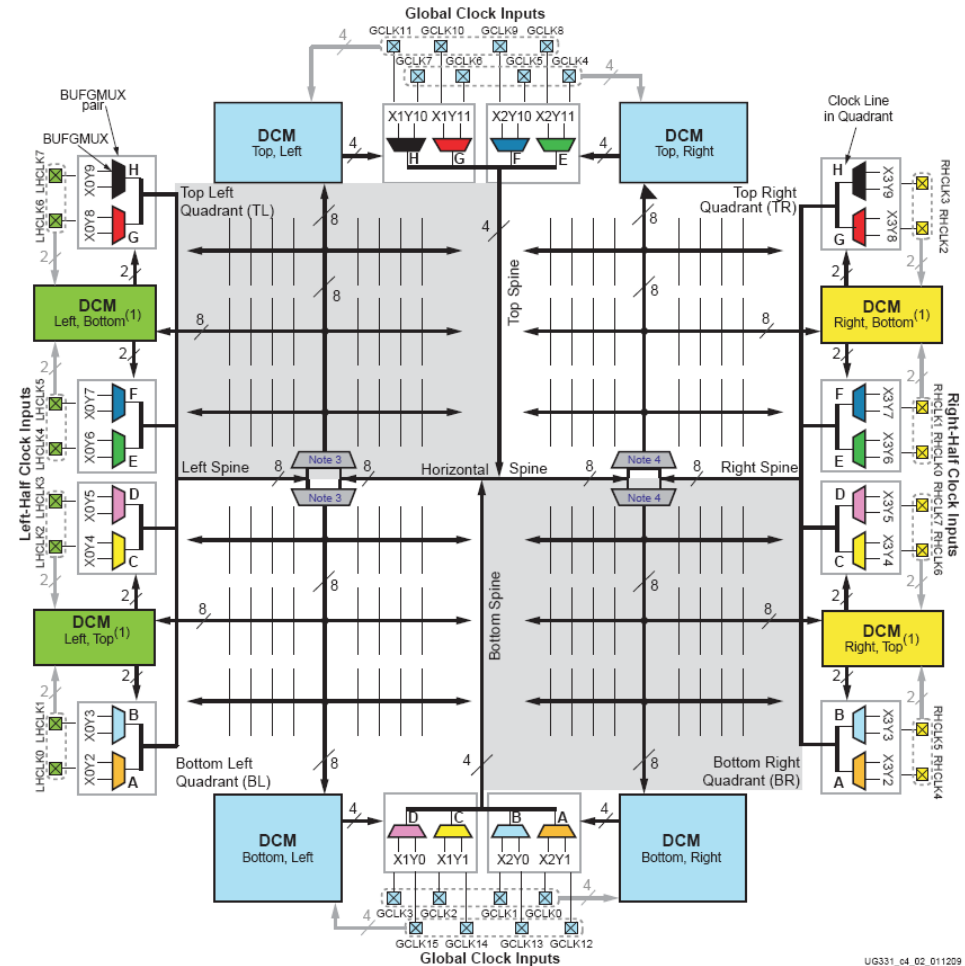

CLK sources

- CLK signal path:
 - CLK input (assigned pins)
 - CLK can be received from even more than one input
 - Differential CLK input is also possible
 - DCM: Digital Clock Management
 - BUFGMUX: multiplexer to choose CLK source
 - Signal on the external CLK pin
 - Output of the CLK management unit
 - Internal signal
 - Clock division unit: Global Routing



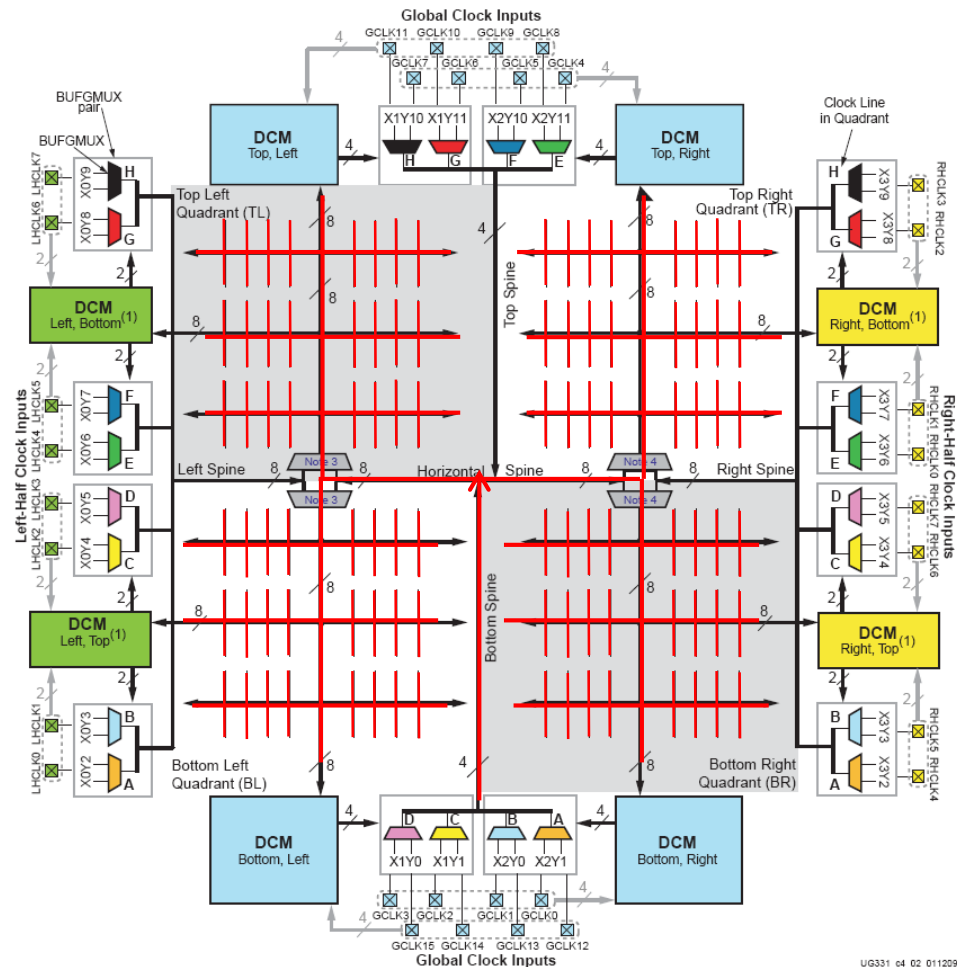
Clock Management Unit

- Special wiring network
- The network can be divided into several CLK domain
- Delay is minimal among the different parts of the IC
 - Fractal-like network structure
 - CLK lines are more or less the same in length



