Introduction to Field Programmable Gate Arrays

Hannes Sakulin
CERN / EP-CMD

International School of Trigger and Data Acquisition 2024

USTC Hefei, China, 20 June 2024

What is a **F**ield **P**rogrammable **G**ate **A**rray ?
… a quick answer for the impatient

- An FPGA is an integrated circuit
  - Mostly digital electronics

- An FPGA is programmable in the in the field (=outside the factory), hence the name "field programmable"
  - Circuit design is specified with a hardware description language or schematics
  - Tools compute a programming file for the FPGA (bitstream)
  - The FPGA is configured with the design (gateware / firmware)
  - Your electronic circuit is ready to use

With an FPGA you can build electronic circuits …
… without using a bread board or soldering iron
… without plugging together NIM modules
… without having a chip produced at a factory

Outline

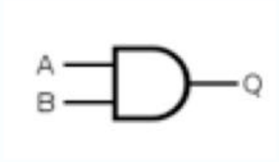
- Quick look at digital electronics
- FPGAs and their features
- Programming techniques
- Design flow
- Example Applications in the Trigger and DAQ domain

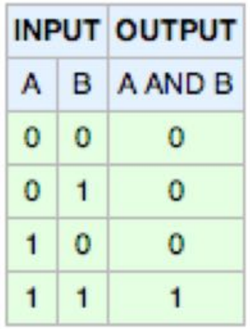
The basic elements of digital electronics

The building blocks: logic gates

Truth table | C equivalent

AND gate

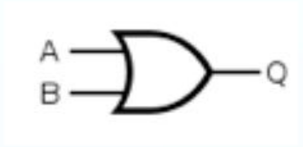


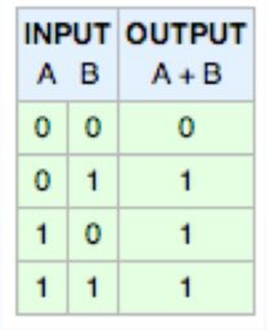


INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

q = a && b;

OR gate

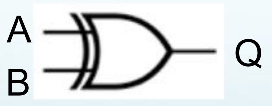


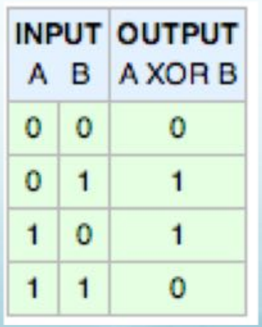


INPUT		OUTPUT
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	1

q = a || b;

Exclusive OR gate
XOR gate



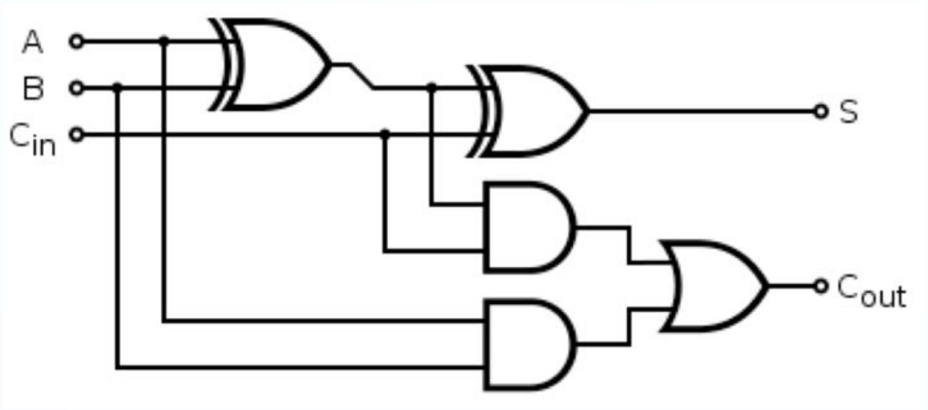


INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

q = a != b;

⋮

Combinatorial logic (asynchronous)



Example: Full adder with carry-in, carry-out

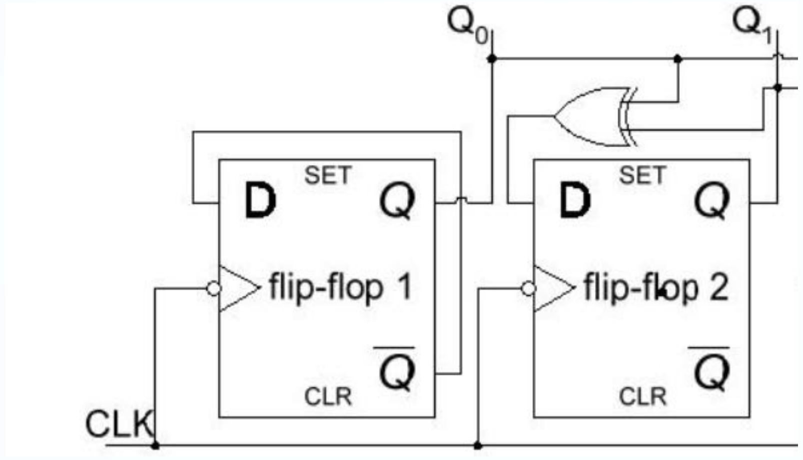
Outputs are determined by Inputs, only

A	B	C_{in}	S	C_{out}
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1

Combinatorial logic may be implemented using Look-Up Tables (LUTs)

LUT = small memory

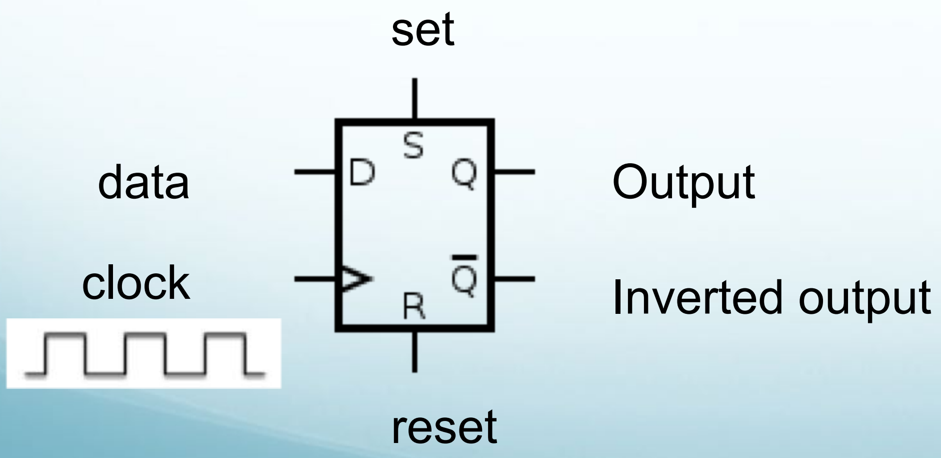
(Synchronous) sequential logic



Outputs are determined
by Inputs and their history
(Sequence)
The logic has an internal state

Example: 2-bit binary counter
https://www.zeepedia.com/read.php?b=9&c=32&d_flip-flop_based_implementation_digital_logic_design

Element that keeps the state: Flip-flop

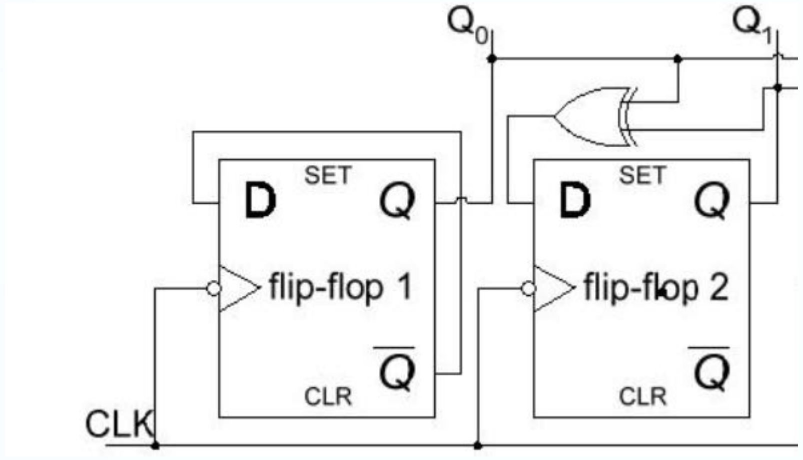


D Flip-flop (D=data, delay) :

samples the data at the rising
(or falling) edge of the clock

The output will be equal to
the last sampled input until the
next rising (or falling) clock edge

(Synchronous) sequential logic

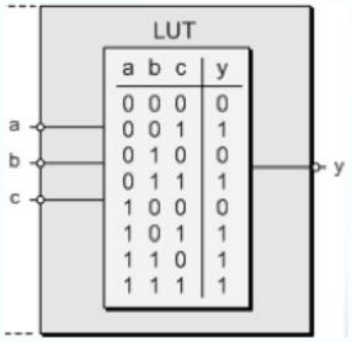


Outputs are determined by Inputs and their history (Sequence)
The logic has an internal state

Example: 2-bit binary counter
https://www.zeepedia.com/read.php?b=9&c=32&d_flip-flop_based_implementation_digital_logic_design

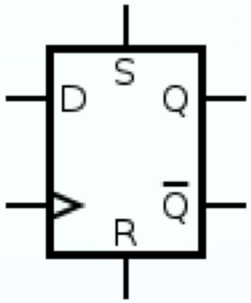
	Flip-flop 1			Flip-flop 2			
time	**D**	**Q**	**Q'**	**D=Q_0 xor Q_1**	**Q**	**Q_1**	**Q_0**
Before clock edge 1	1	0	1	0	0	0	0
after clock edge 1	0	1	0	1	0	0	1
after clock edge 2	1	0	1	1	1	1	0
after clock edge 3	0	1	0	0	1	1	1
after clock edge 4	1	0	1	0	0	0	0

Synchronous sequential logic



a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

+



=

Signal processor

Trigger logic

Data compression logic

Network Interface Card

Neural net classifier

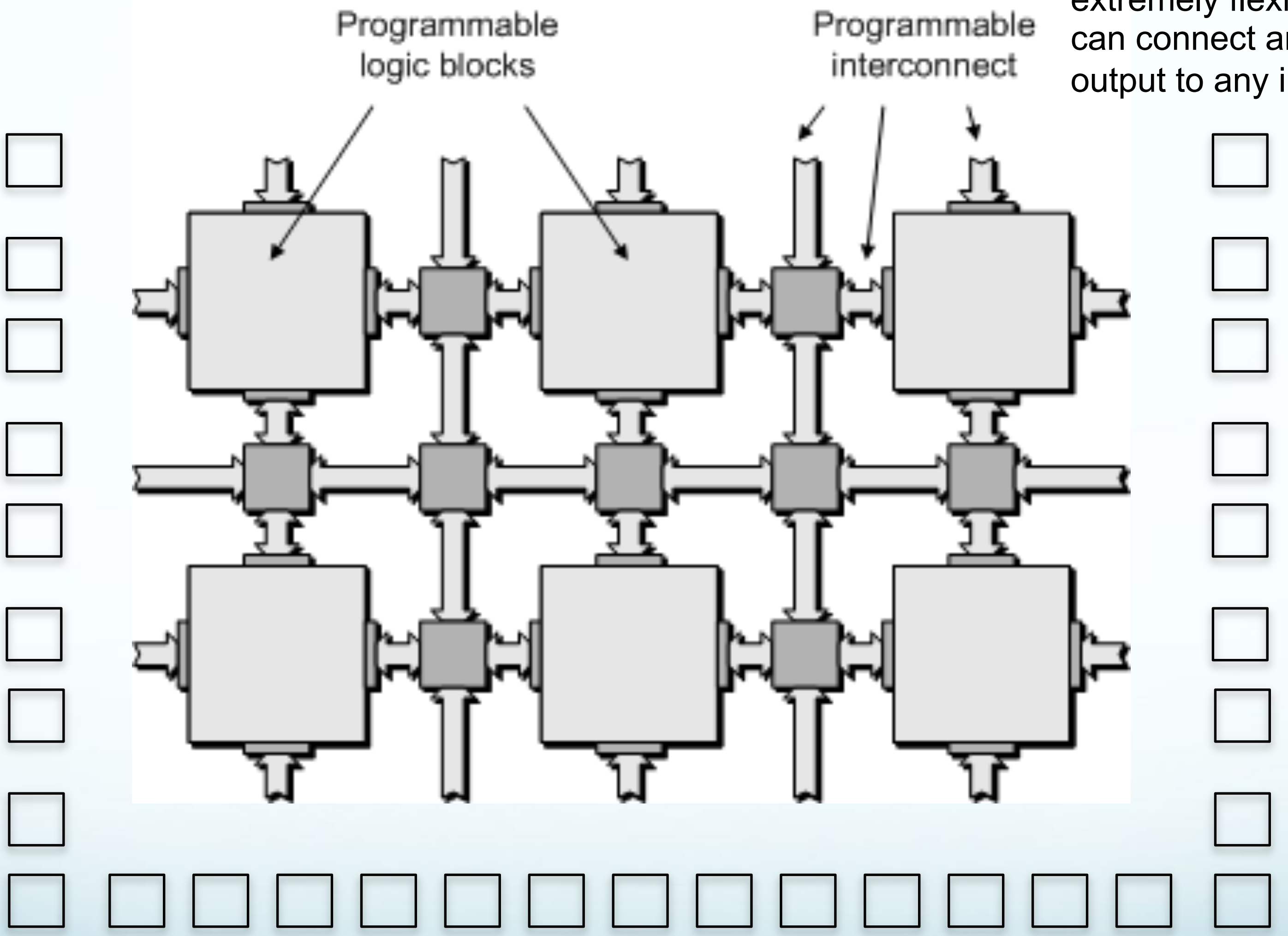
⋮

Using Look-Up-Tables and Flip-Flops
any kind of digital electronics may be implemented

Of course electronics design is an art in itself …

What is inside an FPGA?

Basic elements of an FPGA

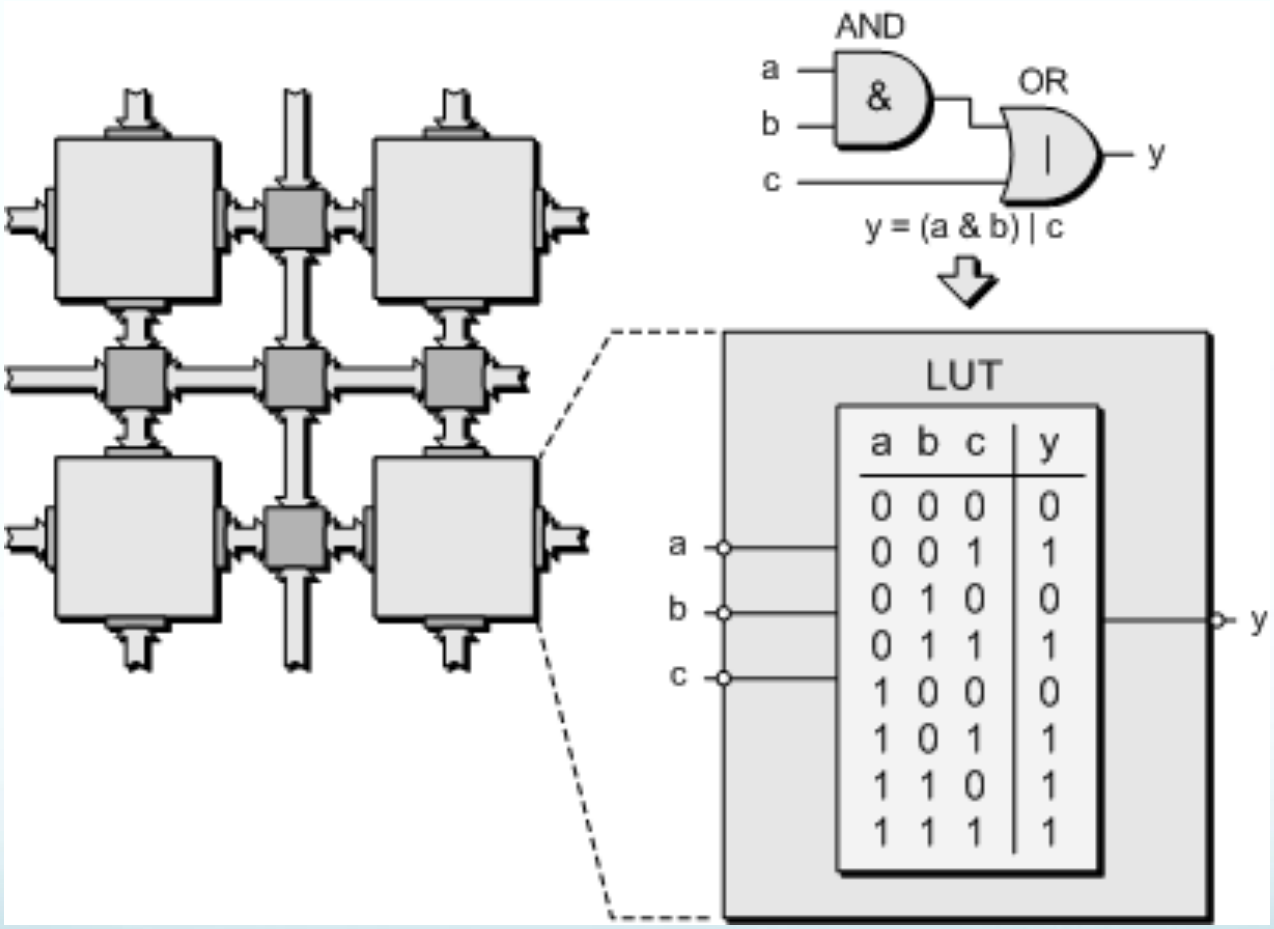


extremely flexible:
can connect any block
output to any input

Fine-grained: 10.000's
up to millions of logic blocks

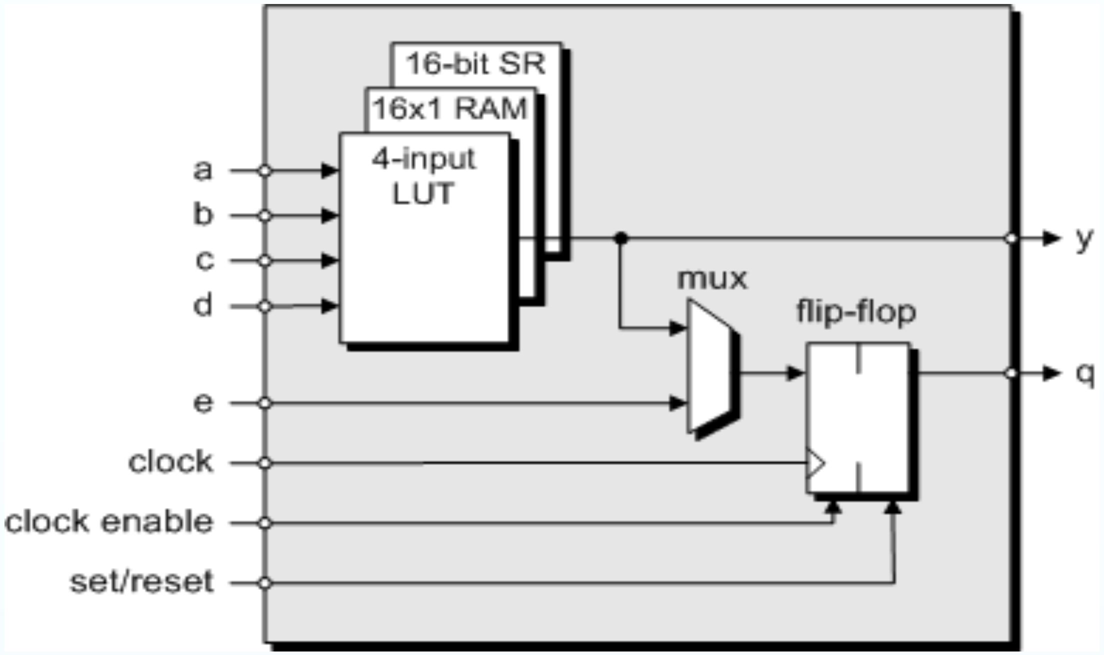
Programmable Input / Output pins

LUT-based Fabrics



a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

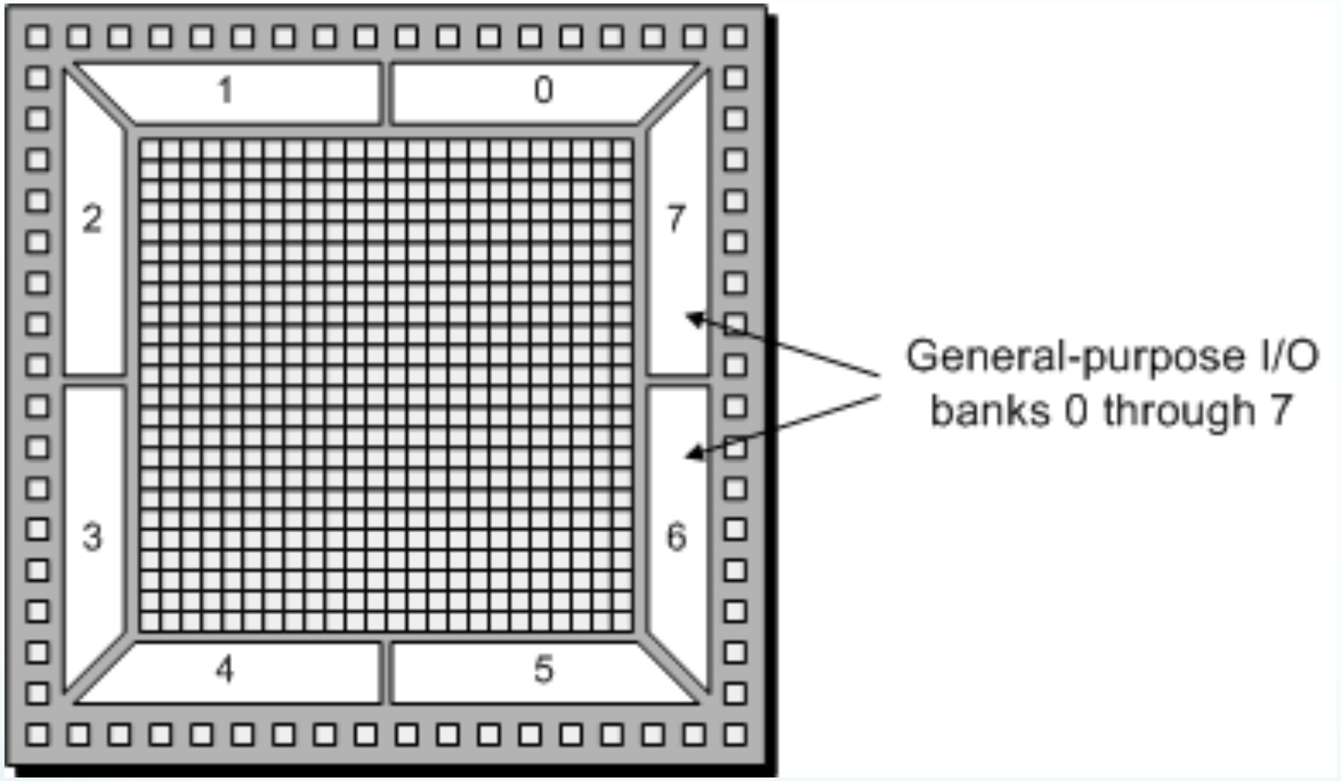
Typical LUT-based Logic Cell



Xilinx: logic cell,
Altera: logic element

- LUT may implement any function of the inputs
- Flip-Flop registers the LUT output
- May use only the LUT or only the Flip-flop
- LUT may alternatively be configured a shift register
- Additional elements (not shown): fast carry logic

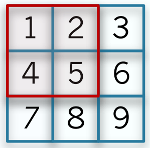
General-Purpose Input/Output (GPIO)



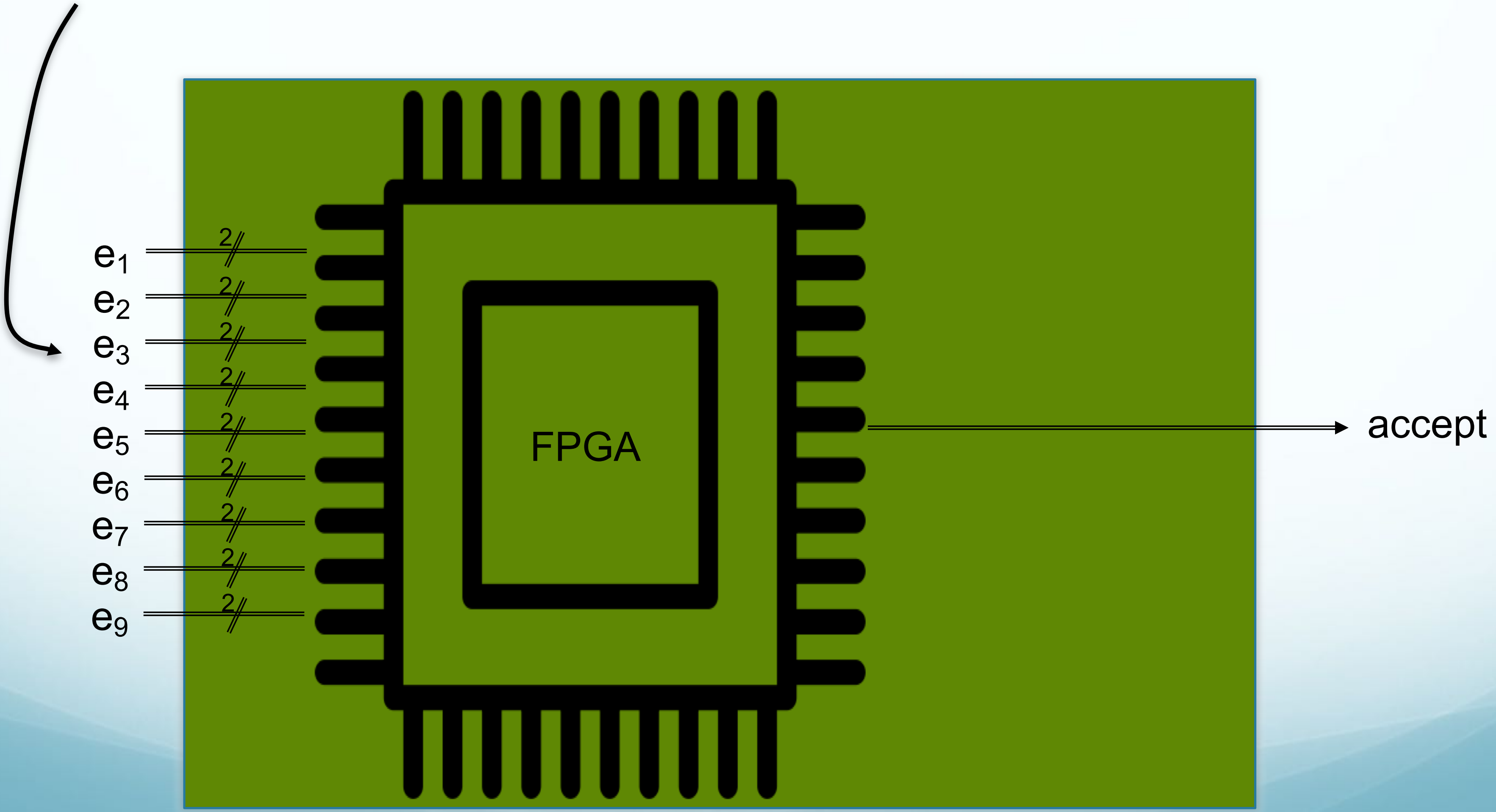
Today: Up to >1000 user I/O pins
- Input and / or output
- Voltages from (1.0), 1.2 .. 3.3 V
- Many IO standards
- Single-ended: LVTTL, LVCMOS, …
- Differential pairs: LVDS, …

