FPGA – VHDL

Cristian Alejandro Sisterna, MSc
Universidad Nacional de San Juan
Argentina

FPGA

FIELD PROGRAMMABLE GATES ARRAY

Agenda

- History and trends
- FPGA Configuration and Routing Cells
- FPGA Logic
- FPGA Dedicated blocks
- Digital System Based on FPGA, Design Flow
  - Synthesis
  - Simulation
- FPGA SoC
- FPGA Configuration options

FPGA: Very Competitive Market

FPGA Vendors

~~2000~~

2015

- Altera Inc.
- ~~Lucent Technology~~
- ~~Intel Corp.~~
- ~~QuickLogic~~
- Lattice Semiconductors
- ~~Vantis~~
- ~~Actel Corp.~~ MicroSemi
- ~~Philips Semiconductors~~
- ~~Motorola Semiconductors~~
- ~~Cypress~~
- ~~AMD~~
- Xilinx

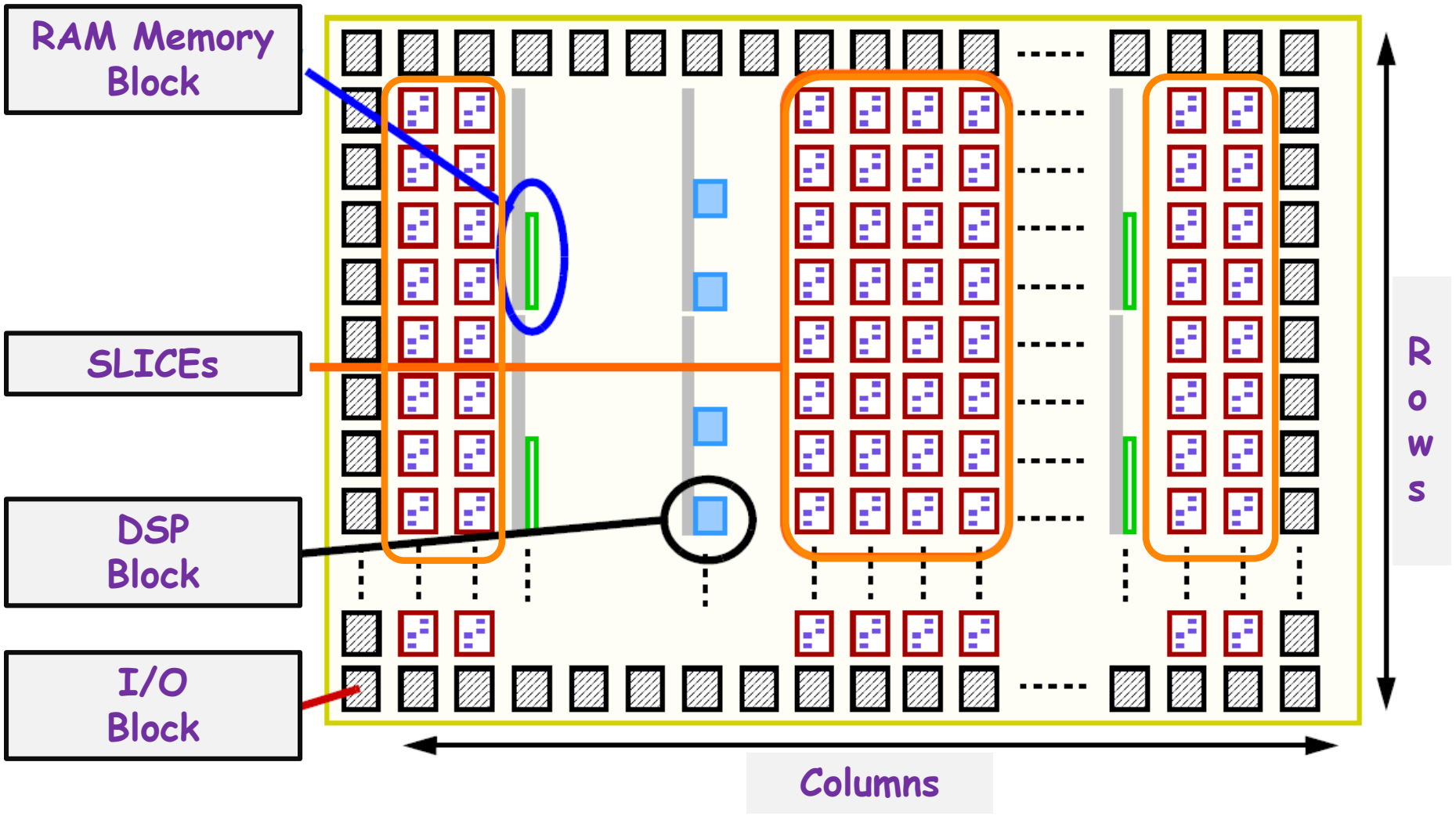
Transistor Counts: FPGA-Microprocessors

Device	Transistor Count	Year	Designer	Process	Area
Intel 4040	2.300	1971	Intel	10µm	12mm^2
MOS 6502	3.510	1975	WDC	8µm	21mm^2
8086	29.000	1978	Intel	3µm	33mm^2
XC2000	1500 (gates)	1985	Xilinx		
80486	1.180.235	1989	Intel	1µm	213mm^2
Pentium II	7.500.000	1997	Intel	0.35µm	113mm^2
Virtex	70.000.000	1997	Xilinx		
Virtex II	350.000.000	2000	Xilinx	130nm	
Pentium 4	184.000.000	2006	Intel	90nm	90mm^2
Virtex-5	1.1000.0000	2006	Xilinx	65nm	
Core 2 Duo	411.000.000	2007	Intel	45nm	107mm^2
Stratix V	3.8000.000.000	2011	Altera	28nm	
Quad Core i7	1.400.000.000	2014	Intel	22nm	177mm^2
Octal Core i7	2.600.000.000.	2014	Intel	22nm	355mm^2
Virtex Ultrascale	20.000.000.000	2014	Xilinx	20nm	

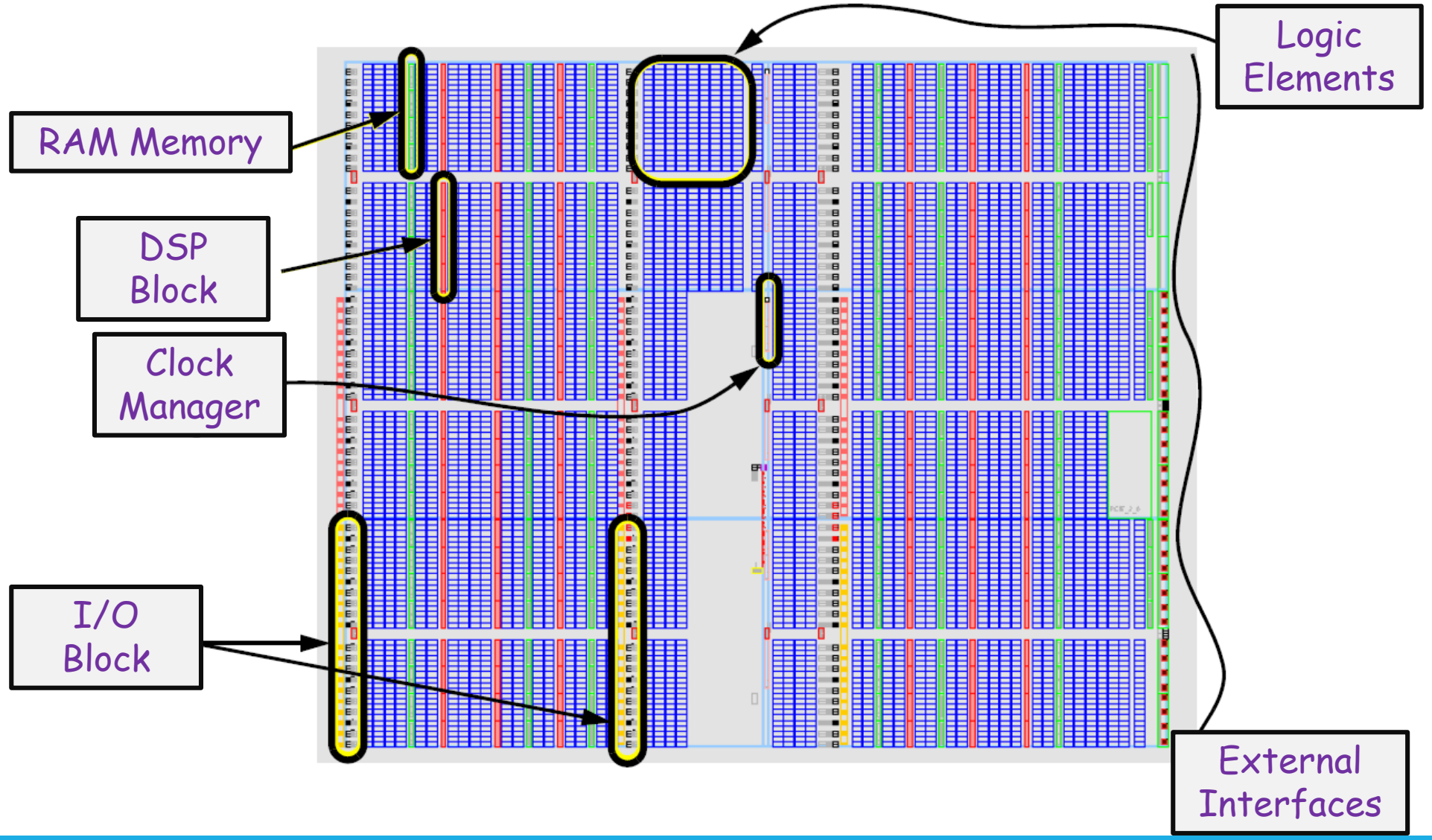
FPGA – What's inside ?

- Programmable Logic
  - Flips-Flops
  - Look-Up-Tables (LUTs)
- Dedicated Blocks:
  - Memory
  - Clock Management
  - DSP blocks
  - Hard coded processor(s) – Hard Core
  - Gigabits serial transmission/reception
  - Ethernet controller
  - External Memory controllers
  - . . .
- IO Blocks
- Interconnections and Routings

FPGA Architecture

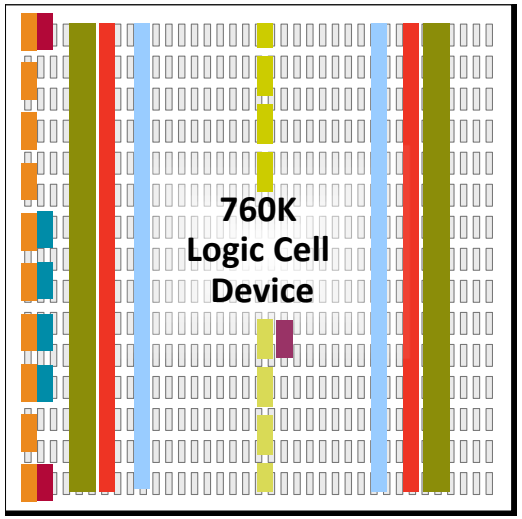


FPGA Architecture



FPGA Architecture

Virtex-6 FPGAs



FIFO Logic

Tri-mode EMAC

System Monitor

Common Resources

LUT-6 CLB

BlockRAM

DSP Slices

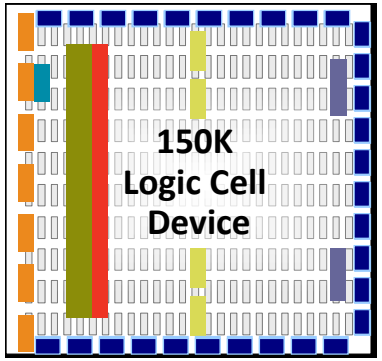
High-performance Clocking

Parallel I/O

HSS Transceivers*

PCIe® Interface

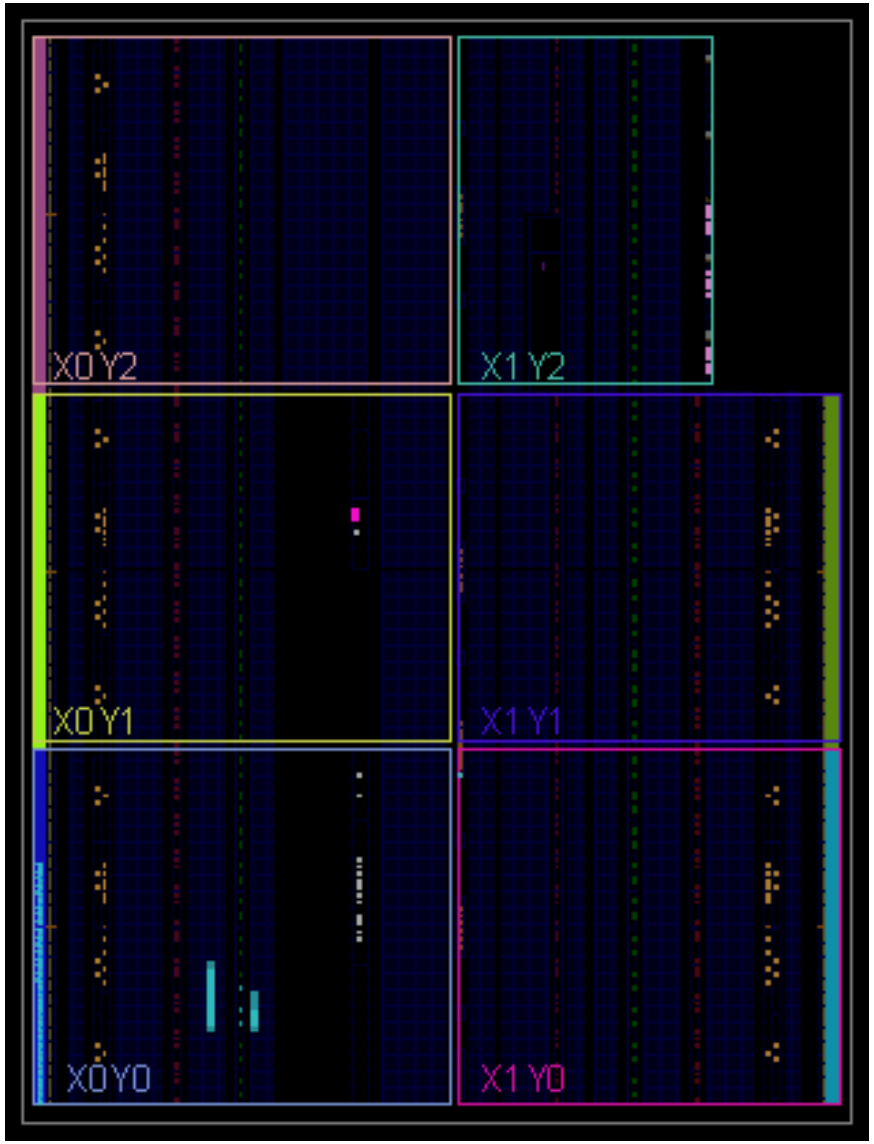
Spartan-6 FPGAs



Hardened Memory Controllers

3.3 Volt compatible I/O

FPGA Architecture - Artix 7 Internal View

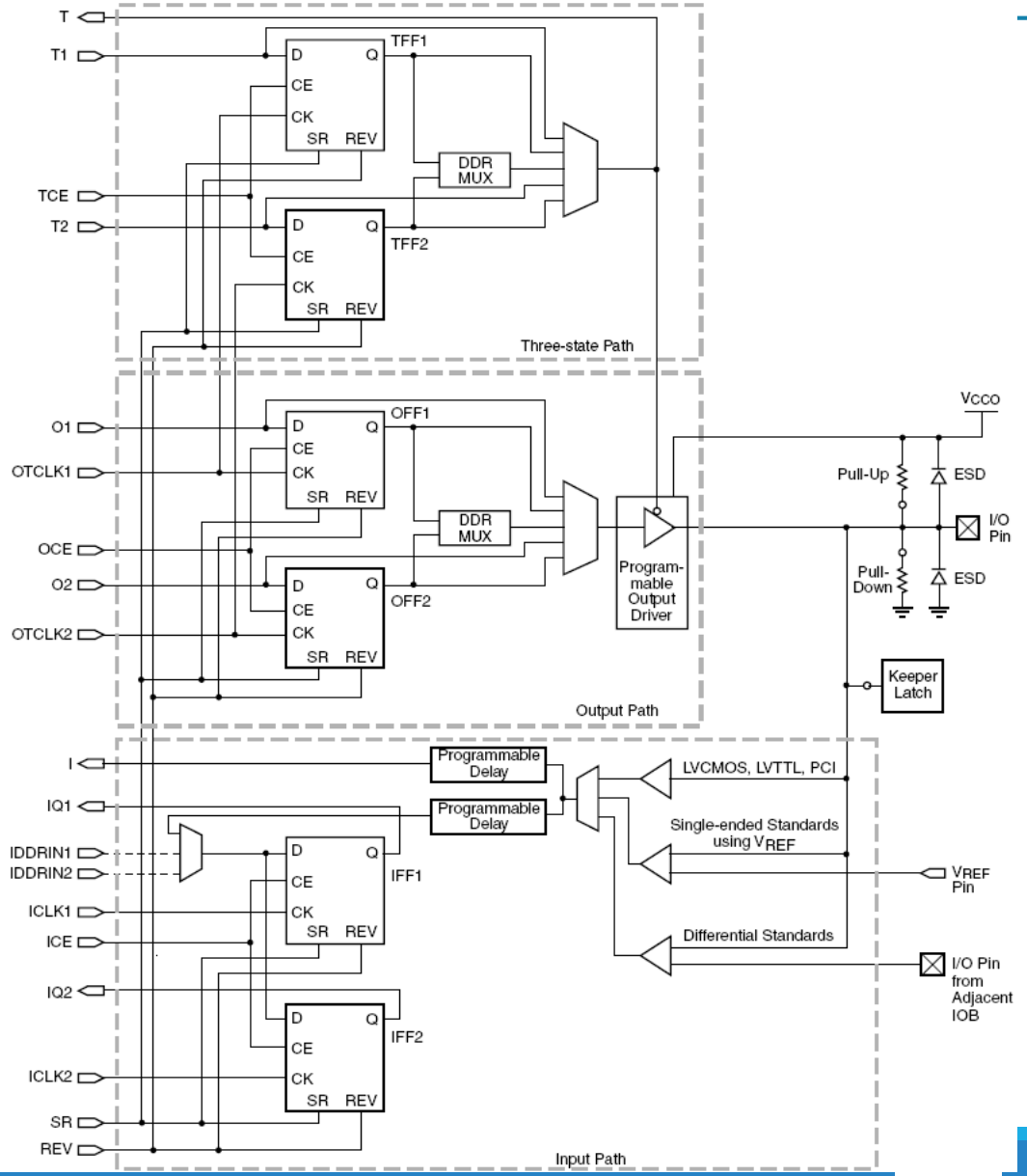


7-Series FPGA Families

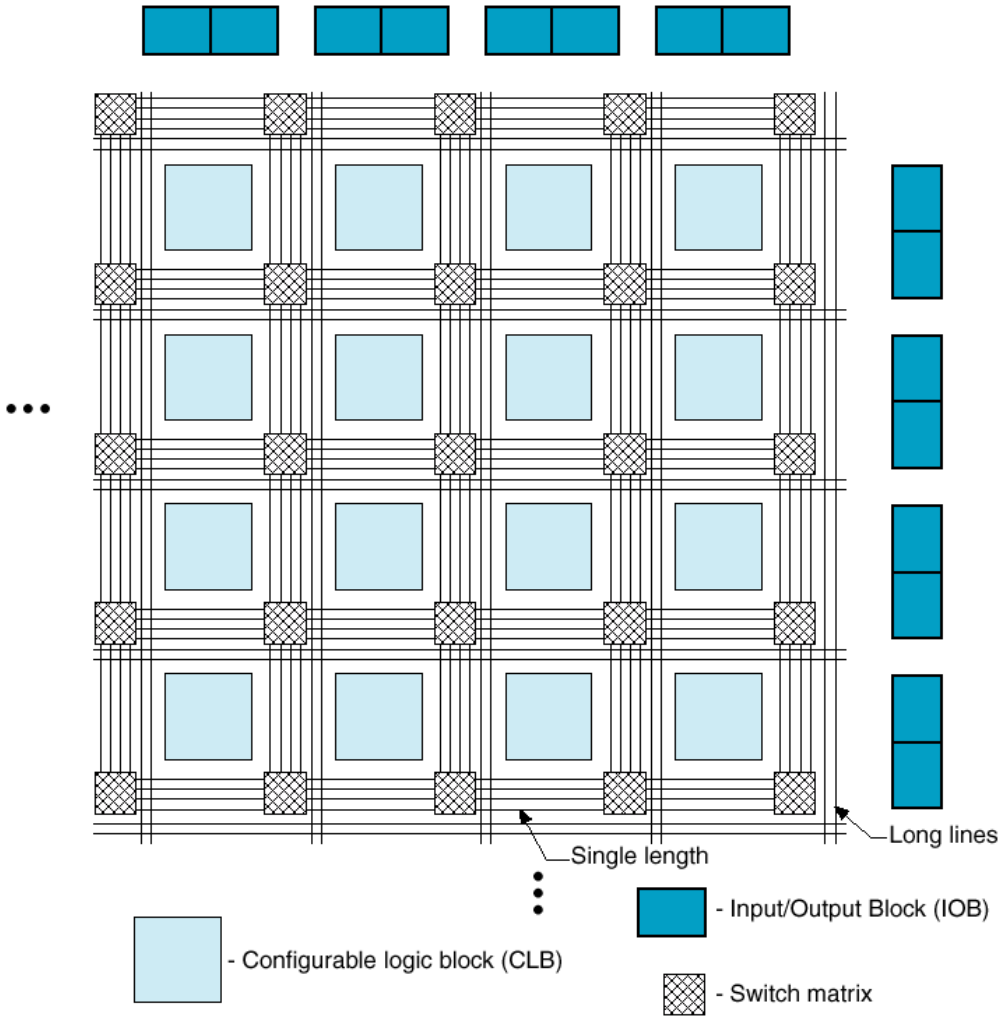
Maximum Capability	ARTIX-7 Lowest Power and Cost	KINTEX-7 Industry's Best Price/Performance	VIRTEX-7 Industry's Highest System Performance	ZYNQ-7000 Extensible Processing Platform
Logic Cells	20K – 355K	70K – 480K	285K – 2,000K	30K – 350K
Block RAM	12 Mb	34 Mb	65 Mb	240KB – 2180KB
DSP Slices	40 – 700	240 – 1,920	700 – 3,960	80 – 900
Peak DSP Perf.	504 GMACS	2,450 GMACs	5,053 GMACS	1080 GMACS
Transceivers	4	32	88	16
Transceiver Performance	3.75Gbps	6.6Gbps and 12.5Gbps	12.5Gbps, 13.1Gbps and 28Gbps	6.6Gbps and 12.5Gbps
Memory Performance	1066Mbps	1866Mbps	1866Mbps	1333Mbps
I/O Pins	450	500	1,200	372
I/O Voltages	3.3V and below	3.3V and below 1.8V and below	3.3V and below 1.8V and below	3.3V and below 1.8V and below