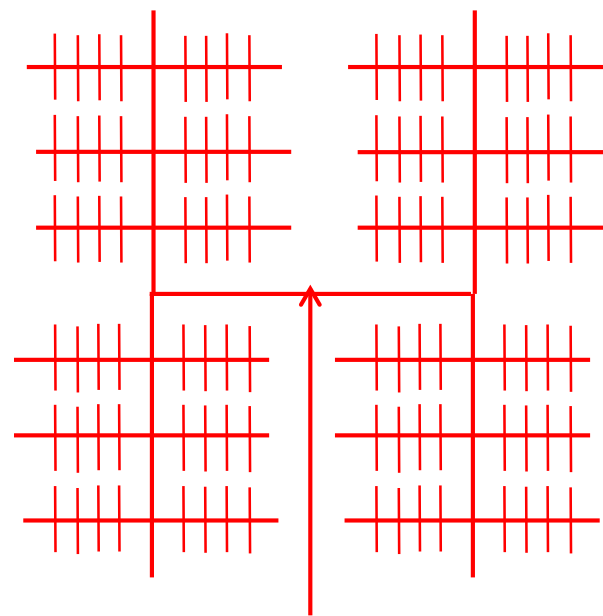
Clock Management Unit

- Special wiring network
- The network can be divided into several CLK domain
- Delay is minimal among the different parts of the IC
 - Fractal-like network structure
 - CLK lines are more or less the same in length



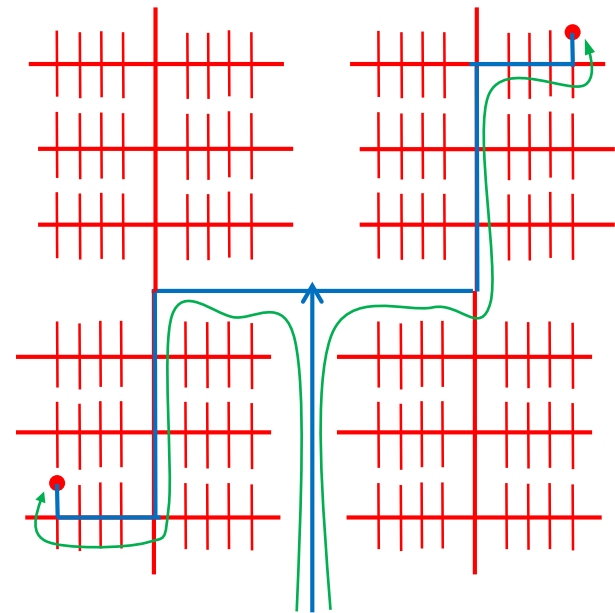
Clock Management Unit

- Special wiring network
- The network can be divided into several CLK domain
- Delay is minimal among the different parts of the IC
 - Fractal-like network structure
 - CLK lines are more or less the same in length



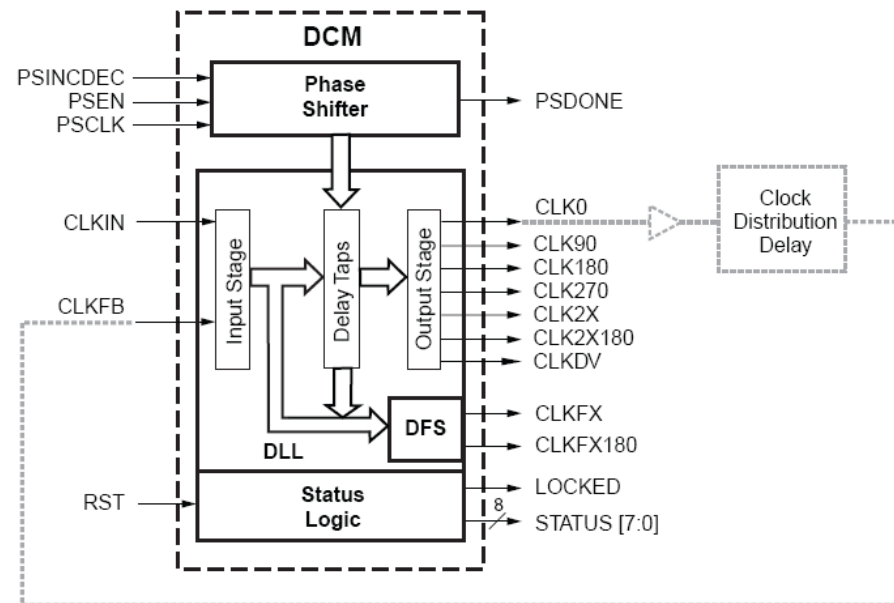
Clock Management Unit

- Special wiring network
- The network can be divided into several CLK domain
- Delay is minimal among the different parts of the IC
 - Fractal-like network structure
 - CLK lines are more or less the same in length