A toy example

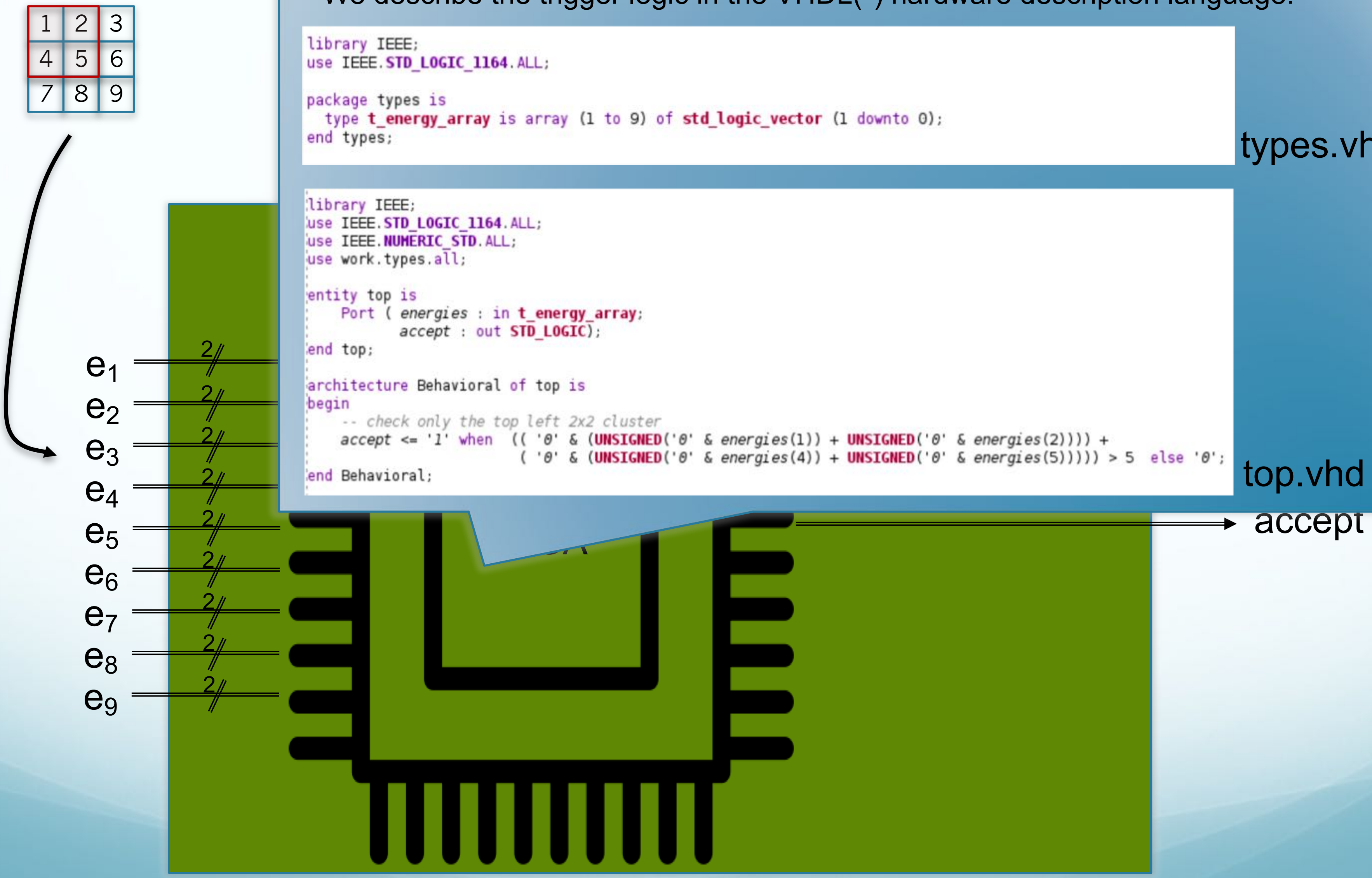
Toy example: trigger on energy cluster



Say, we have a 3x3 pixel detector
Each pixel can measure deposited energy with 2 bit resolution
Trigger condition: the sum of energies deposited in a 2x2 pixel area exceeds 5 counts.



Toy example: VHDL code



We describe the trigger logic in the VHDL(*) hardware description language:

types.vhd

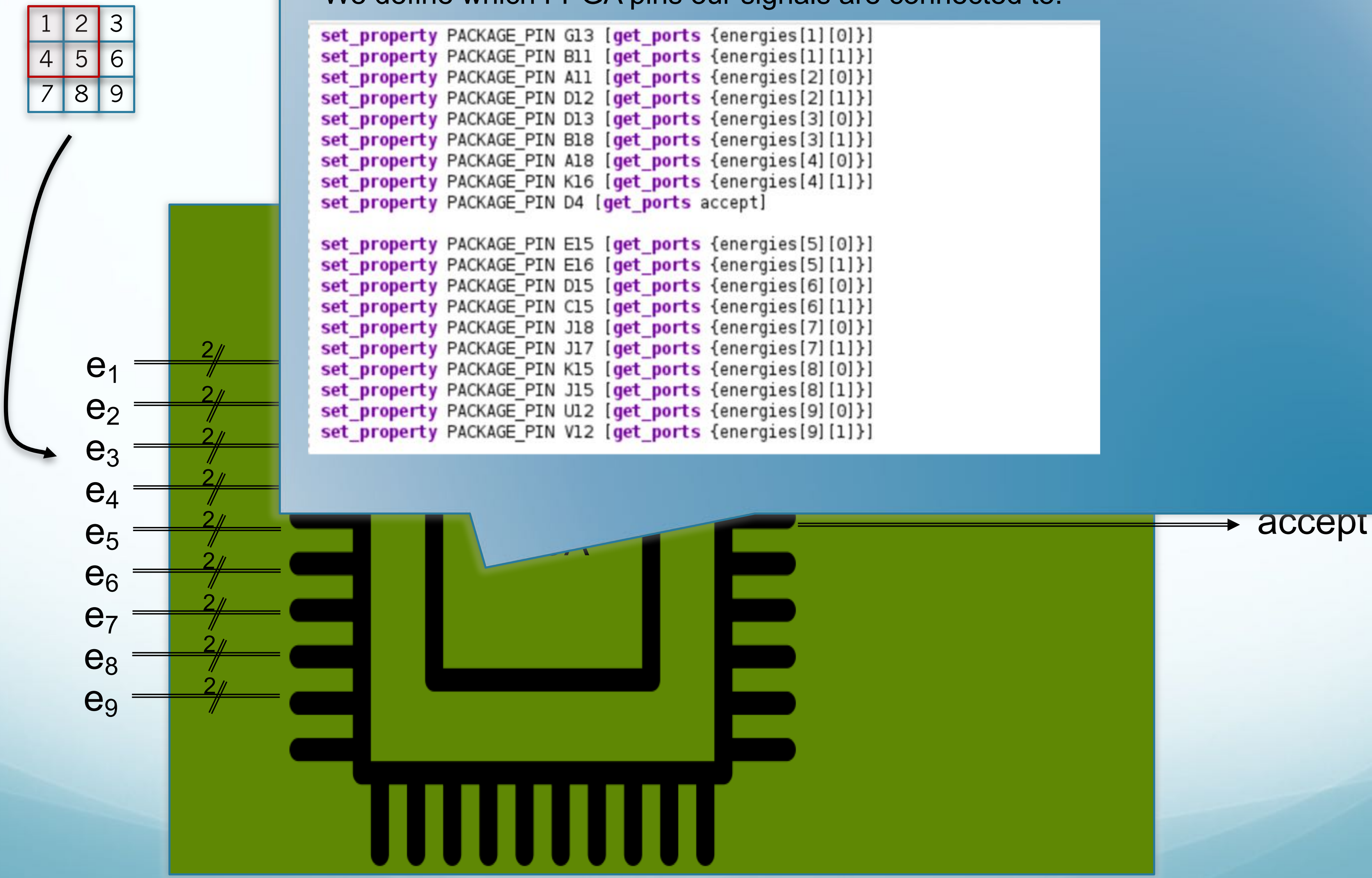
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

package types is
  type t_energy_array is array (1 to 9) of std_logic_vector (1 downto 0);
end types;
```

top.vhd

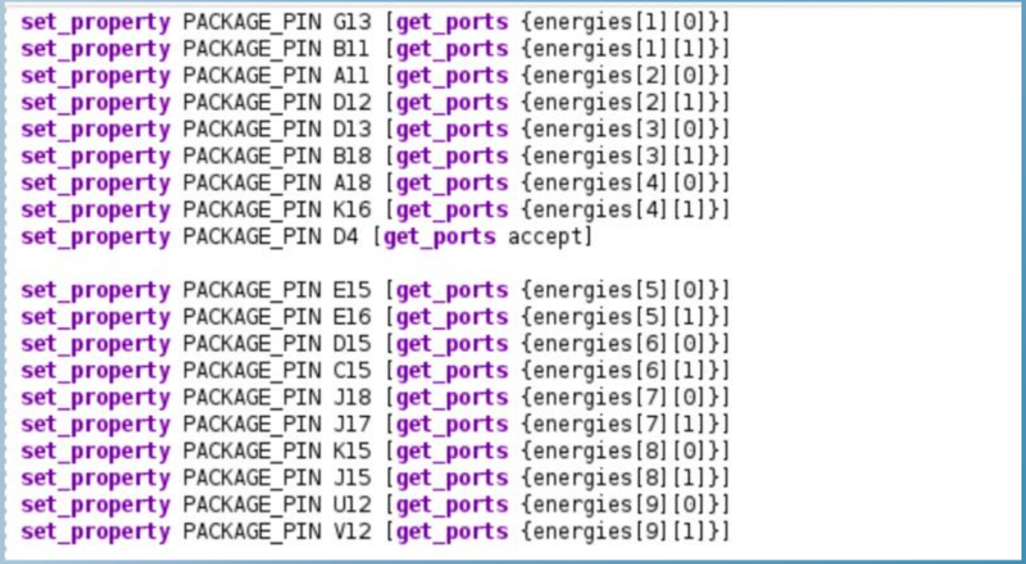
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.types.all;

entity top is
    Port ( energies : in t_energy_array;
           accept : out STD_LOGIC);
end top;

architecture Behavioral of top is
begin
    -- check only the top left 2x2 cluster
    accept <= '1' when  (( '0' & (UNSIGNED('0' & energies(1)) + UNSIGNED('0' & energies(2)))) +
                          ( '0' & (UNSIGNED('0' & energies(4)) + UNSIGNED('0' & energies(5))))) > 5  else '0';
end Behavioral;
```

(*) Very High-Speed Integrated Circuit Hardware Description Language

Toy example: constraints



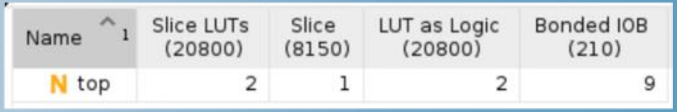
We define which FPGA pins our signals are connected to:



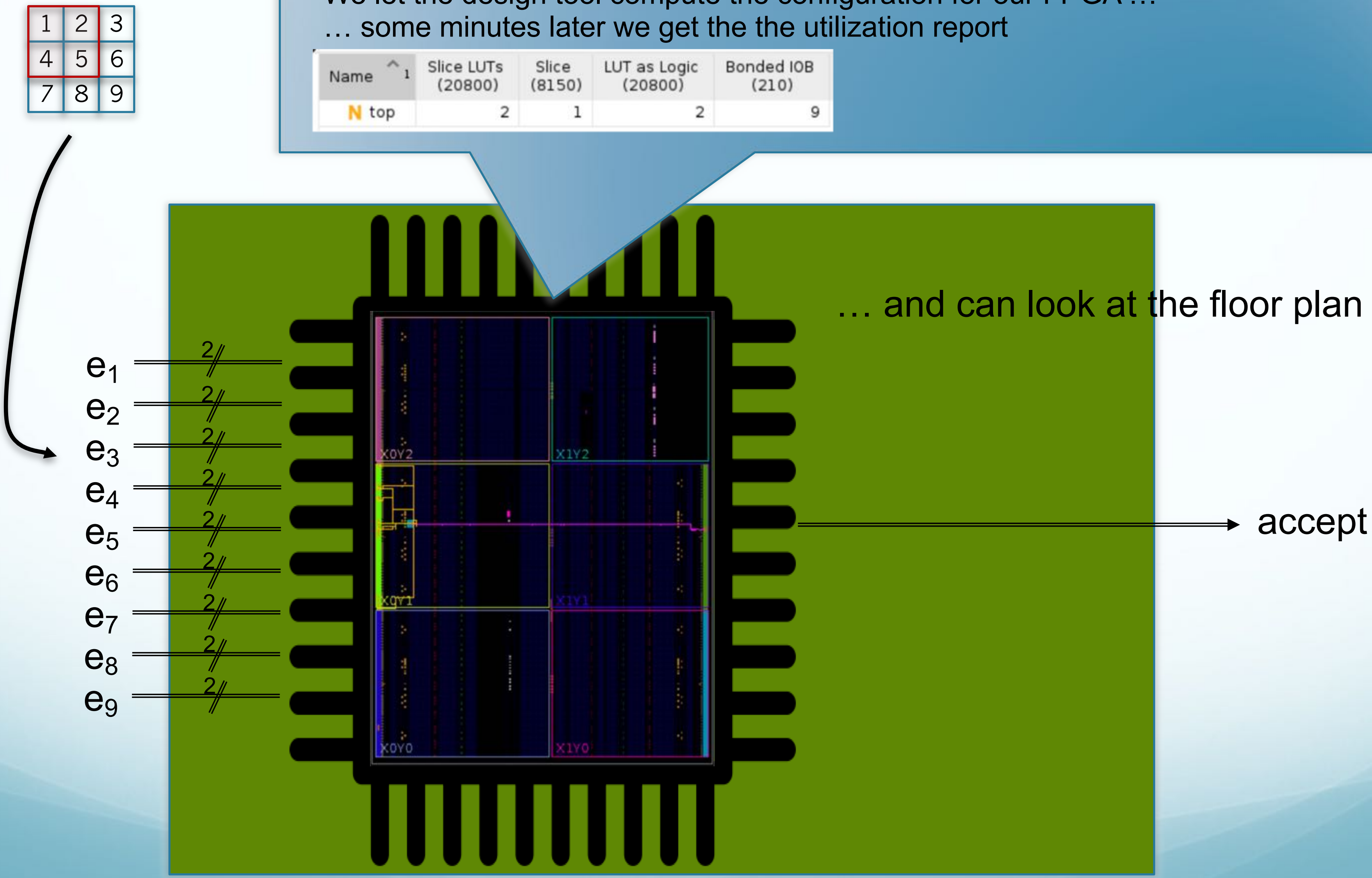
```
set_property PACKAGE_PIN G13 [get_ports {energies[1][0]}]
set_property PACKAGE_PIN B11 [get_ports {energies[1][1]}]
set_property PACKAGE_PIN A11 [get_ports {energies[2][0]}]
set_property PACKAGE_PIN D12 [get_ports {energies[2][1]}]
set_property PACKAGE_PIN D13 [get_ports {energies[3][0]}]
set_property PACKAGE_PIN B18 [get_ports {energies[3][1]}]
set_property PACKAGE_PIN A18 [get_ports {energies[4][0]}]
set_property PACKAGE_PIN K16 [get_ports {energies[4][1]}]
set_property PACKAGE_PIN D4 [get_ports accept]

set_property PACKAGE_PIN E15 [get_ports {energies[5][0]}]
set_property PACKAGE_PIN E16 [get_ports {energies[5][1]}]
set_property PACKAGE_PIN D15 [get_ports {energies[6][0]}]
set_property PACKAGE_PIN C15 [get_ports {energies[6][1]}]
set_property PACKAGE_PIN J18 [get_ports {energies[7][0]}]
set_property PACKAGE_PIN J17 [get_ports {energies[7][1]}]
set_property PACKAGE_PIN K15 [get_ports {energies[8][0]}]
set_property PACKAGE_PIN J15 [get_ports {energies[8][1]}]
set_property PACKAGE_PIN U12 [get_ports {energies[9][0]}]
set_property PACKAGE_PIN V12 [get_ports {energies[9][1]}]
```

Toy example: timing and floorplan

We let the design tool compute the configuration for our FPGA …
… some minutes later we get the the utilization report

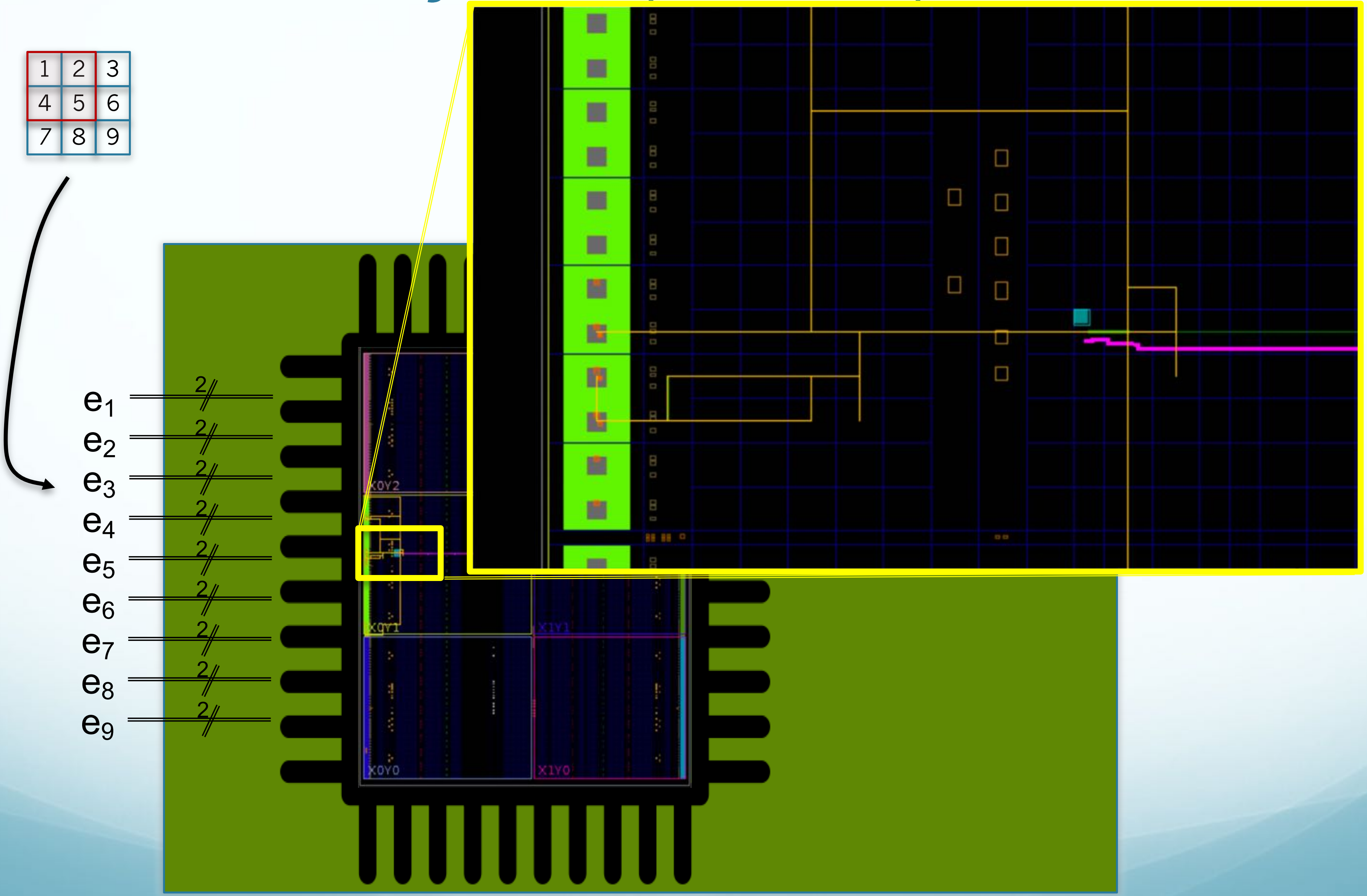


Name	Slice LUTs (20800)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (210)
N top	2	1	2	9