FPGA - Configuration & Routing Cells

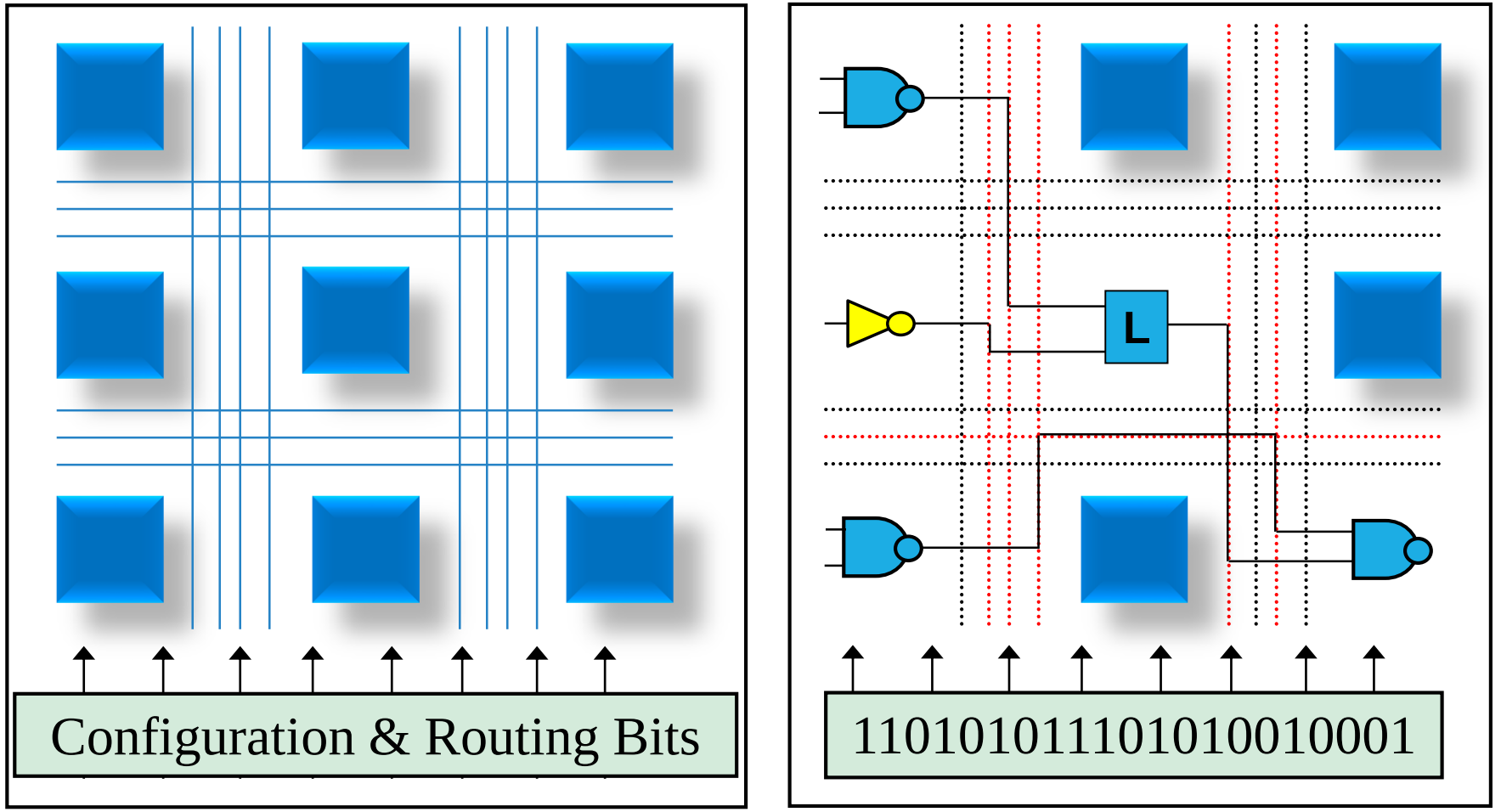
FPGA Configuration & Routing



FPGA Routing Options



Logic and Routing Configuration



Source: Actel

Technology for Configuring the FPGA

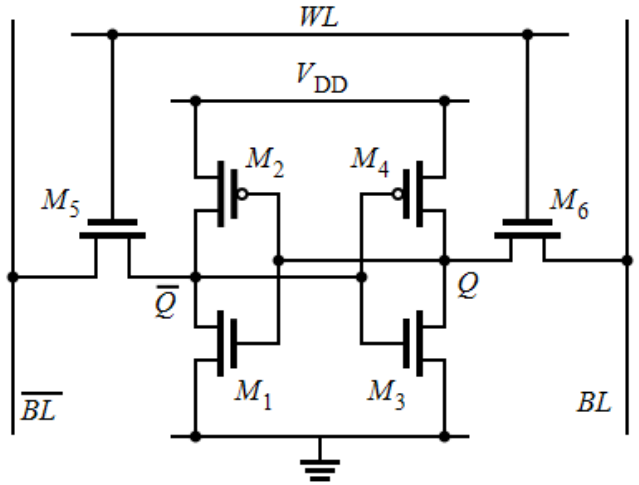
The basic cell for configuring both the logic elements and the internal routings and interconnections can be based on one of the following technologies:

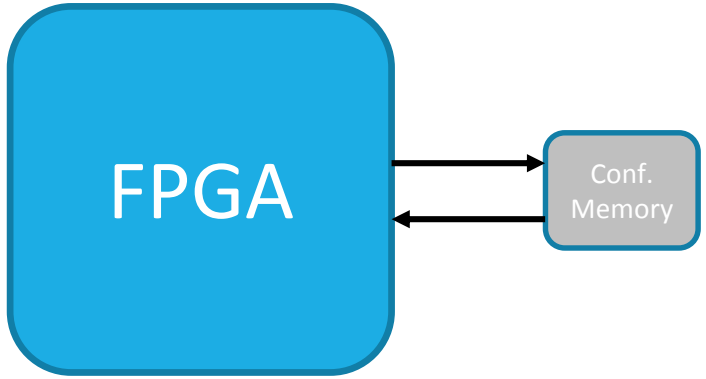
- SRAM
- Anti-Fuse
- Flash
- Flash & SRAM

SRAM Based FPGA

Advantages

- Standard fab process
- Very low cost
- Process highly tested
- High yield
- Infinitely reprogrammable





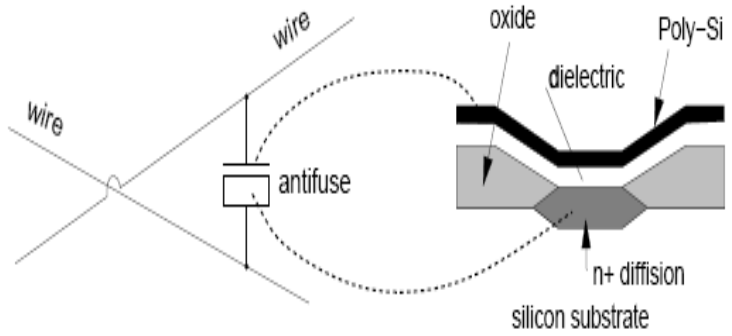
Disadvantages

- Volatile cell
- Long routing delays
- Slow configuration process (~500ms)
- Need an external configuration memory
- Unsafe connection between FPGA-memory

Anti-Fuse Based FPGA

Advantages

- No Volatile
- Small interconnection delays
- No sensible to ionic particles
- Commonly used in special systems
- No need of external configuration memory



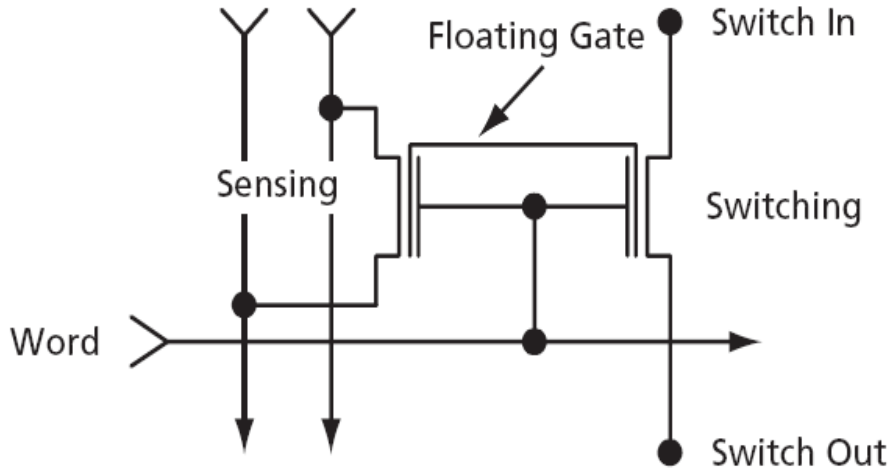
Disadvantages

- Specific fab process
- Very high cost
- One-time programmable (OTP)
- Long and tough verification process
- Yield relatively low

Flash Based FPGA

Advantages

- No volatile
- Small interconnection delays
- Low sensibility to ionic particles
- Used in special systems



Disadvantages

- Fab process still expensive
- Slow configuration process (~3-5 seg)

Flash-SRAM Based FPGA

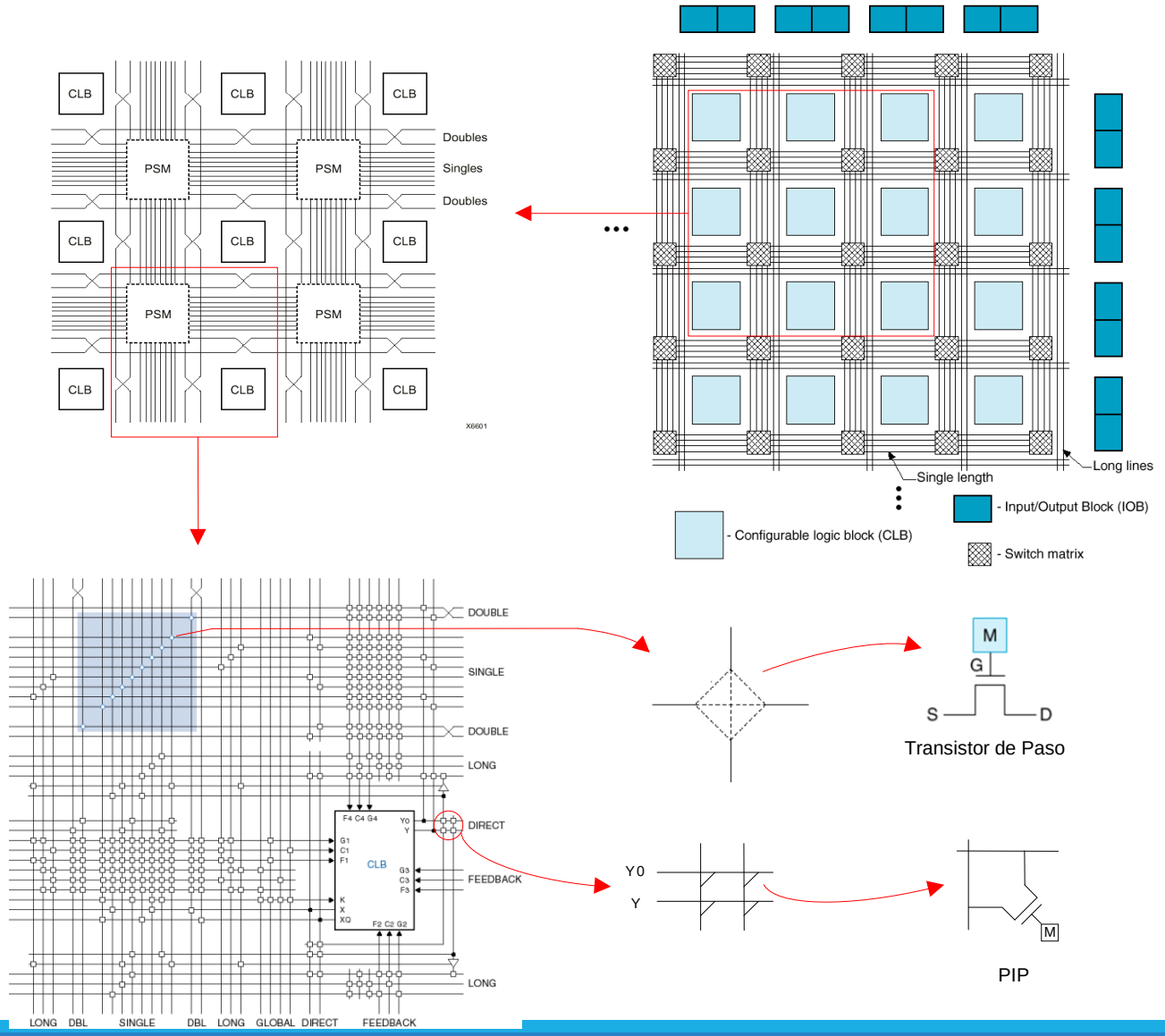
Advantages

- It can be said that is No Volatile
- Configuration time slow (~<1ms)
- SRAM cells can be used during the debug process
- No need of external configuration memory
- Safe system (safe bitstream)