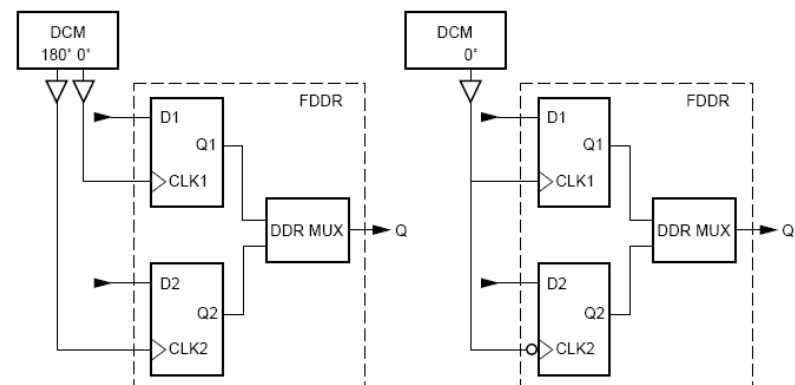
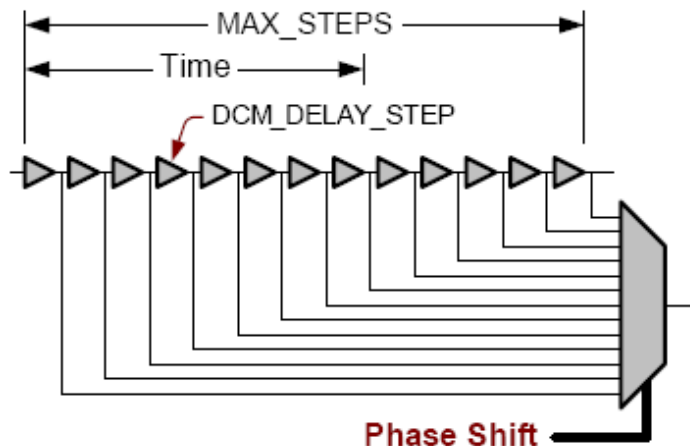
DCM: Digital Clock Manager

- DCM used for:
 - Eliminating CLK shift
 - Phase shifting
 - Multiplication/division of CLK frequency
 - CLK conditioning, restoring duty cycle of CLK
 - CLK buffering, relay
- Capable of generation of high quality CLK from internal signals
- Components:
 - Delay Locked Loop (DLL)
 - Digital Frequency Synthesizer (DFS)
 - Phase Shift (PS)
 - Status Logic (SL)
 - Internal- and external buffer stages



DCM – phase shift

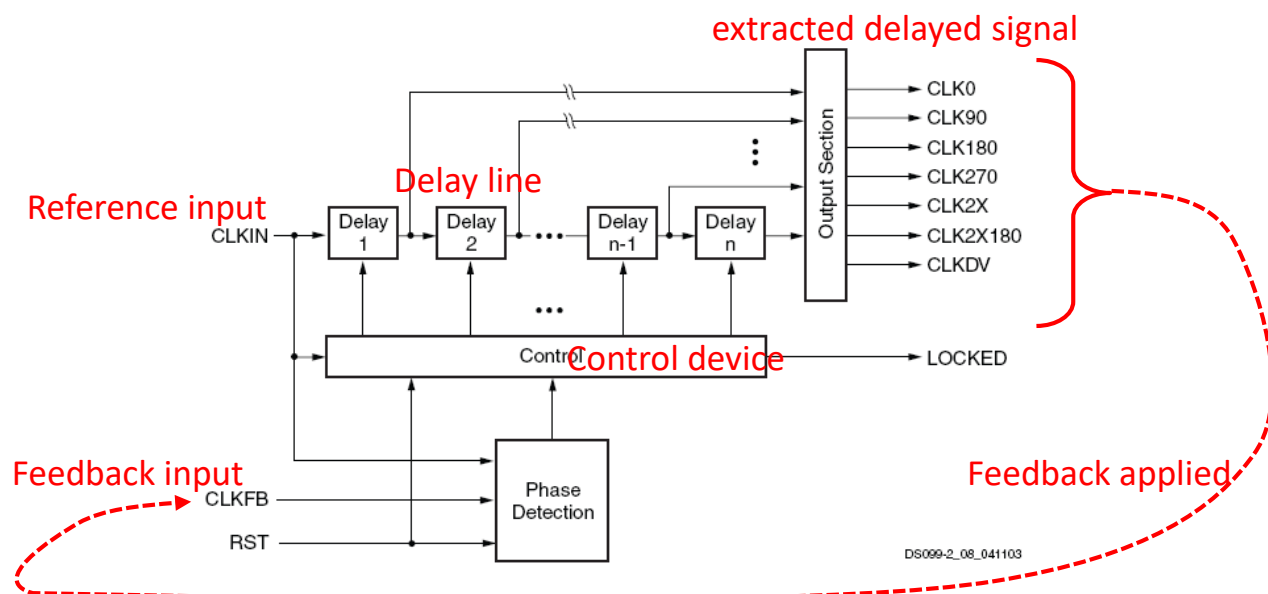
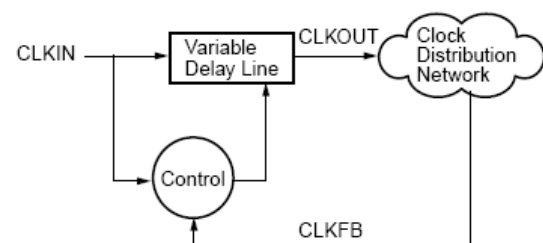
- Phase shift: the incoming CLK is shifted by a certain value of delay by the appropriate signal extraction from the delay line
 - Basically it implements a DLL without feedback
- Example: DDR (Dual Data Rate)
 - Generation of negated and not-negated CLK signal
 - Is it better than inversion? Yes, since 180° phase-shift can be set accurately: in case of inversion the delay of the inverter should be accounted.