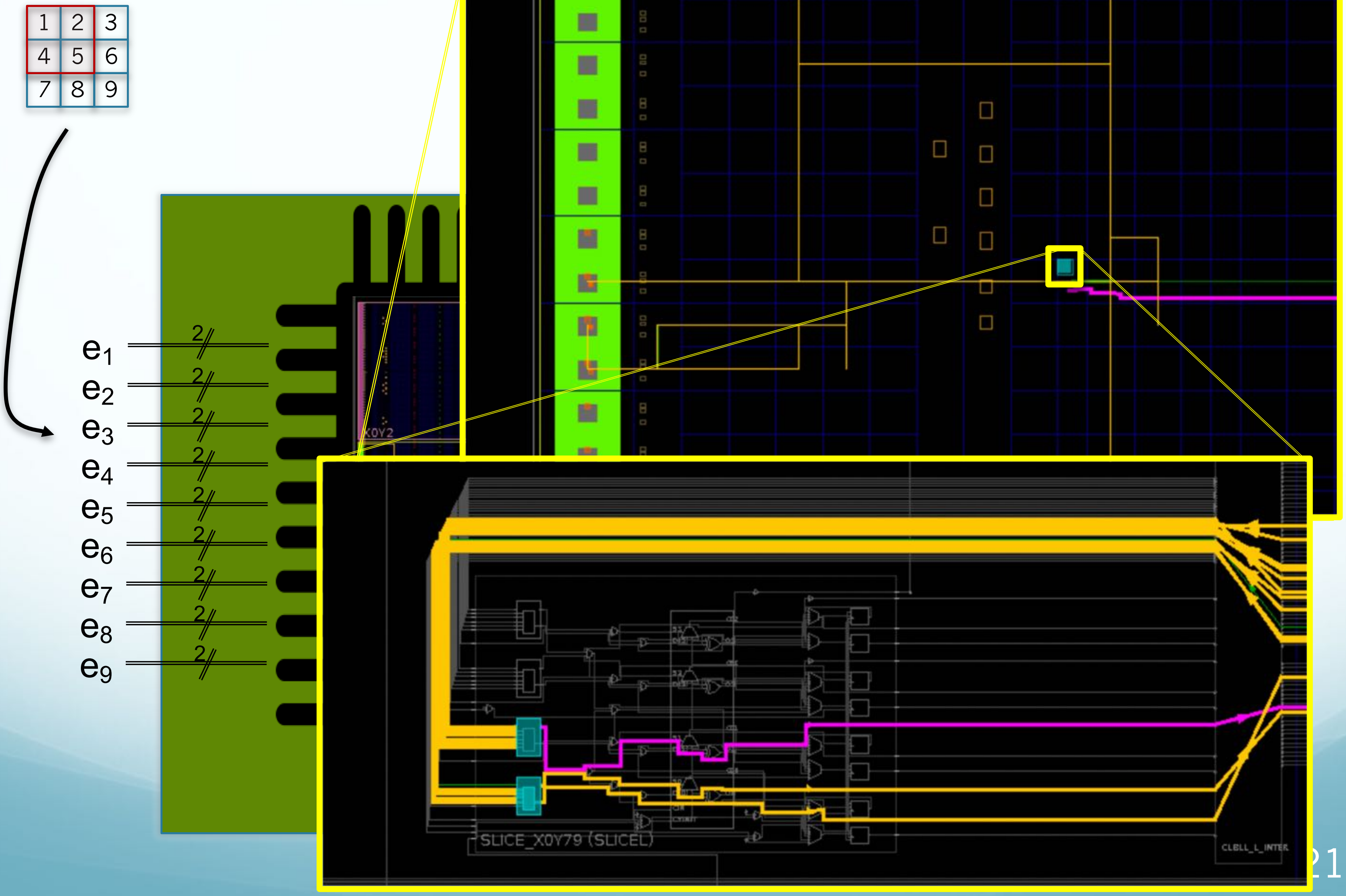


… and can look at the floor plan

Toy example: floorplan

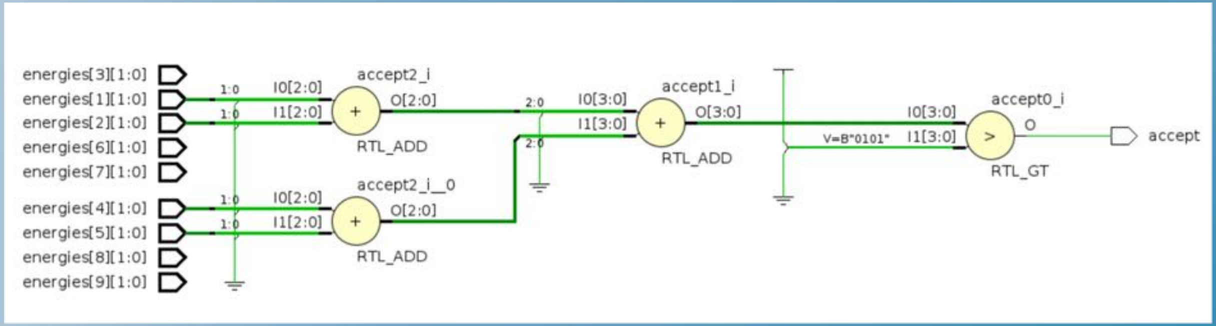


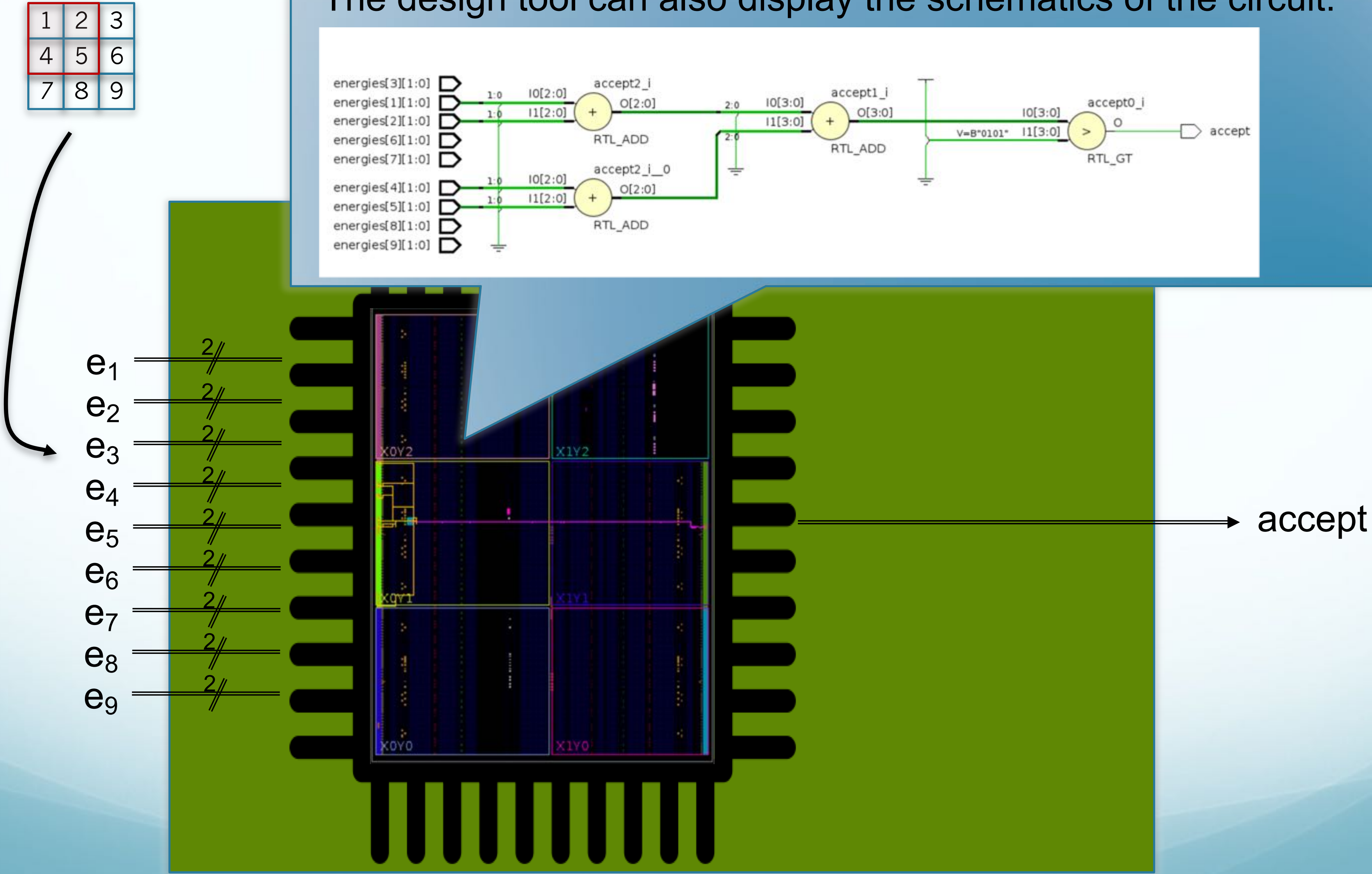
Toy example: floorplan



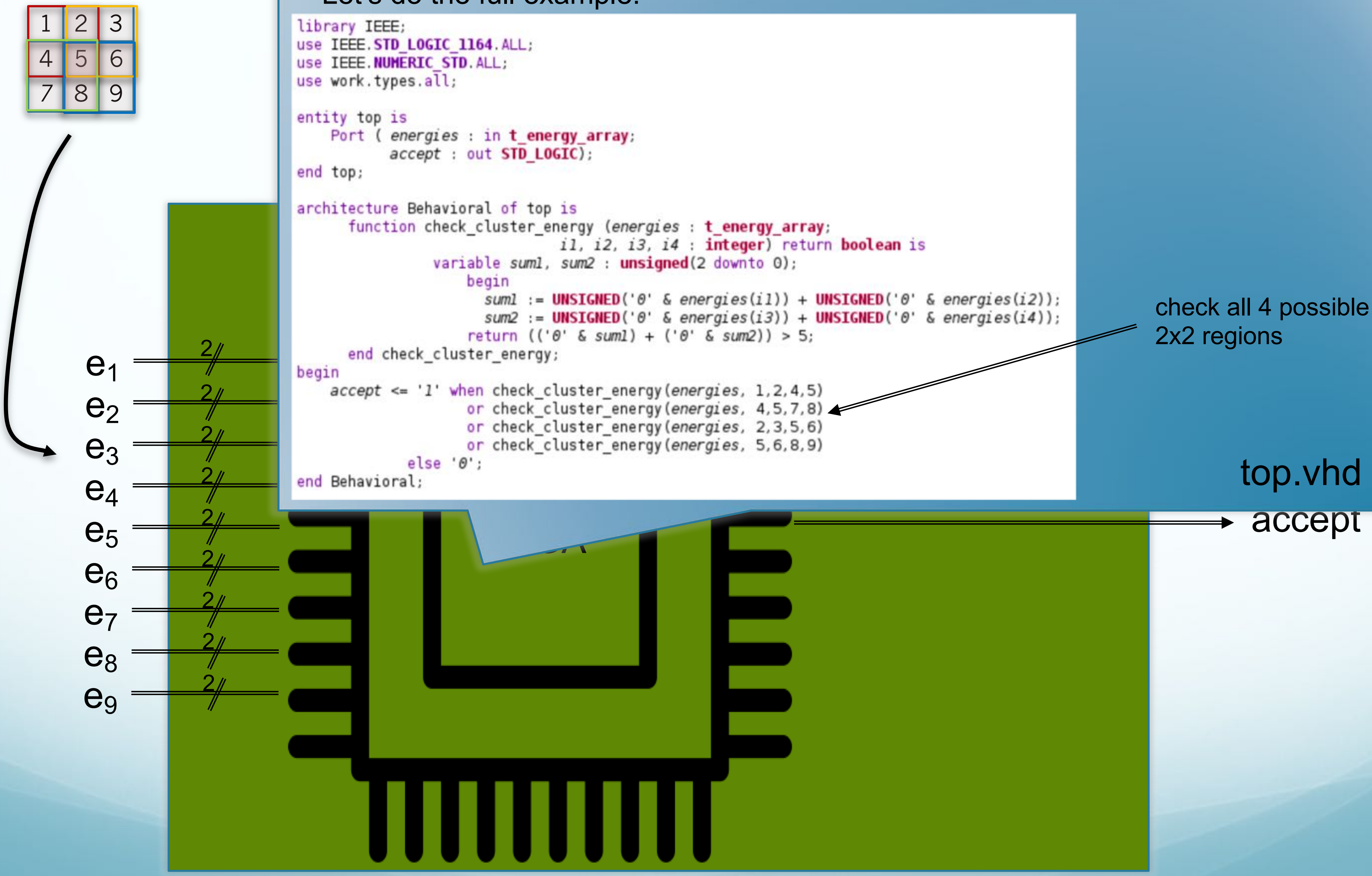
Toy example: Register Transfer Level (RTL) design

The design tool can also display the schematics of the circuit:





Toy example: Full example, 4 possible clusters



Let's do the full example:

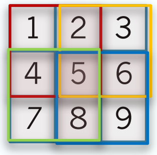
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use work.types.all;

entity top is
    Port ( energies : in t_energy_array;
           accept : out STD_LOGIC);
end top;

architecture Behavioral of top is
      function check_cluster_energy (energies : t_energy_array;
                                     i1, i2, i3, i4 : integer) return boolean is
            variable sum1, sum2 : unsigned(2 downto 0);
                begin
                  sum1 := UNSIGNED('0' & energies(i1)) + UNSIGNED('0' & energies(i2));
                  sum2 := UNSIGNED('0' & energies(i3)) + UNSIGNED('0' & energies(i4));
                return (('0' & sum1) + ('0' & sum2)) > 5;
      end check_cluster_energy;
begin
    accept <= '1' when check_cluster_energy(energies, 1,2,4,5)
                    or check_cluster_energy(energies, 4,5,7,8)
                    or check_cluster_energy(energies, 2,3,5,6)
                    or check_cluster_energy(energies, 5,6,8,9)
               else '0';
end Behavioral;
```

check all 4 possible 2x2 regions

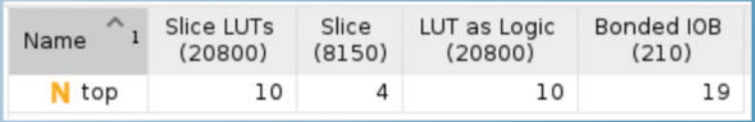
top.vhd

(*) Very High-Speed Integrated Circuit Hardware Description Language

Toy example: resource usage and floorplan

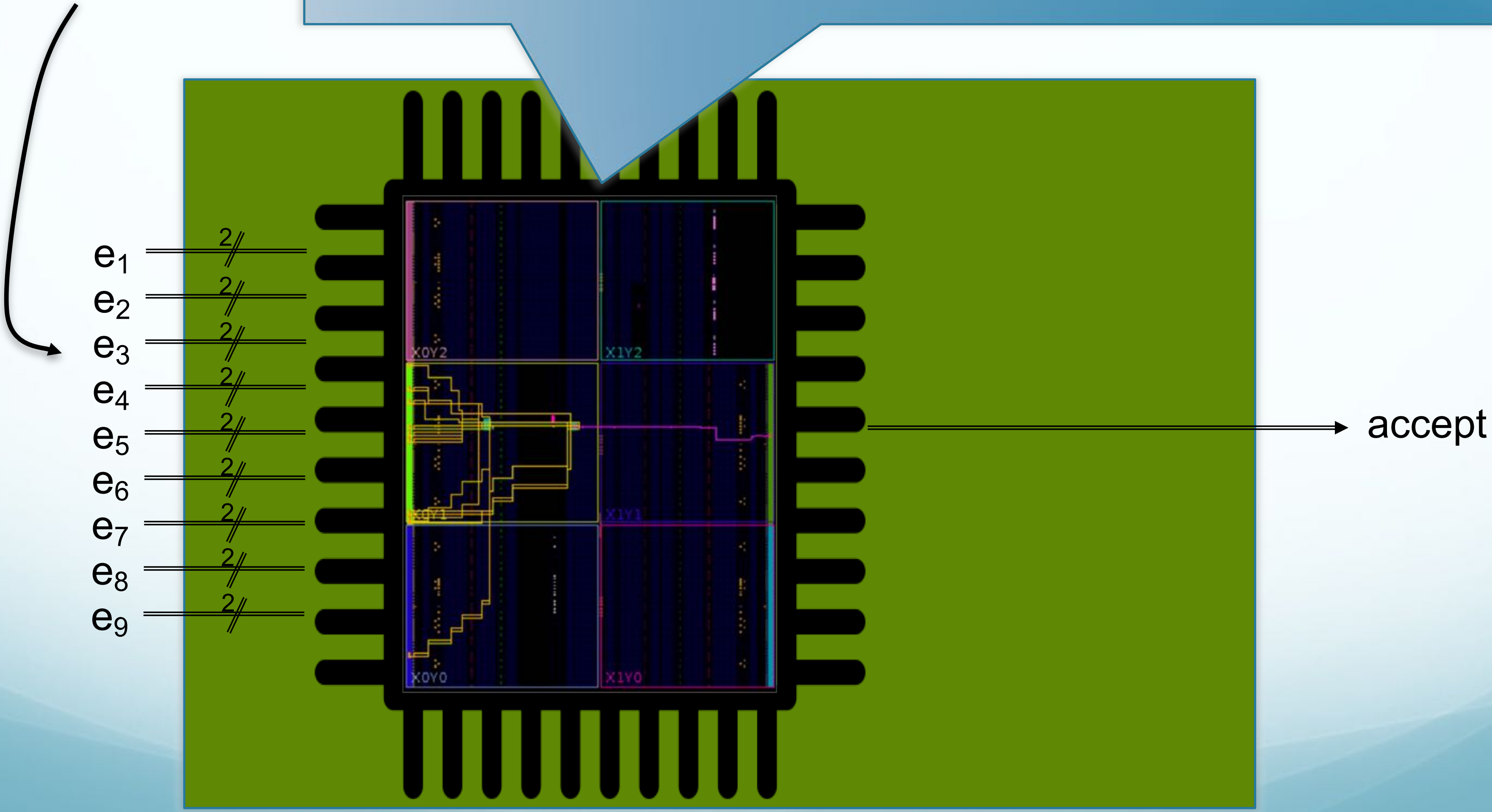


We let the design tool compute the configuration for our FPGA



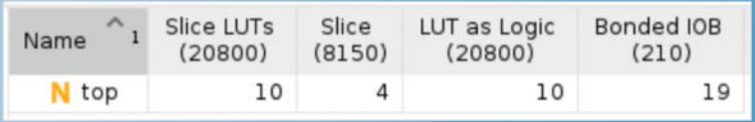
Name	Slice LUTs (20800)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (210)
N top	10	4	10	19

utilization report



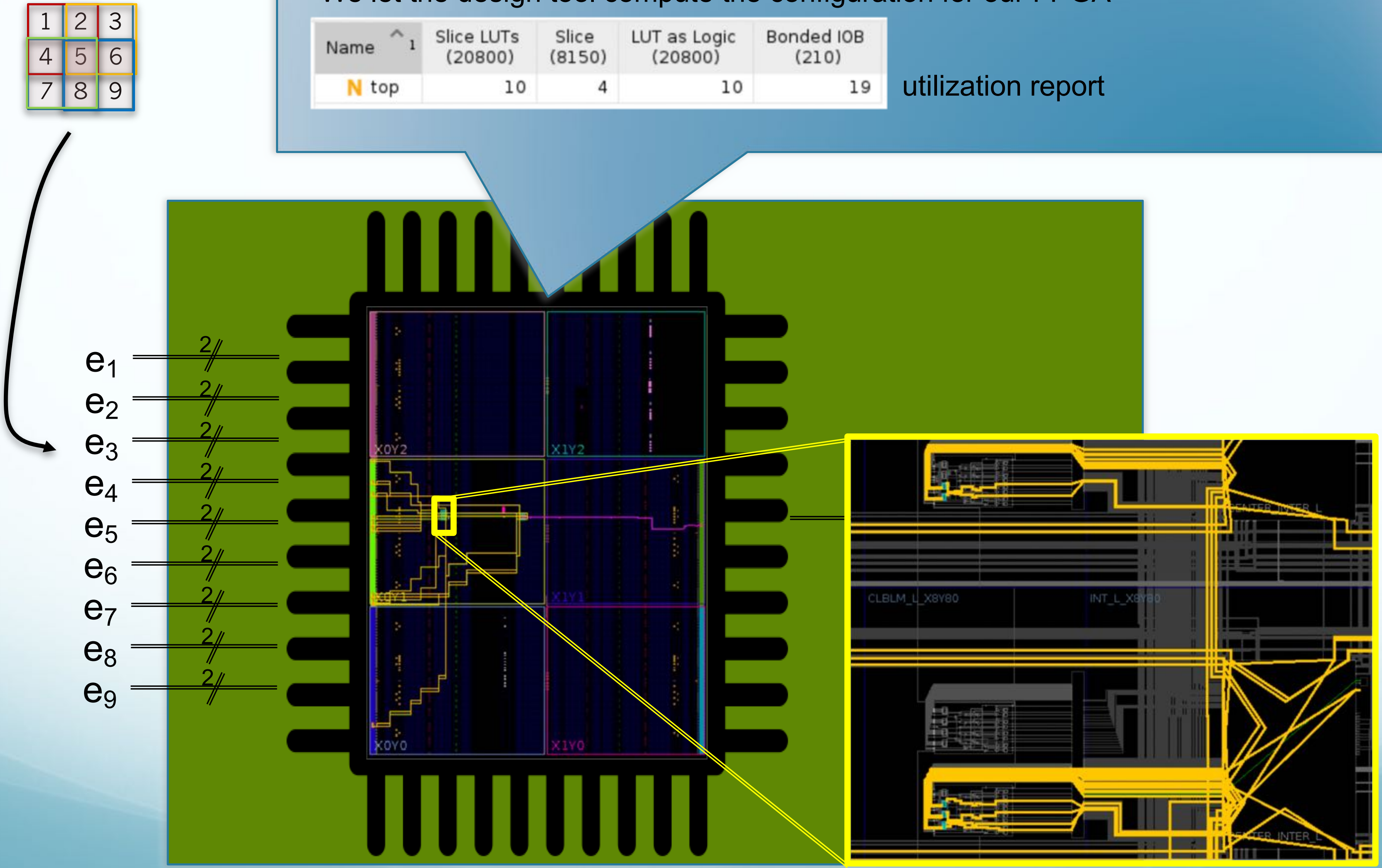
Toy example: floorplan

We let the design tool compute the configuration for our FPGA



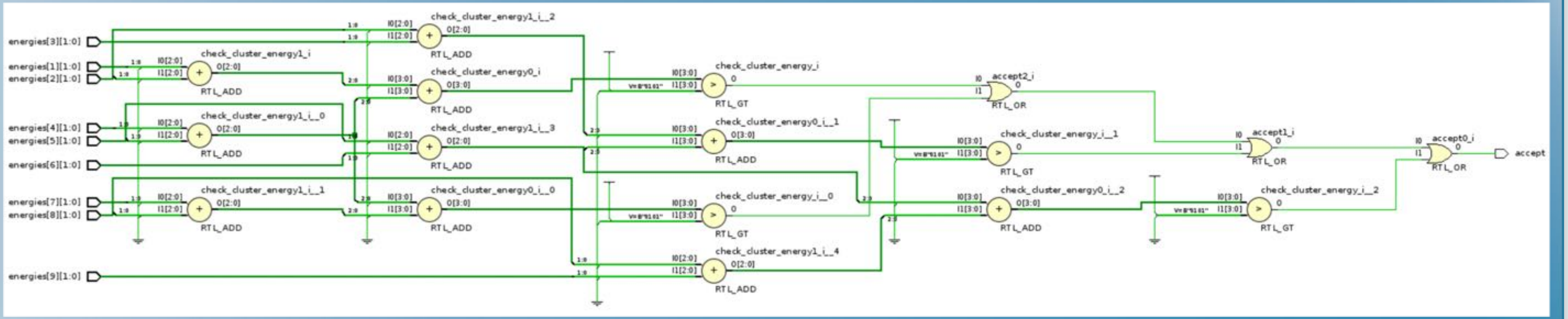
Name	Slice LUTs (20800)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (210)
N top	10	4	10	19

utilization report

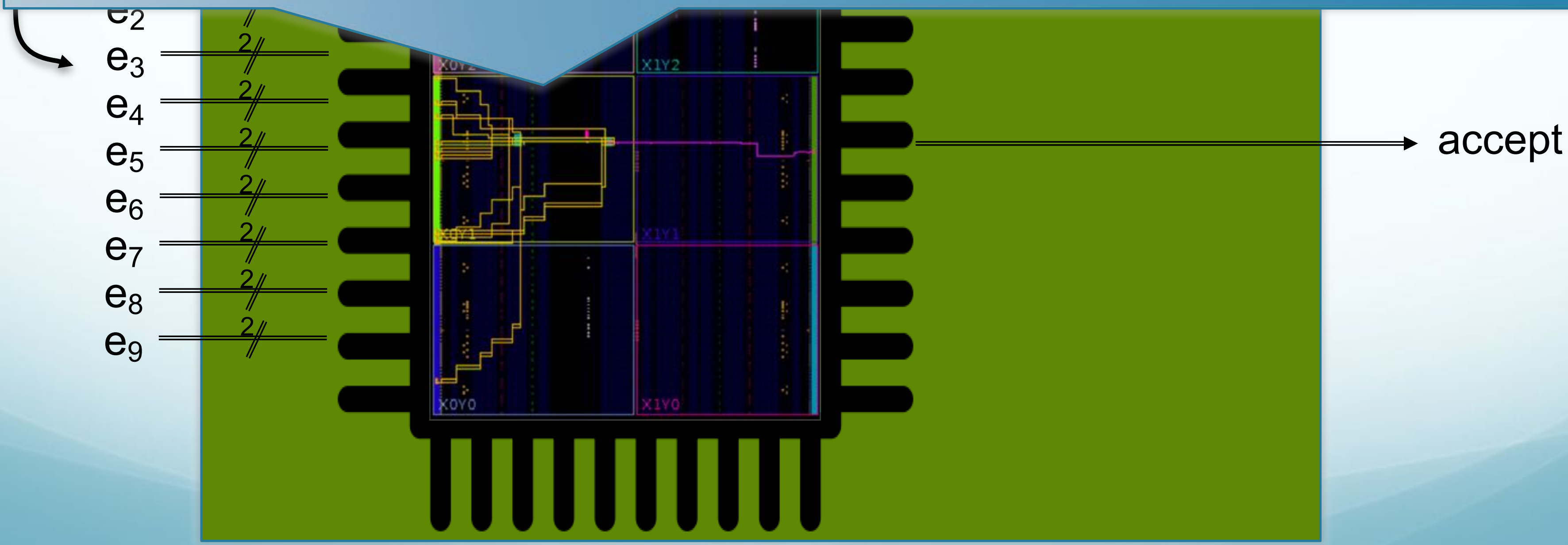


Toy example: RTL design

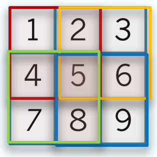
Again, we can look at the generated schematics:


energies[1][1:0]
energies[2][1:0]
energies[3][1:0]
energies[4][1:0]
energies[5][1:0]
energies[6][1:0]
energies[7][1:0]
energies[8][1:0]
energies[9][1:0]
check_cluster_energy1_i
check_cluster_energy1_i__2
check_cluster_energy0_i
check_cluster_energy1_i__0
check_cluster_energy1_i__3
check_cluster_energy1_i__1
check_cluster_energy0_i__0
check_cluster_energy_i
check_cluster_energy0_i__1
check_cluster_energy_i__0
check_cluster_energy1_i__4
check_cluster_energy_i__1
check_cluster_energy0_i__2
check_cluster_energy_i__2
accept2_i
accept1_i
accept0_i
RTL_ADD
RTL_GT
RTL_OR
accept

If we look closely, we can see that adders that are shared between adjacent 2x2 areas, are only implemented once.

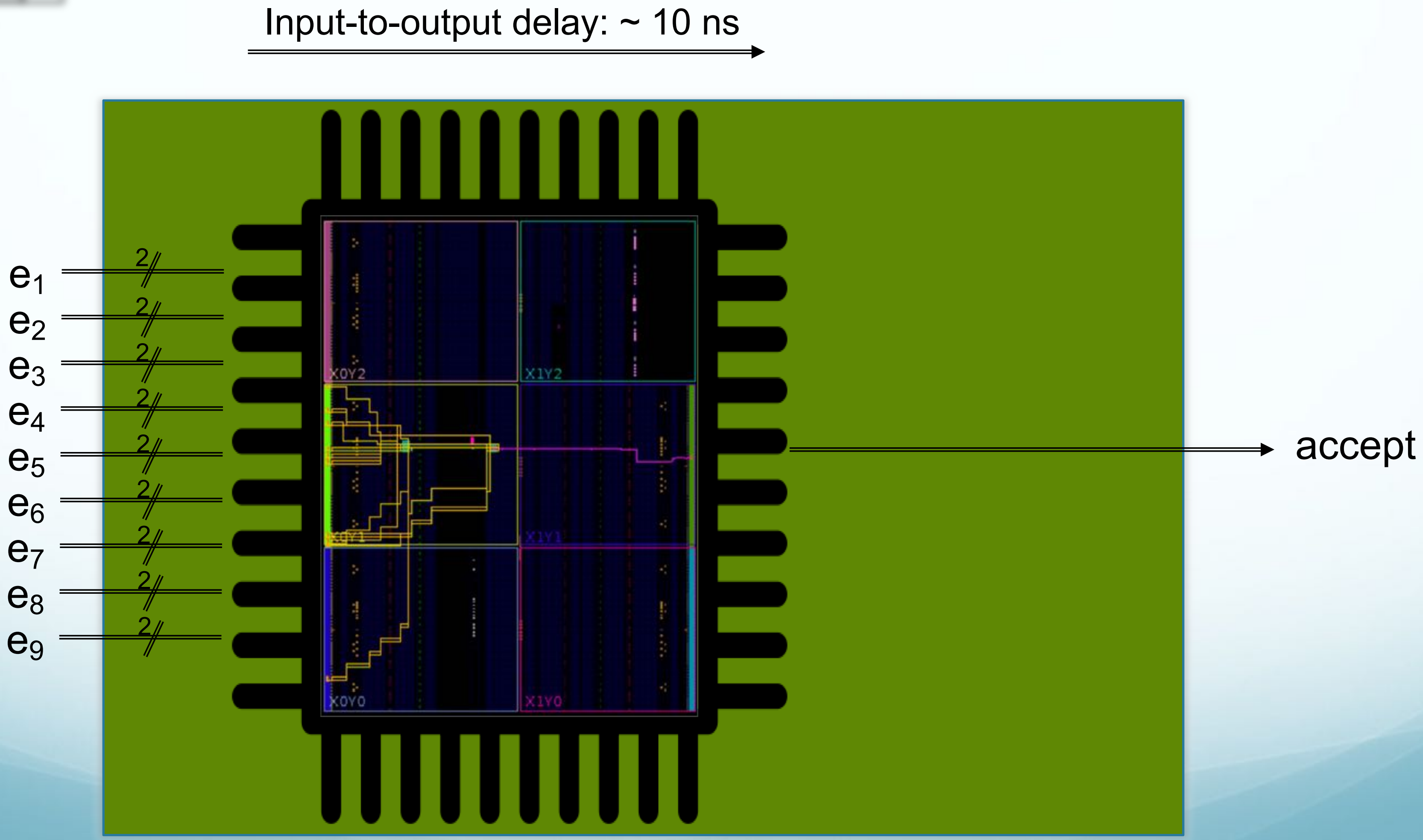


Toy example: Timing

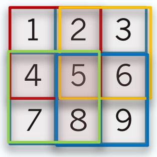


Timing:

In this example asynchronous design using a Xilinx Artix 7
(note that asynchronous designs are not how we typically use FPGAs)

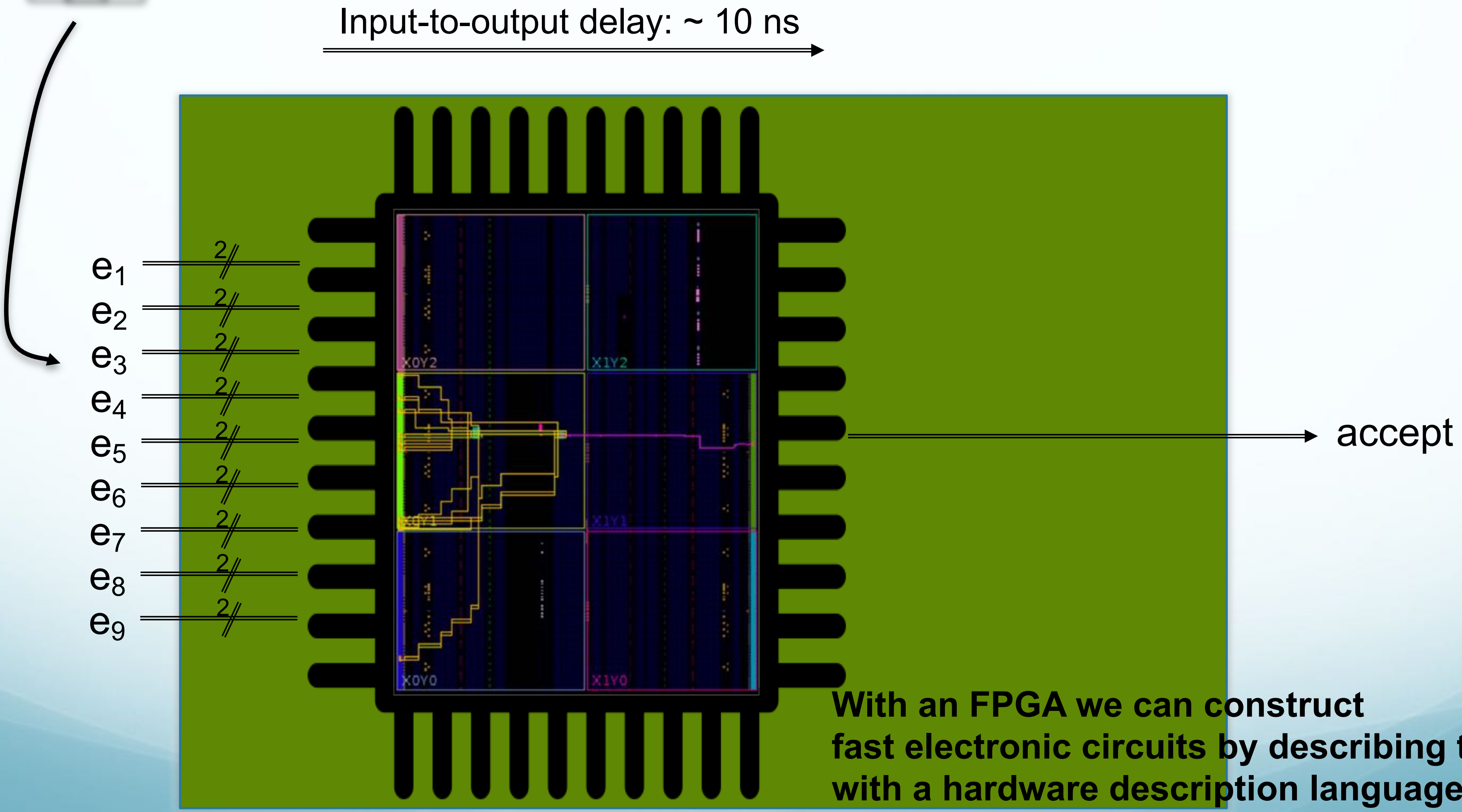


Toy example: Timing



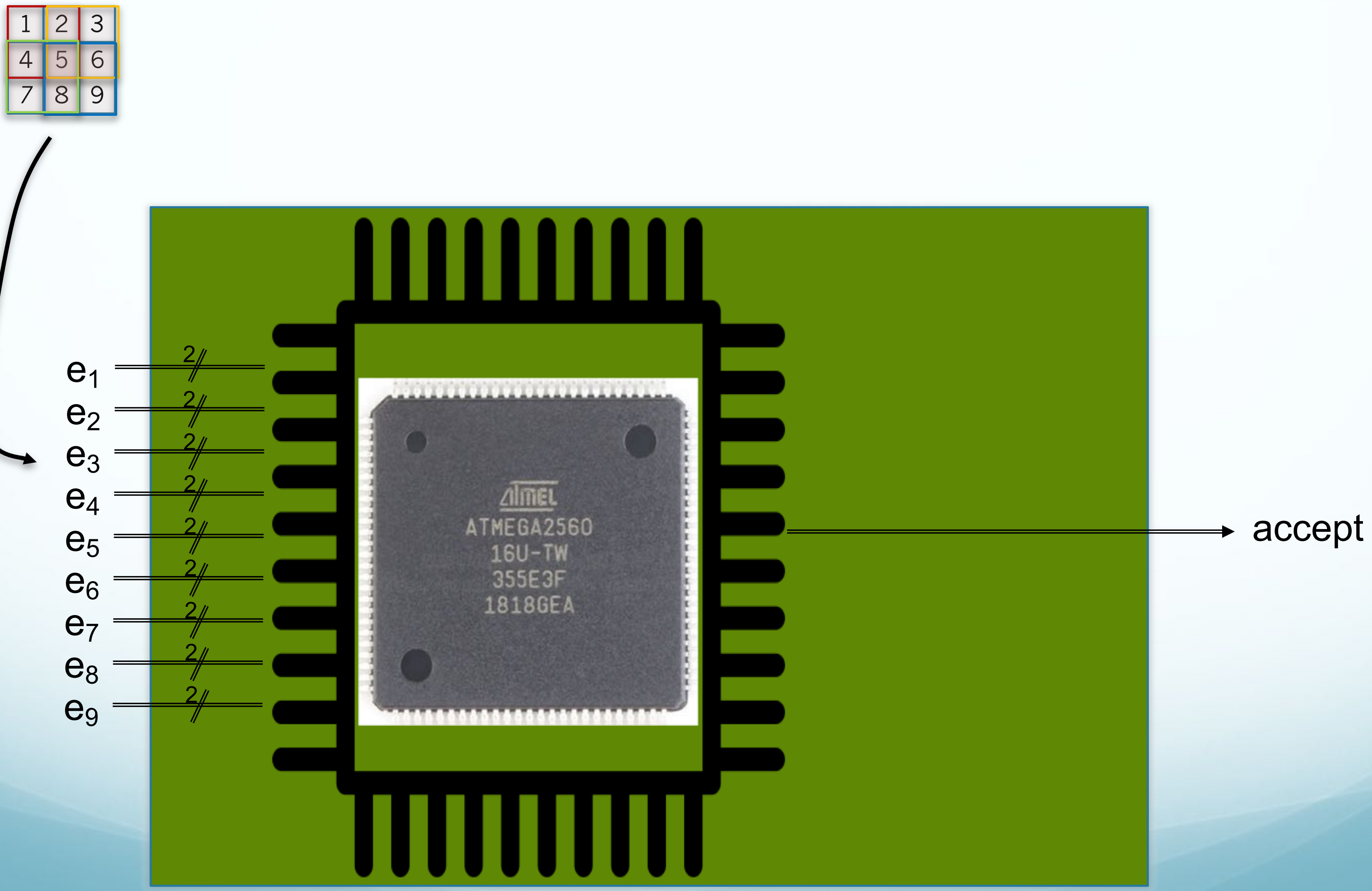
Timing:

In this example asynchronous design using a Xilinx Artix 7
(note that asynchronous designs are not how we typically use FPGAs)

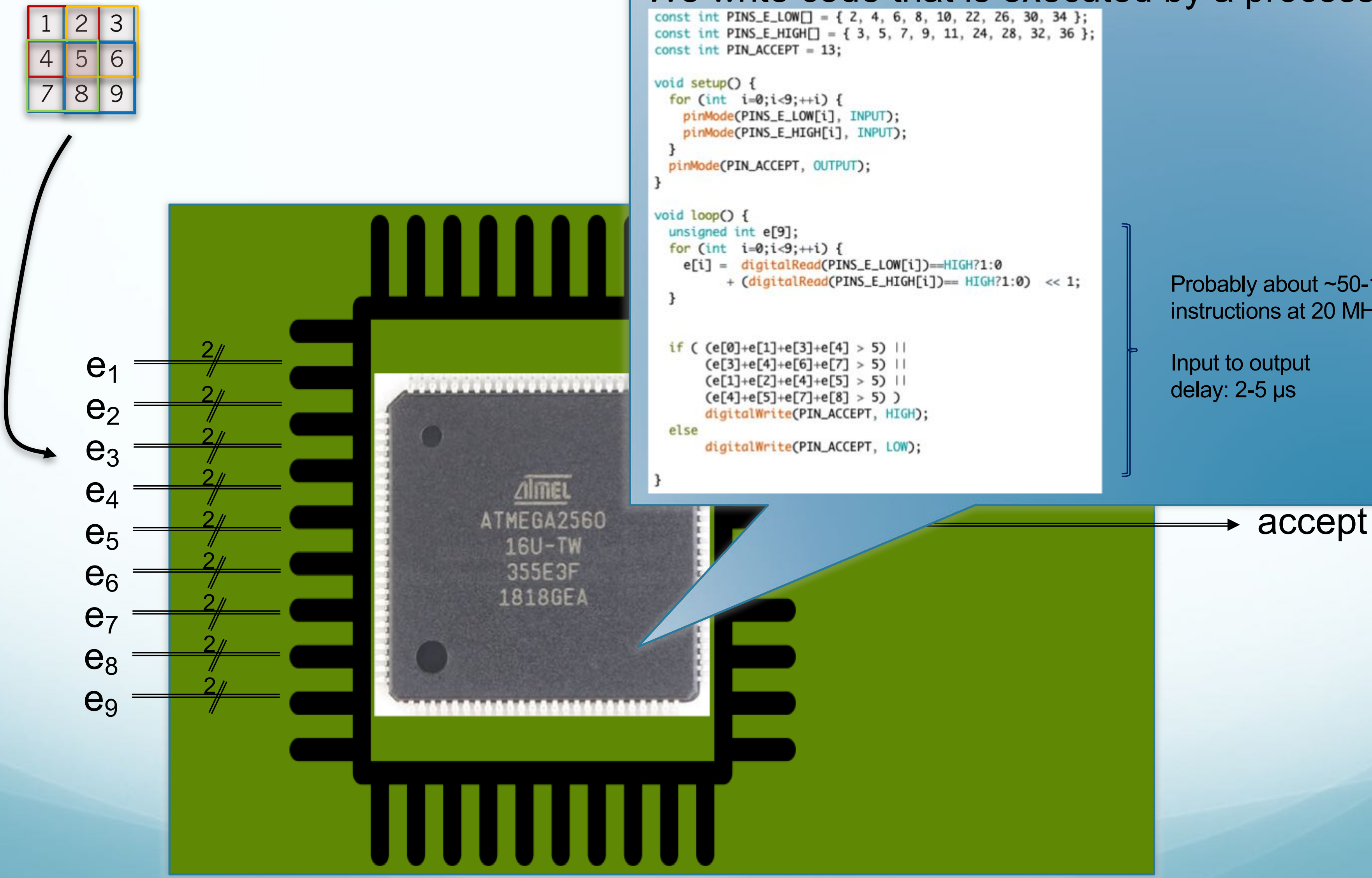


With an FPGA we can construct fast electronic circuits by describing them with a hardware description language.

Doing the same with a microcontroller



Doing the same with a microcontroller



We write code that is executed by a processor