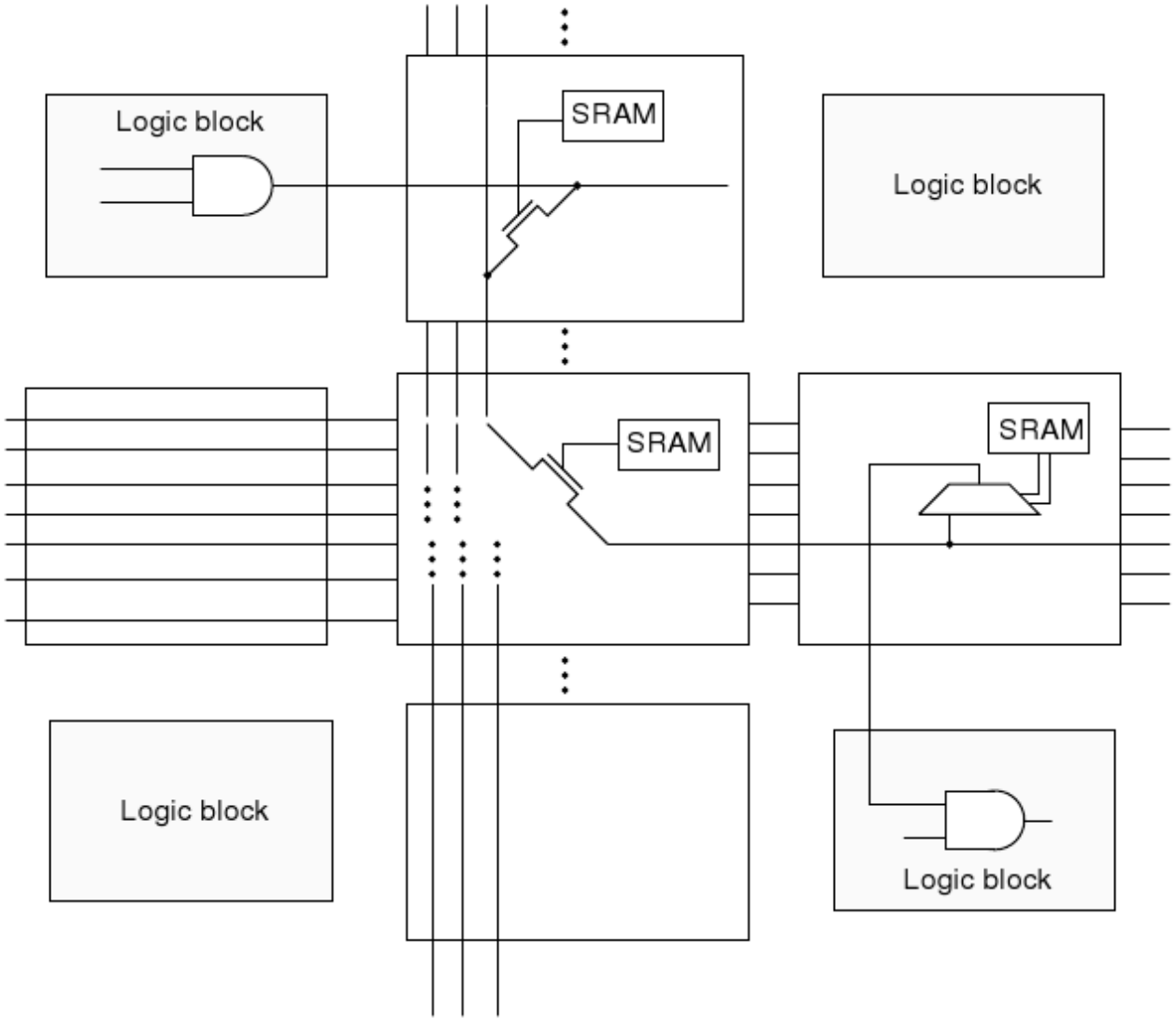
Disadvantages

- Fab process still expensive

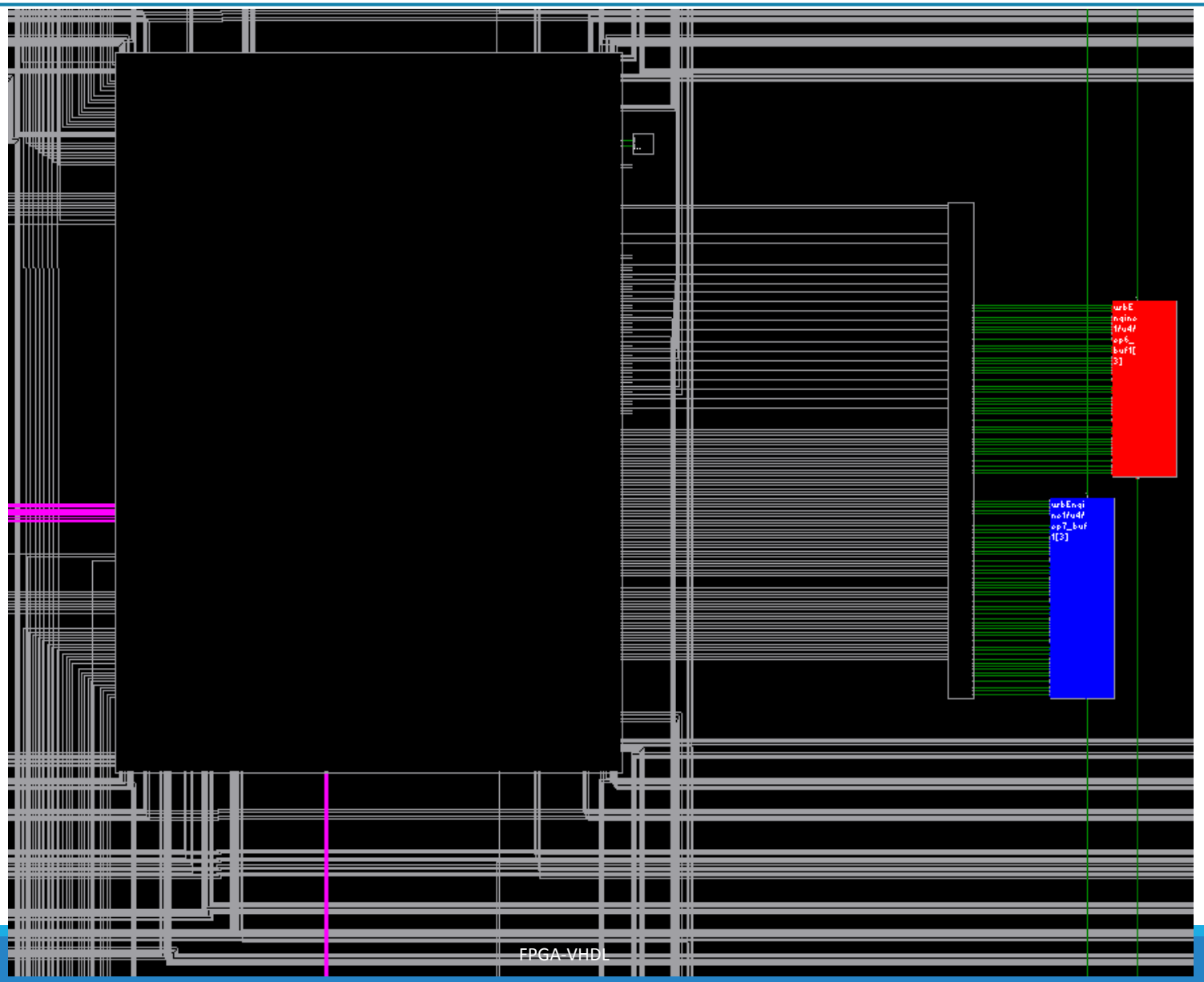
FPGA Routing & Interconnecting



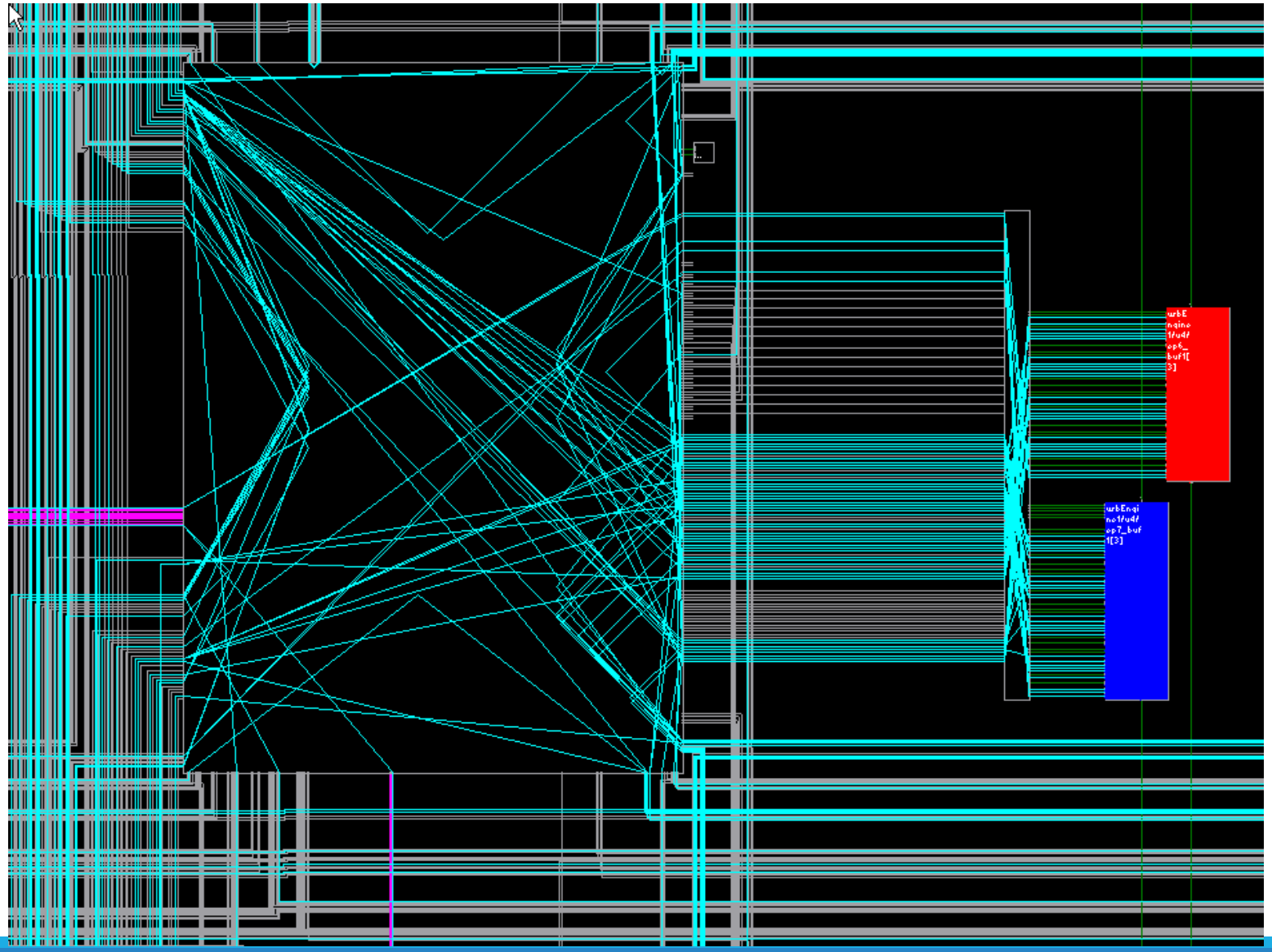
FPGA Routing & Interconnecting



FPGA Routing & Interconnecting

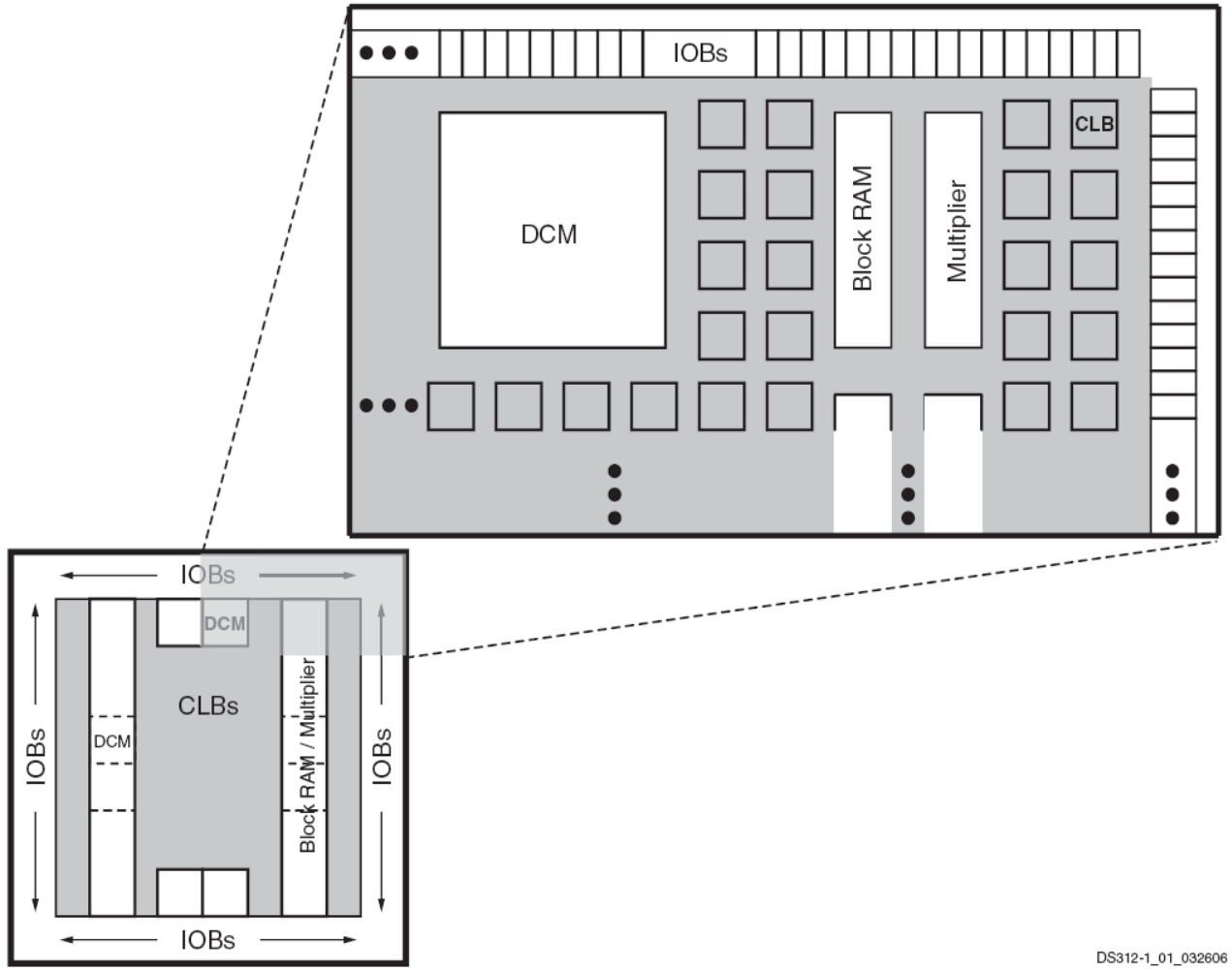


FPGA Routing & Interconnecting

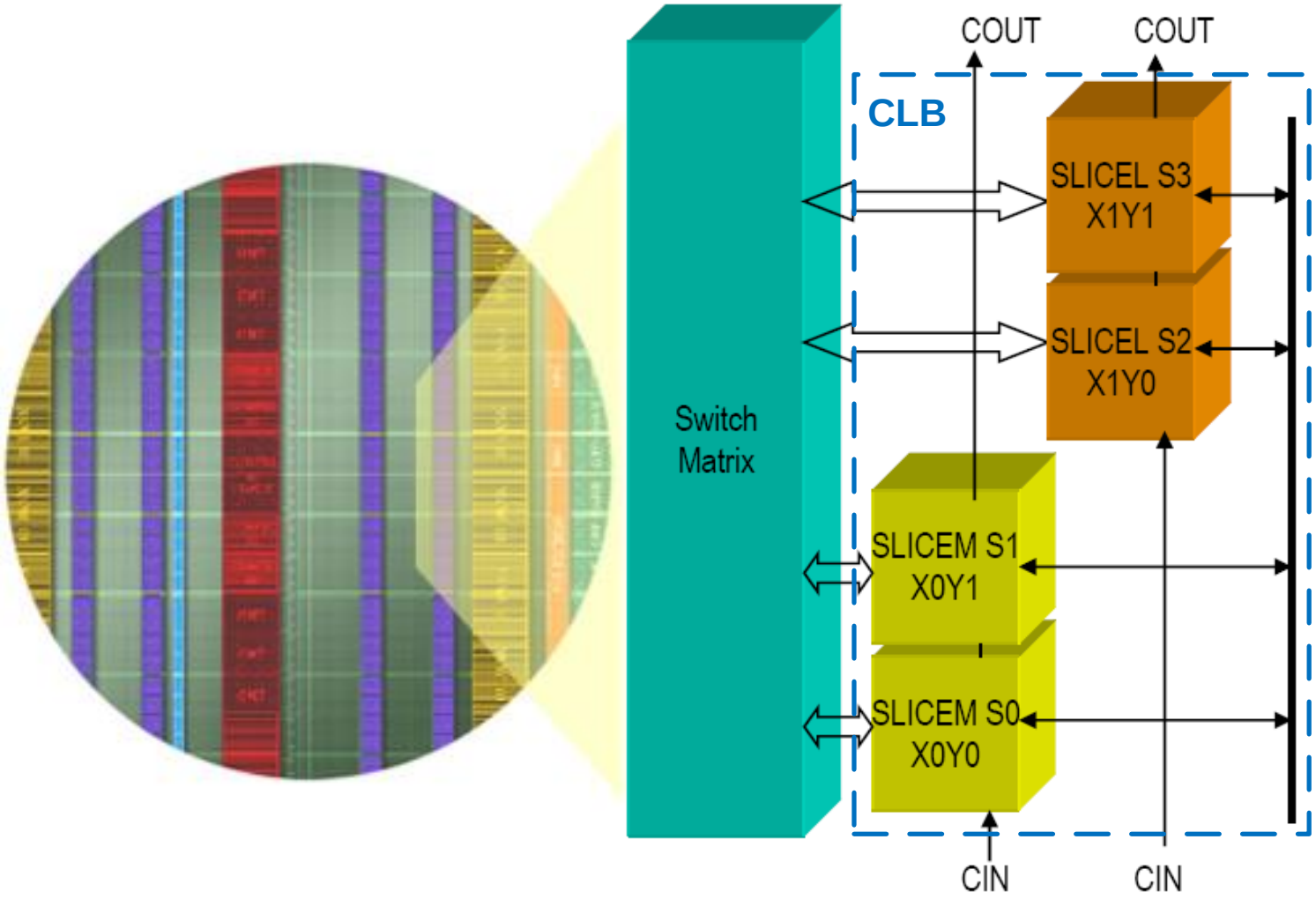


FPGA Resources

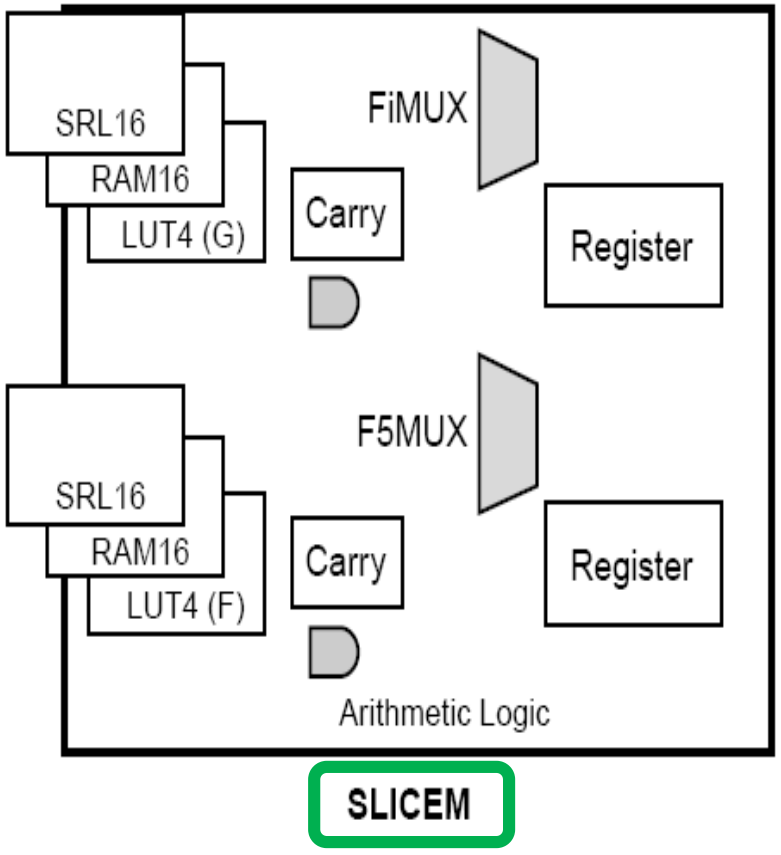
FPGA Programmable Logic



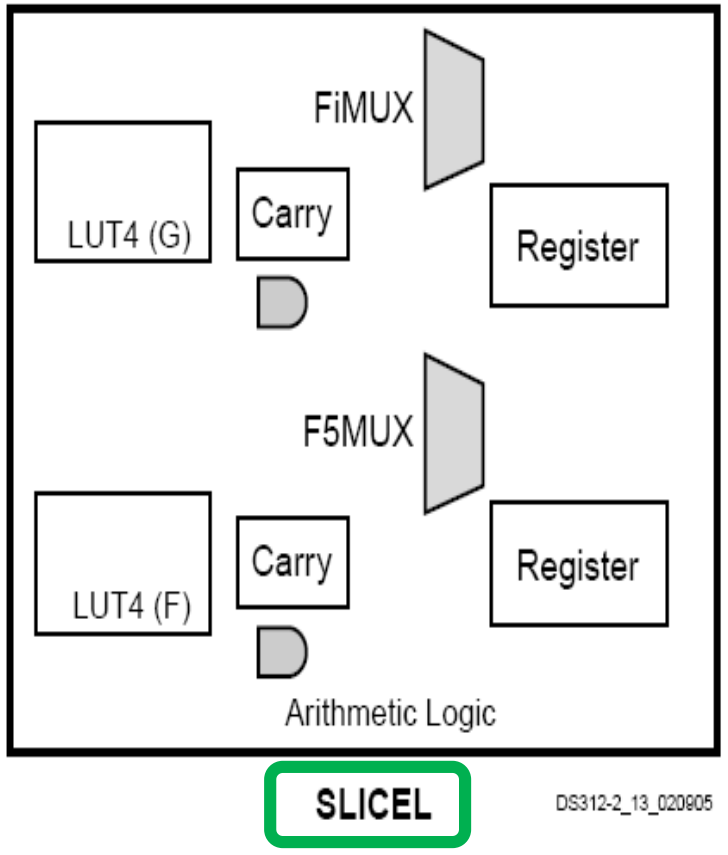
Configurable Logic Block and SLICEs



Different Type of SLICEs

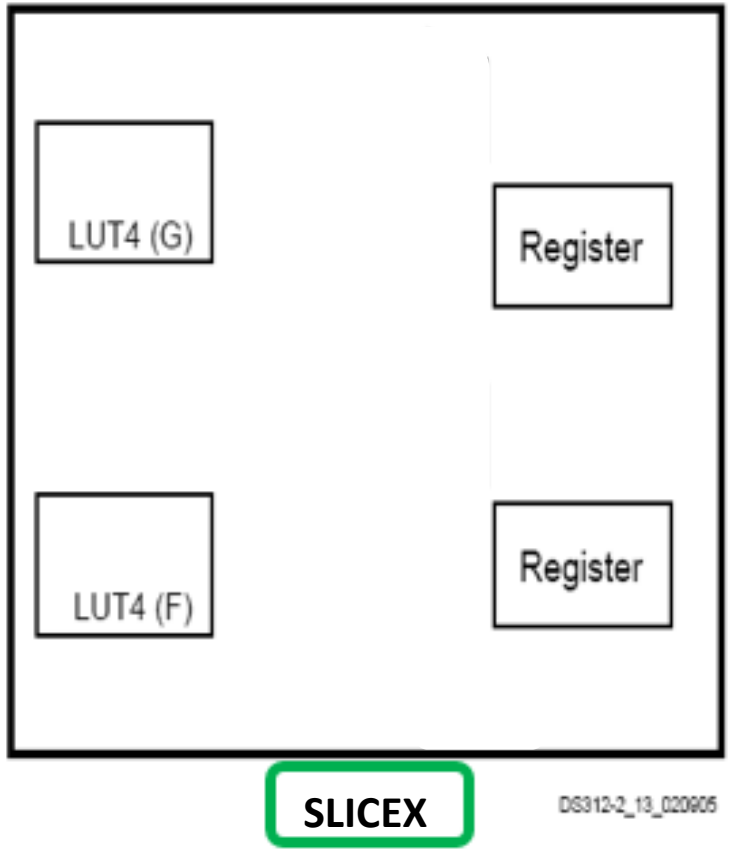


SLICEM



SLICEL

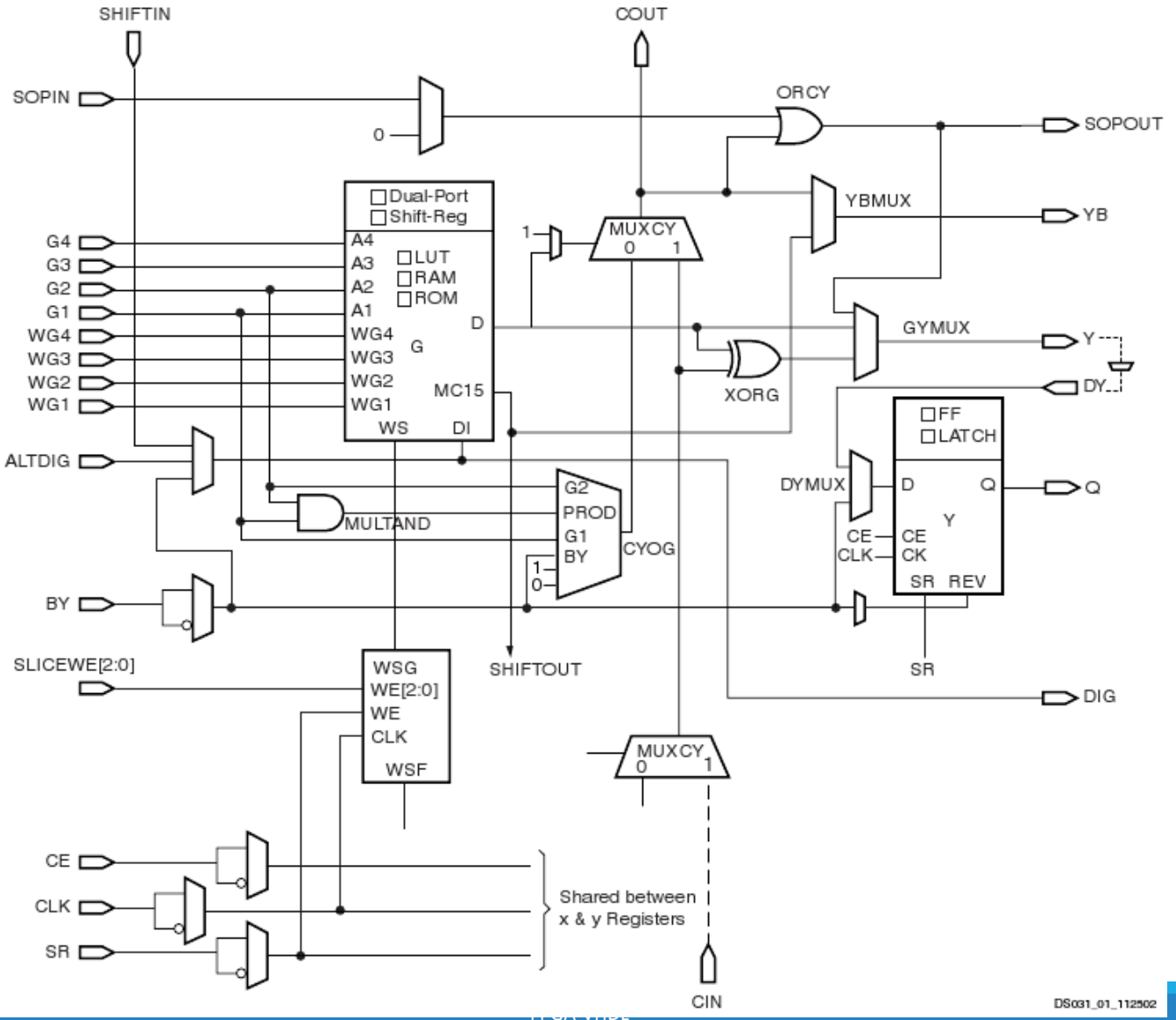
DS312-2_13_020905



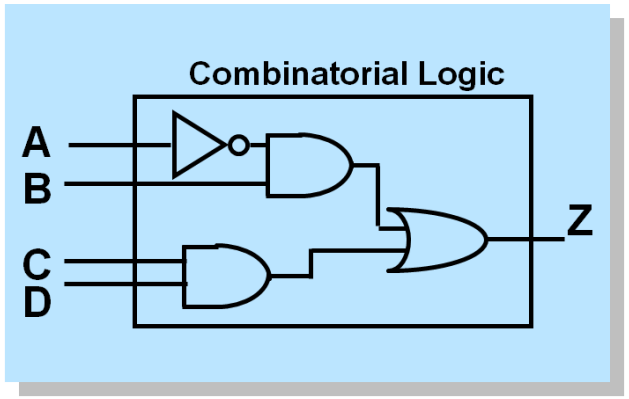
SLICEX

DS312-2_13_020905

Half SLICE(?) Detailed View

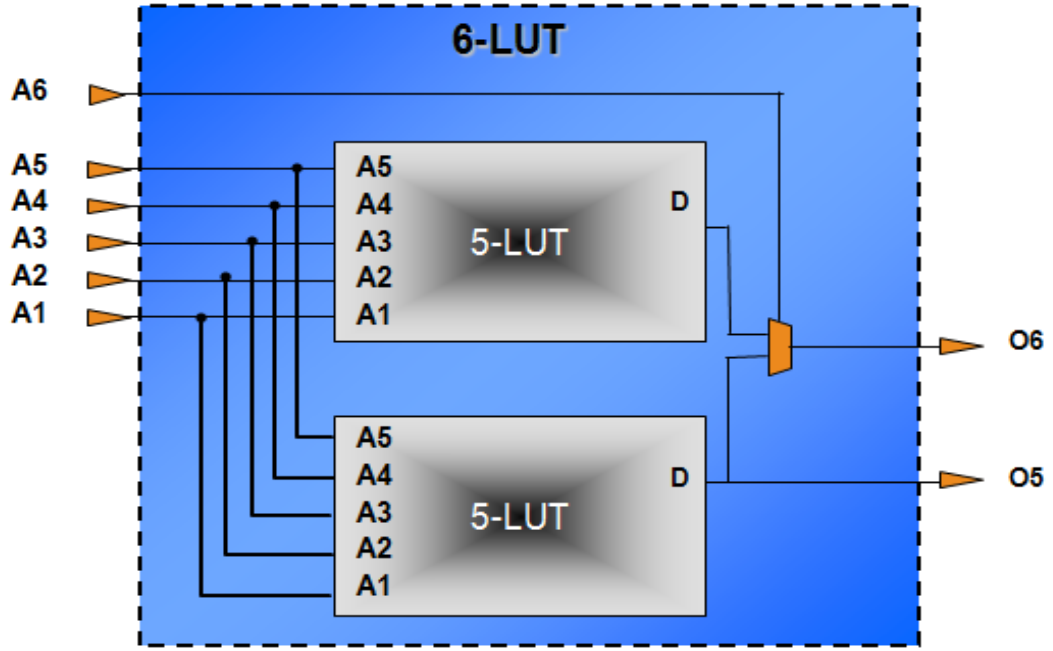


Look Up Table

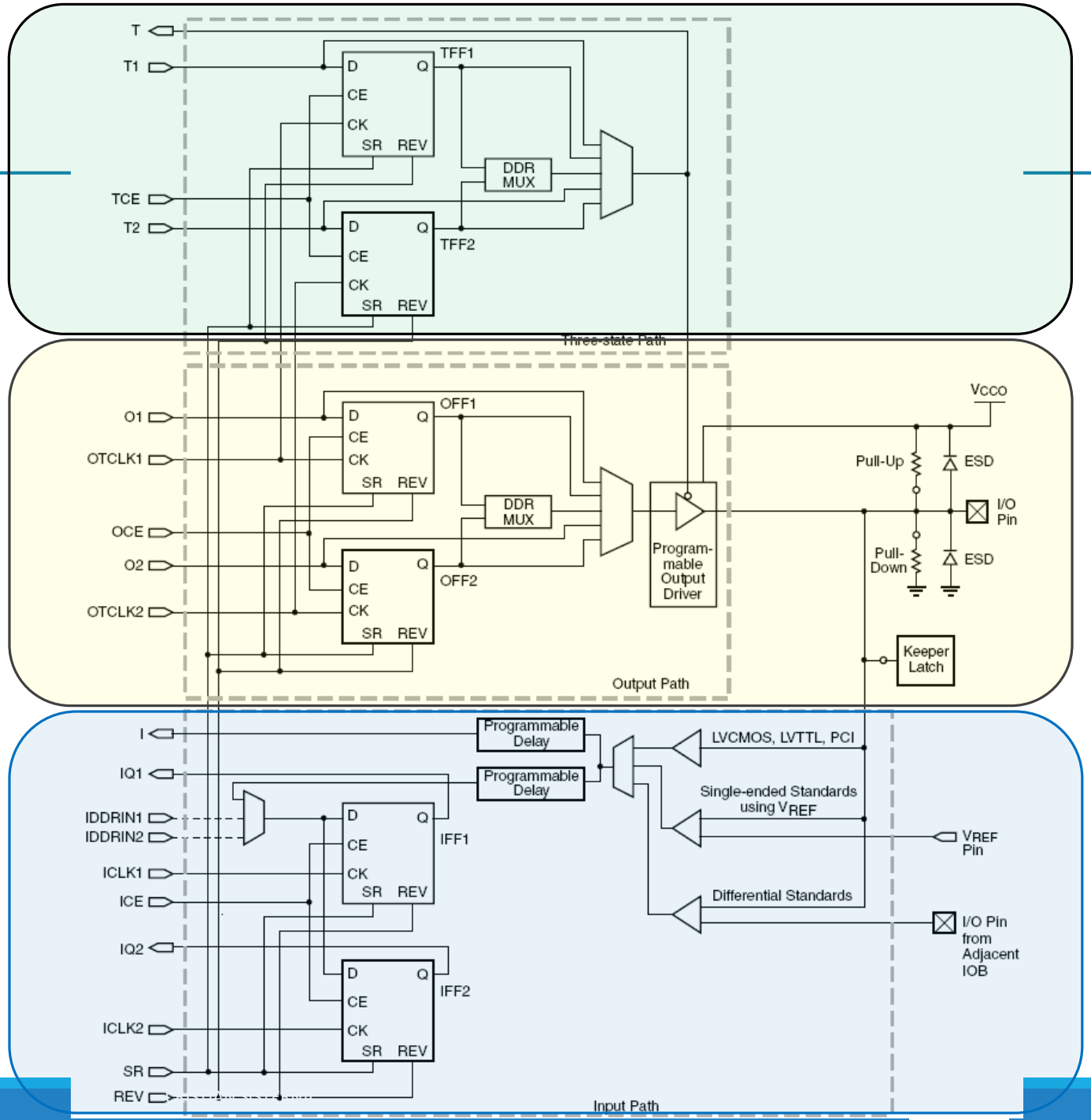


A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
	.	.	.	
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

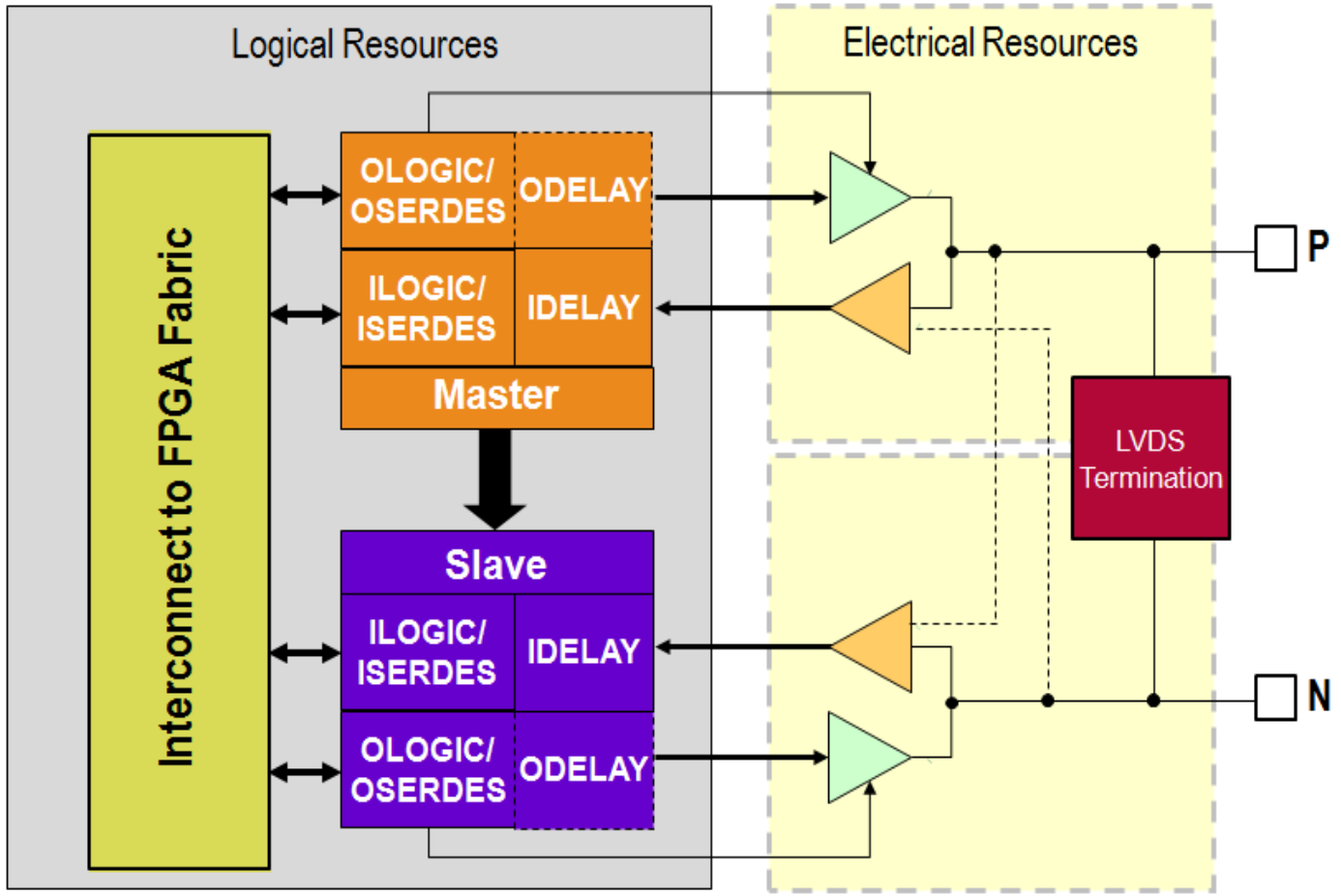
VHDL



I/O Block (IOB)


T
T1
TCE
T2
D
Q
CE
CK
SR
REV
TFF1
TFF2
DDR MUX
Three-state Path
O1
OTCLK1
OCE
O2
OTCLK2
OFF1
OFF2
Vcco
Pull-Up
ESD
I/O Pin
Pull-Down
Program-mable Output Driver
Keeper Latch
Output Path
I
IQ1
IDDRIN1
IDDRIN2
ICLK1
ICE
IQ2
ICLK2
SR
REV
Programmable Delay
IFF1
IFF2
LVCMOS, LVTTL, PCI
Single-ended Standards using VREF
VREF Pin
Differential Standards
I/O Pin from Adjacent IOB
Input Path

7-Series I/O Block



I/O Resources

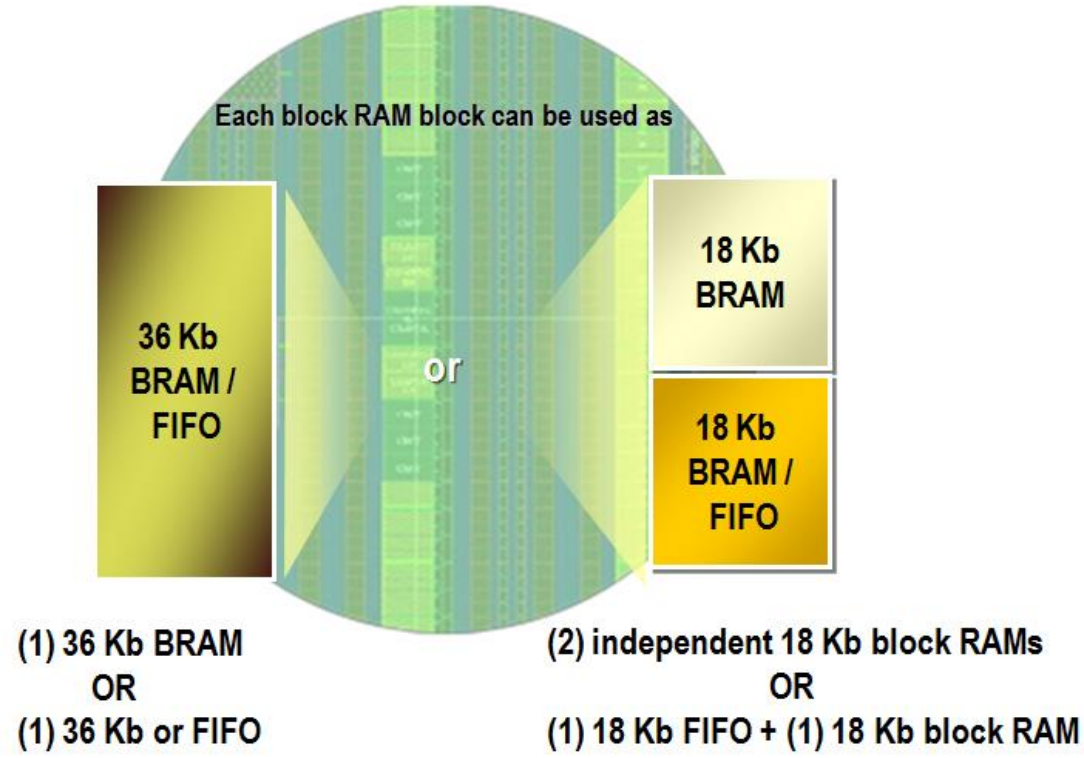
- Digital Controlled Impedance (DCI)
- Drive Strength
- Slew Rate
- Bus Hold (Bus keeper)
- Pull-up/Pull-down
- Differential Termination
- IODelay (V5, V6, V7, S6)
  - Fixed
  - Variable

FPGA More Common I/O Standards

Standard	Description	Usage	Input Buffer	Output Buffer
LVTTL	Low-Voltage TTL	Propósito general 3.3V	LVTTL	Push-pull
LVCMOS	Low-Voltage CMOS	Propósito general 3.3V, 2.5V, 1.8V, 1.5V	CMOS	Push-pull