DCM – delay line & DLL

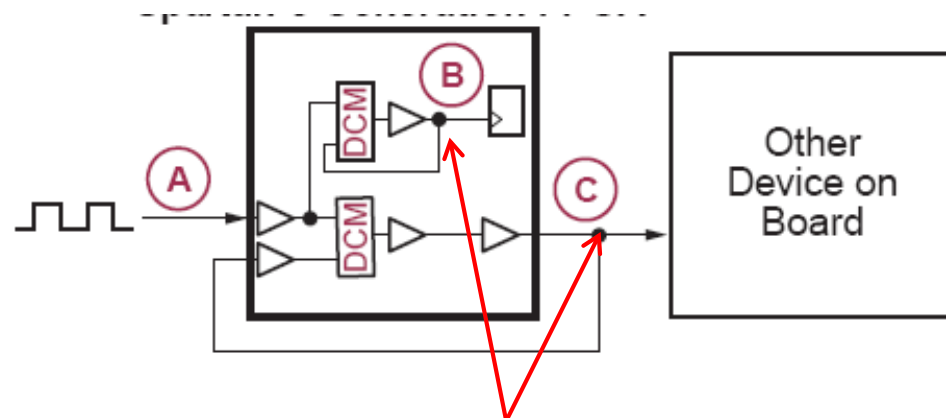
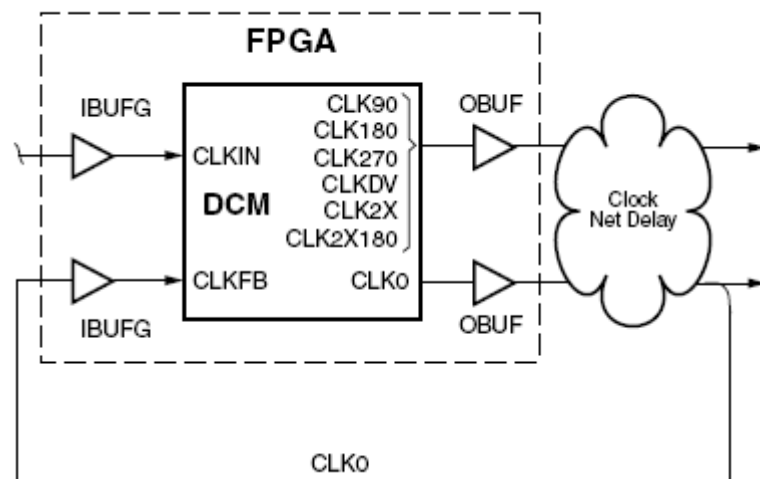
■ Delay Locked Loop

- Similar to phase-locked-loop but now the control is based on the delay between two signals
- The incoming signal is routed to a delay line with variable delay and the delay is tuned until the delay between the two inputs disappears.



DCM – delay line & DLL

- DLL application: compensation of CLK shift inside or outside of the device:
 - CLK signal connected to an internal module or external device is fed back to the DLL input
 - DLL set the delay of the output signal in such a way that no delay occur between the reference signal and the measured output
 - It basically implements a control loop



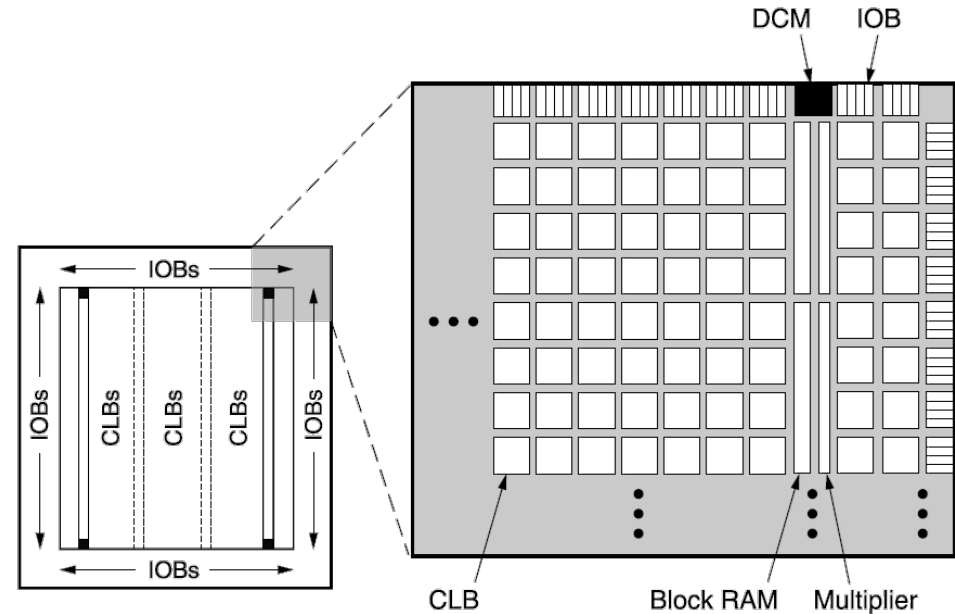
CLK signals appearing at points C and B are measured and both are controlled by a DCM module based on a common reference (A)

Considerations in terms of CLK signals

- Design only synchronous logic networks
 - Use one CLK signal inside one block
 - If slower operation is needed use enabling signals
 - Changing between different CLK domains is complicated, avoid it if possible, take care of it during development
- Synthesizer can buffer the CLK inputs in an autonomous manner in normal case

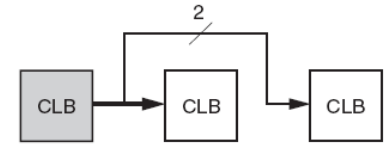
Resources

- Main components:
 - IO Blocks
 - DCM (digital clock management)
 - CLB (Configurable Logic Block)
 - RAM
 - Multiplier
- Synthesiser usually allocate automatically the resources based on the HW description. Nevertheless it is worth to keep in mind the resources available and their main features since it determines the HW description.
 - E.g.: if a shift register does not need a parallel input/output it will not be used up therefore can be implemented more efficiently (see later)



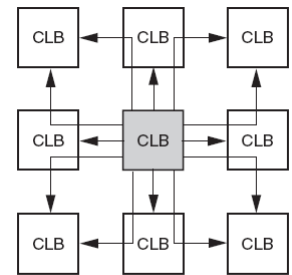
Type of connections

- Connections are basically done by the synthesizer
- Different type of connections exist. Basis of grouping: how far the components to be connected are
- Components logically connected should be close to each other physically as well and let them be connected by a 'rapid line'
- There exist direct connections inside a logic, e.g., carry logic (see later)
- For more details: https://www.xilinx.com/support/documentation/user_guides/ug331.pdf



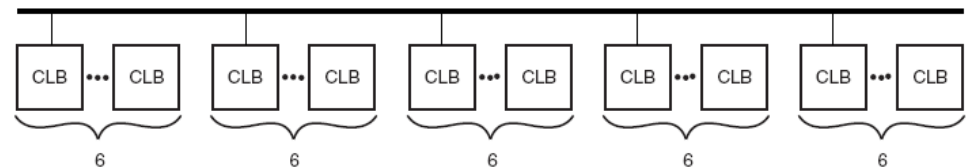
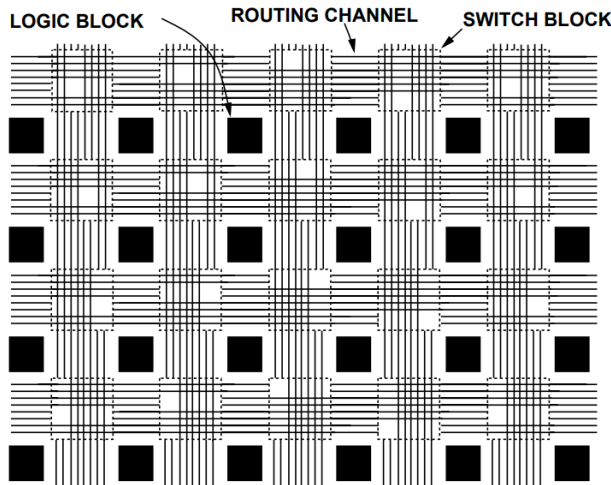
(c) Double Lines