```
const int PINS_E_LOW[] = { 2, 4, 6, 8, 10, 22, 26, 30, 34 };
const int PINS_E_HIGH[] = { 3, 5, 7, 9, 11, 24, 28, 32, 36 };
const int PIN_ACCEPT = 13;

void setup() {
  for (int  i=0;i<9;++i) {
    pinMode(PINS_E_LOW[i], INPUT);
    pinMode(PINS_E_HIGH[i], INPUT);
  }
  pinMode(PIN_ACCEPT, OUTPUT);
}

void loop() {
  unsigned int e[9];
  for (int  i=0;i<9;++i) {
    e[i] =  digitalRead(PINS_E_LOW[i])==HIGH?1:0
          + (digitalRead(PINS_E_HIGH[i])== HIGH?1:0)  << 1;
  }


  if ( (e[0]+e[1]+e[3]+e[4] > 5) ||
       (e[3]+e[4]+e[6]+e[7] > 5) ||
       (e[1]+e[2]+e[4]+e[5] > 5) ||
       (e[4]+e[5]+e[7]+e[8] > 5) )
       digitalWrite(PIN_ACCEPT, HIGH);
  else
       digitalWrite(PIN_ACCEPT, LOW);

}
```

Probably about ~50-100 instructions at 20 MHz

Input to output delay: 2-5 μs

Microcontroller vs FPGA

	µC / CPU	FPGA
Principle of operation	Source code is translated to machine instructions and executed by CPU	Hardware Description Language is translated to configuration of FPGA, defining an electronic circuit
Processing time	Microseconds	10’s of nanoseconds

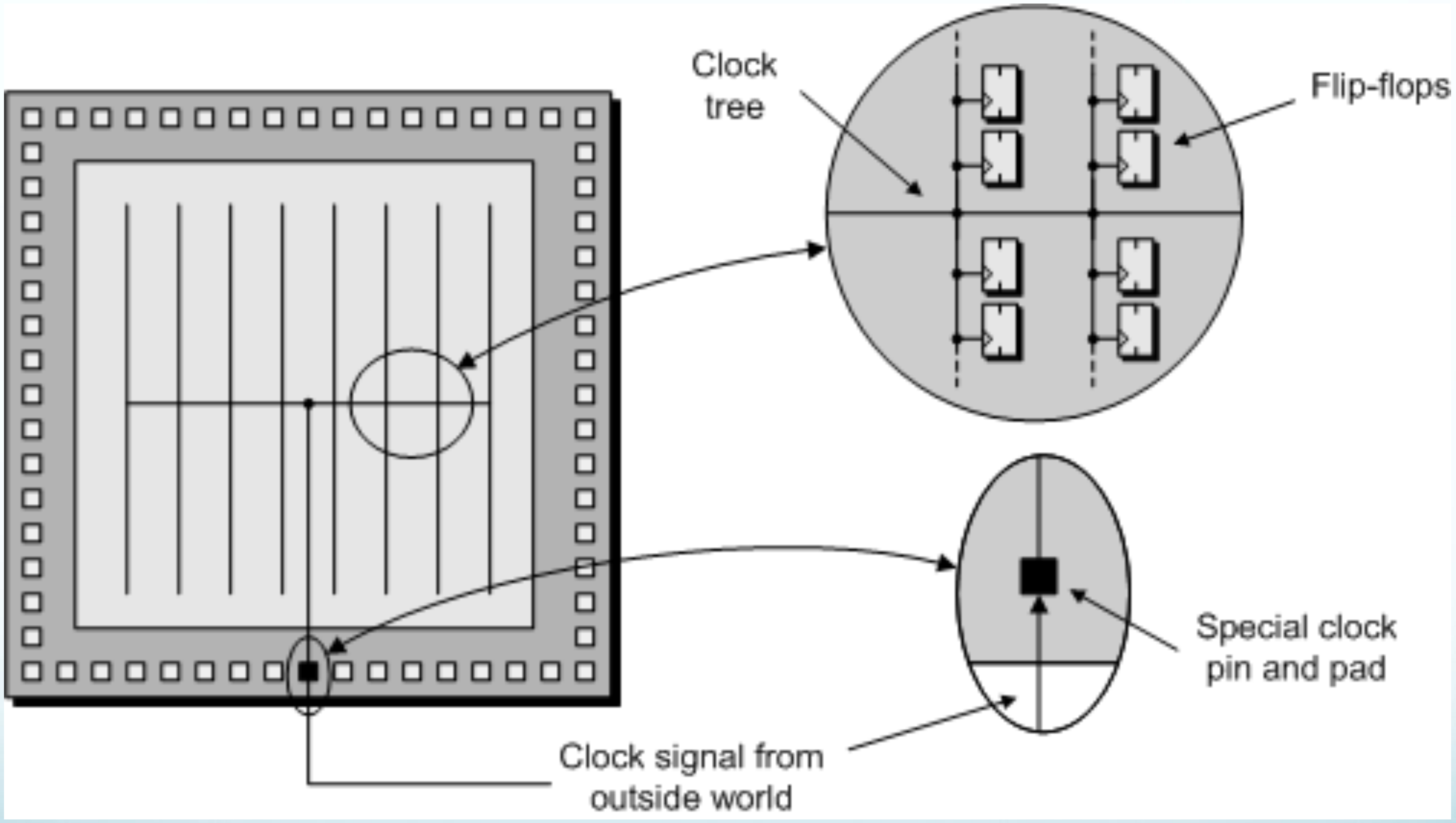
A closer look at FPGAs

Additional elements in an FPGA

- Besides logic cells and interconnect (distributed logic) we have additional elements in an FPGA:
  - Either to provide functions that cannot be implemented with distributed logic (because the logic would be too slow)
    - Clock resources, clock Managers
    - Gigabit transceivers
    - ...
  - Or to provide functionality that could also be implemented with distributed logic, but is more efficiently(*) implemented as a hard macro (in silicon)
    - Multipliers, DSPs
    - RAM
    - Processors

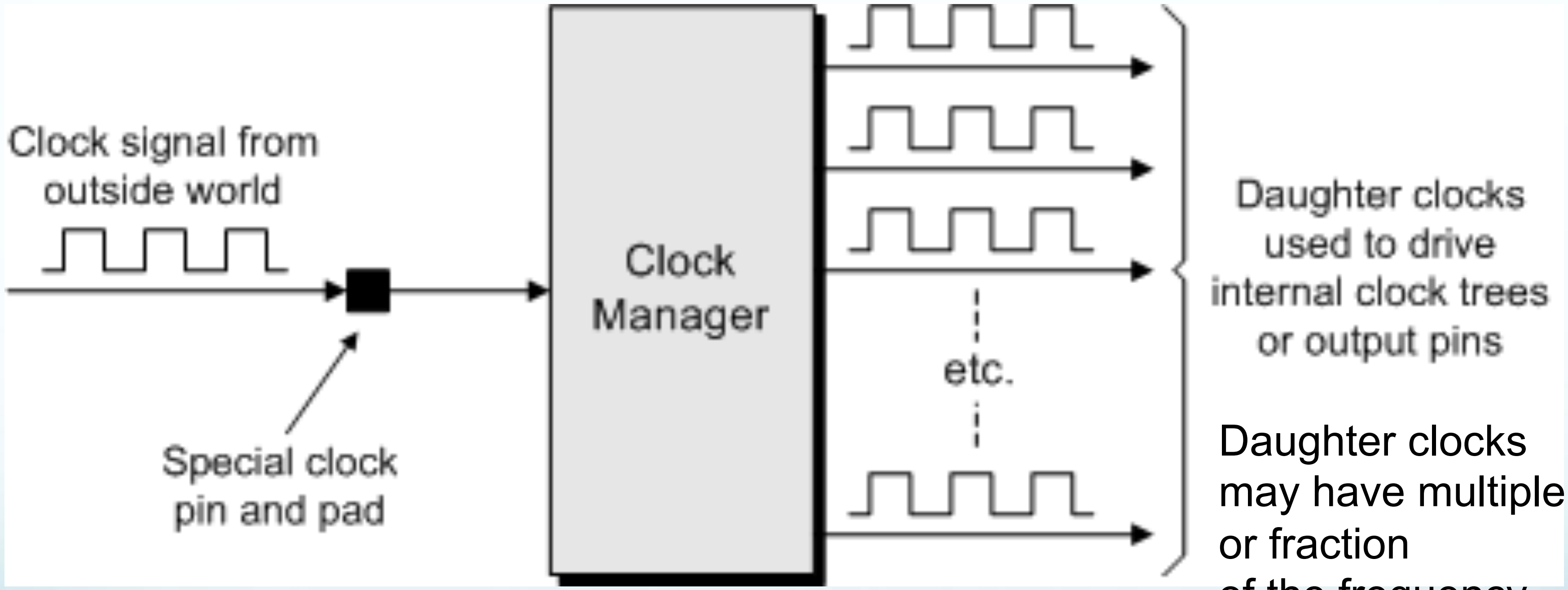
(*) smaller chip area, smaller power consumption, faster

Clock Trees



Typical FPGA designs use one or multiple clocks
Clock trees guarantee that the clock arrives at the same time at all flip-flops
Typical fabric clock 10’s to 100’s of MHz up to ~ 1 GHz

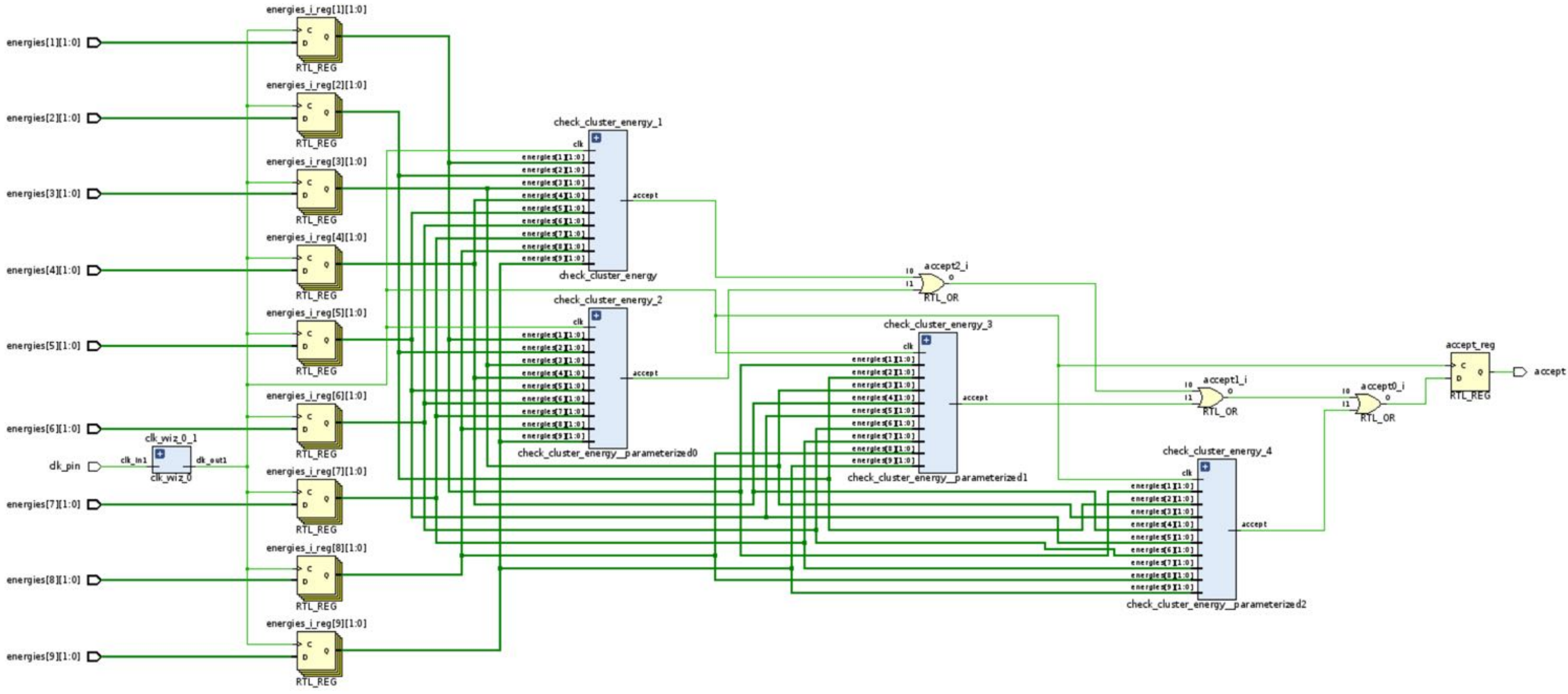
Clock Managers


Clock signal from outside world
Special clock pin and pad
Clock Manager
etc.
Daughter clocks used to drive internal clock trees or output pins

Daughter clocks may have multiple or fraction of the frequency

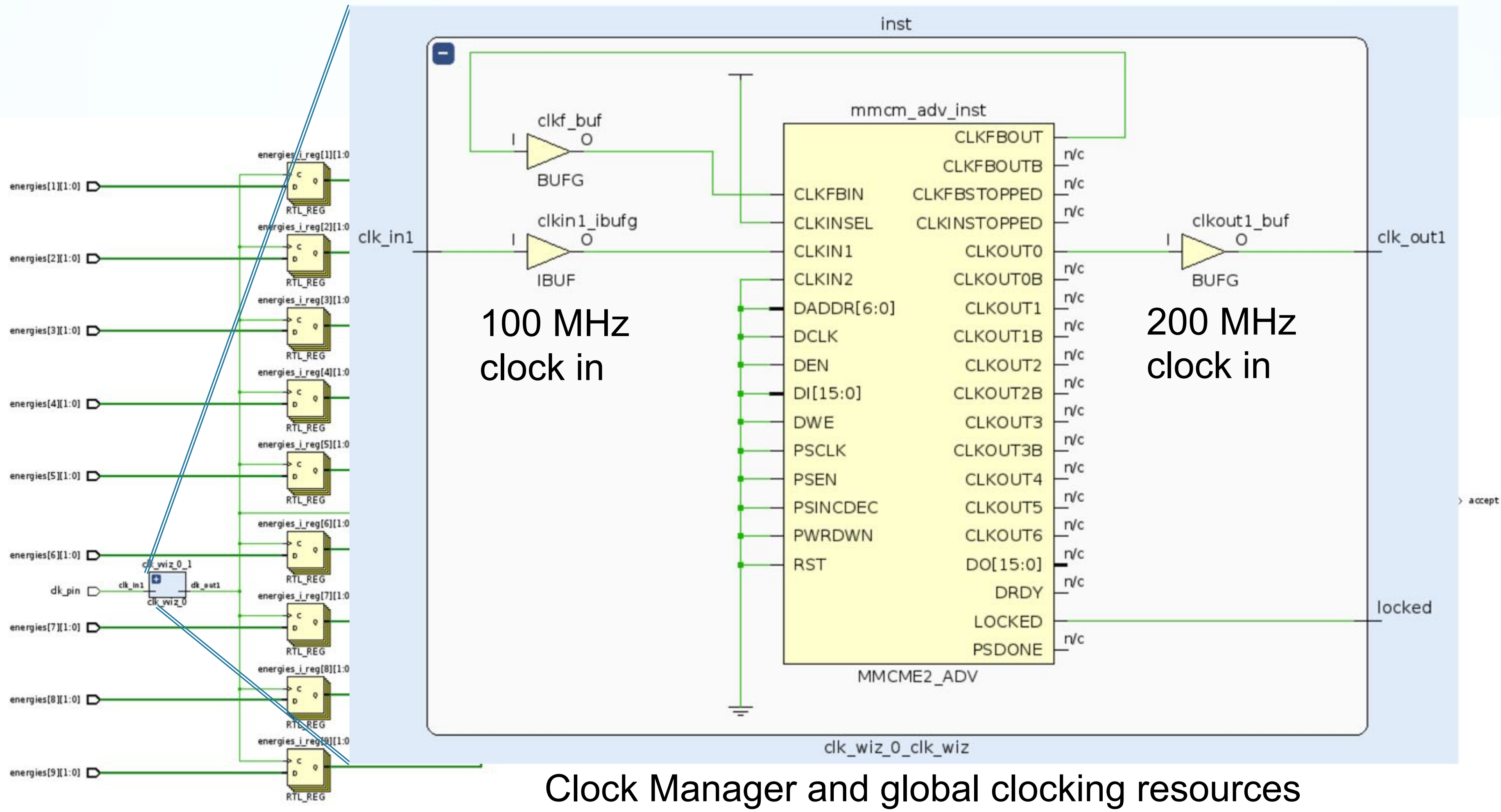
Our toy example with clock

Adding a clock and flip-flops



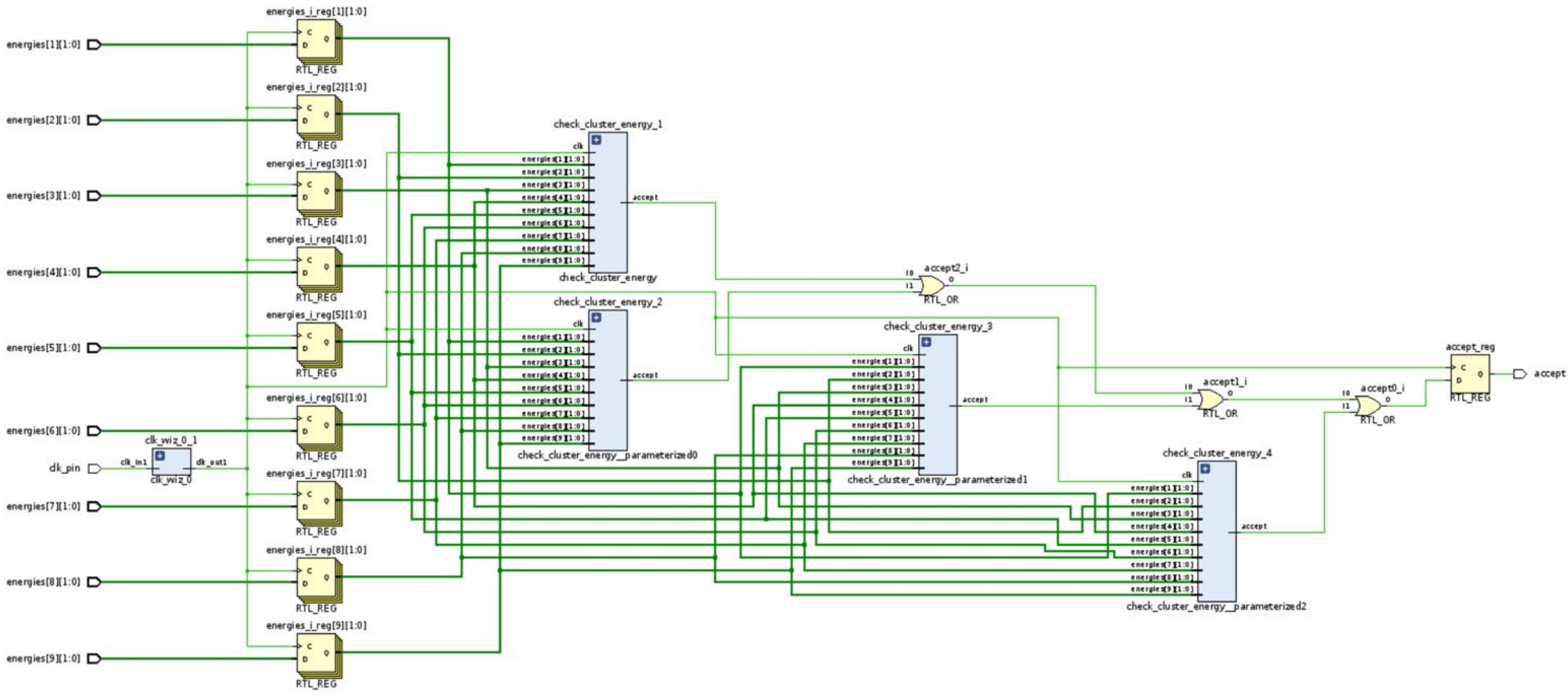
We transform our design into a pipeline with 3 processing stages

Adding a clock and flip-flops



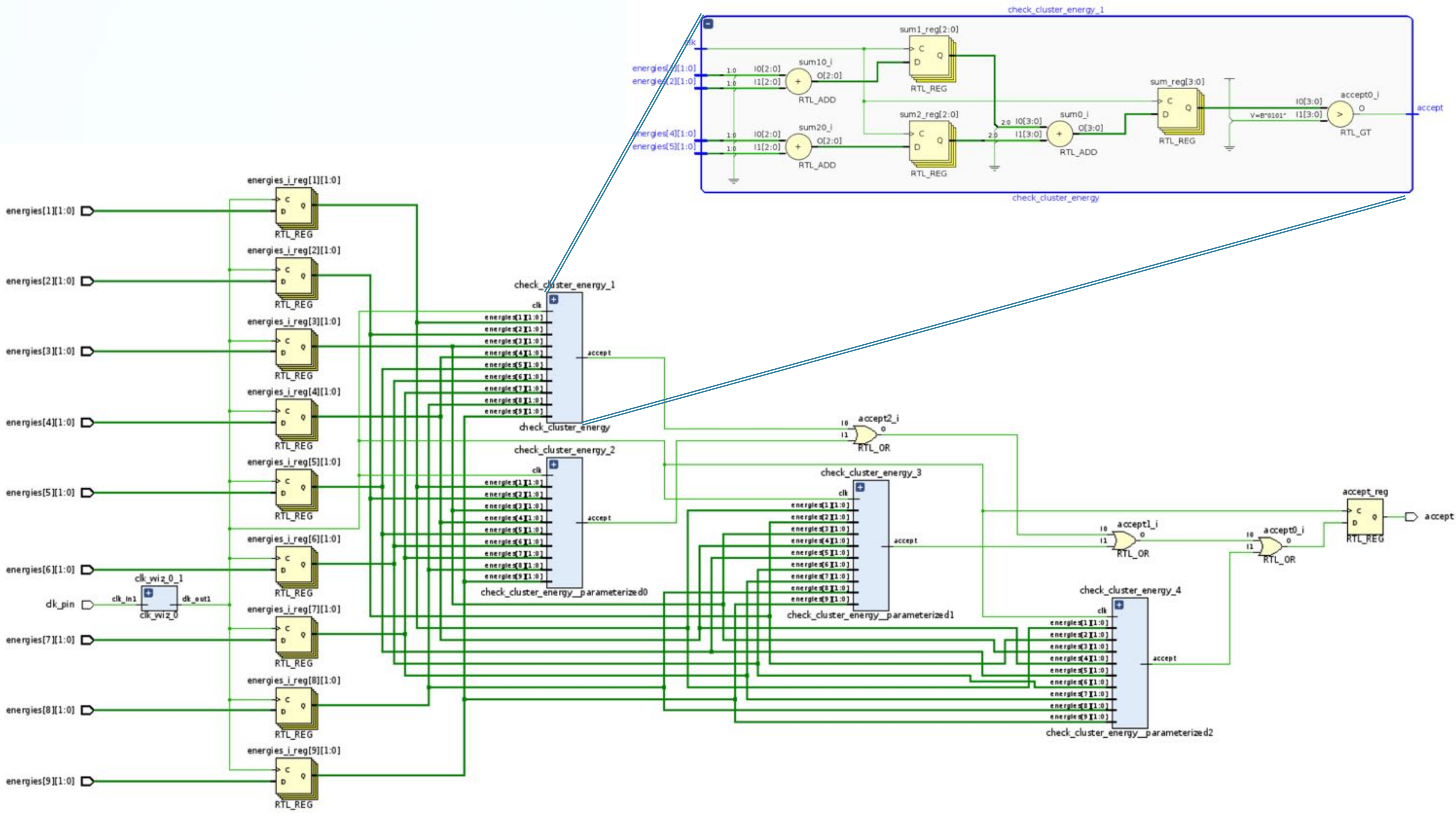
Clock Manager and global clocking resources

Adding a clock and flip-flops



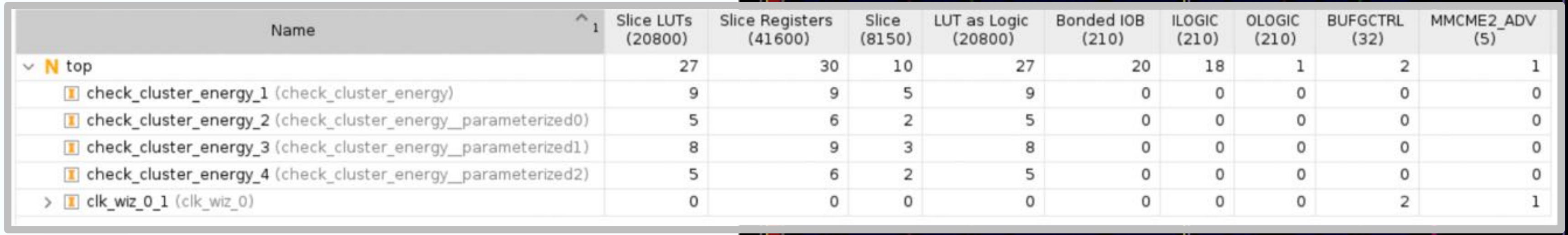
We transform our design into a pipeline with 3 processing stages

Adding a clock and flip-flops

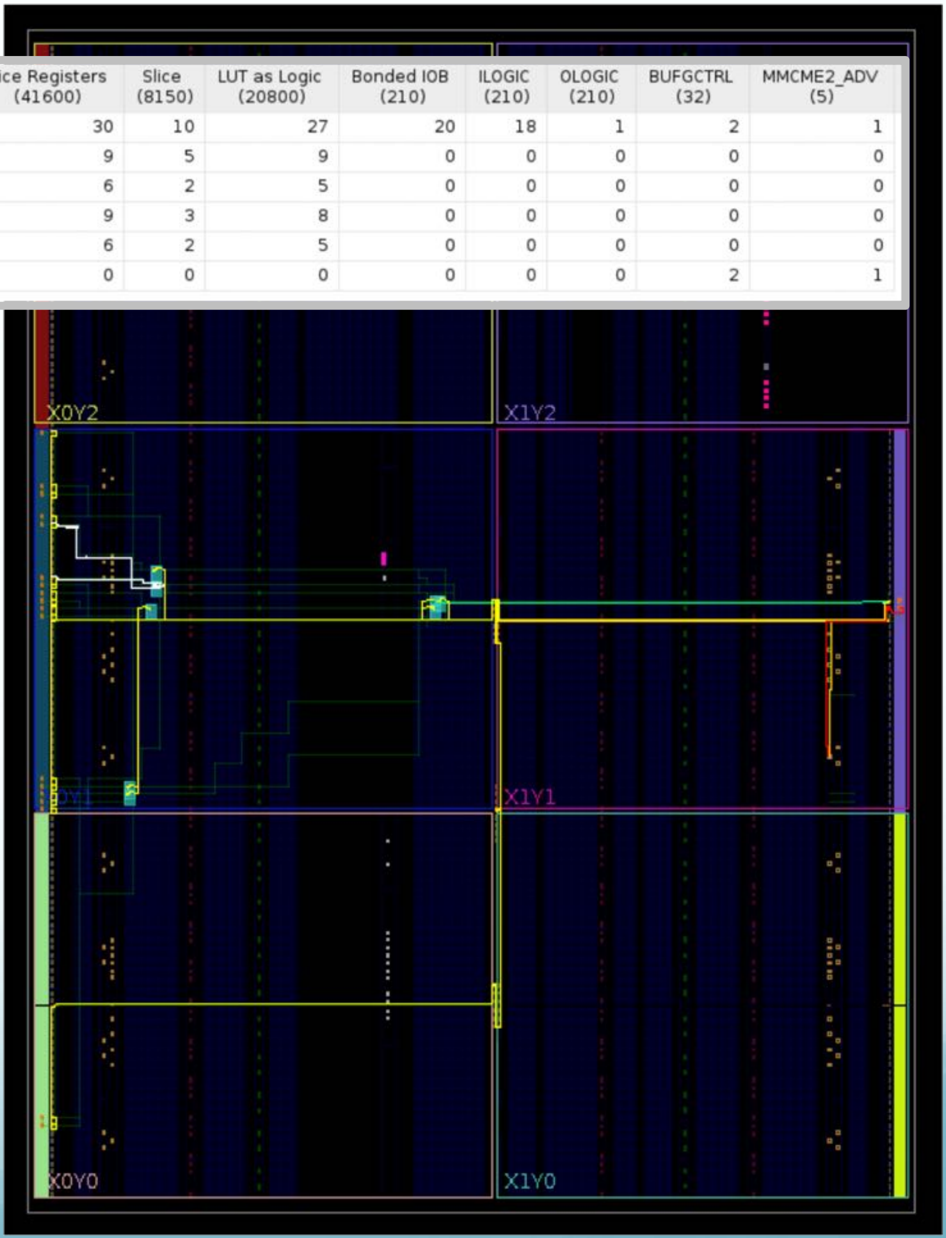


We transform our design into a pipeline with 3 processing stages

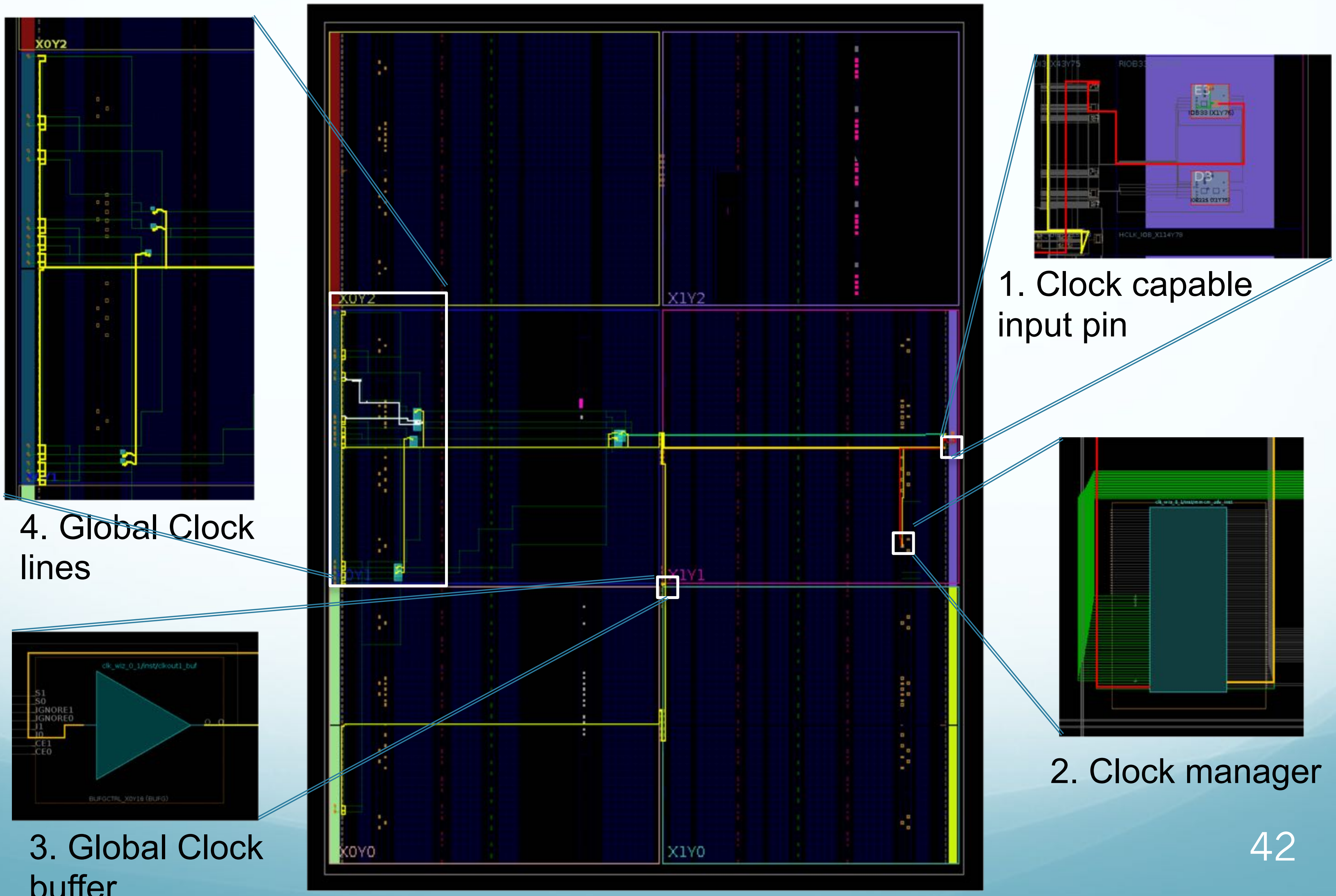
Resource usage & floorplan



Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (210)	ILOGIC (210)	OLOGIC (210)	BUFGCTRL (32)	MMCME2_ADV (5)
top	27	30	10	27	20	18	1	2	1
check_cluster_energy_1 (check_cluster_energy)	9	9	5	9	0	0	0	0	0
check_cluster_energy_2 (check_cluster_energy__parameterized0)	5	6	2	5	0	0	0	0	0
check_cluster_energy_3 (check_cluster_energy__parameterized1)	8	9	3	8	0	0	0	0	0
check_cluster_energy_4 (check_cluster_energy__parameterized2)	5	6	2	5	0	0	0	0	0
clk_wiz_0_1 (clk_wiz_0)	0	0	0	0	0	0	0	2	1



Floorplan: clock resources



1. Clock capable input pin

2. Clock manager

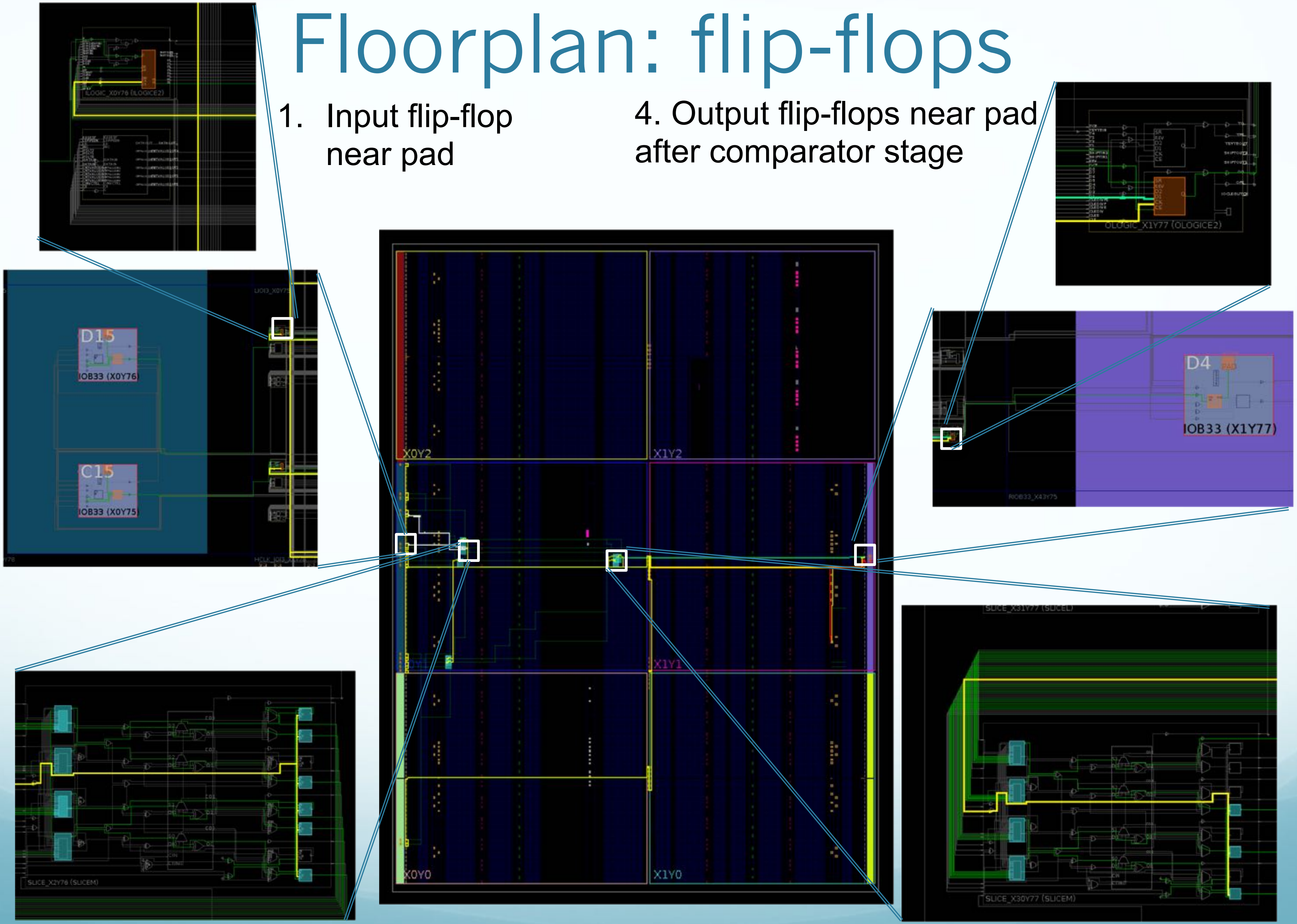
3. Global Clock buffer

4. Global Clock lines

Floorplan: flip-flops

1. Input flip-flop near pad

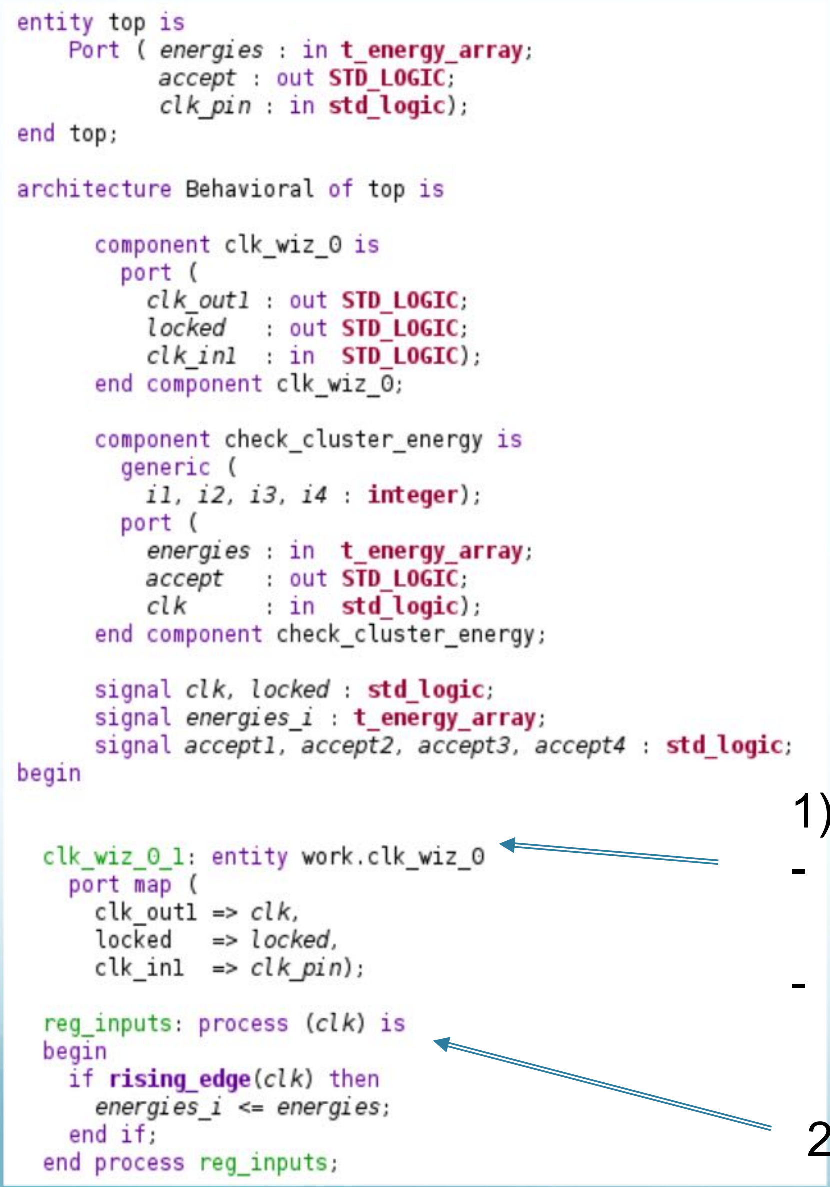
4. Output flip-flops near pad after comparator stage



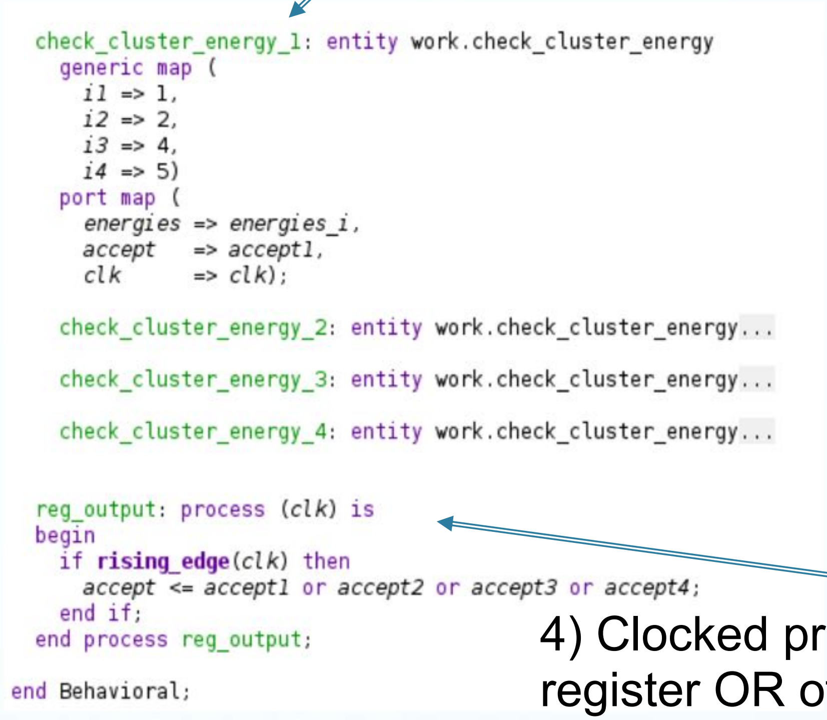
2. Flip-flops after first adder stage

3. Flip-flops after second adder stage

VHDL code



```
entity top is
    Port ( energies : in t_energy_array;
           accept : out STD_LOGIC;
           clk_pin : in std_logic);
end top;

architecture Behavioral of top is

      component clk_wiz_0 is
        port (
          clk_out1 : out STD_LOGIC;
          locked   : out STD_LOGIC;
          clk_in1  : in  STD_LOGIC);
      end component clk_wiz_0;

      component check_cluster_energy is
        generic (
          i1, i2, i3, i4 : integer);
        port (
          energies : in  t_energy_array;
          accept   : out STD_LOGIC;
          clk      : in  std_logic);
      end component check_cluster_energy;

      signal clk, locked : std_logic;
      signal energies_i : t_energy_array;
      signal accept1, accept2, accept3, accept4 : std_logic;
begin


  clk_wiz_0_1: entity work.clk_wiz_0
    port map (
      clk_out1 => clk,
      locked   => locked,
      clk_in1  => clk_pin);

  reg_inputs: process (clk) is
  begin
    if rising_edge(clk) then
      energies_i <= energies;
    end if;
  end process reg_inputs;
```



```
  check_cluster_energy_1: entity work.check_cluster_energy
    generic map (
      i1 => 1,
      i2 => 2,
      i3 => 4,
      i4 => 5)
    port map (
      energies => energies_i,
      accept   => accept1,
      clk      => clk);

    check_cluster_energy_2: entity work.check_cluster_energy...

    check_cluster_energy_3: entity work.check_cluster_energy...

    check_cluster_energy_4: entity work.check_cluster_energy...