PCI	Peripheral Component Interconnect	Bus PCI	LVTTL	Push-pull
GTL	Gunning Transceiver Logic	Bus alta velocidad, backplane	V_{REF}	Open Drain
GTL+	GTL Plus	Intel Pentium Pro	V_{REF}	Open Drain
HSTL	High Speed Transceiver Logic	Interface con SRAM	V_{REF}	Push-pull
SSTL3	Stub Series Terminated Logic 3.3V	SRAM/SDRAM	V_{REF}	Push-pull
SSTL2	Stub Series Terminated Logic 2.5V	SRAM/SDRAM	V_{REF}	Push-pull
SSTL18	Stub Series Terminated Logic 1.8V	SRAM/SDRAM	V_{REF}	Push-pull
Differential Standards				
LVDS	Low-Voltage Differential Signaling	High speed interface	Diferencial	Diferencial
BLVDS	Bus LVDS	Multipoint LVDS	Diferencial	Diferencial
LVPECL	Low Voltage Positive ECL	High-speed clocks	Diferencial	Diferencial
LDT	Lightning Data Transport	Bidireccional serie/paralelo (Hyper Transport)	Diferencial	Diferencial
Mini-LVDS	Mini-LVDS	Flat Panel Displays	Diferencial	Diferencial
LVDSExt	Extensión de LVDS	Hard Drive interface	Diferencial	Diferencial
RSDS	Reduced Swing Differencial Signaling	DVI/HDMI	Diferencial	Diferencial

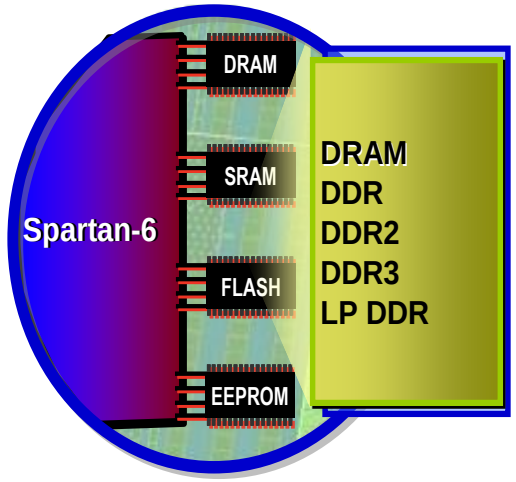
FPGA Memory Blocks

- Fully synchronous operation
- Optional internal pipeline register for higher frequency operation
- Two independent ports access common data
- Multiple configuration options
  - True dual-port, simple dual-port, single-port
- Integrated cascade logic
- Byte-write enable in wider configurations
- Integrated control for fast and efficient FIFOs

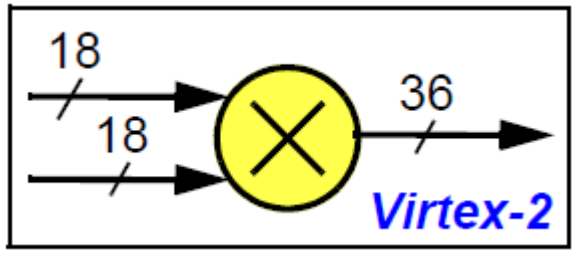


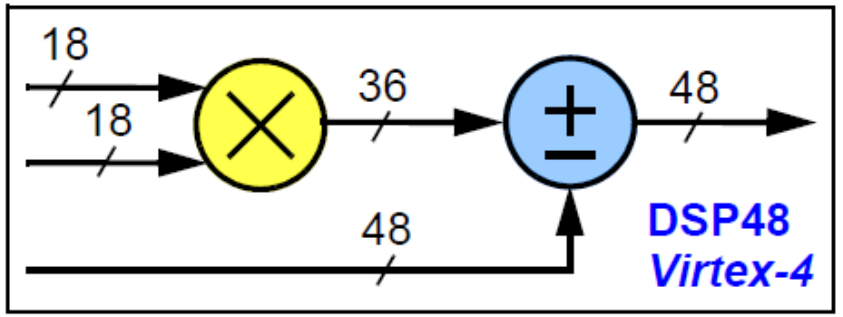
‘Hard’ Memory Controller

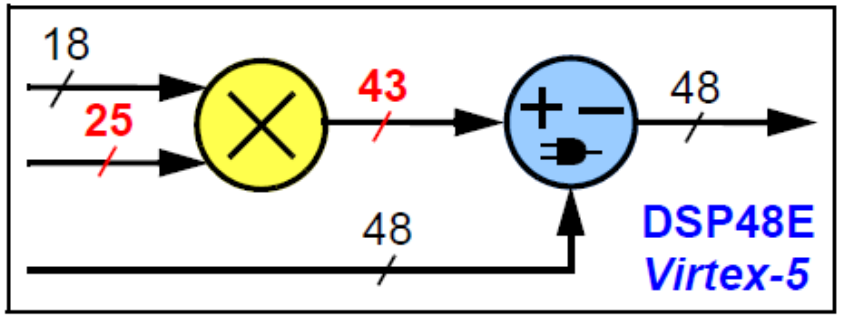
- Guaranteed memory interface performance providing
  - Reduced engineering & board design time
  - DDR, DDR2, DDR3 & LP DDR support
  - Up to 12.8Mbps bandwidth for each memory controller
- Automatic calibration features
- Multiport structure for user interface
  - Six 32-bit programmable ports from fabric
  - Controller interface to 4, 8 or 16 bit memories devices