DS099-2_21_040103



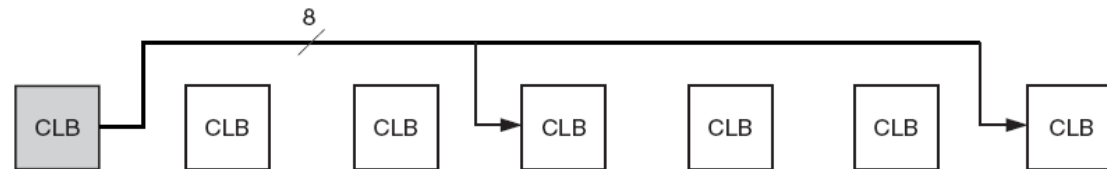
(d) Direct Lines

DS099-2_22_040103



(a) Long Lines

DS099-2_19_040103

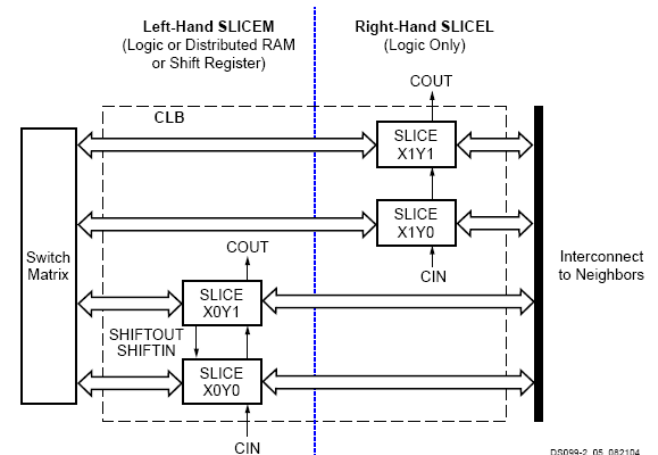
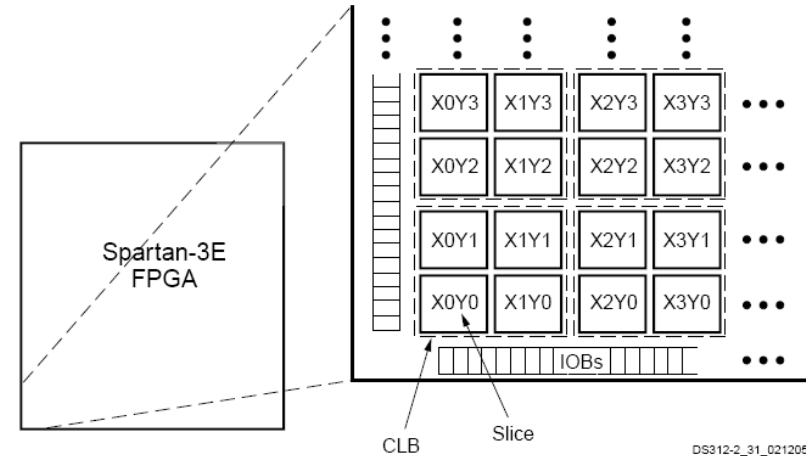


(b) Hex Lines

DS099-2_20_040103

Architecture of CLBs

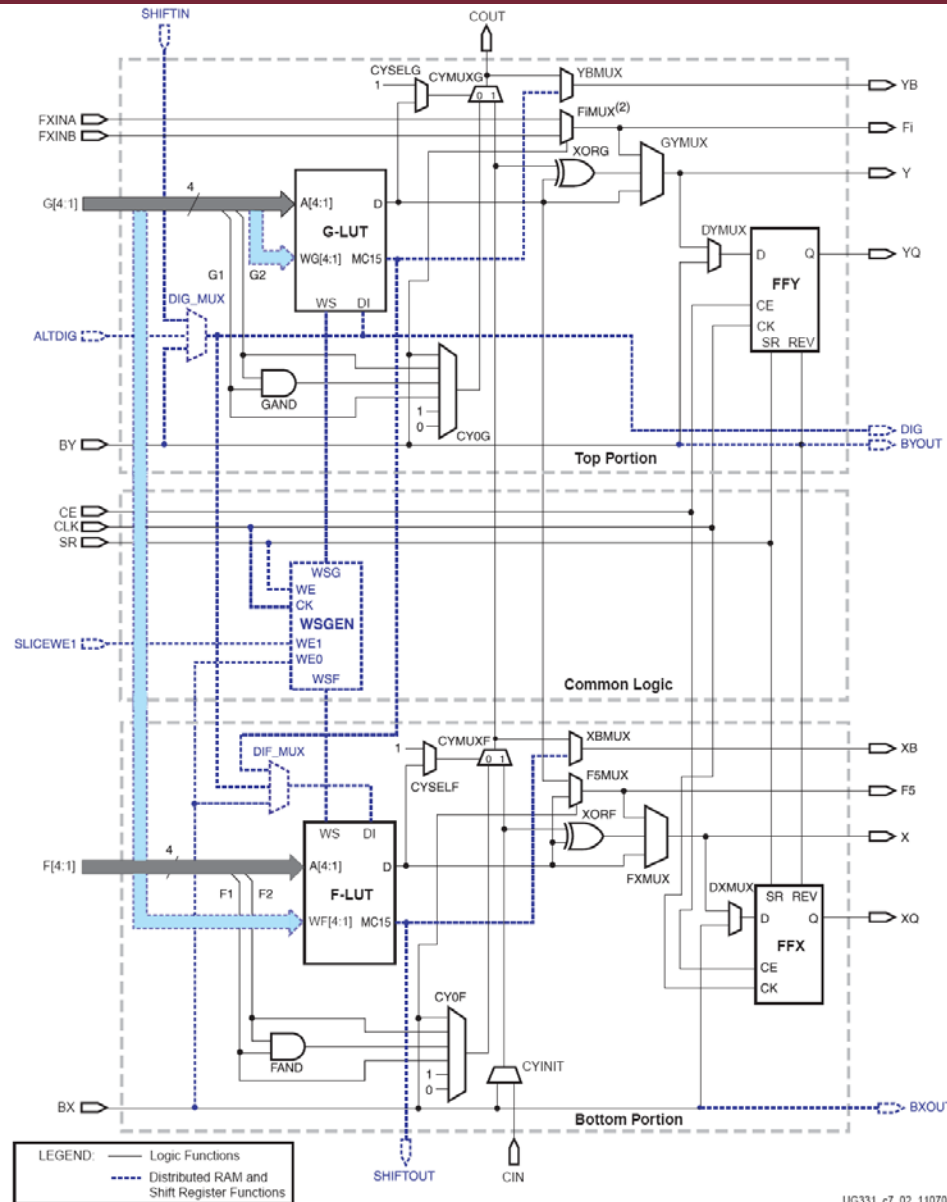
- CLB: Configurable Logic Block
- CLB building element: Slice
 - 1 CLB contains 4 slices
 - The basic functions of the slices are the same but differences can be found
- CLBs can be connected to each other in several ways