  reg_output: process (clk) is
  begin
    if rising_edge(clk) then
      accept <= accept1 or accept2 or accept3 or accept4;
    end if;
  end process reg_output;

end Behavioral;
```

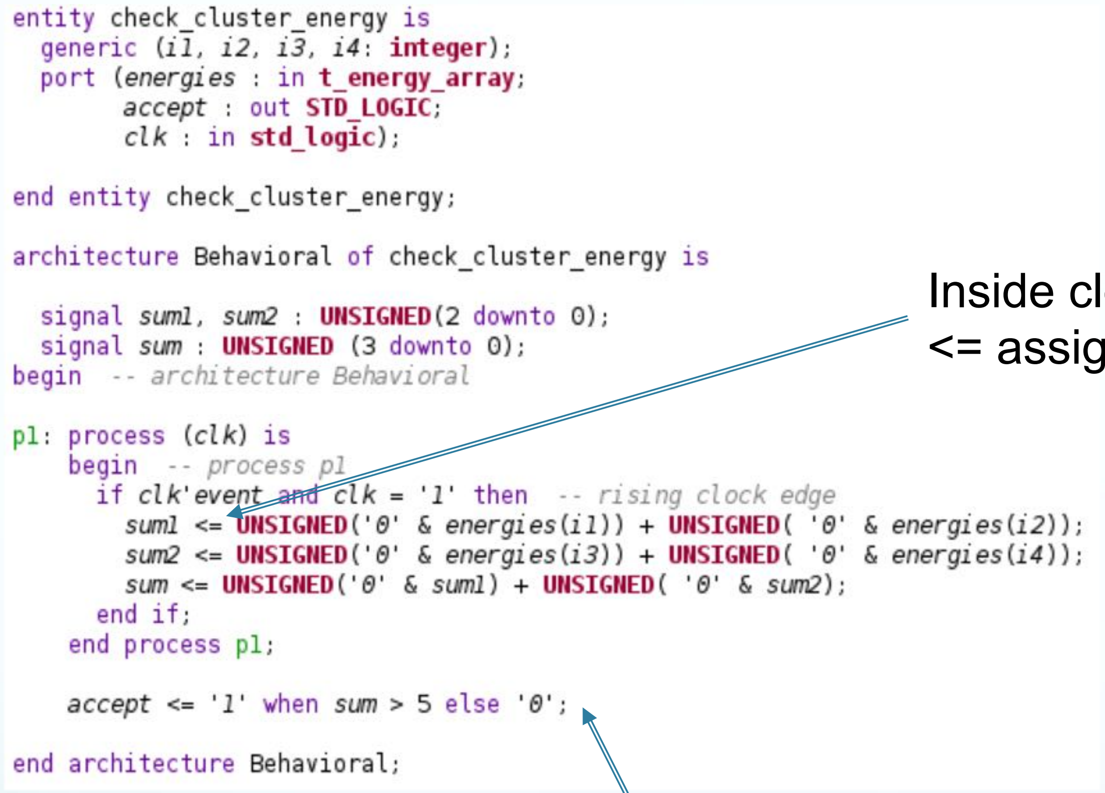
3) Instantiate cluster energy check module

4) Clocked process: register OR of Cluster checks

1) Instantiate clocking logic
- Customized firmware block produced by FPGA design tool
- Also called IP (Intellectual Property) Core

2) Clocked process: register inputs

VHDL code – cluster energy check

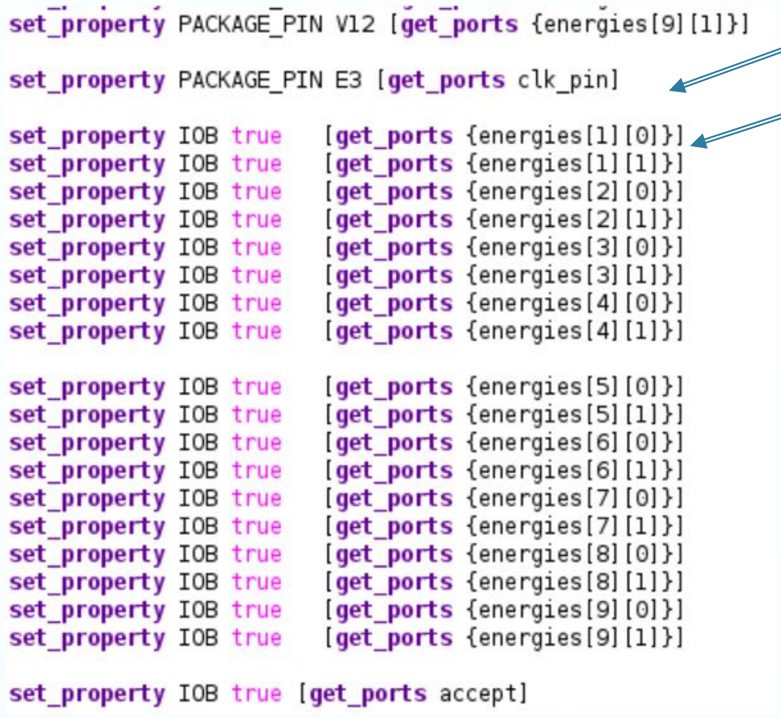


```
entity check_cluster_energy is
  generic (i1, i2, i3, i4: integer);
  port (energies : in t_energy_array;
        accept : out STD_LOGIC;
        clk : in std_logic);

end entity check_cluster_energy;

architecture Behavioral of check_cluster_energy is

  signal sum1, sum2 : UNSIGNED(2 downto 0);
  signal sum : UNSIGNED (3 downto 0);
begin  -- architecture Behavioral

p1: process (clk) is
    begin  -- process p1
      if clk'event and clk = '1' then  -- rising clock edge
        sum1 <= UNSIGNED('0' & energies(i1)) + UNSIGNED( '0' & energies(i2));
        sum2 <= UNSIGNED('0' & energies(i3)) + UNSIGNED( '0' & energies(i4));
        sum <= UNSIGNED('0' & sum1) + UNSIGNED( '0' & sum2);
      end if;
    end process p1;

    accept <= '1' when sum > 5 else '0';

end architecture Behavioral;
```

Inside clocked process:
<= assignment creates flip-flop

Outside process: asynchronous logic

Constraints

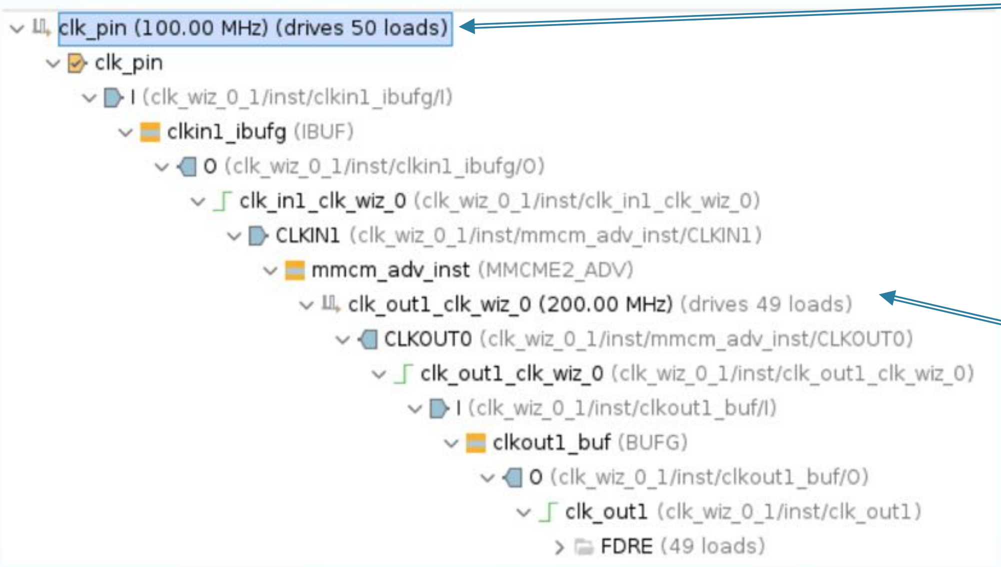


```
set_property PACKAGE_PIN V12 [get_ports {energies[9][1]}]

set_property PACKAGE_PIN E3 [get_ports clk_pin]

set_property IOB true    [get_ports {energies[1][0]}]
set_property IOB true    [get_ports {energies[1][1]}]
set_property IOB true    [get_ports {energies[2][0]}]
set_property IOB true    [get_ports {energies[2][1]}]
set_property IOB true    [get_ports {energies[3][0]}]
set_property IOB true    [get_ports {energies[3][1]}]
set_property IOB true    [get_ports {energies[4][0]}]
set_property IOB true    [get_ports {energies[4][1]}]

set_property IOB true    [get_ports {energies[5][0]}]
set_property IOB true    [get_ports {energies[5][1]}]
set_property IOB true    [get_ports {energies[6][0]}]
set_property IOB true    [get_ports {energies[6][1]}]
set_property IOB true    [get_ports {energies[7][0]}]
set_property IOB true    [get_ports {energies[7][1]}]
set_property IOB true    [get_ports {energies[8][0]}]
set_property IOB true    [get_ports {energies[8][1]}]
set_property IOB true    [get_ports {energies[9][0]}]
set_property IOB true    [get_ports {energies[9][1]}]

set_property IOB true [get_ports accept]
```

Assign clock pin

Use input/output flip-flops

Timing

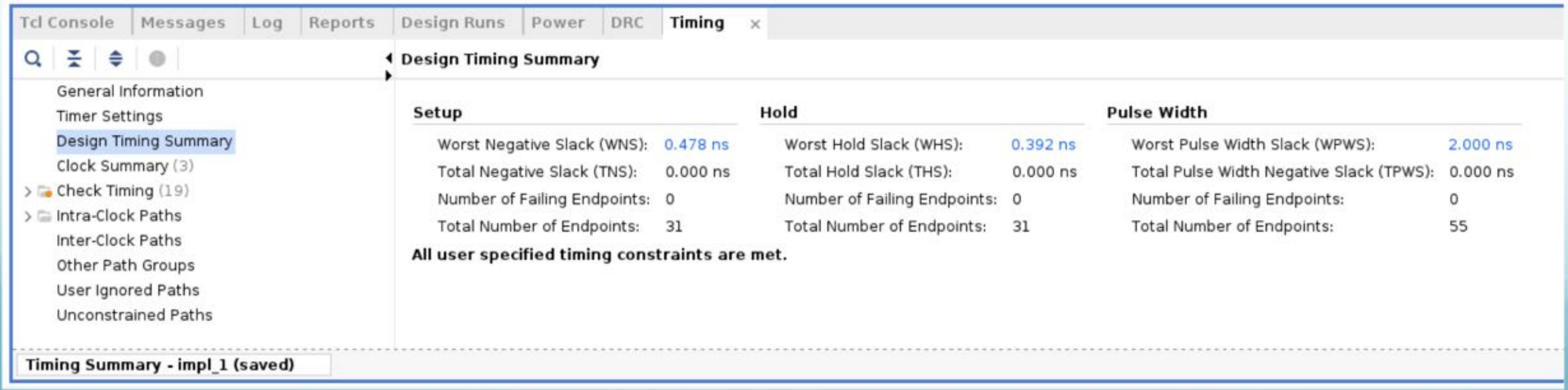