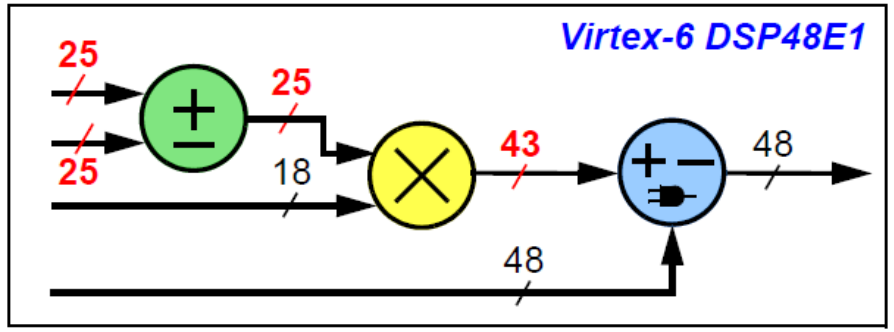


FPGA Multipliers & DSP Blocks

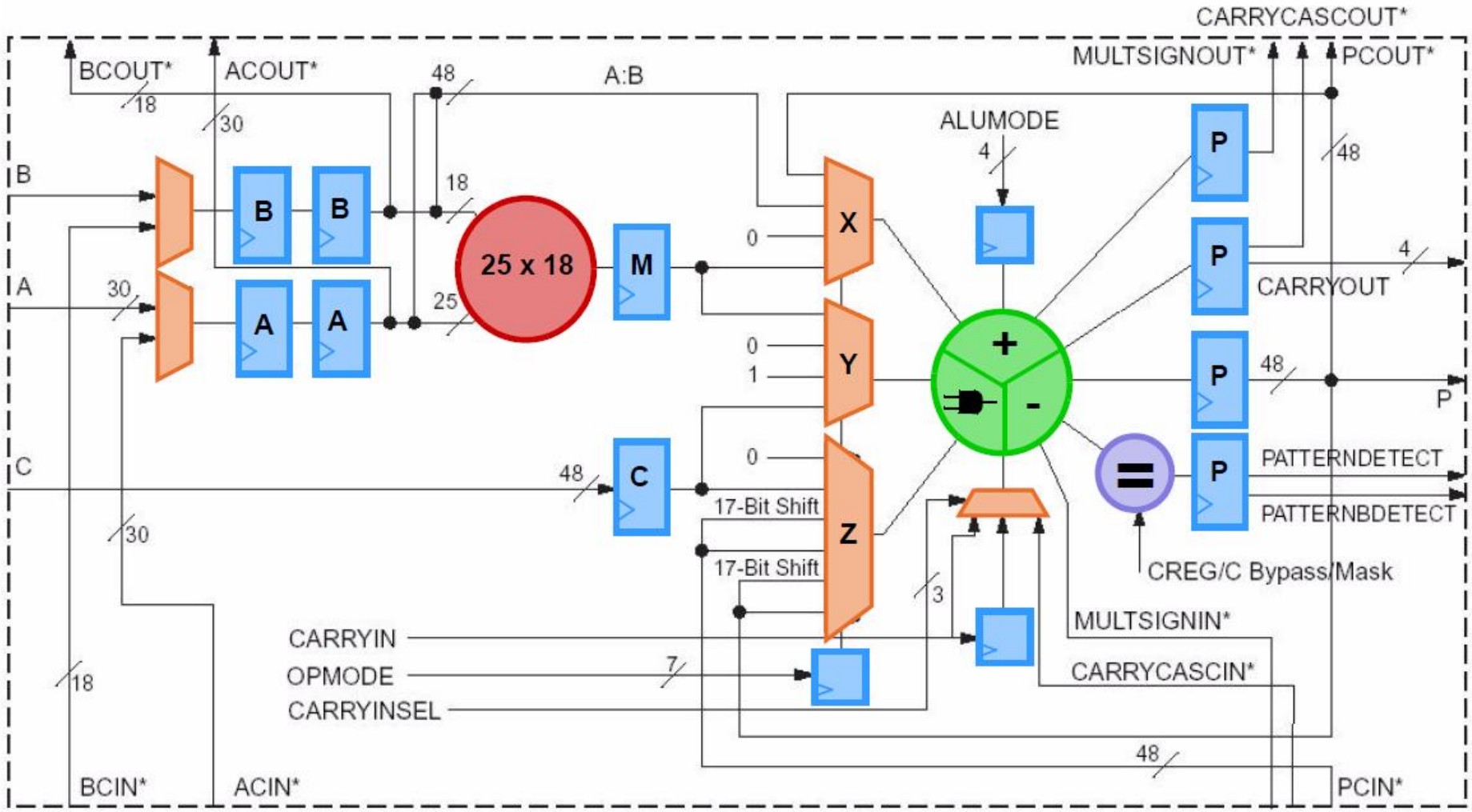




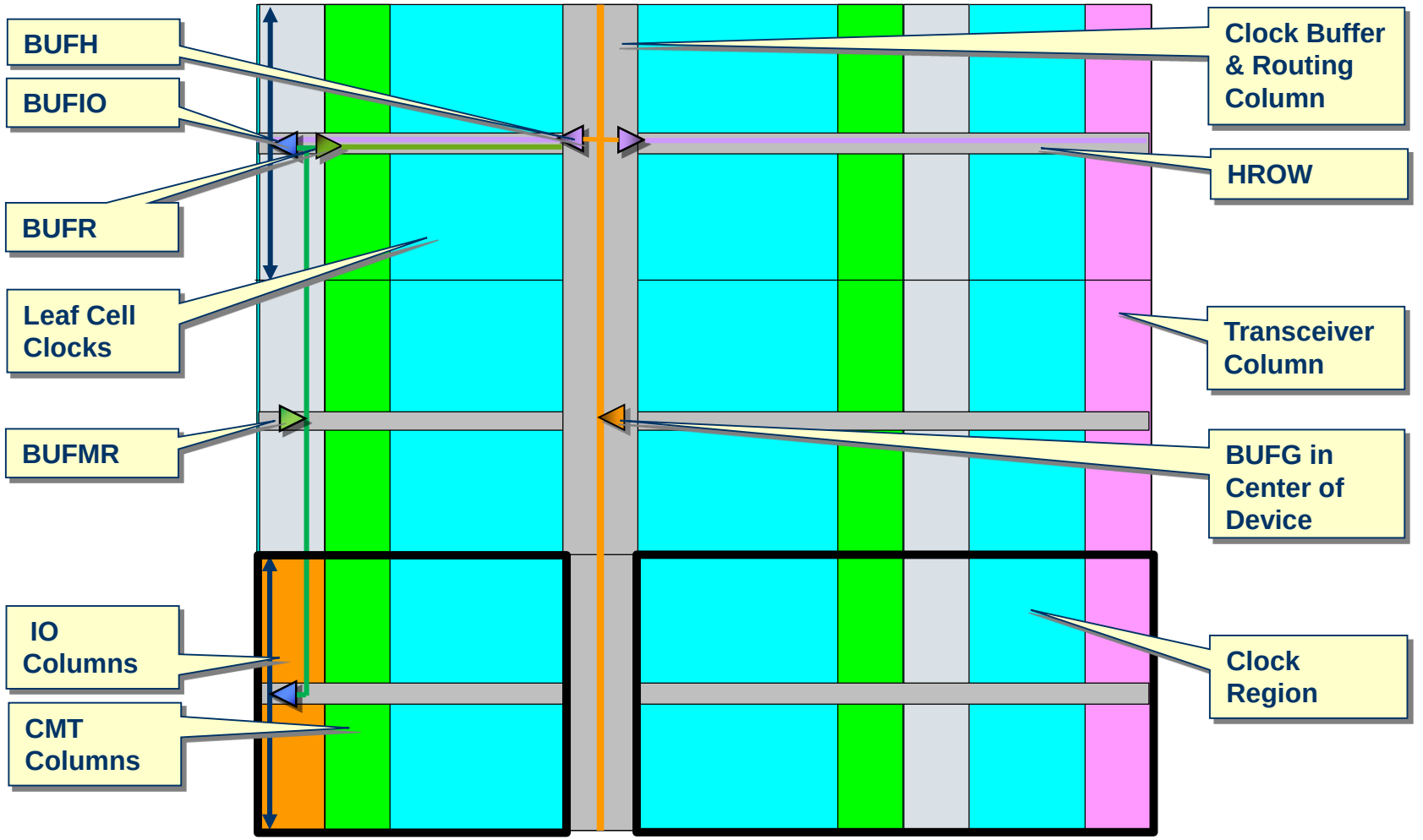




Virtex / Zynq DSP Block



FPGA Clock Resources



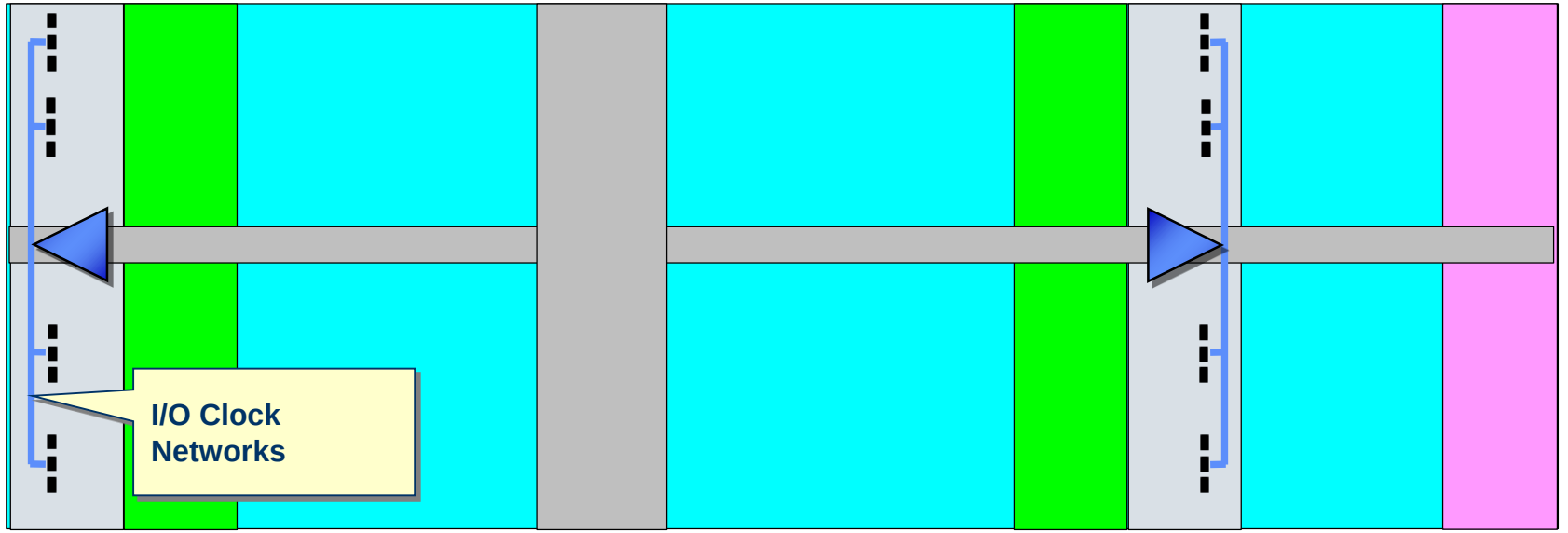
I/O Clock Networks

Each clock region has four I/O clock networks

These networks can drive **only** the clock ports of the ILOGIC/OLOGIC resources and the high-speed clock ports (CLK) of the ISERDES/OSERDES

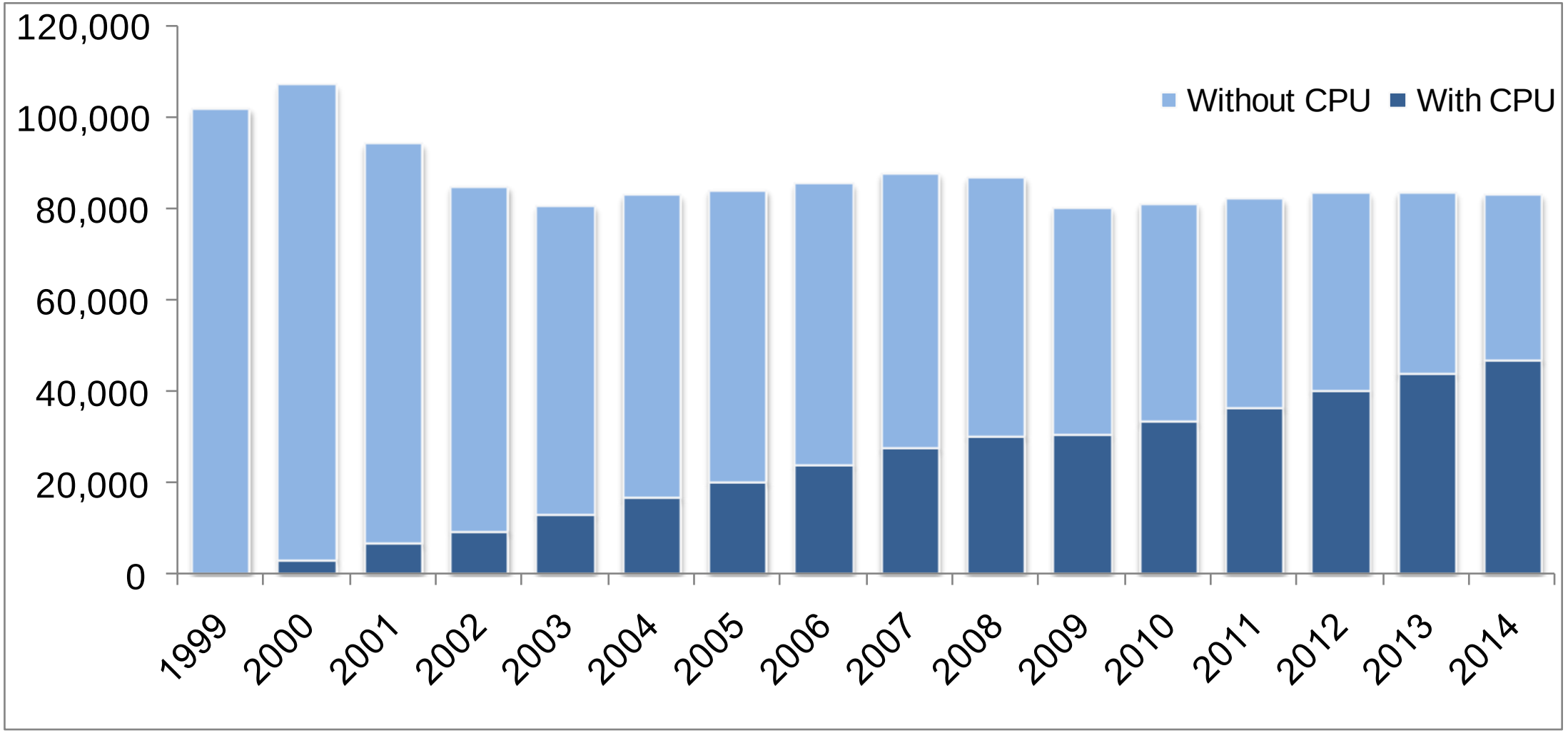
Each I/O clock network is driven by a BUFIO

Highest quality clock available



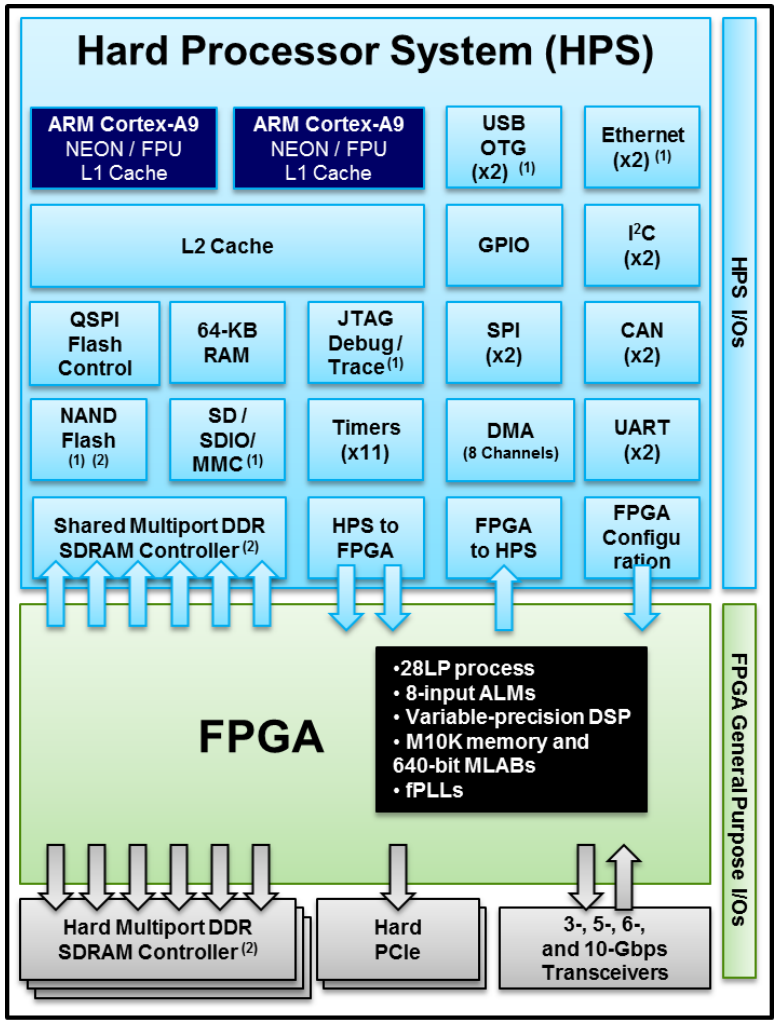
FPGA SoC

Growth of Processors in FPGAs



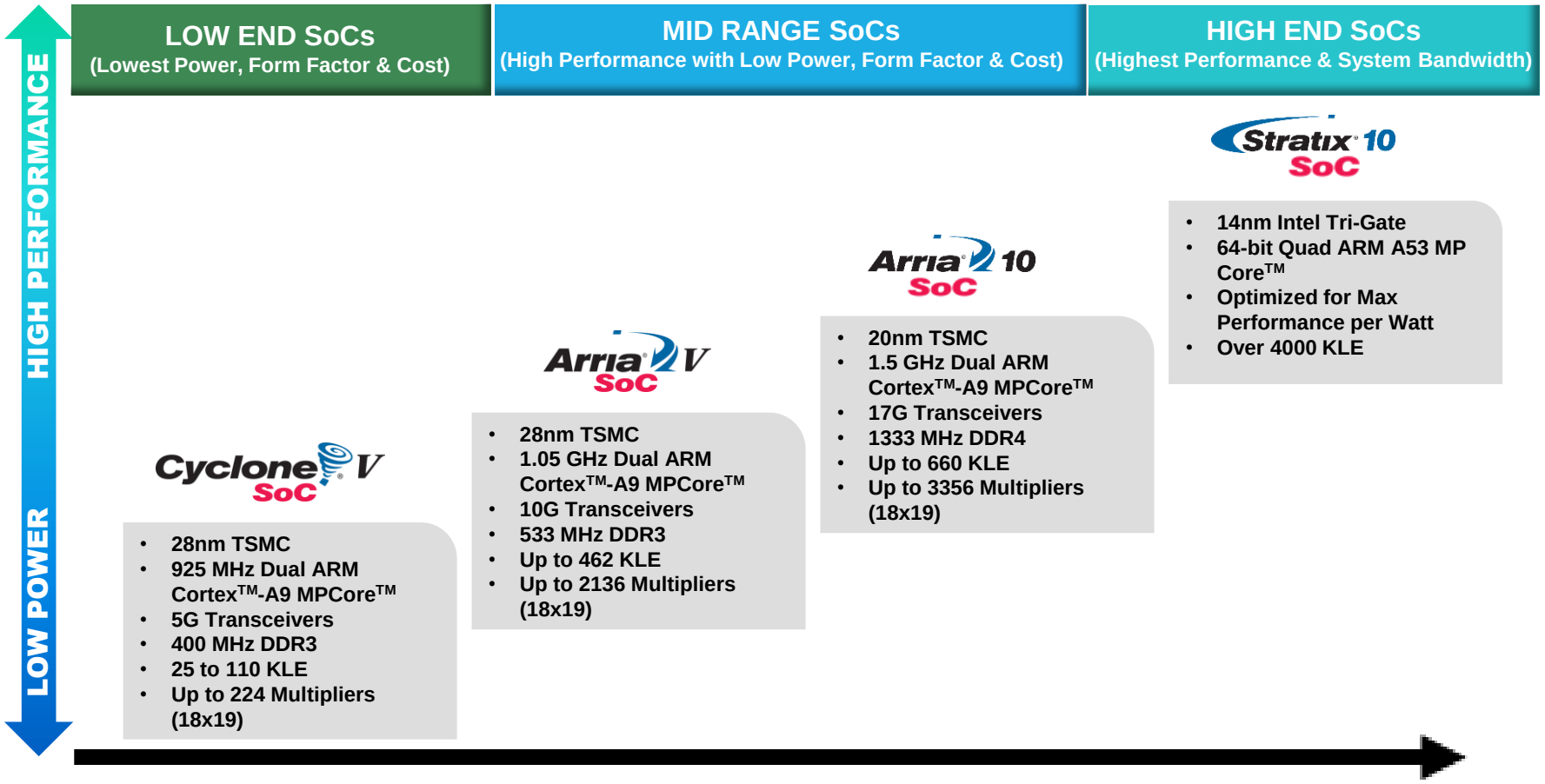
Altera SoC

- **Processor**
  - Dual-core ARM® Cortex™-A9 MPCore™ processor
  - Up to 5,250 MIPS (1050 MHz per core maximum)
  - NEON coprocessor with double-precision FPU
  - 32-KB/32-KB L1 caches per core
  - 512-KB shared L2 cache
- **Multiport SDRAM controller**
  - DDR3, DDR3L, DDR2, LPDDR2
  - Integrated ECC support
- **High-bandwidth on-chip interfaces**
  - > 125-Gbps HPS-to-FPGA interface
  - > 125-Gbps FPGA-to-SDRAM interface
- **Cost- and power-optimized FPGA fabric**
  - Lowest power transceivers
  - Up to 1,600 GMACS, 300 GFLOPS
  - Up to 25Mb on-chip RAM
  - More hard intellectual property (IP)



Notes:
(1) Integrated direct memory access (DMA)
(2) Integrated ECC

Altera SoC Portfolio

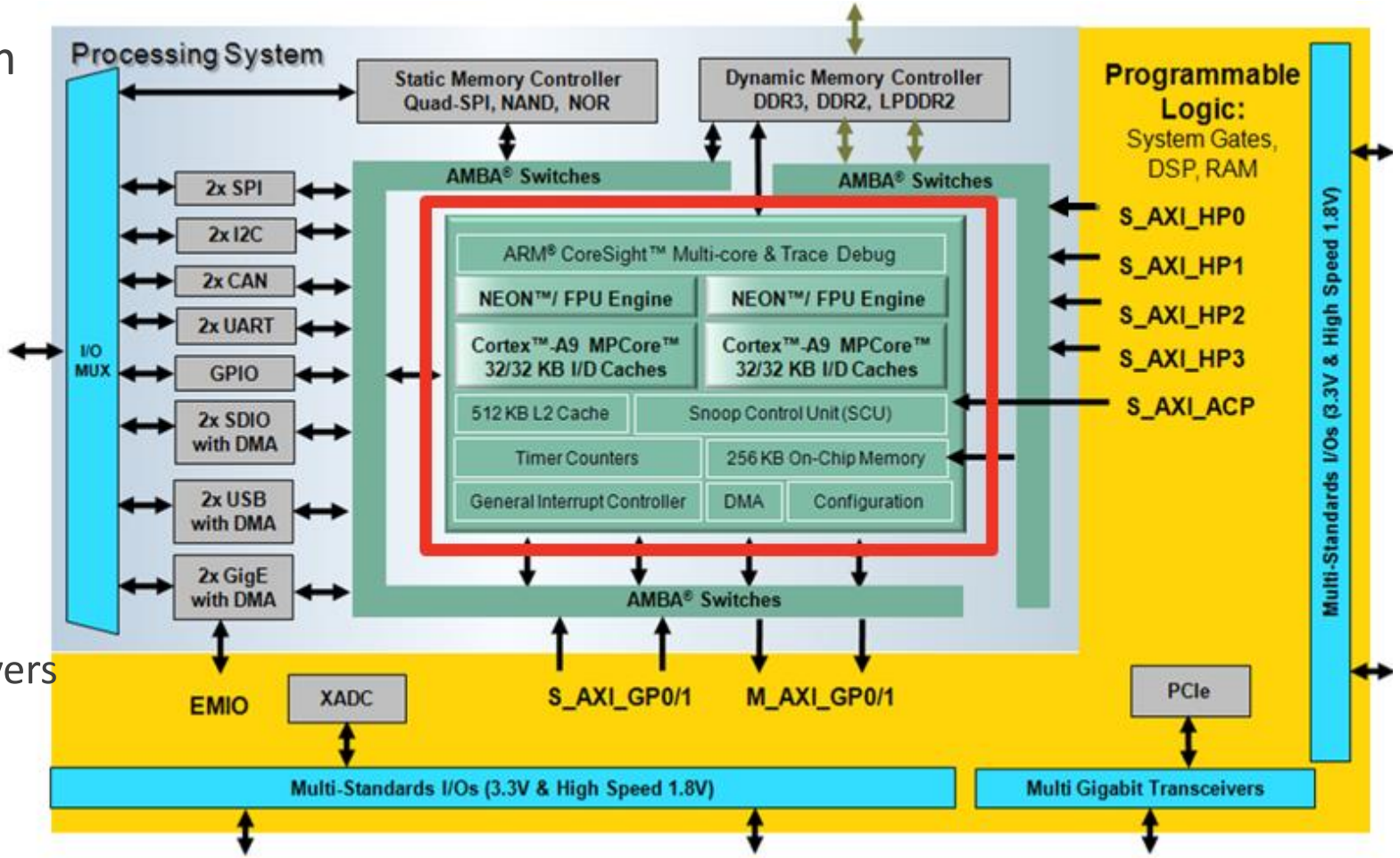


Xilinx SoC – Zynq

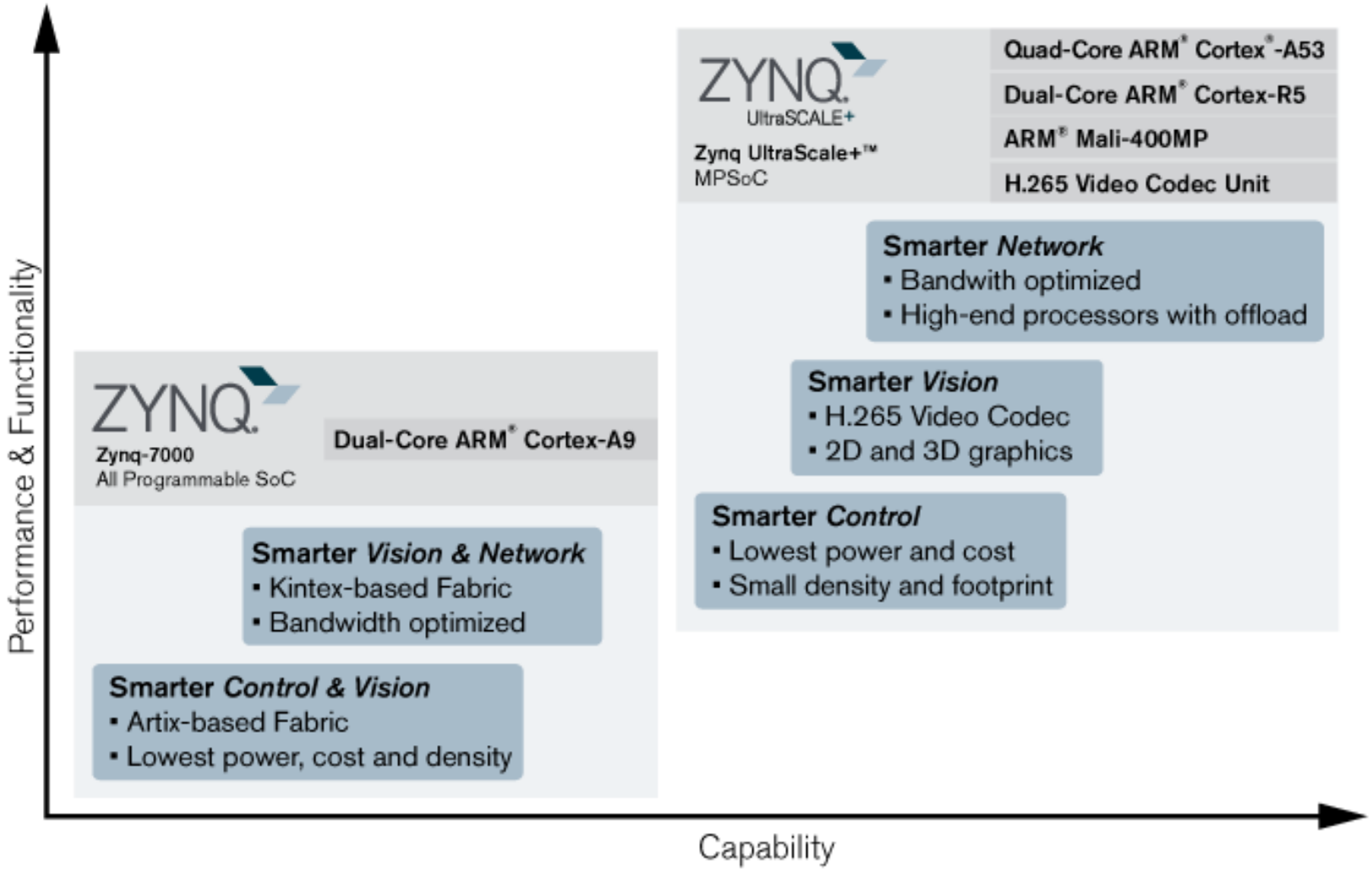
Complete ARM®-based processing system
- Application Processor Unit (APU)
  - Dual ARM Cortex™-A9 processors
  - Caches and support blocks
- Fully integrated memory controllers
- I/O peripherals