Device	CLB Rows	CLB Columns	CLB Total	Slices	LUTs / Flip-Flops	Equivalent Logic Cells	RAM16 / SRL16	Distributed RAM Bits
XC3S250E	34	26	612	2,448	4,896	5,508	2,448	39,168

Architecture of a slice

- LUT: look-up table
 - Implementation of combinational logic
- Flip-flop:
 - Implementation of synchronous logic
- Accelerating arithmetical operation
 - carry propagation
 - direct AND and XOR gates
- Multiplexers

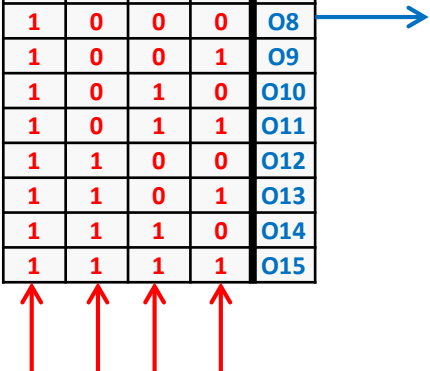


UG331_e7_02_110708

LUT: Look-up Table

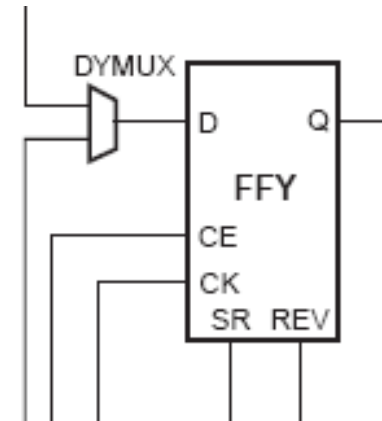
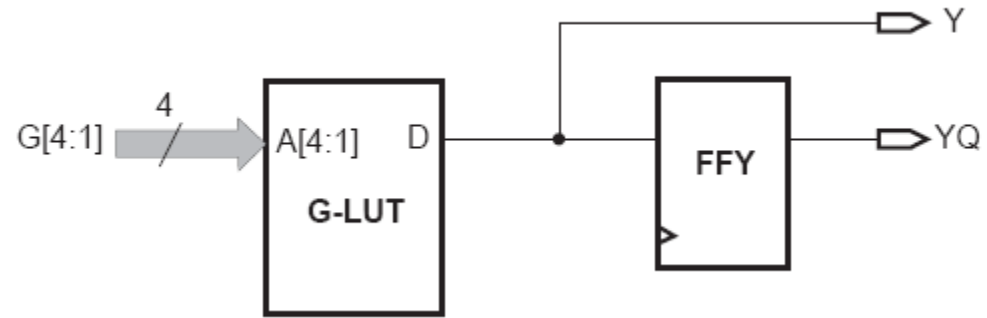
- LUT: can be considered as a 1-bit wide memory of 4 addressing bits (RAM)
- Implementation of 4-bit input 1-bit output logic functions
 - Direct implementation: the output is given to each input code
- If more variables are used then the LUTs inside and outside of the slices can be connected to each other hence expanding the inputs

Input code				out
0	0	0	0	00
0	0	0	1	01
0	0	1	0	02
0	0	1	1	03
0	1	0	0	04
0	1	0	1	05
0	1	1	0	06
0	1	1	1	07
1	0	0	0	08
1	0	0	1	09
1	0	1	0	010
1	0	1	1	011
1	1	0	0	012
1	1	0	1	013
1	1	1	0	014
1	1	1	1	015



Flip flop

- Basic component for synchronous digital network
 - State machine: combinational logic (LUT)+FF
- Widely configurable
 - Set/reset input
 - Clock enable
 - Data input can be selected
 - CLK can be inverted in some cases
 - Latch

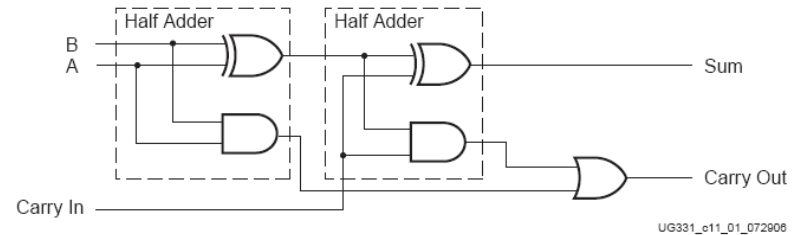


Implementation of functional blocks

- Architecture of FPGAs facilitates the implementation of functional blocks
 - ALU: arithmetical and logical unit (addition, subtraction)
 - multiplication: not always but there exist solutions that support multiplication
 - Multiplexer
 - Shift regiszter
 - RAM