```
clk_pin (100.00 MHz) (drives 50 loads)
  clk_pin
    I (clk_wiz_0_1/inst/clkin1_ibufg/I)
      clkin1_ibufg (IBUF)
        O (clk_wiz_0_1/inst/clkin1_ibufg/O)
          clk_in1_clk_wiz_0 (clk_wiz_0_1/inst/clk_in1_clk_wiz_0)
            CLKIN1 (clk_wiz_0_1/inst/mmcm_adv_inst/CLKIN1)
              mmcm_adv_inst (MMCME2_ADV)
                clk_out1_clk_wiz_0 (200.00 MHz) (drives 49 loads)
                  CLKOUT0 (clk_wiz_0_1/inst/mmcm_adv_inst/CLKOUT0)
                    clk_out1_clk_wiz_0 (clk_wiz_0_1/inst/clk_out1_clk_wiz_0)
                      I (clk_wiz_0_1/inst/clkout1_buf/I)
                        clkout1_buf (BUFG)
                          O (clk_wiz_0_1/inst/clkout1_buf/O)
                            clk_out1 (clk_wiz_0_1/inst/clk_out1)
                              FDRE (49 loads)
```

100 MHz clock at pin

200 MHz internal clock (we set this as a parameter to the clock manager)

Design can check for clusters at 200 MHz (every 5 ns),
but needs 4 clock cycles (20 ns) to compute the trigger decision



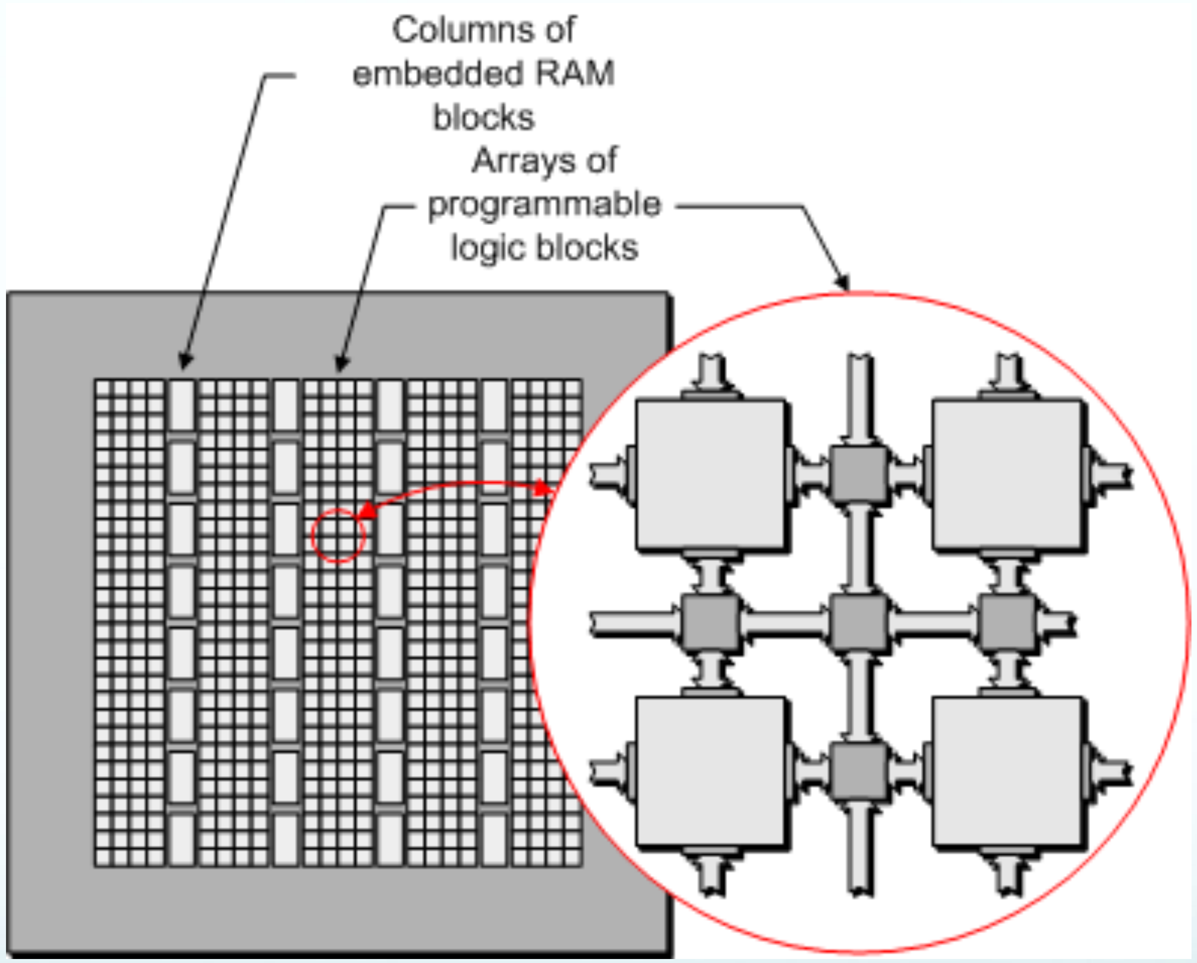
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.478 ns	Worst Hold Slack (WHS):	0.392 ns	Worst Pulse Width Slack (WPWS):	2.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	31	Total Number of Endpoints:	31	Total Number of Endpoints:	55

All user specified timing constraints are met.

Other elements in FPGAs

Embedded RAM blocks

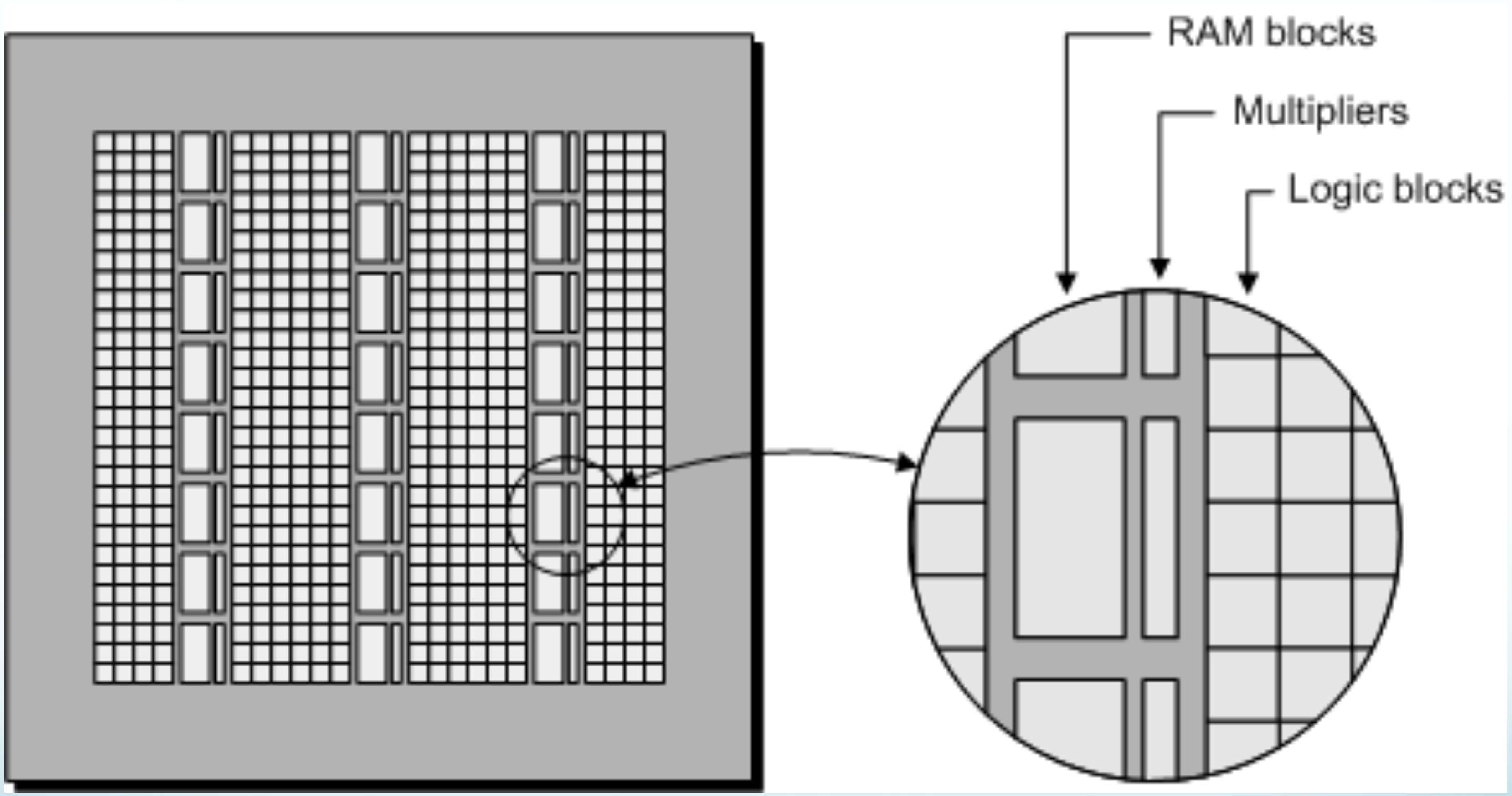

Columns of embedded RAM blocks
Arrays of programmable logic blocks

Can be used in many ways:

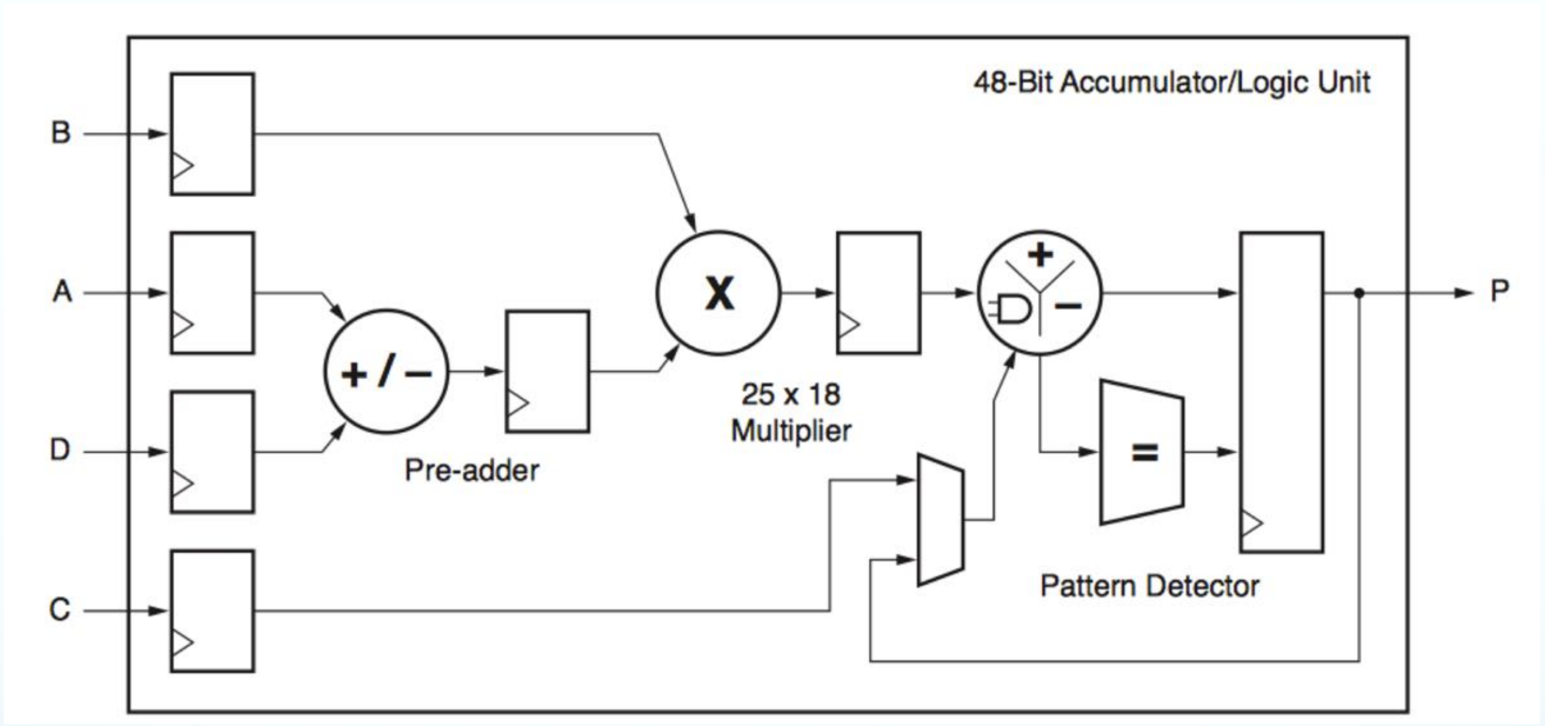
- Look-up of mathematical function
- Buffer memory
- …

Today: Up to ~500 Mbit of RAM

Embedded Multipliers & DSPs



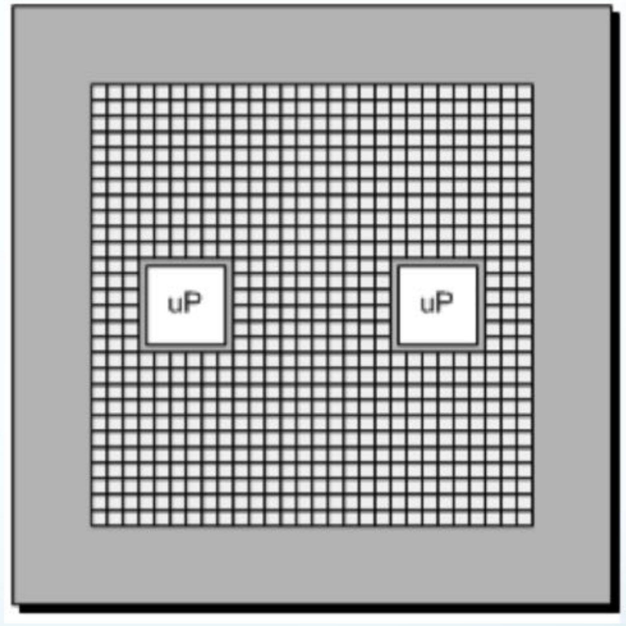
Digital Signal Processor (DSP)



DSP block (Xilinx 7-series)
Up to several 1000 per chip

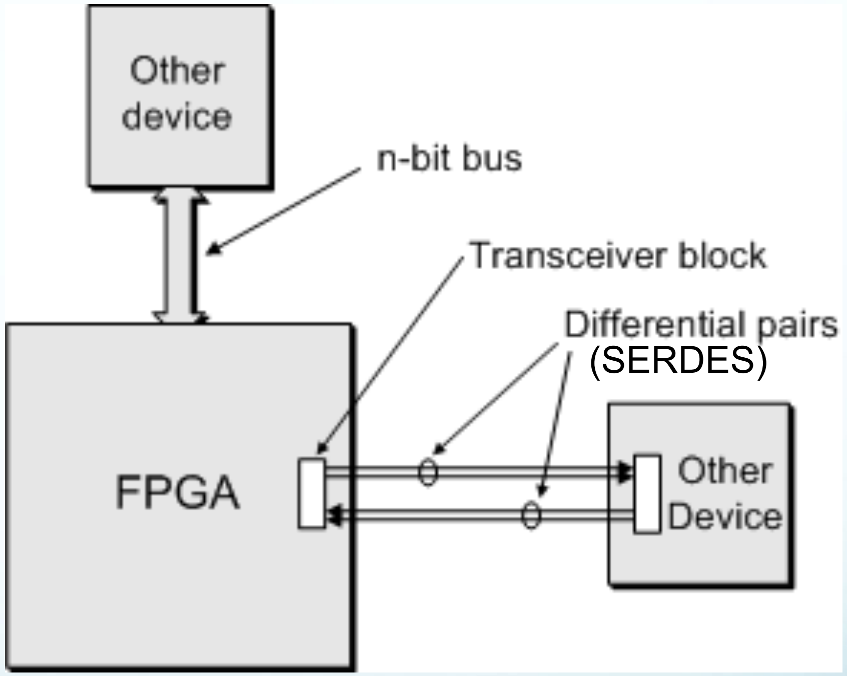
Soft and Hard Processor Cores

- Soft core
  - Design implemented with the programmable resources (logic cells) in the chip
- Hard core
  - Processor core that is available in addition to the programmable resources
  - E.g.: Power PC, ARM

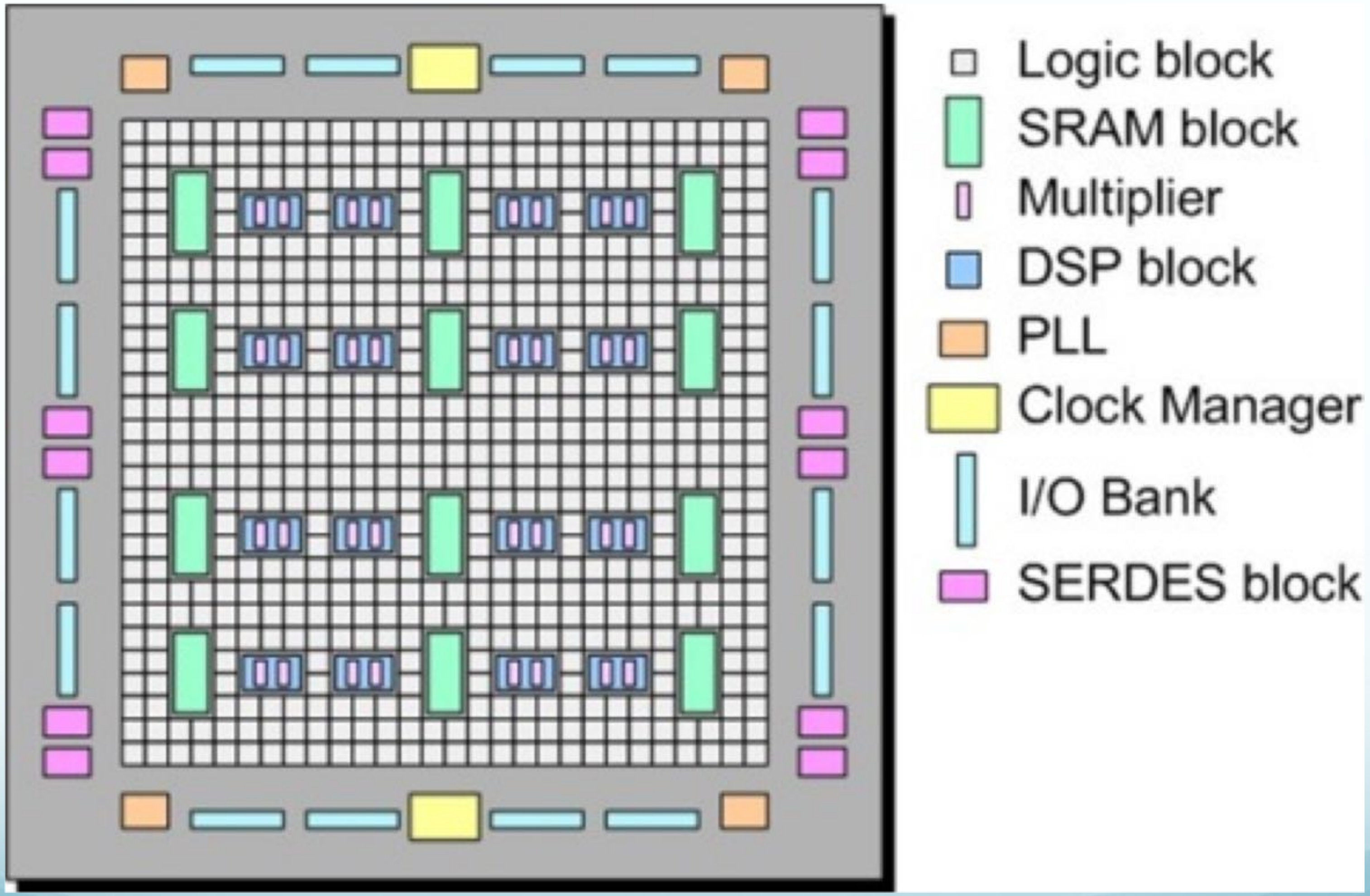


High-Speed Serial Interconnect

- Using differential pairs
- Standard I/O pins limited to about 1 Gbit/s
- Latest serial transceivers: typically 25 Gb/s
  - up to 112 Gb/s with Pulse Amplitude Modulation (PAM)
- FPGAs with multi-Tbit/s IO bandwidth

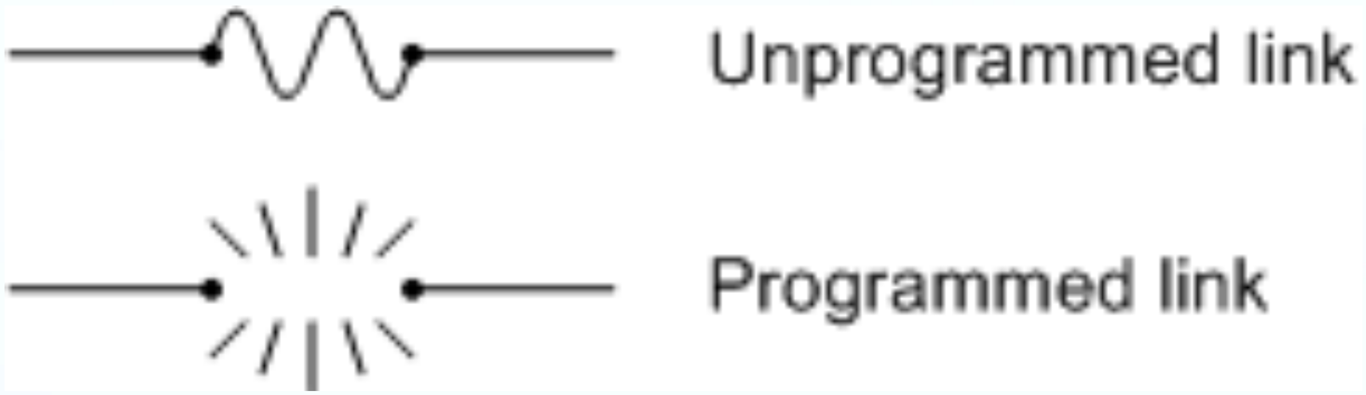


Components in a modern FPGA

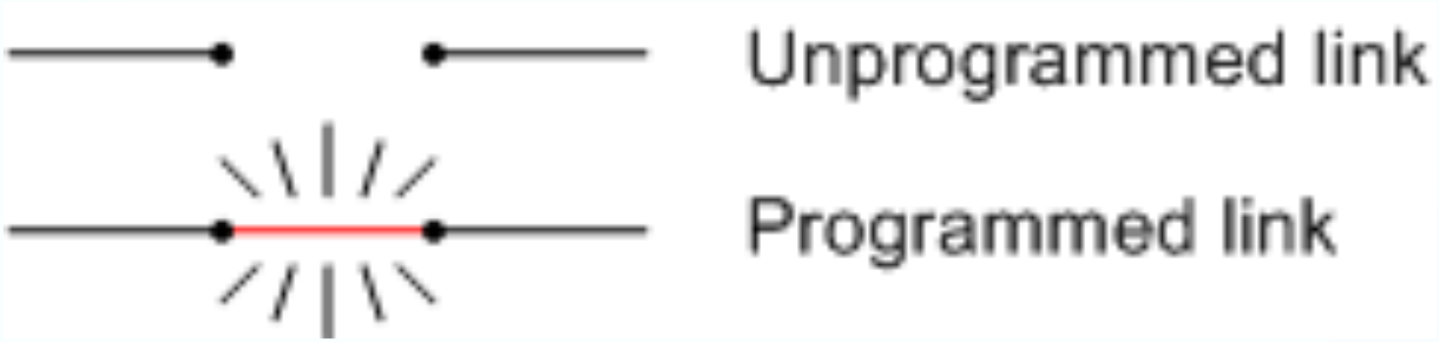


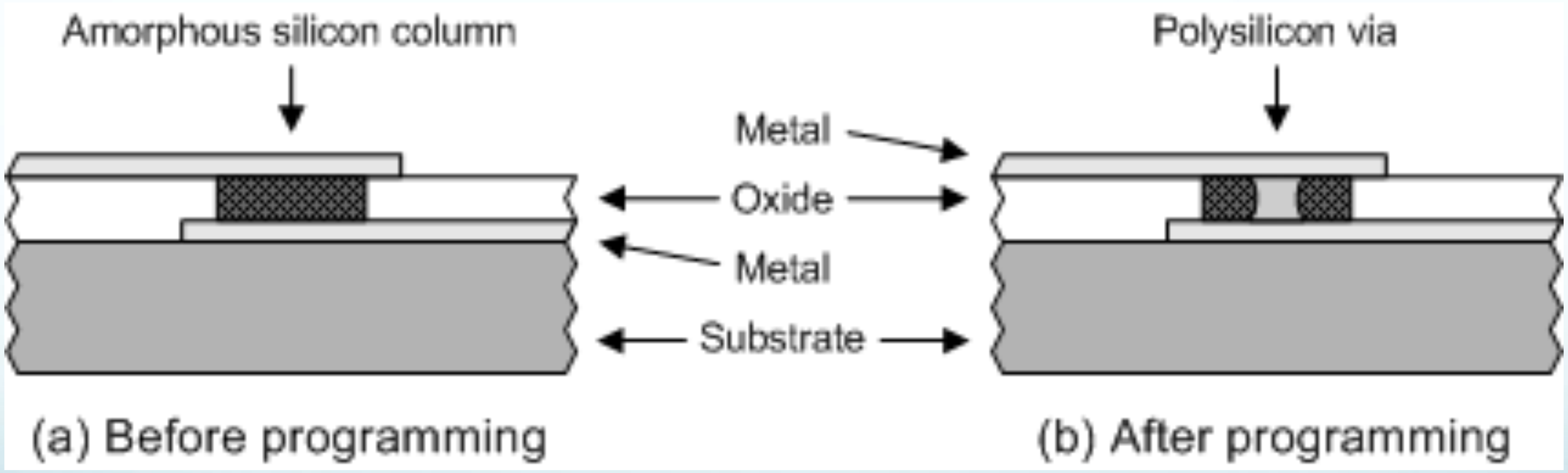
Programming techniques

Fusible Links (not used in FPGAs)



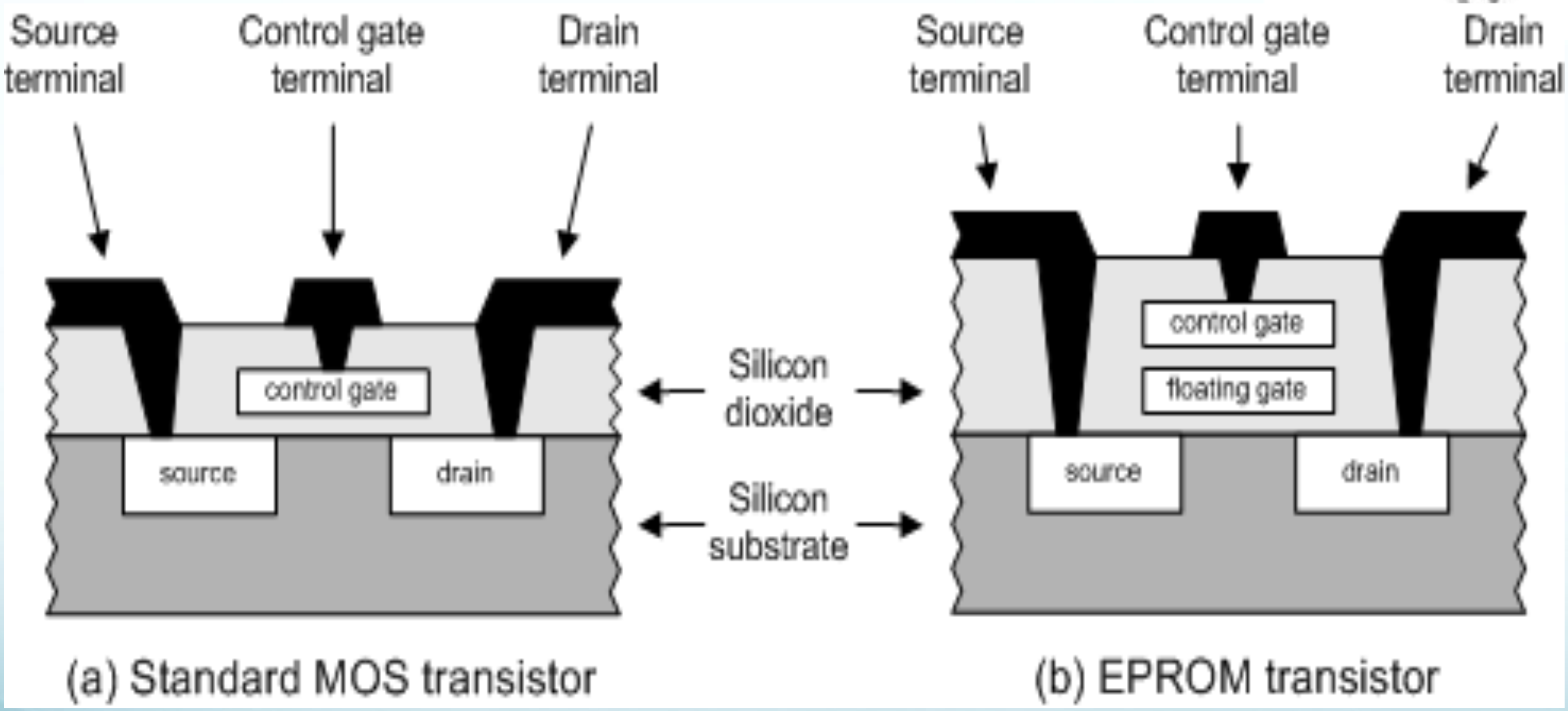
Antifuse Technology





EPROM Technology

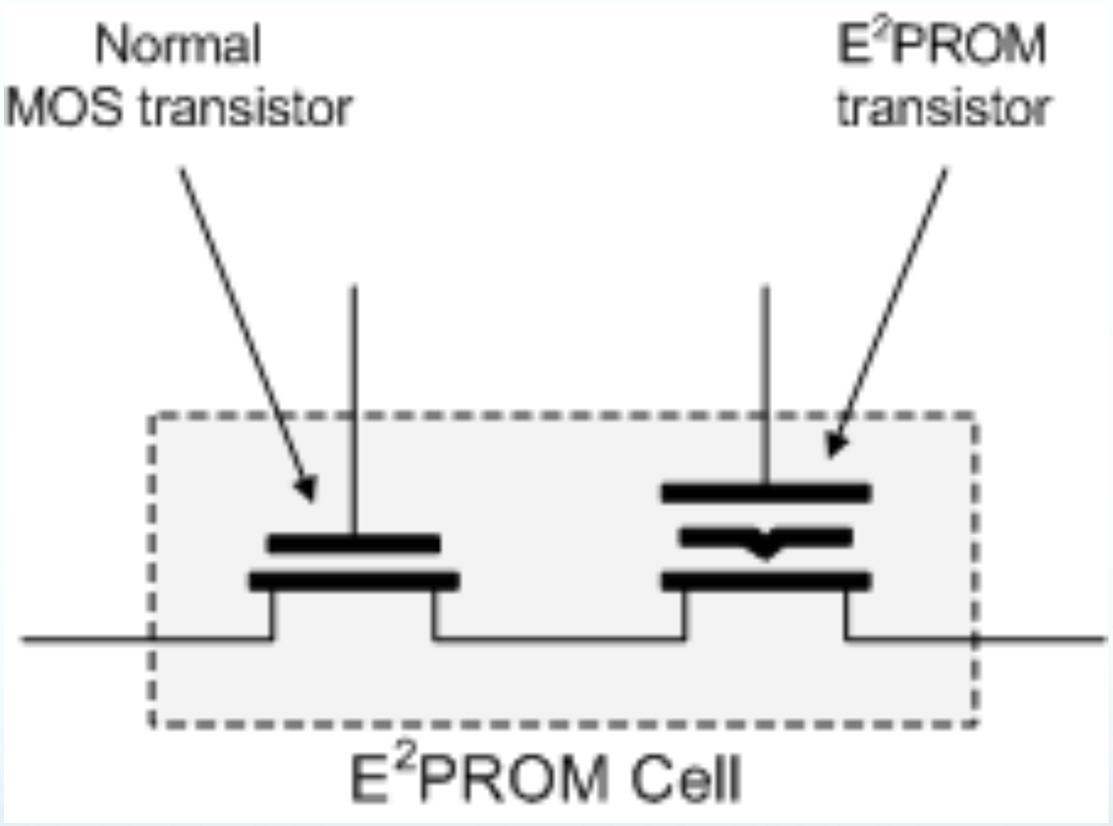
Erasable Programmable Read Only Memory



Intel, 1971

EEPROM and FLASH Technology

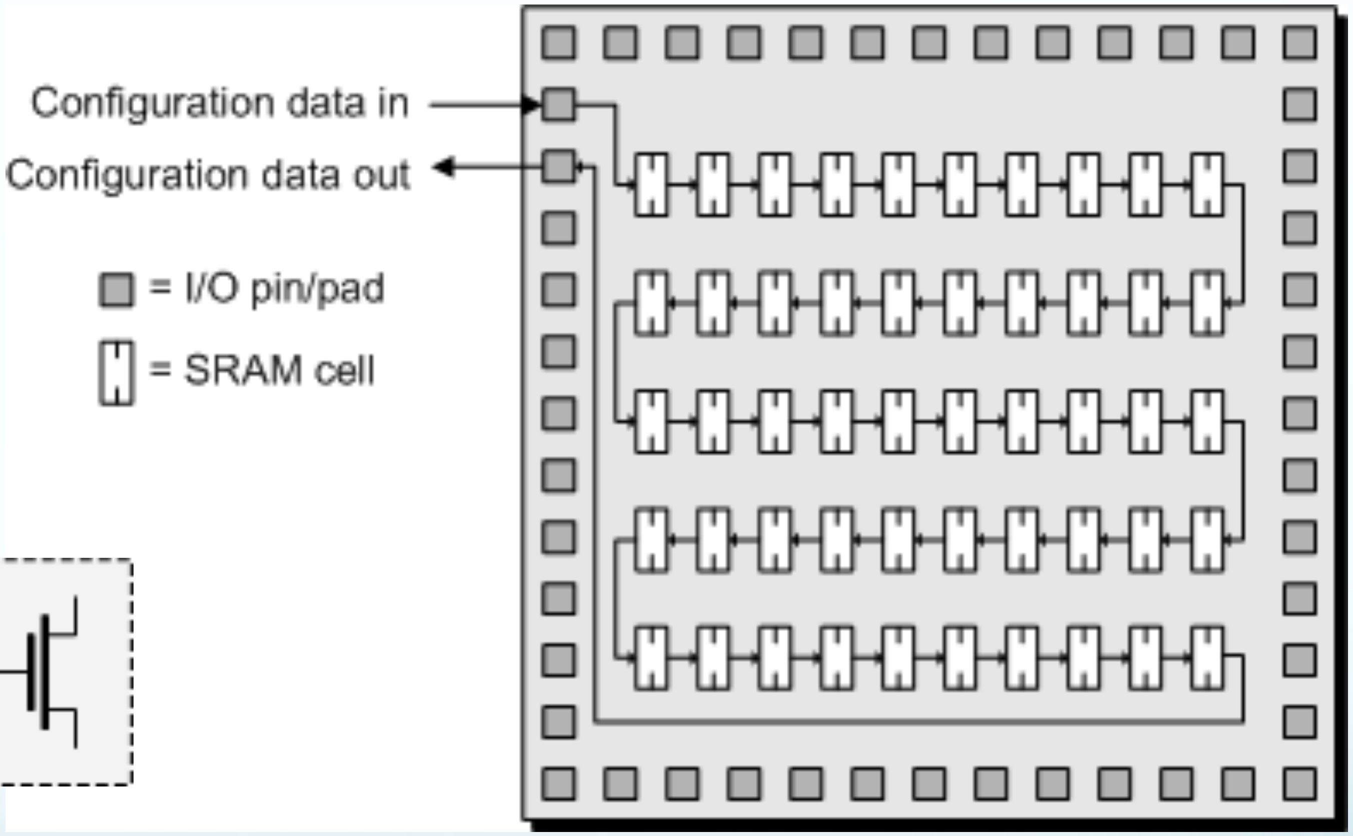
Electrically Erasable Programmable Read Only Memory

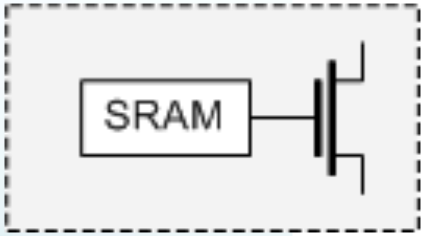


EEPROM: erasable word by word
FLASH: erasable by block or by device

Erasure by “Fowler-Nordheim” Tunneling

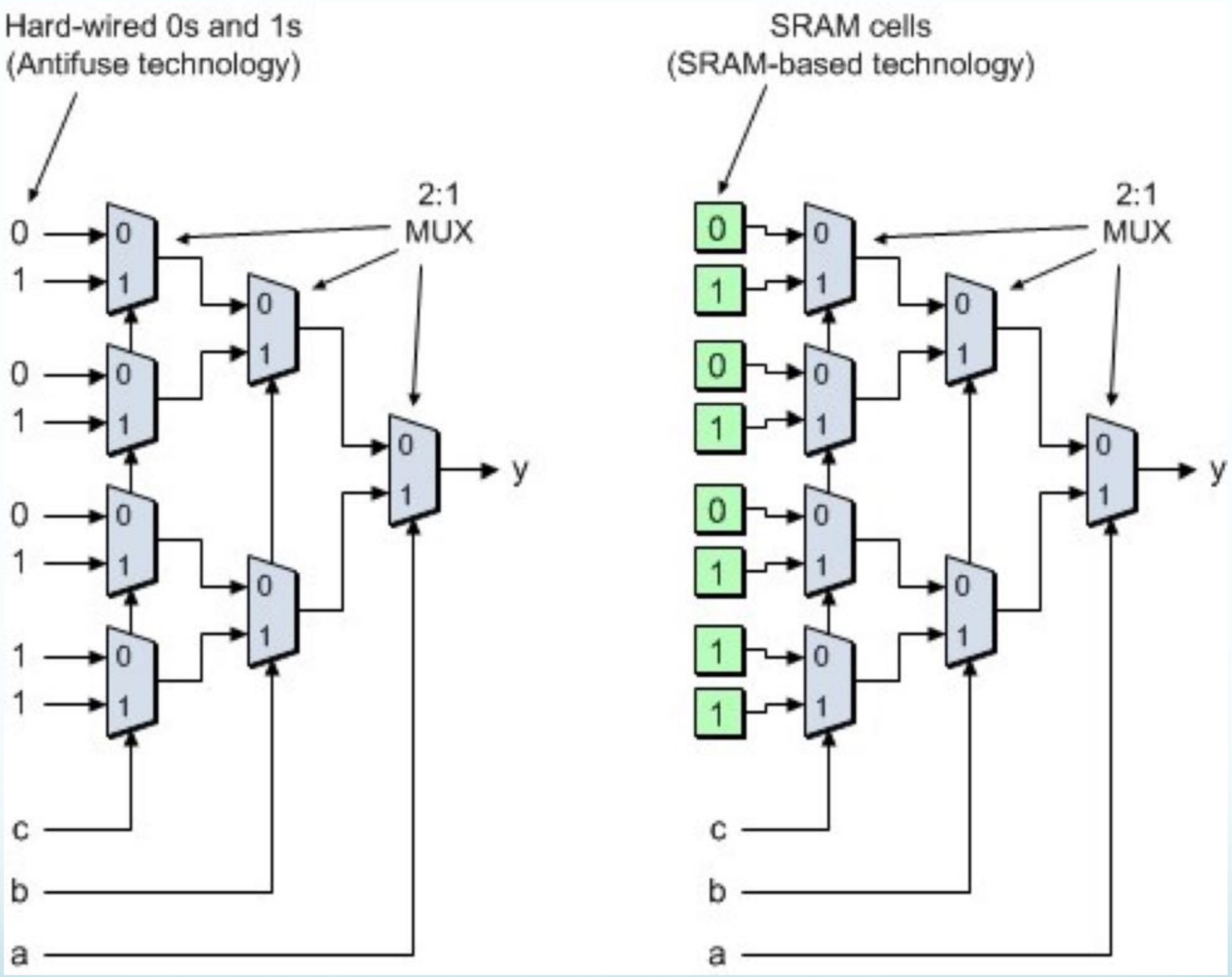
SRAM-Based Devices



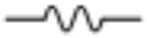


Multi-transistor SRAM cell

Programming a 3-bit wide LUT



Summary of Technologies

Technology	Symbol	Predominantly associated with ...
Fusible-link		SPLDs
Antifuse		FPGAs
EPROM		SPLDs and CPLDs
E^2PROM/ FLASH		SPLDs, CPLDs, and FPGAs
SRAM		FPGAs (some CPLDs)

Antifuse: Rad-tolerant secure

E^2PROM/ FLASH: Rad-tolerant (e.g. Alice)

SRAM: Used in most FPGAs

Major Manufacturers