Tightly integrated programmable logic
- Used to extend the processing system
- Scalable density and performance

Flexible array of I/O
- Wide range of external multi-standard I/O
- High-performance integrated serial transceivers
- Analog-to-digital converter inputs

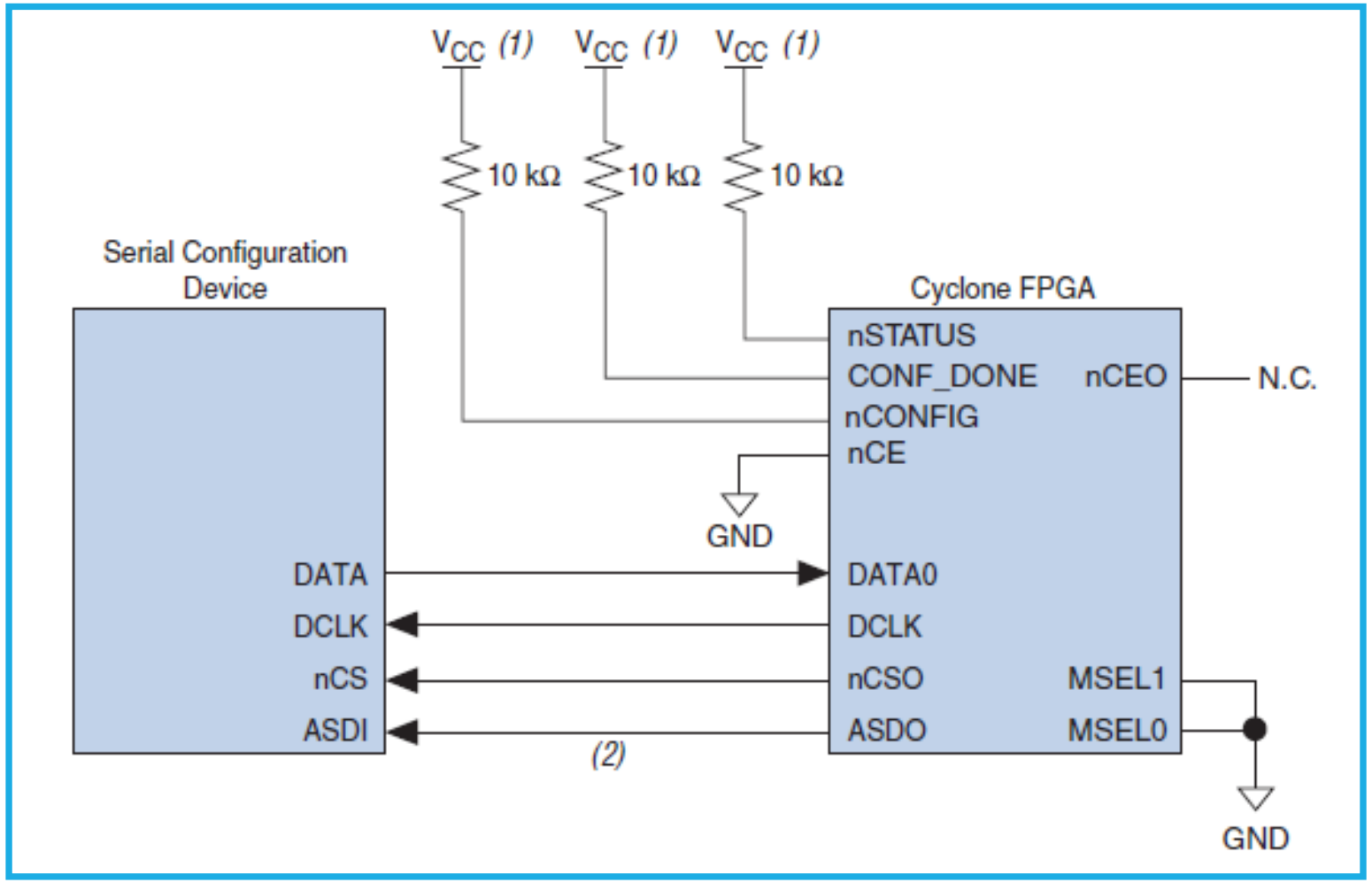


Xilinx SoC Portfolio

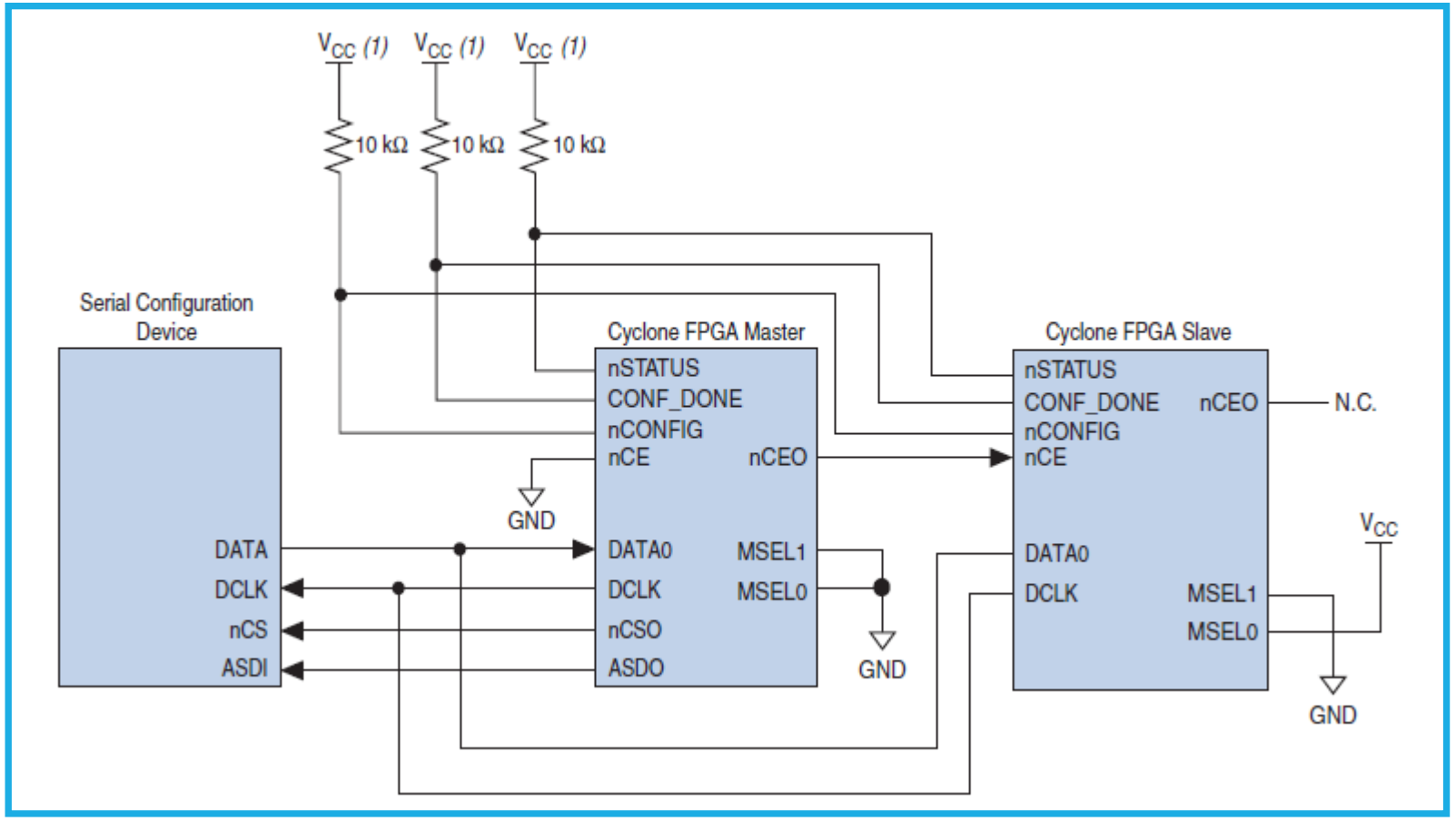


FPGA – Configuration Methods

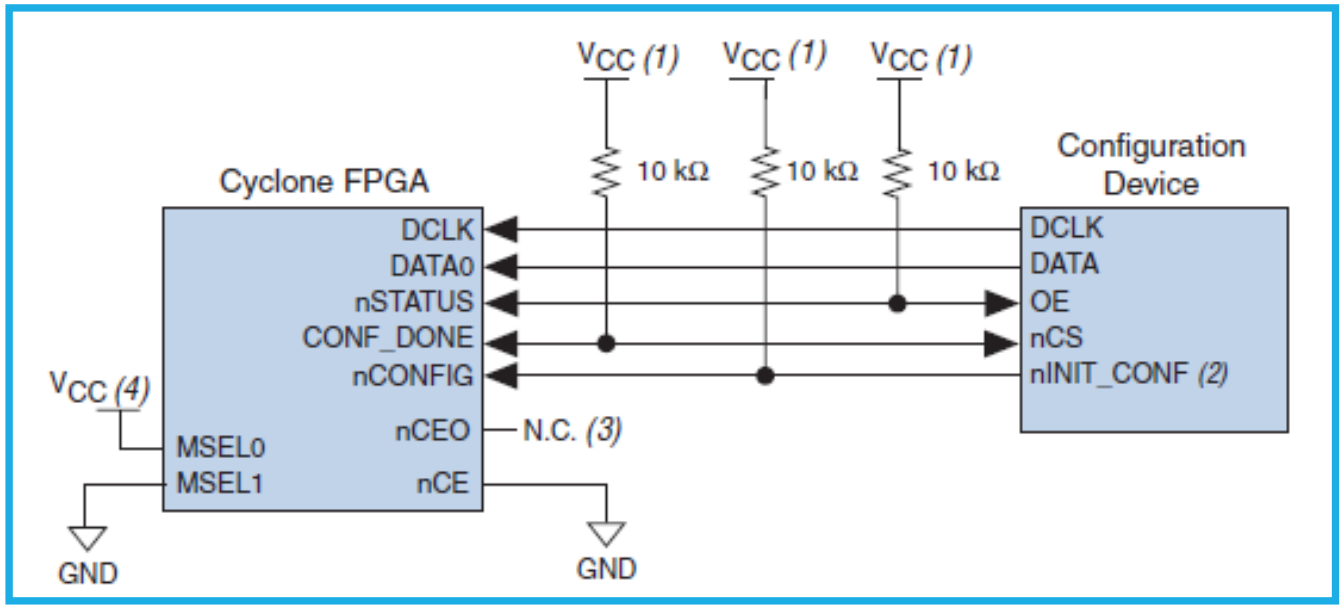
SRAM FPGA Configuration



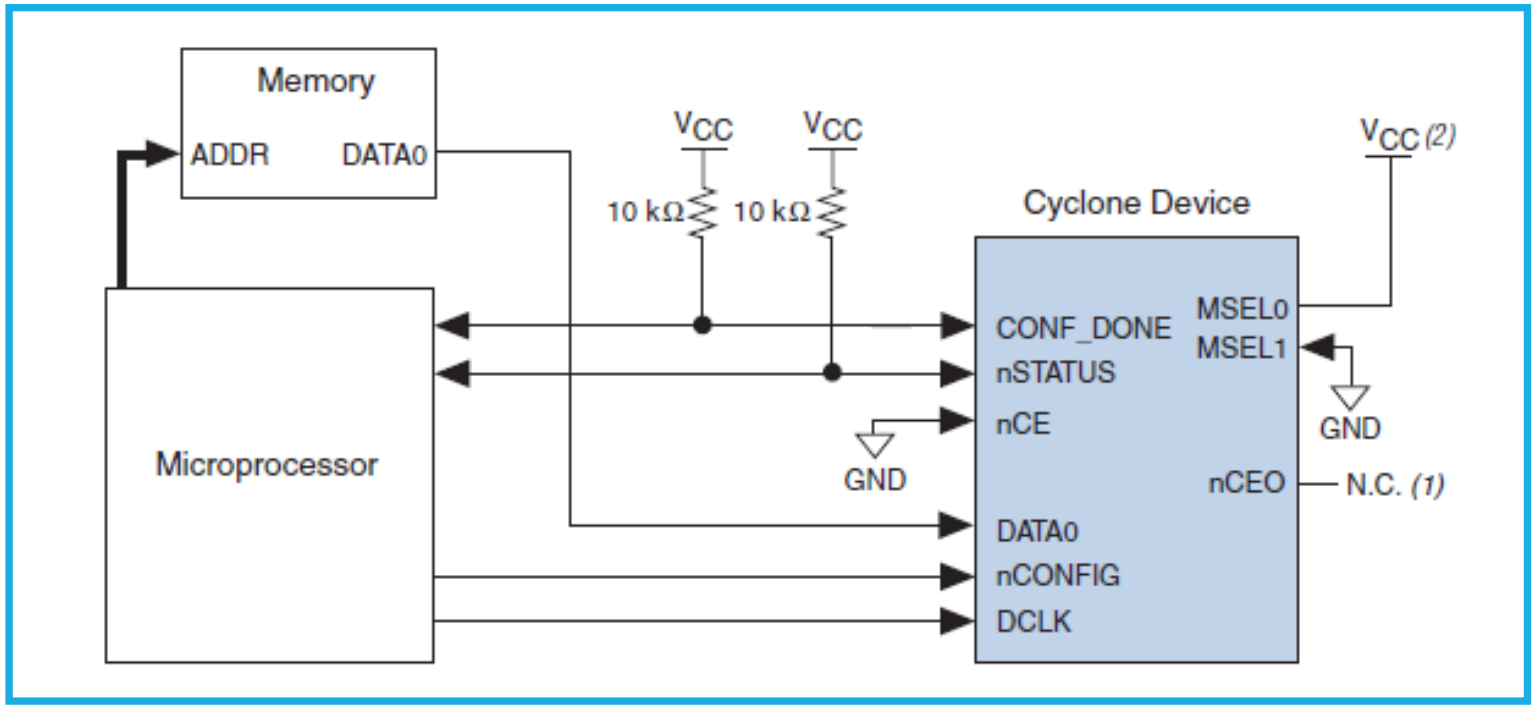
Configuring Multiple SRAM FPGA Devices



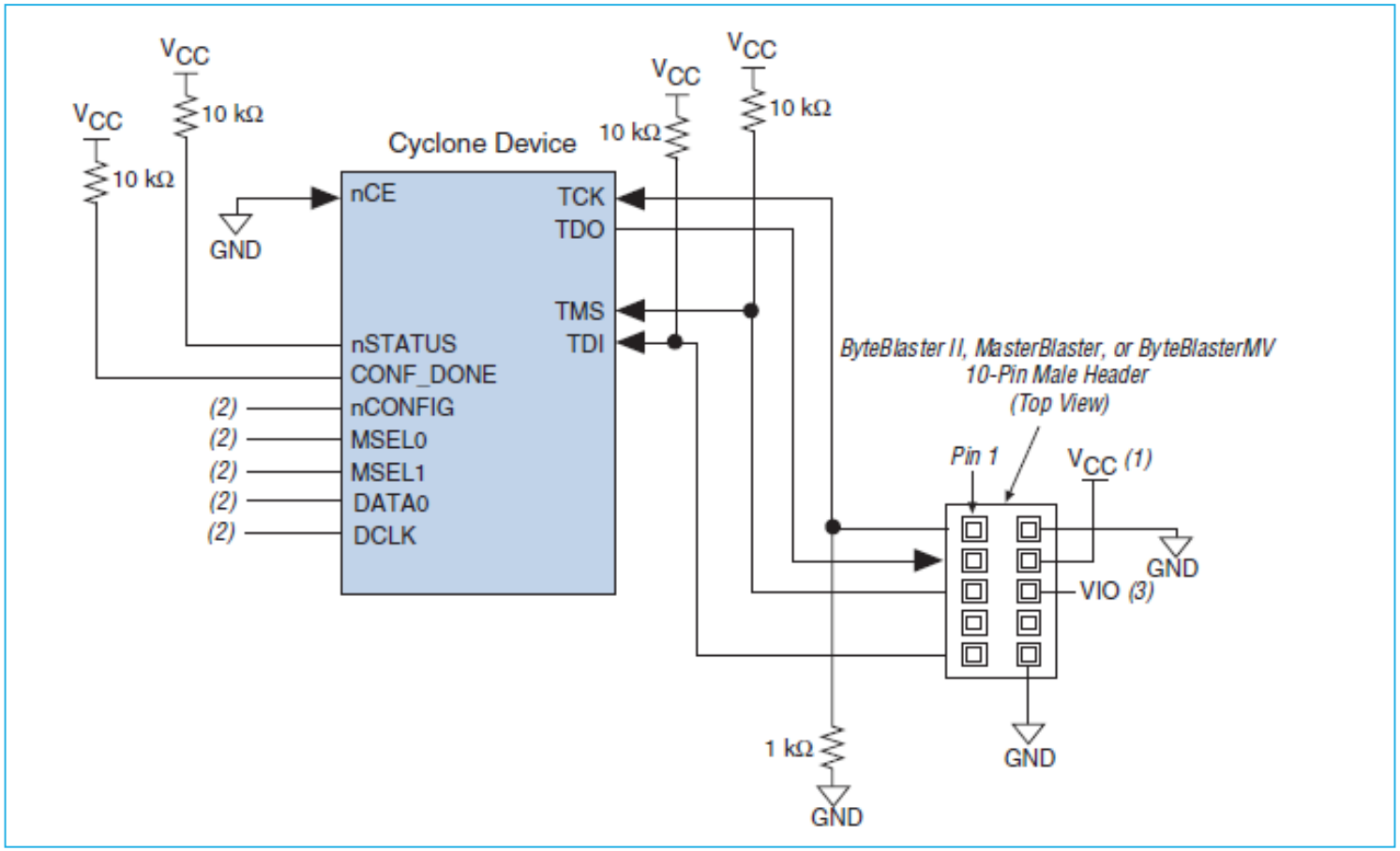
FPGA Passive Serial Configuration



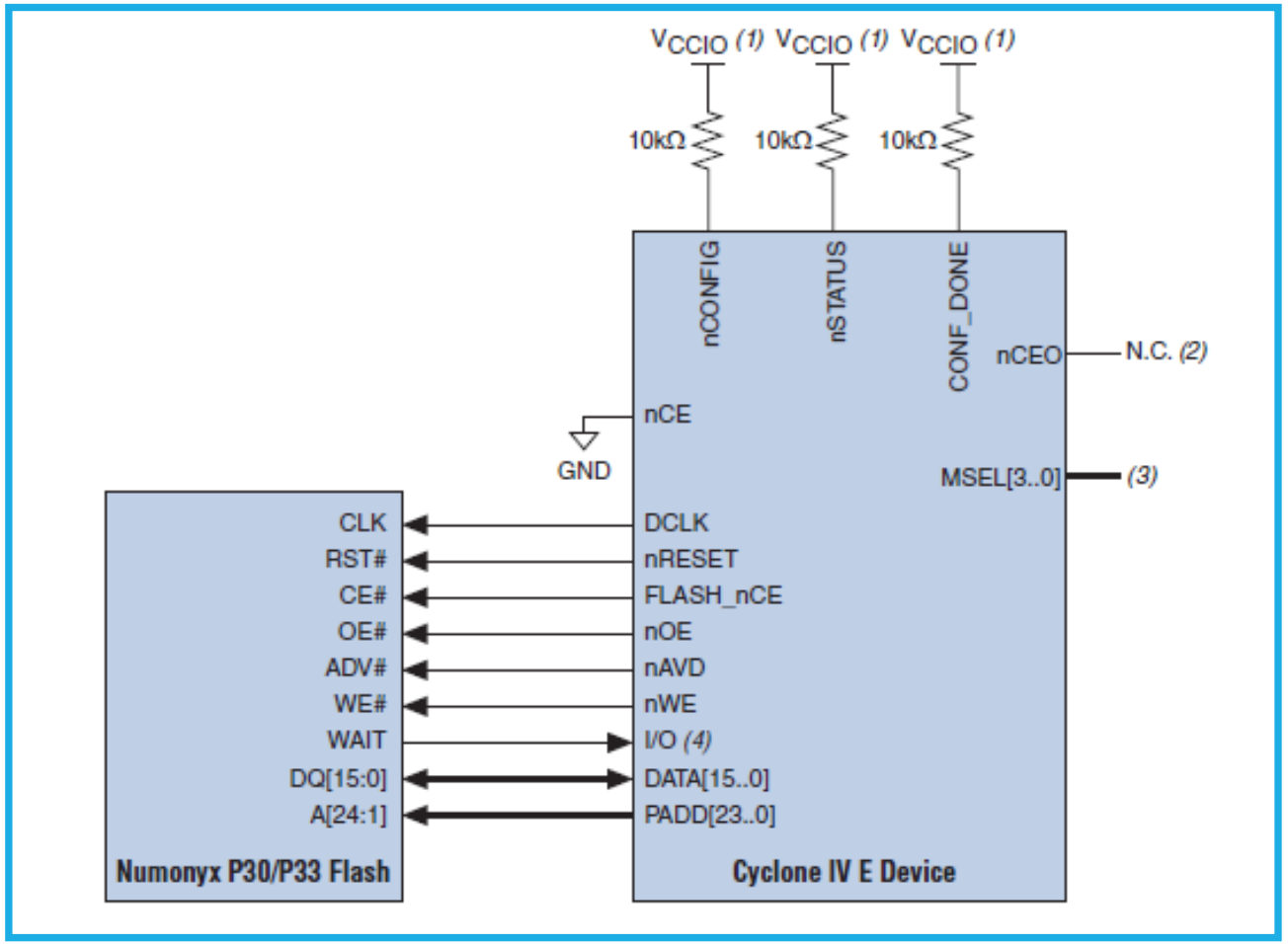
FPGA Configured By a Microprocessor



JTAG Configuration



FPGA Active Parallel Configuration



VHDL

VHDL

Very High Speed ICs

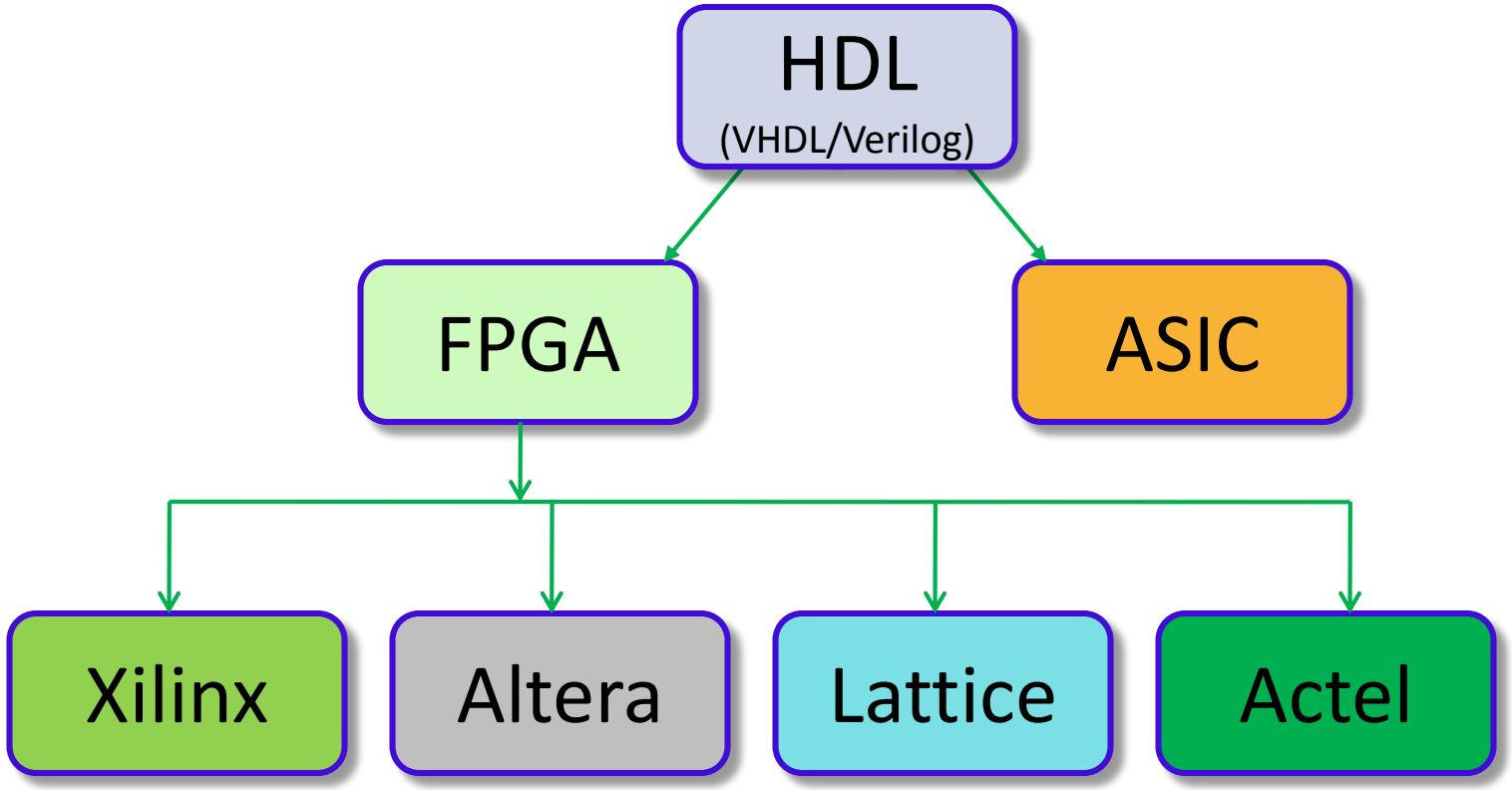
Hardware

Description

Language

V
H
D
L

Hardware Description Language



HDL – Main Features

- High level of abstraction

```
if(reset='1') then
      count <= 0;
elsif(rising_edge(clk)) then
      count <= count+1;
end if;
```

- Easy to debug
- Parameterized designs
- Re-uso
- IP Cores (free) available

What is not HDL

Neither Verilog, nor VHDL ARE A **programming** language;

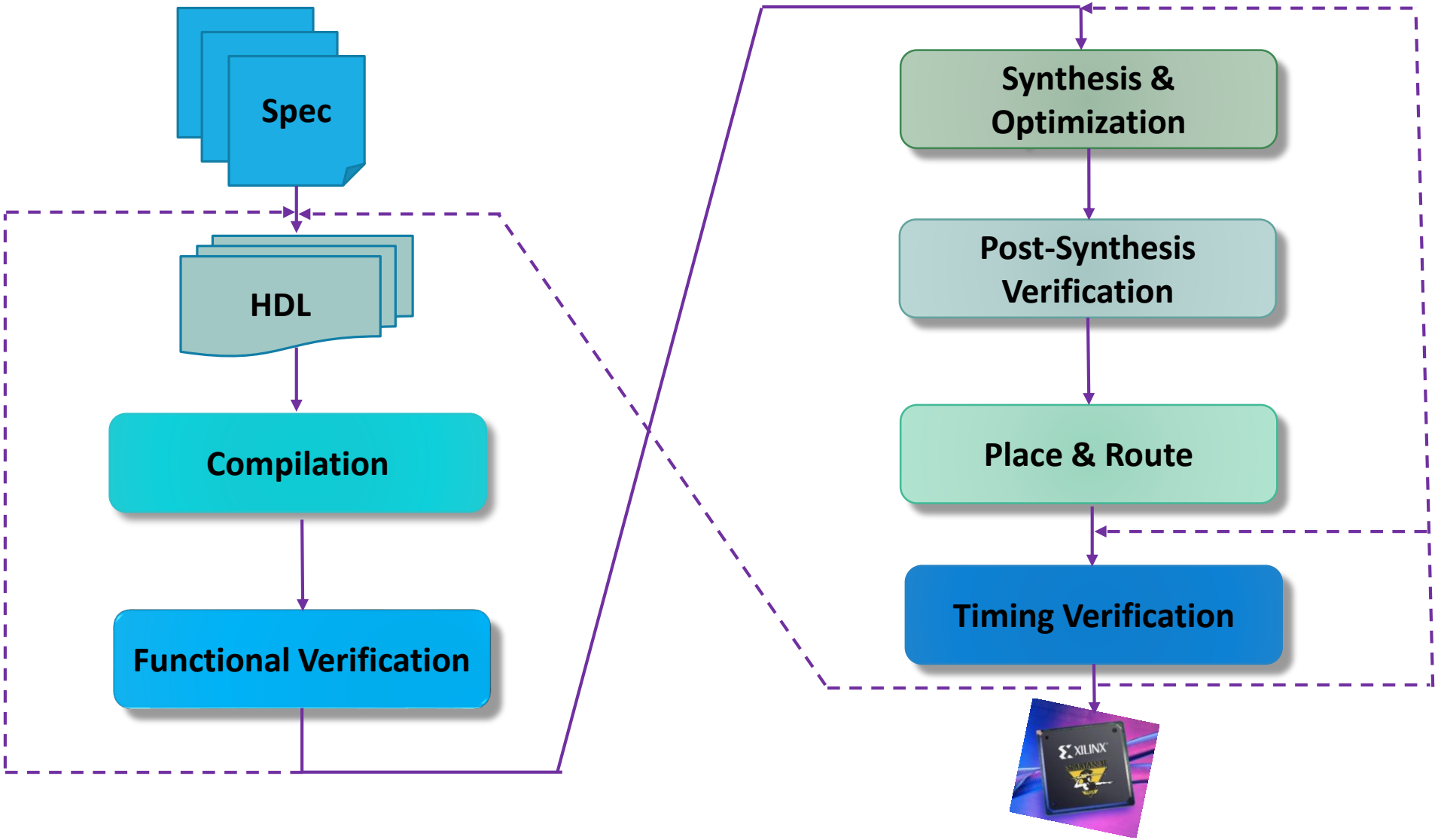