ALU: addition

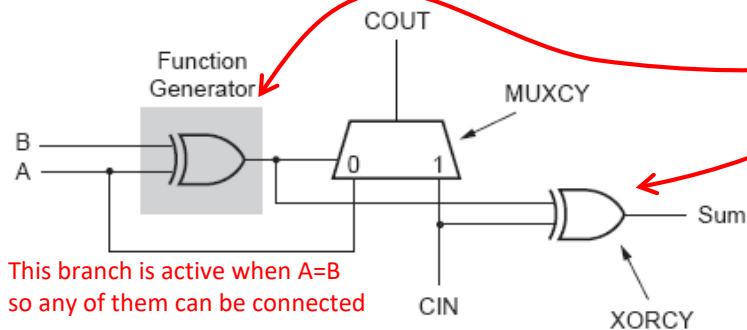
- Addition: $A \text{ xor } B \text{ xor } \text{Carry}$
 - Partial result of inputs A and B is generated by the LUT
 - Result $(A+B)$ and the sum with C is generated by a XOR gate \rightarrow no need for two LUTs
- carry: determines whether the carry bit shall be propagated or modified:



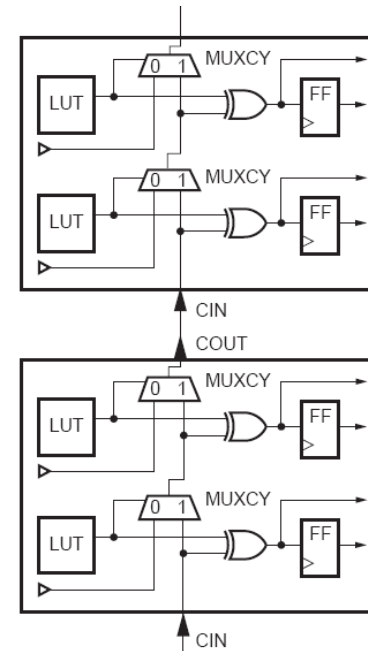
UG331_e11_01_072808

A	B	Propagate	Generate
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

COUT?=CIN



több bites összeadó lánc:



Implementation of comparators

Evaluation of equivalence ($A=B$?):

- All bits shall be matched
- Using a LUT equivalence of two bit pairs can be evaluated (two XNOR applied to an AND, but in a LUT it is mapped into a truth-table)
- Using an internal multiplexer-chain it is propagated whether previously matched all the bits or not. In case of difference, 0 is propagated forward the chain

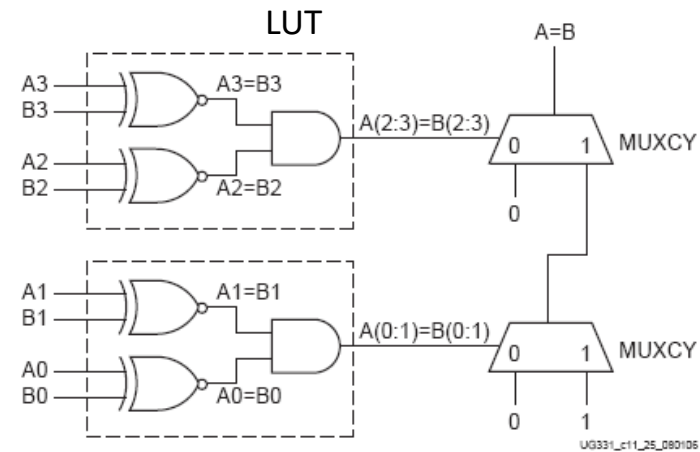


Figure 9-25: Equality Comparator in One Slice

Evaluation of inequality ($A>B$):

- The results for lower significant bits are propagated in this case as well
- Let's have an N bit variable
- If $A_n=B_n$ ($n: N-1...0$):
 - The result of the lower stages are propagated
- If $A_n \neq B_n$:
 - If $A_n=1$, then $A>B$
 - Otherwise $A \leq B$

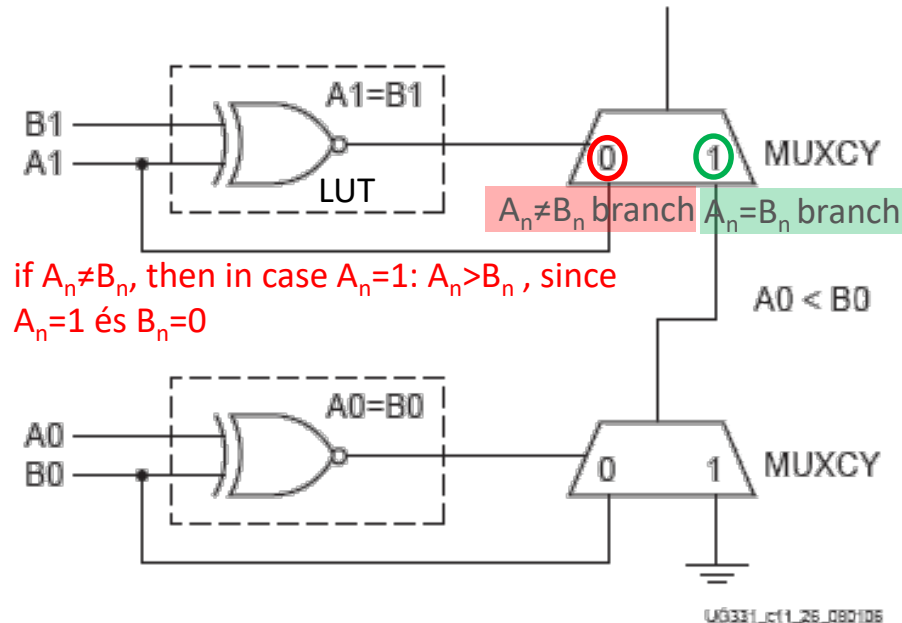


Figure 9-26: Magnitude Comparator in One Slice

Comparison can be efficiently implemented using fast propagation

Binary counter

Counter:

- 000
- 001
- 010
- 011
- 100
- 101
- 110
- 111

Lower bit is kept changing

Bit n-th changes only when every previous bits are 1

The propagation multiplexers propagate the lower bits if the actual bit is a 1. The propagation will be a 1 for bit n-th if all the lower bits are one.