- AMD Xilinx (formerly Xilinx)
  - First company to produce FPGAs in 1985
  - About 55% market share, today
  - SRAM based CMOS devices

Bought by AMD in 2022

- Intel FPGA (formerly Altera)
  - About 35% market share
  - SRAM based CMOS devices

Altera bought by Intel in 2015

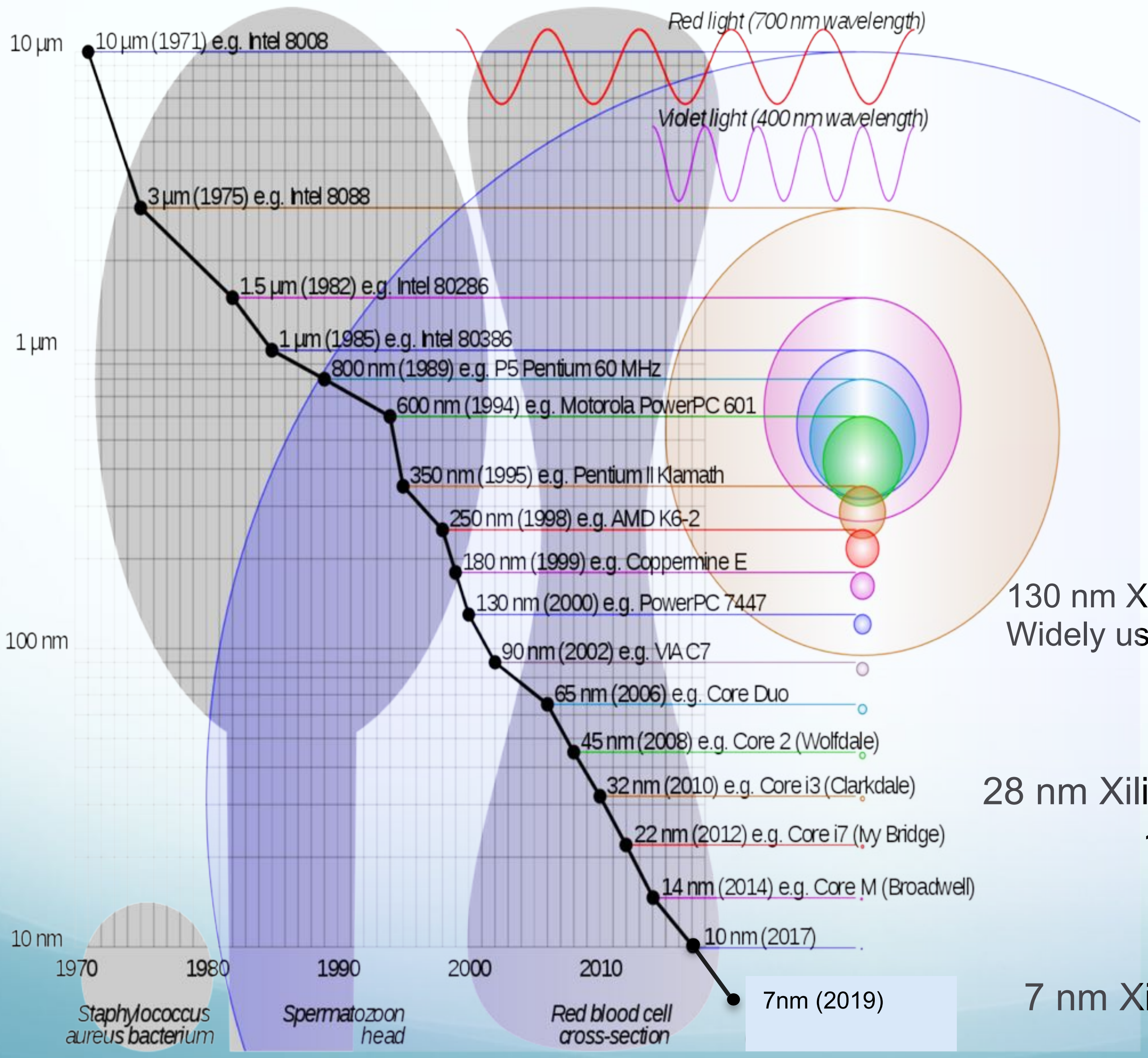
- Microchip (Microsemi, Actel)
  - Anti-fuse FPGAs
  - Flash based FPGAs
  - Mixed Signal

(Formerly Microsemi)
(Formerly Actel)

- Lattice Semiconductor
  - SRAM based with integrated Flash PROM
  - low power

Trends

Ever-decreasing feature size



- Higher capacity
- Higher speed
- Lower power consumption

130 nm Xilinx Virtex-2
Widely used at LHC startup

28 nm Xilinx Virtex-7 / Altera Stratix V

16 nm Xilinx UltraScale +

14 nm Intel Stratix 10

7 nm Xilinx Versal ACAP(*)

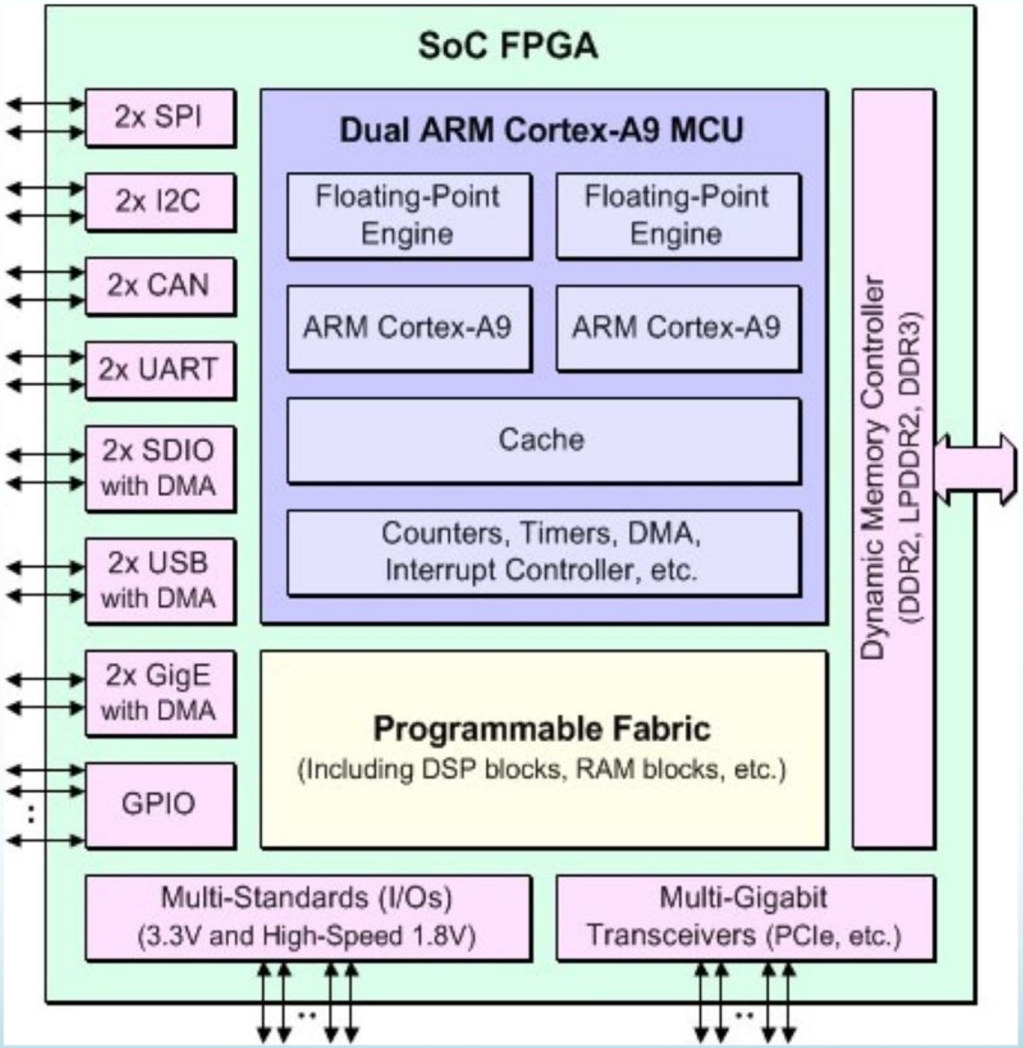
Technology nodes < 28 nm are commercial names and do not represent any geometry of transistors.

(*) Adaptive Compute Acceleration Platform

Trends

- Speed of logic keeps increasing
- Look-up-tables with more inputs (5 or 6)
- Speed of serial links increasing (multiple Gb/s)
- More integrated memory
  - Integrated High Bandwidth Memory (HBM) in-package
    - 10x faster than DDR4 (Xilinx: up to 8 GB, Intel: up to 16GB)
- More and more hard macro cores on the FPGA
  - PCI Express
    - Gen2: 5 Gb/s per lane
    - Gen3: 8 Gb/s per lane (typically up to 16 lanes)
    - Gen4: 16 Gb/s per lane
  - 10 Gb/s, 40 Gb/s, 100 Gb/s Ethernet, 150 Gb/s Interlaken
- Sophisticated soft macros
  - CPUs
  - Gb/s MACs
  - Memory interfaces (DDR2/3/4)
- Processor-centric architectures – see next slide

System-On-a-Chip (SoC) FPGAs

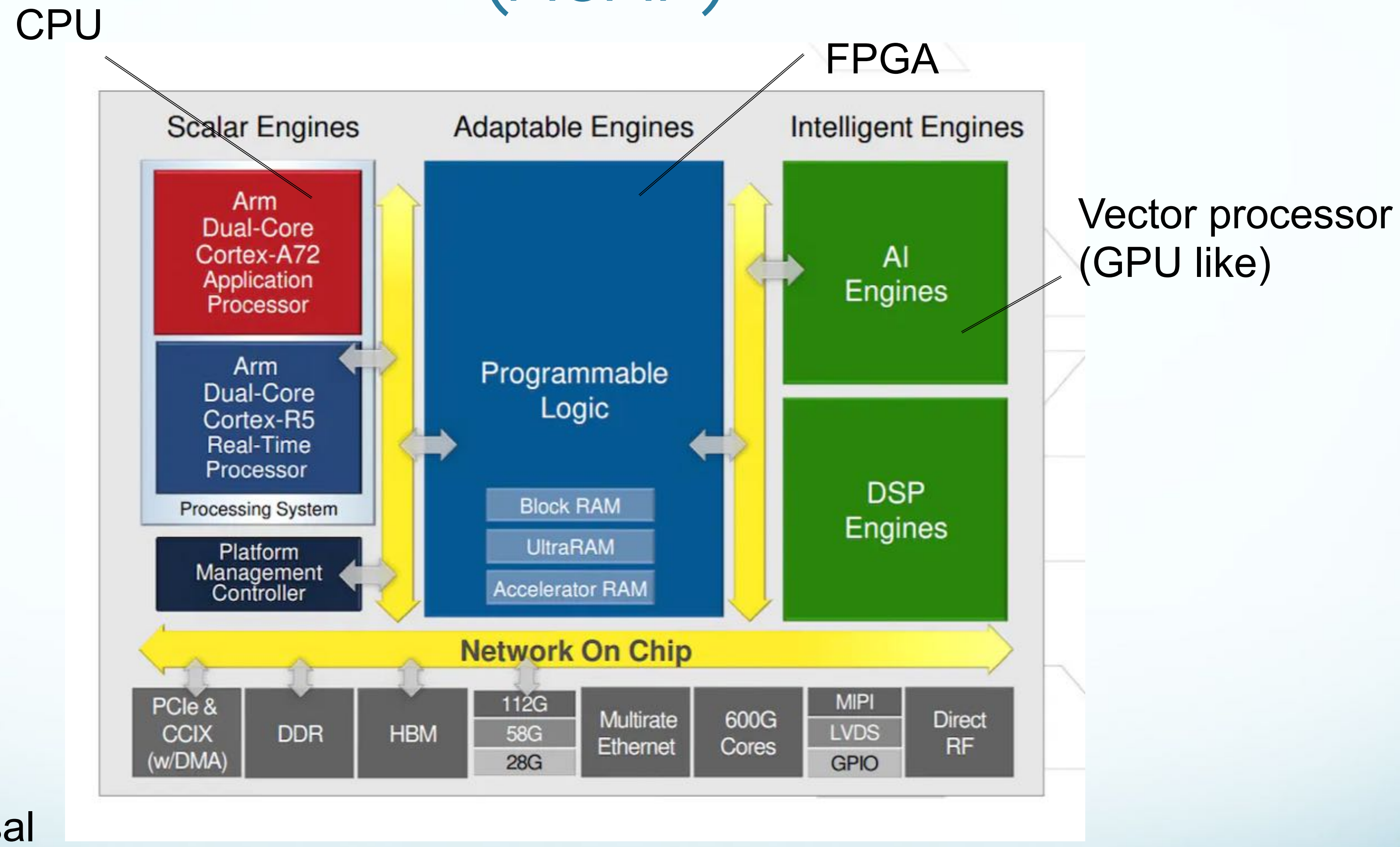

SoC FPGA
2x SPI
2x I2C
2x CAN
2x UART
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
GPIO
Dual ARM Cortex-A9 MCU
Floating-Point Engine
Floating-Point Engine
ARM Cortex-A9
ARM Cortex-A9
Cache
Counters, Timers, DMA, Interrupt Controller, etc.
Programmable Fabric
(Including DSP blocks, RAM blocks, etc.)
Dynamic Memory Controller (DDR2, LPDDR2, DDR3)
Multi-Standards (I/Os) (3.3V and High-Speed 1.8V)
Multi-Gigabit Transceivers (PCIe, etc.)

Xlinix Zynq

Intel Stratix 10 SoC

CPU(s) + Peripherals + FPGA in one package

Adaptive Compute Acceleration Platform (ACAP)



Xlinix Versal

https://www.electronicdesign.com/markets/automation/video/21234012/electronic-design-versal-card-streamlines-acap-fpga-ai-development

CPU(s) + Peripherals + FPGA + AI (Adaptable Intelligence) Engines in one package

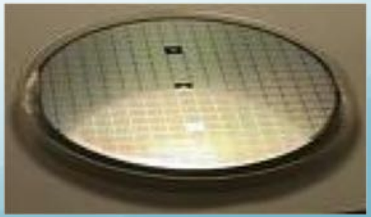
FPGA – ASIC comparison

FPGA

- A chip (the FPGA) is configured to represent a digital circuit
- May be reprogrammed in the field (gateware upgrade)
  - New features
  - Bug fixes
- Rapid development cycle (minutes / hours)
- Only digital designs are possible
- Low development cost
  - You can get started with a development board (< $100) and free software
- High-end FPGAs rather expensive

ASIC(*)

- A chip is produced in a foundry for a specific purpose
- Design cannot be changed once it is produced
- Long development cycle (weeks / months)
- Analog designs possible
- Higher performance
  - Speed, Area, Power
- Better radiation hardness
- Extremely high development cost
  - ASICs are produced at a semiconductor fabrication facility ("fab") according to your design
- Lower cost per device compared to FPGA, when large quantities are needed



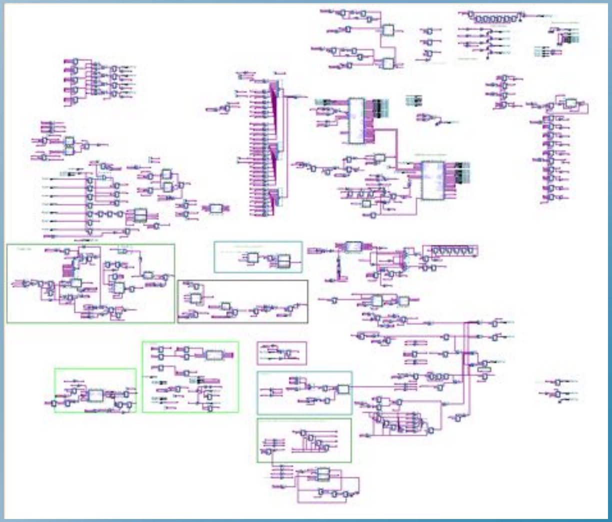
(*) Application Specific Integrated Circuit

FPGA design flow

Design entry

Schematics

Hardware description language
VHDL, Verilog