They ARE A **HARDWARE DESCRIPTION LENGUAGE**

Verilog o VHDL is not (yet) a highly abstract language:

$$y(n) = 0.75y(n-1) + 0.3x(n) ;$$

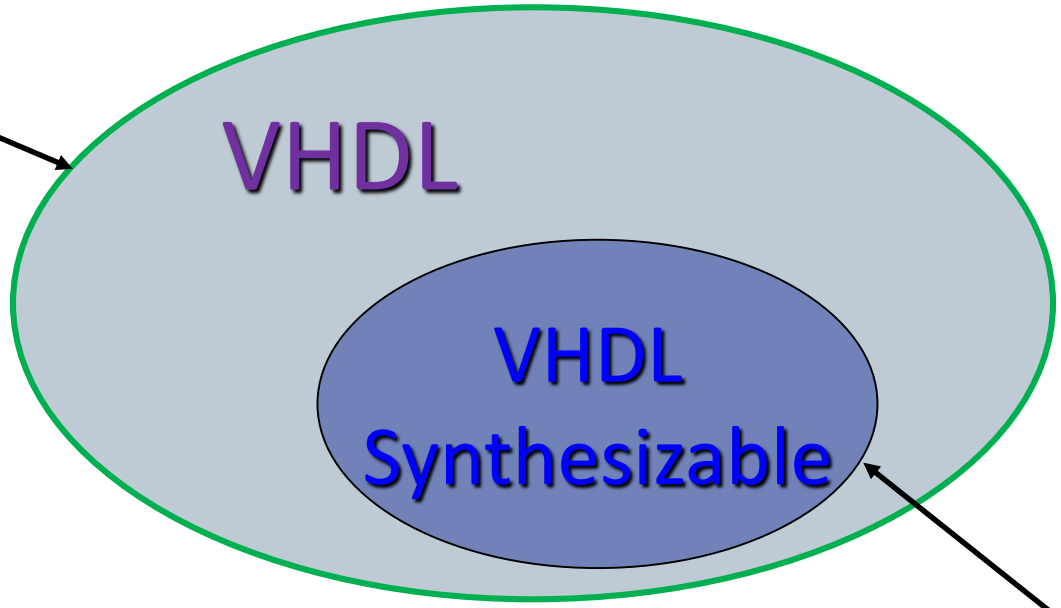
(Simulink/FPGA design flow)

VHDL - FPGA Flow Process



VHDL Synthesis – VHDL Simulation

Used to write code to simulate the behavior of a design



Used to implement the design into hardware (for instance in an FPGA)

VHDL Synthesis – FPGA Place & Route

VHDL Code

```
with tmp select
 j <= w when "1000",
      x when "0100",
      z when "0001",
     '0'when others;
```

Constraints

```
NET CLOCK PERIOD = 50 ns;
NET LOAD LOC = P14
```

Attributes

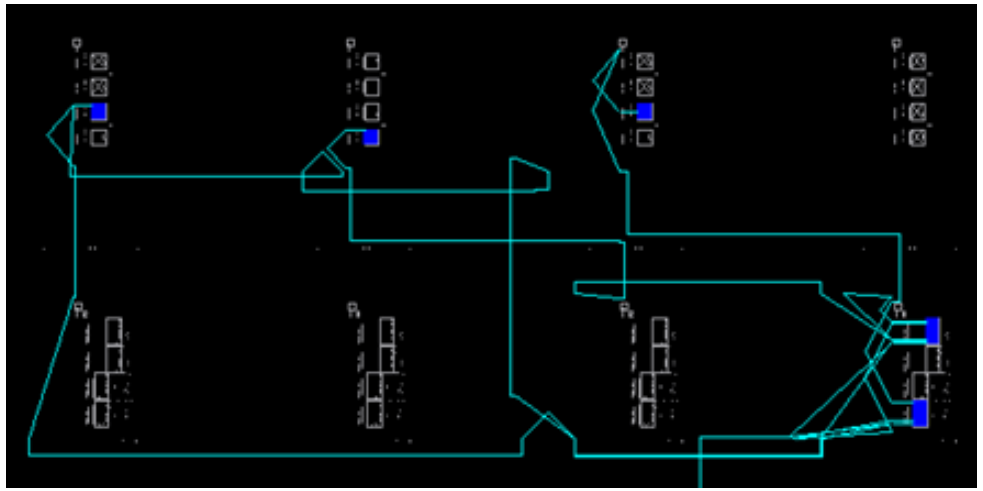
```
attribute syn_encoding of
   my_fsm: type is "one-hot";
```

FPGA Library of Components

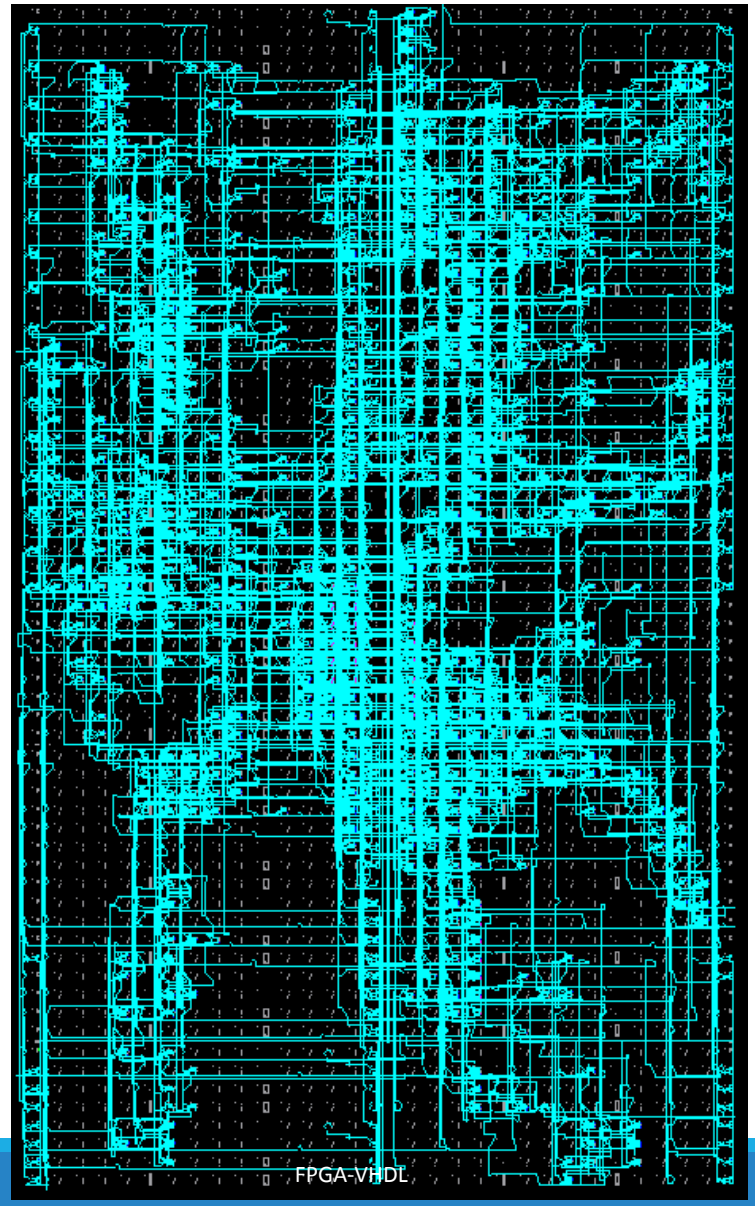
```
Cyclone
Spartan
```

Synthesis Tool

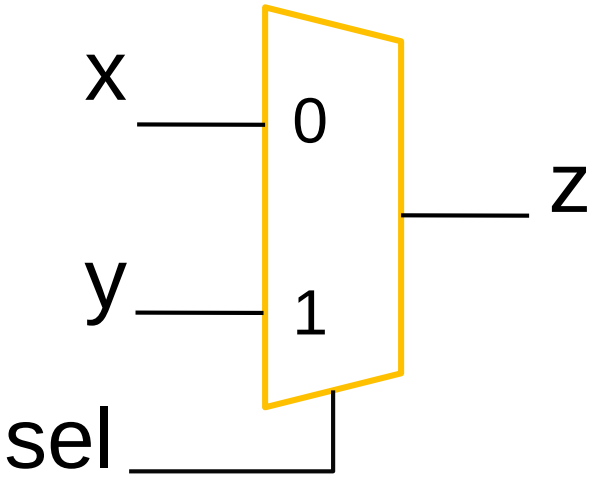
Place & Route Tool



Implemented Design in an FPGA



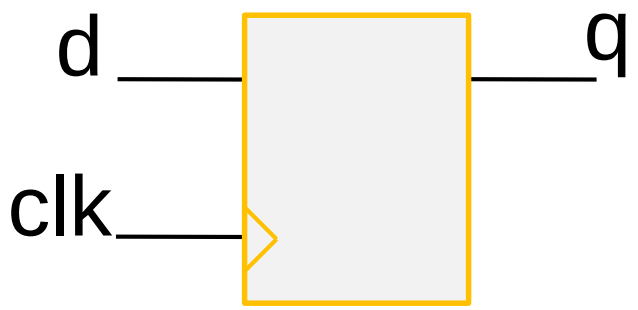
VHDL – Simple Hardware Description Example




```
if(sel='1') then
      z <= y;
else
      z <= x;
end if;
```

```
z <= y when sel='1' else x;
```

VHDL – Simple Hardware Description Example

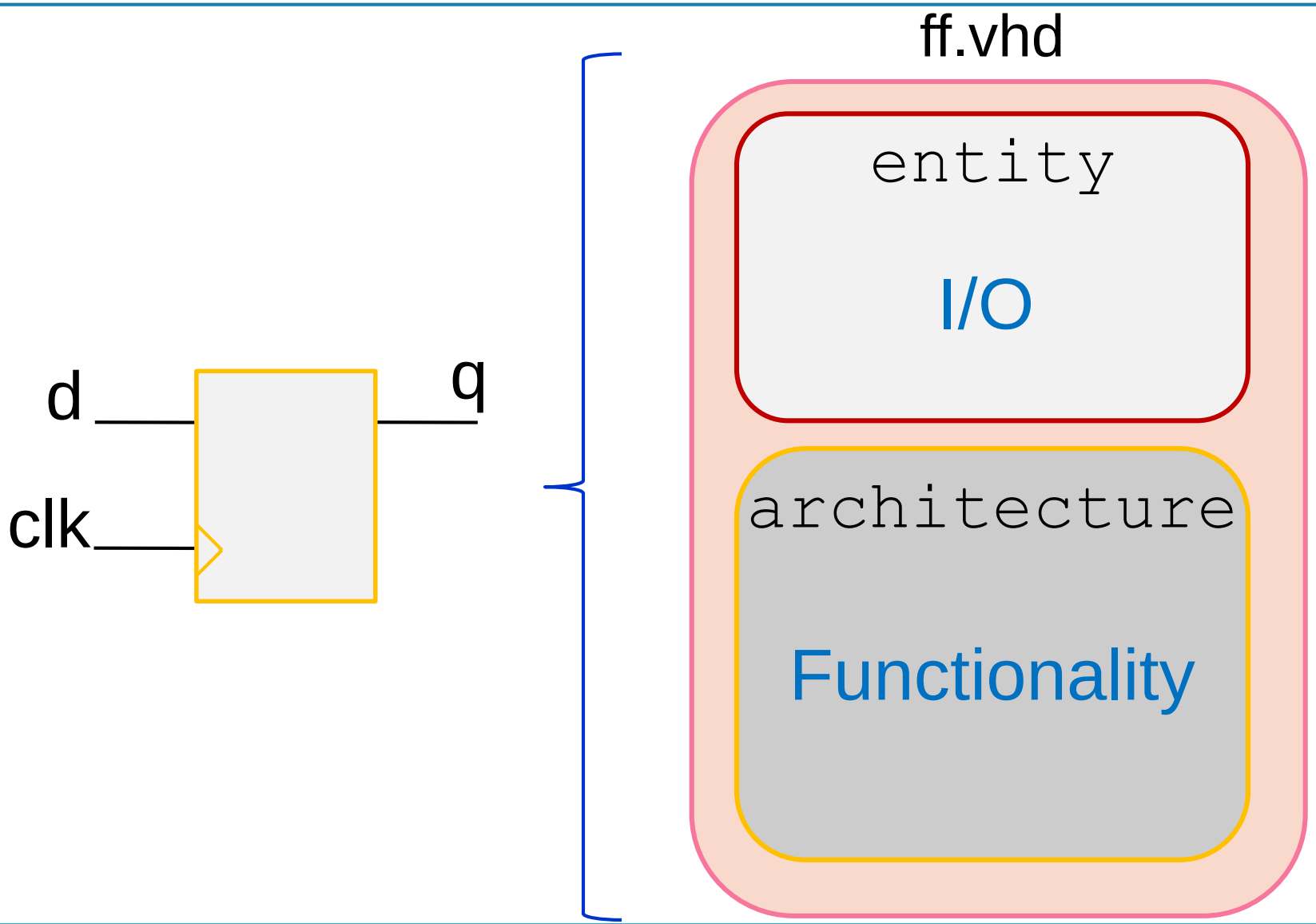



```
if(clk'event and clk='1')then
      q <= d;
else
      q <= q;
end if;
```