Using this technique addition logic is not needed for the implementation of counter.

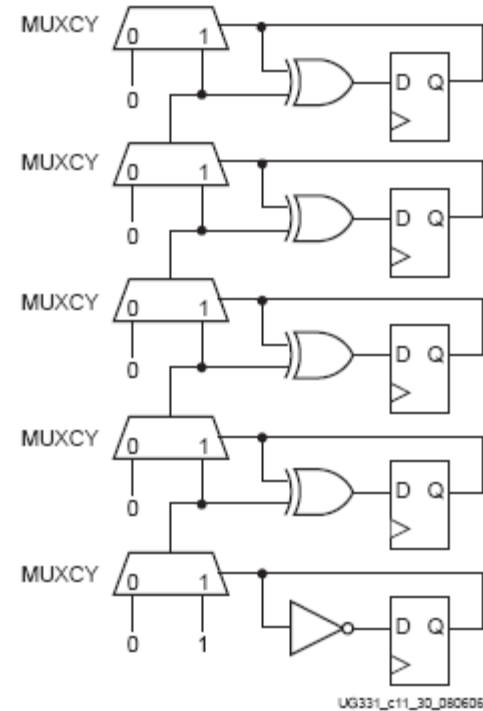
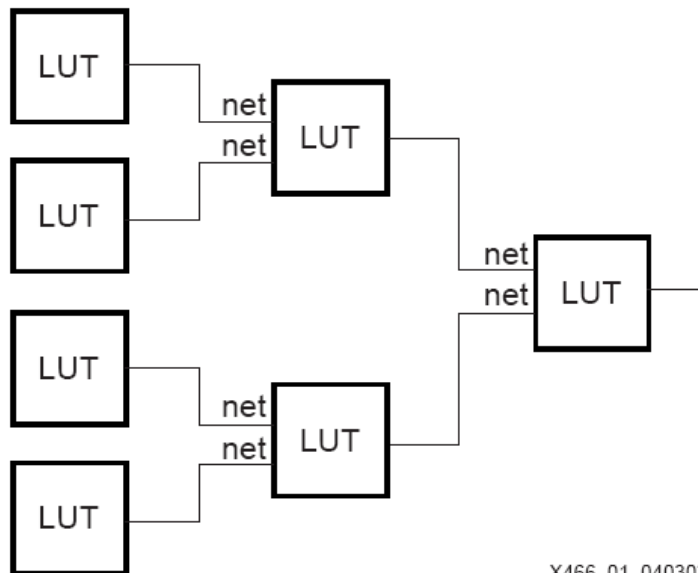


Figure 9-30: Binary Counter Using D Flip-Flops

Multiplexer

- 1-input multiplexer can be implemented using a LUT :
 - Enable line, select line and two inputs are needed
- Multiple-input multiplexer:
 - Implementation based on LUTs: cascading 1-input multiplexers



X466_01_040303

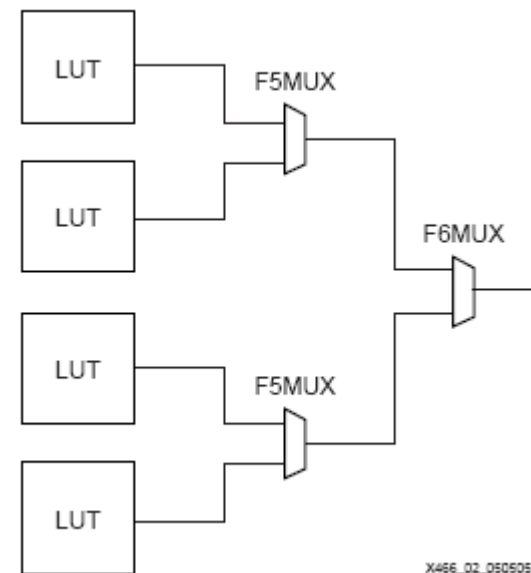
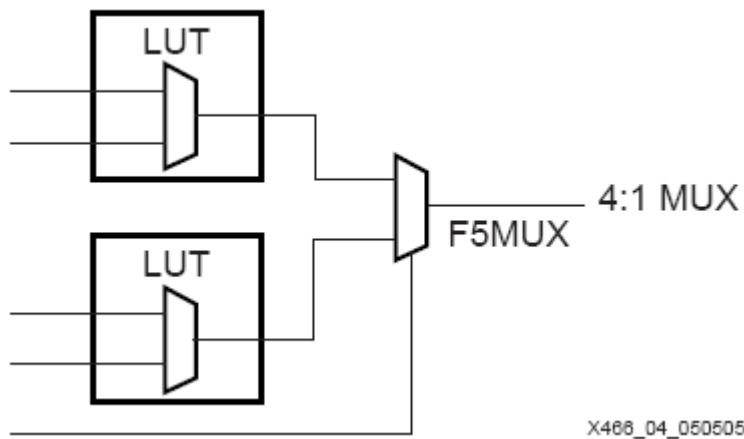
	Input code				out
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	0	1	1	1
1	0	1	0	0	0
1	0	1	1	1	1
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	1	1
1	1	1	1	1	1

en sel in1 in0

A blue arrow points from the right side of the table to the right.

Multiplexer

- Cascading based on only LUTs: multi-level logic → slower signal propagation
- Logic inside slices: using further auxiliary components LUT-multiplexers can be connected to each other inside a slice
- Using internal multiplexers multi-input functions can be implemented with 4-input LUTs



Multiplexer

- Synthesizer recognize it and maps into the FPGA, not needed to implement is 'manually'
- If-then-else structures: not always recognized
 - Use simple, clean structures
 - Output is required in each branches
 - Minimize the number of input variables: easier to recognize what is the intended operation
- case structure: it should be used for describing a multiplexer
 - All possible outcomes has to be covered

Shift register

- General implementation: using flip-flops in slices can be synthesized a general purpose shift-register
 - Flip-flops in slices should be connected in series
 - Serial/parallel in-/output can be implemented
- Resource-saving solution (SRL16 operation mode):
 - Configuration SRAM cells of LUTs + LUT logic can be used as shift register
 - A LUT is basically a 4-input multiplexer, therefore input code defines the extraction point of the shift register