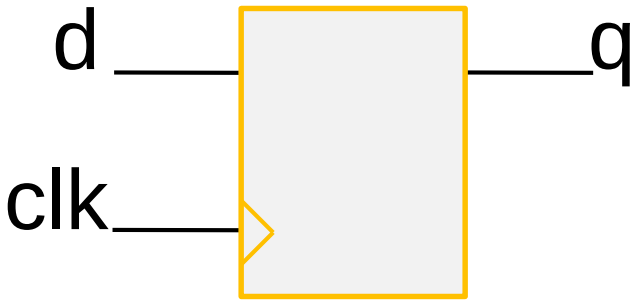
```
if(clk'event and clk='1')then
      q <= d;
end if;
```

```
if(rising_edge(clk))then
      q <= d;
end if;
```

VHDL Module Structure



VHDL Module Structure



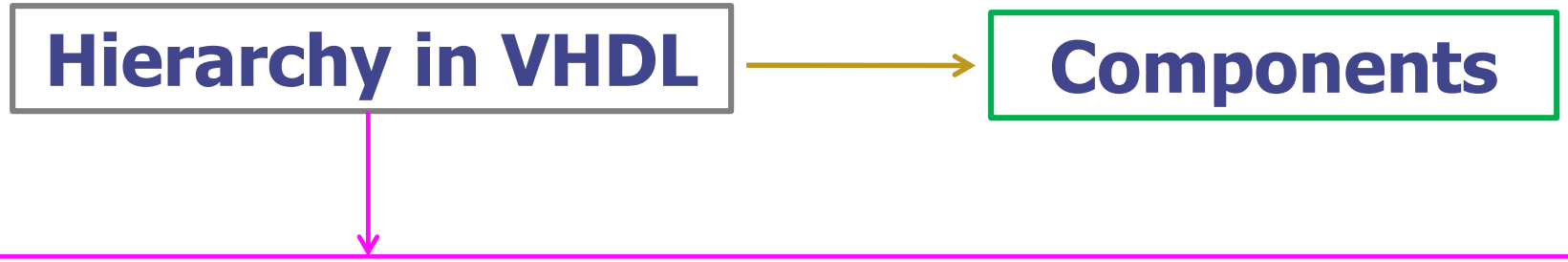
ff.vhd

```
entity ff is
port(
    d,clk : in  std_logic;
    q     : out std_logic);
end ff;
```

```
architecture test of ff is
begin
process(clk)
begin
 if(rising_edge(clk)) then
       q <= d;
 end if;
end test;
```

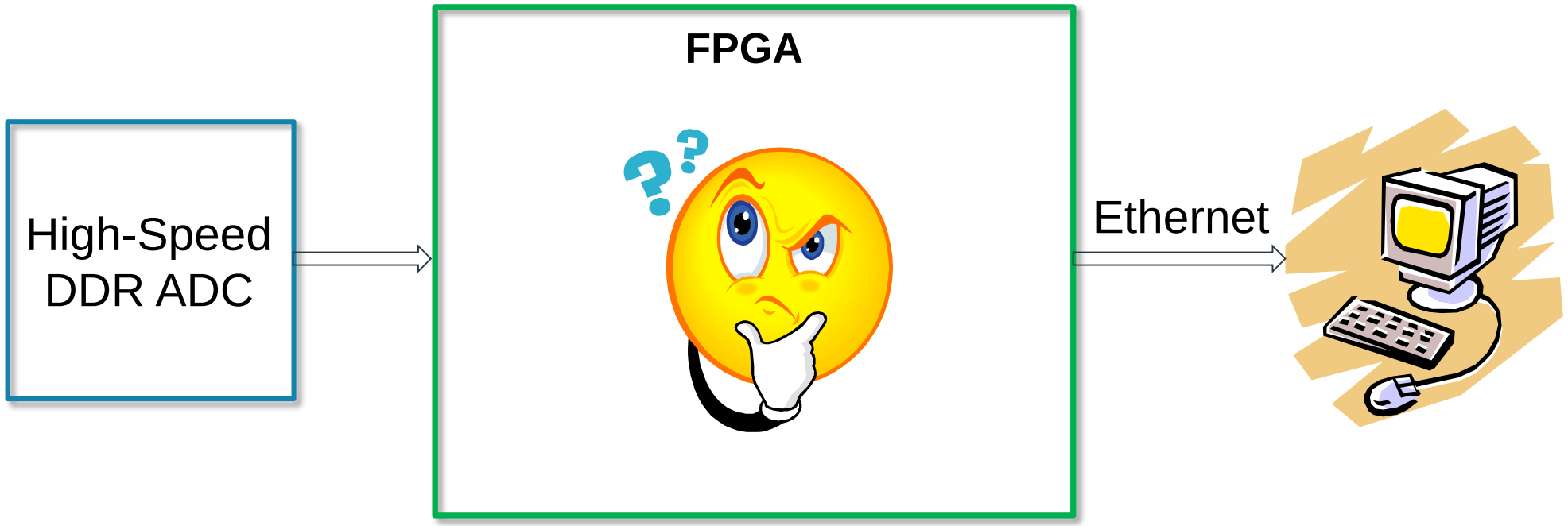
Components in VHDL

The Role of Components in VHDL

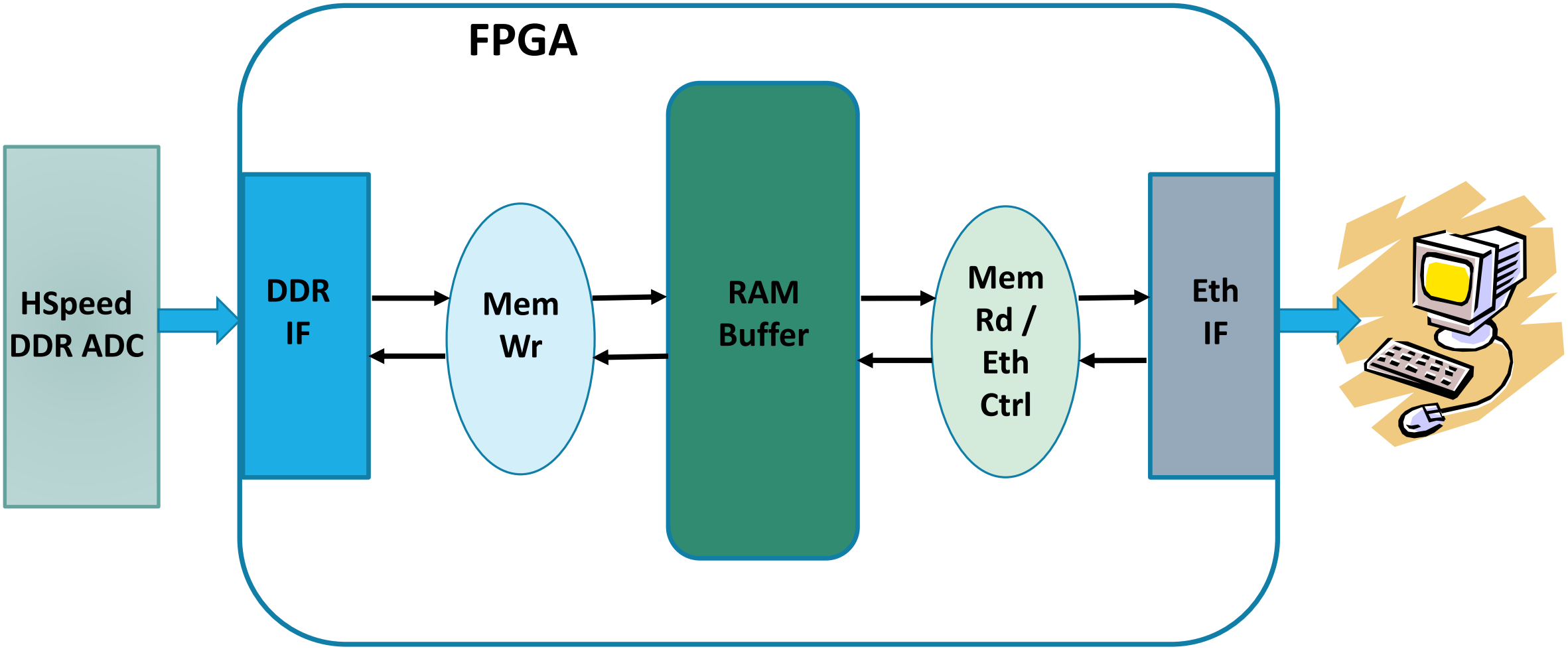


- Divide & Conquer
- Each subcomponent can be designed and completely tested
- Create library of components (technology independent if possible)
- Third-party available components
- Code for reuse

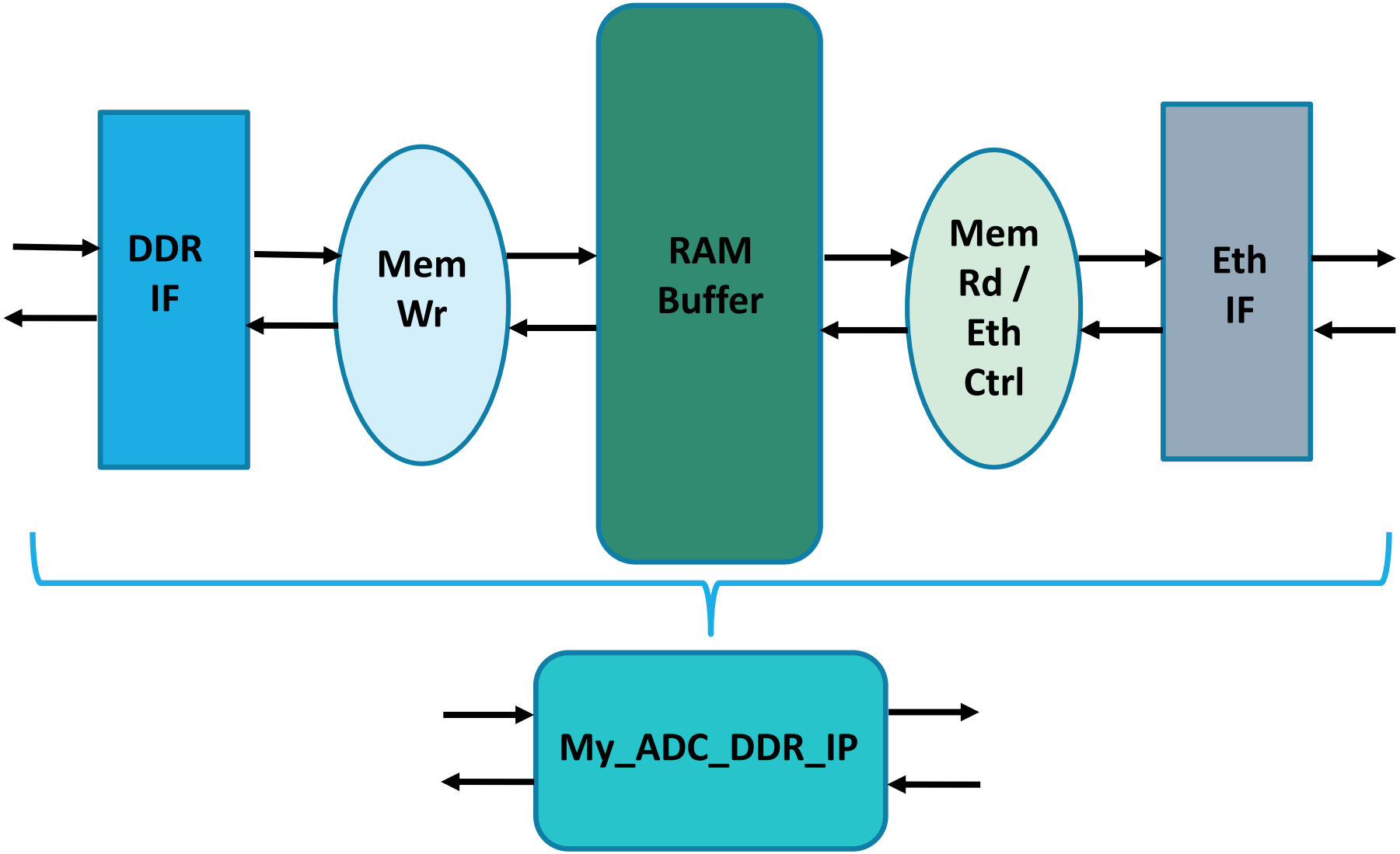
Hierarchy in VHDL - Components



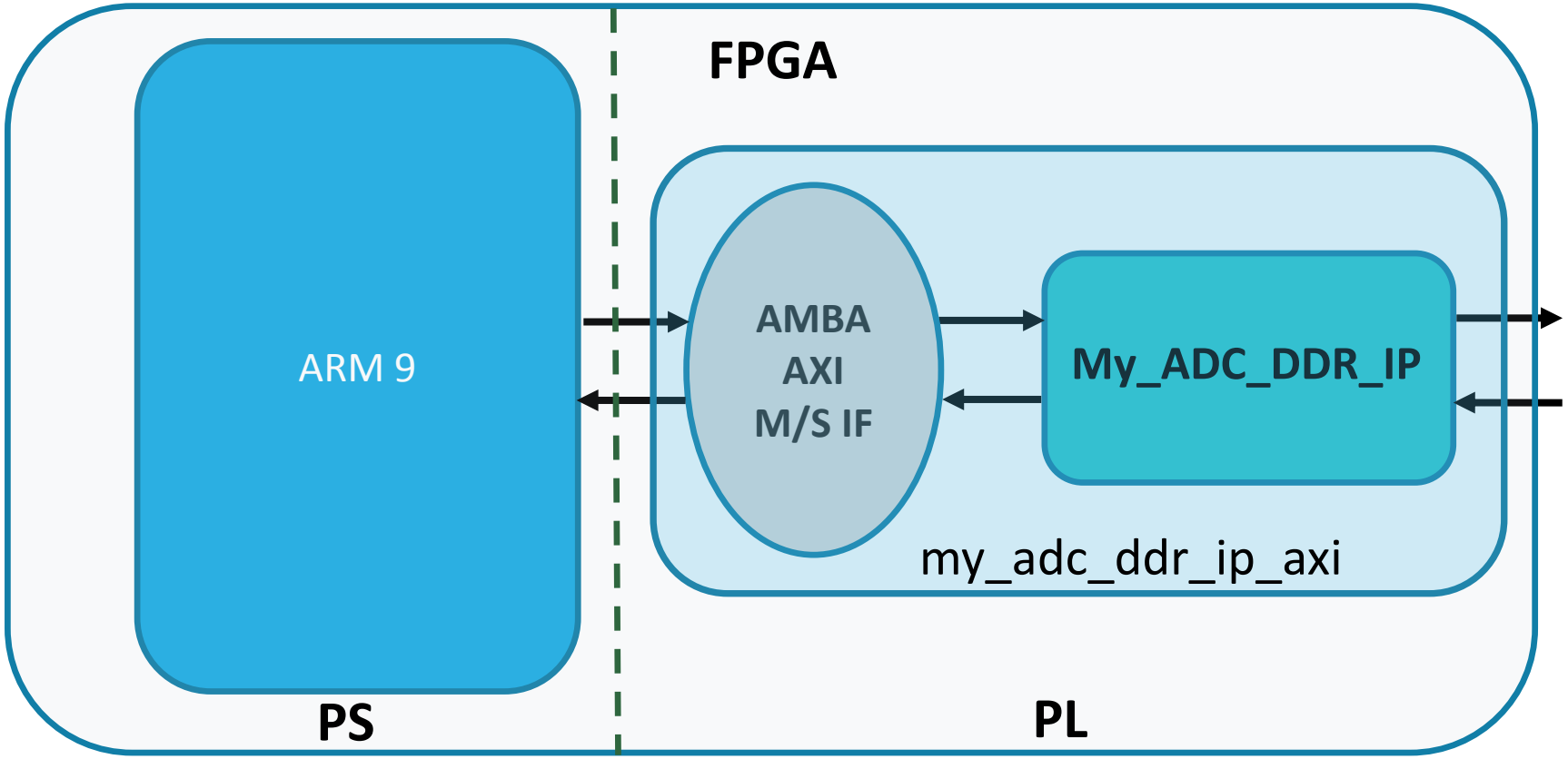
Importance of Hierarchy in VHDL



Importance of Hierarchy in VHDL: IP

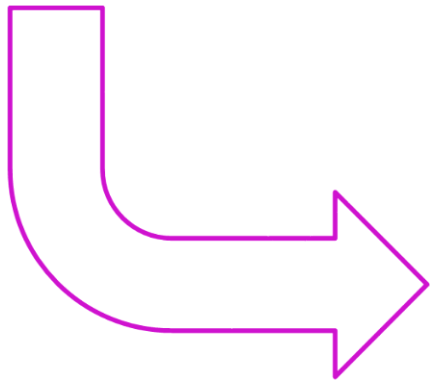


IP I/O Interfaces on a SoC FPGA (Zynq)



Component Declaration

```
entity nand2 is
       port (a, b: in std_logic,
                z: out std_logic);
end;
architecture rtl of nand2 is
…
end;
```



```
entity top is
    port(…
      );
end;
achitecture structural of top is
component nand2
      port (a, b: in  std_logic,
               z   : out std_logic);
end component;
…
begin
….
end;
```

Component Instantiation

```
component_label: component_name
      [generic map (generic_assocation_list)]
          port map (port_association_list);
```

- `component_label` it labels the instance by giving a name to the component to be instantiated
- `generic_assocation_list` assigns new values to the default generic values (given in the entity declaration)
- `port_association_list` associate the signals in the top entity/architecture with the ports of the component. There are two ways of specifying the port map:
  - Positional Association / Name Association

Association by Position

In positional association, an association list is of the form

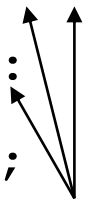
```
(actual1, actual2, actual3, … actualn);
```

- Each actual in the component instantiation is mapped *by position* with each port in the component declaration
- That is, the first port in the component declaration corresponds to the first *actual* in the component instantiation, the second with the second and so on
- The I/Os on the component declaration, are called *formals*

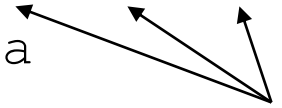
Association by Position

```
-- component declaration
component NAND2
      port (a, b: in std_logic,
            z:    out std_logic);
end component;

-- component instantiation
U1: NAND2 port map (S1, S2, S3);
-- S1 associated with a
-- S2 associated with b
-- S3 associated with z
```



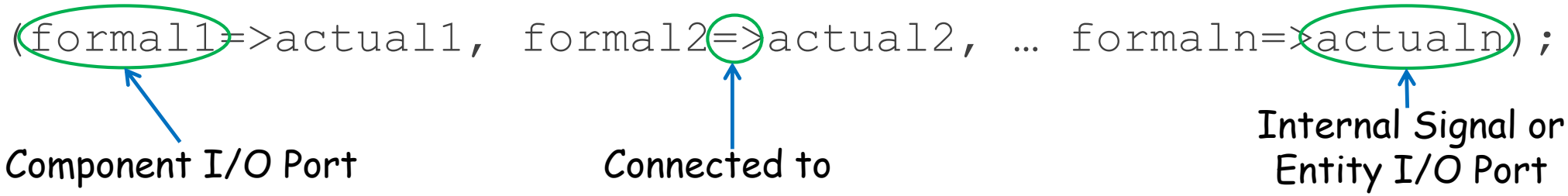
formals



actuals

Association by Name

In named association, an association list is of the form




```
-- component declaration
component NAND2
        port (a, b: in std_logic;
                z: out std_logic);
end component;
-- component instantiation
U1: NAND2 port map (a=>S1, z=>S3, b=>S2);
-- S1 associated with a, S2 with b and S3 with z
```

Unconnected Outputs

When a component is instanced, one of the outputs sometimes has to be unconnected

This can be done using the keyword *open*

```
architecture rtl of top_level is
component ex4
    port (a, b: in std_logic;
          q1, q2: out std_logic;
end component;
begin
       U1: ex4 port map(a=>a, b=>b, q1=>dout, q2=>open);
end;
```

Generic Map

```
generic map (generic_assocation_list);
```

- If a *generic* have been specified in the component to be instanced, their value can be changed during instantiation using the command *generic map*
- By using generics, *it is possible to design components which can be parameterized*
- Positional and named association can be used

Component Example Using *generic*

```
architecture ejemplo of regist_variable is
component dff
   generic (width: positive);
   port (rst, clk: in std_logic;
                d: in std_logic_vector(width-1 downto 0);
                q: out std_logic_vector(width-1 downto 0));
end component;
constant width_8: positive:= 8;
constant width_16: positive:= 16;
constant width_32: positive:= 32;
signal d8, q8: std_logic_vector(7 downto 0);
signal d16, q16: std_logic_vector(15 downto 0);
signal d32, q32: std_logic_vector(31 downto 0);
```

Component Example Using *generic*

```
architecture ejemplo of regist_variable is
component dff
   generic (width: positive);
   port (rst, clk: in std_logic;
                d: in std_logic_vector(width-1 downto 0);
                q: out std_logic_vector(width-1 downto 0));
end component;
constant width_8: positive:= 8;
constant width_16: positive:= 16;
constant width_32: positive:= 32;
signal d8, q8: std_logic_vector(7 downto 0);
signal d16, q16: std_logic_vector(15 downto 0);
signal d32, q32: std_logic_vector(31 downto 0);
```

```
begin
 ff8: dff generic map(width_8)
           port map (rst, clk, d8, q8);
 ff16:dff generic map(width_16)
           port map (rst, clk, d16, q16);
 ff32:dff generic map(width_32)
          port map (rst=>rst,clk=>clk,d=>d32, q=>q32);
end ejemplo;
```