```
entity DelayLine is

  generic (
    n_halfcycles : integer := 2);

  port (
    x          : in std_logic_vector;
    x_delayed  : out std_logic_vector;
    clk        : in std_logic);

end entity DelayLine;
```

- Graphical overview
- Can draw entire design
- Use pre-defined blocks

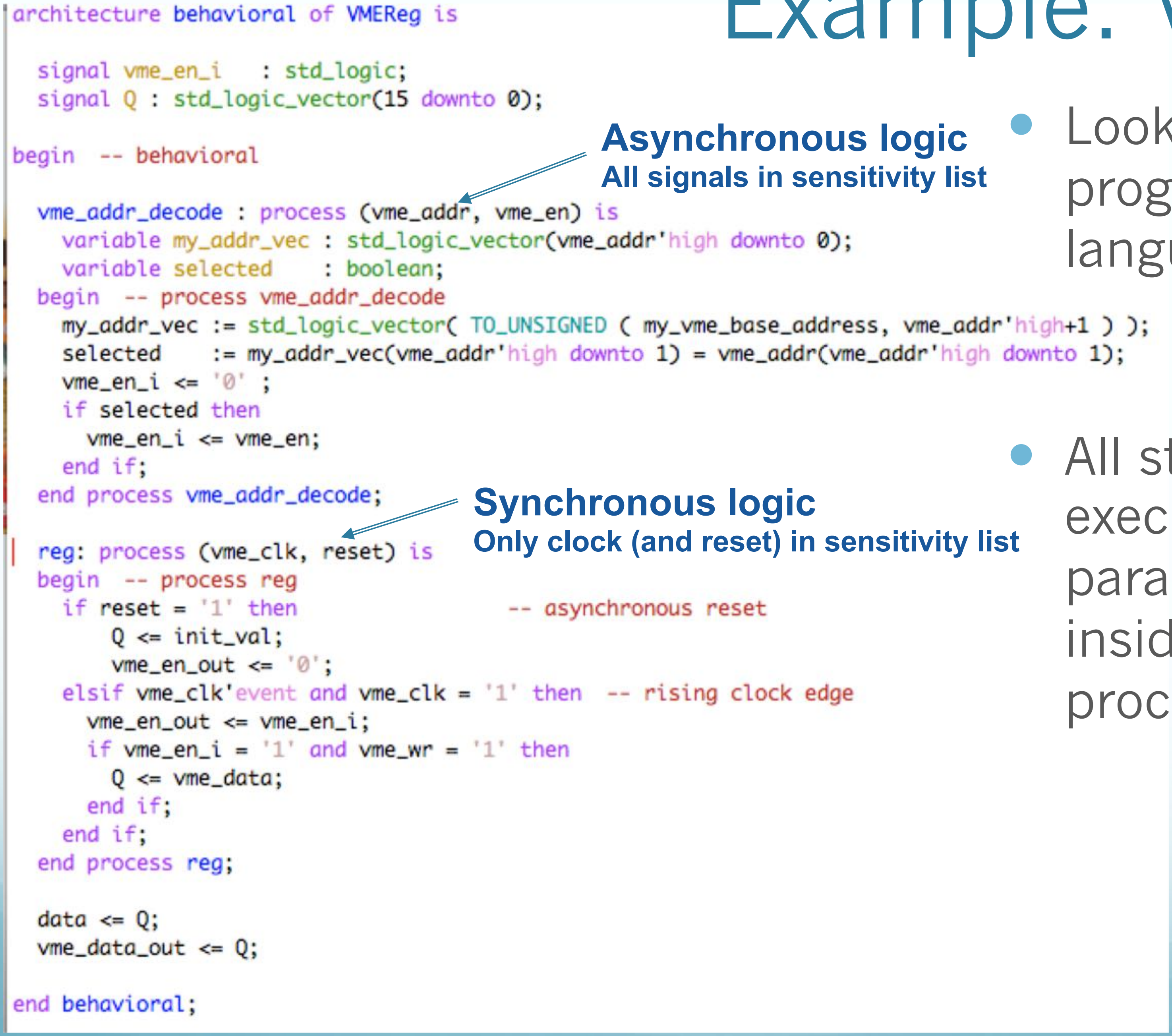
rarely used today

- Can generate blocks using loops
- Can synthesize algorithms
- Independent of design tool
- May use tools used in SW development (SVN, git ...)

Hardware Description Language

- Looks similar to a programming language
  - BUT be aware of the difference
    - Programming language => translated into machine instructions that are executed by a CPU
    - HDL => translated into gateware (logic gates & flip-flops)
- Common HDLs
  - VHDL
  - Verilog
  - AHDL (Altera specific)
- Newer trends
  - C-like languages (handle-C, System C)
  - Labview
  - High Level Synthesis (HLS) from C/C++

Example: VHDL



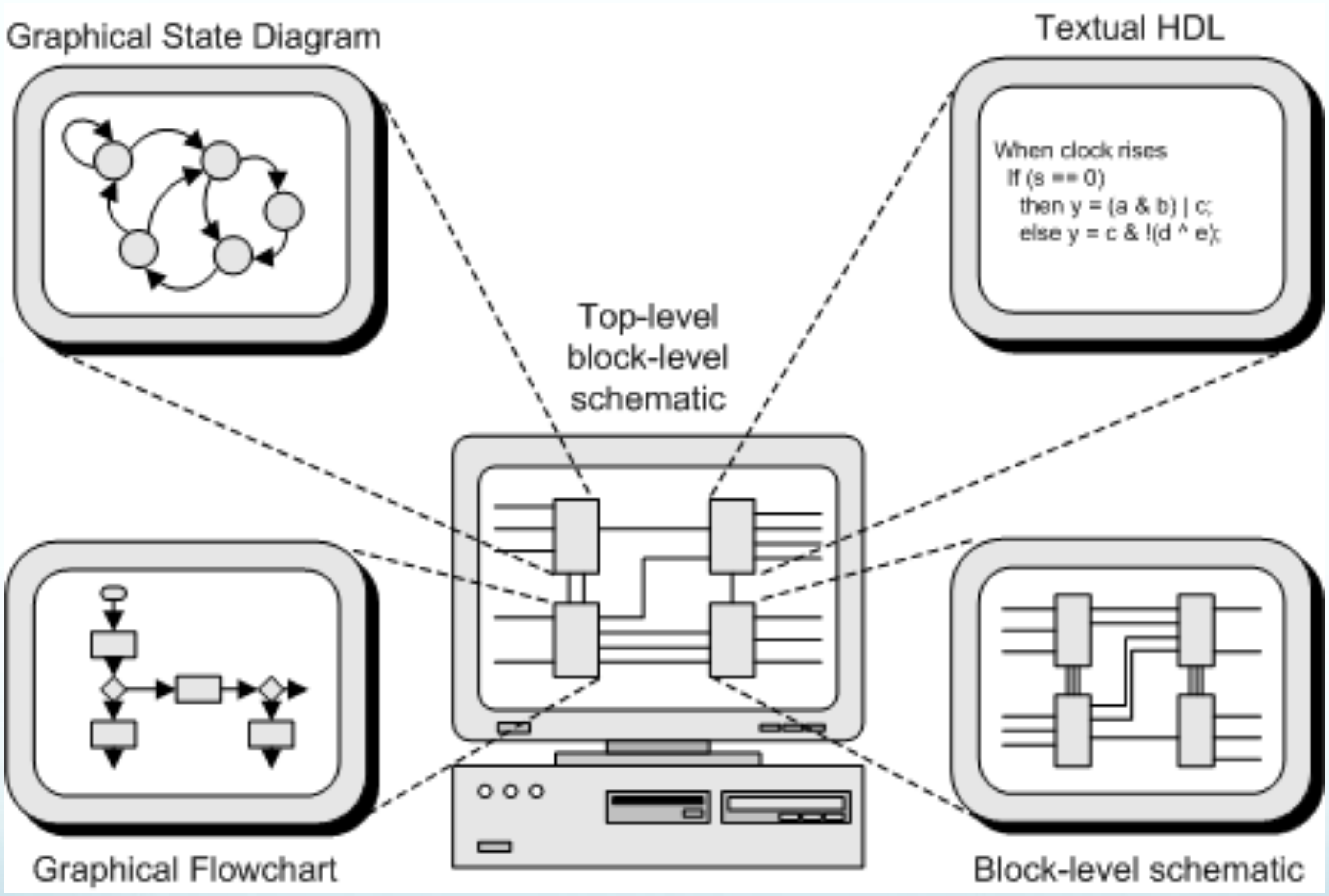
```
architecture behavioral of VMEReg is

  signal vme_en_i    : std_logic;
  signal Q : std_logic_vector(15 downto 0);

begin  -- behavioral

  vme_addr_decode : process (vme_addr, vme_en) is
    variable my_addr_vec : std_logic_vector(vme_addr'high downto 0);
    variable selected    : boolean;
  begin  -- process vme_addr_decode
    my_addr_vec := std_logic_vector( TO_UNSIGNED ( my_vme_base_address, vme_addr'high+1 ) );
    selected    := my_addr_vec(vme_addr'high downto 1) = vme_addr(vme_addr'high downto 1);
    vme_en_i <= '0' ;
    if selected then
      vme_en_i <= vme_en;
    end if;
  end process vme_addr_decode;

  reg: process (vme_clk, reset) is
  begin  -- process reg
    if reset = '1' then                 -- asynchronous reset
        Q <= init_val;
        vme_en_out <= '0';
    elsif vme_clk'event and vme_clk = '1' then  -- rising clock edge
      vme_en_out <= vme_en_i;
      if vme_en_i = '1' and vme_wr = '1' then
        Q <= vme_data;
      end if;
    end if;
  end process reg;

  data <= Q;
  vme_data_out <= Q;

end behavioral;
```

Asynchronous logic
All signals in sensitivity list

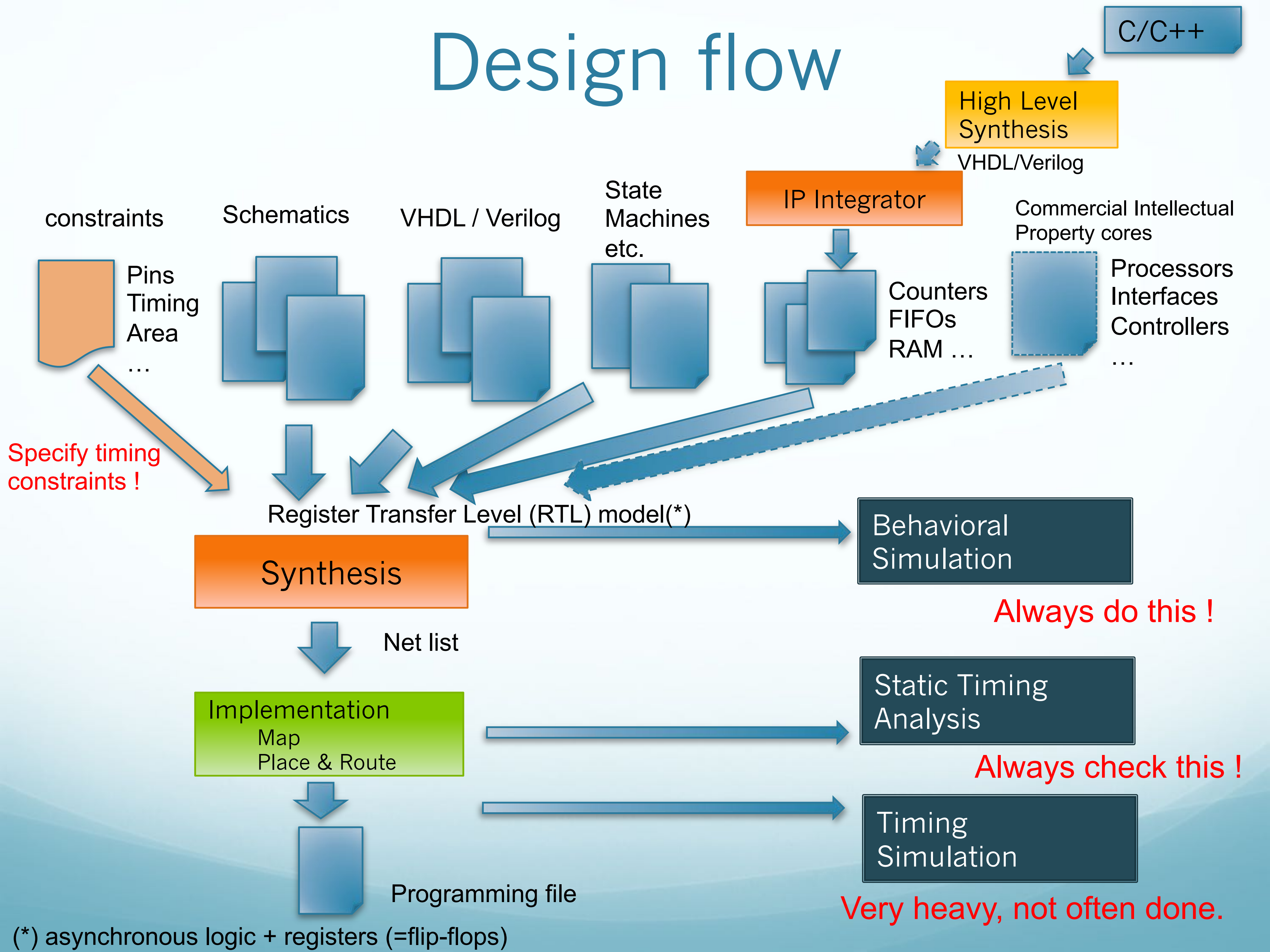
Synchronous logic
Only clock (and reset) in sensitivity list

- Looks like a programming language
- All statements executed in parallel, except inside processes

Schematics & HDL combined

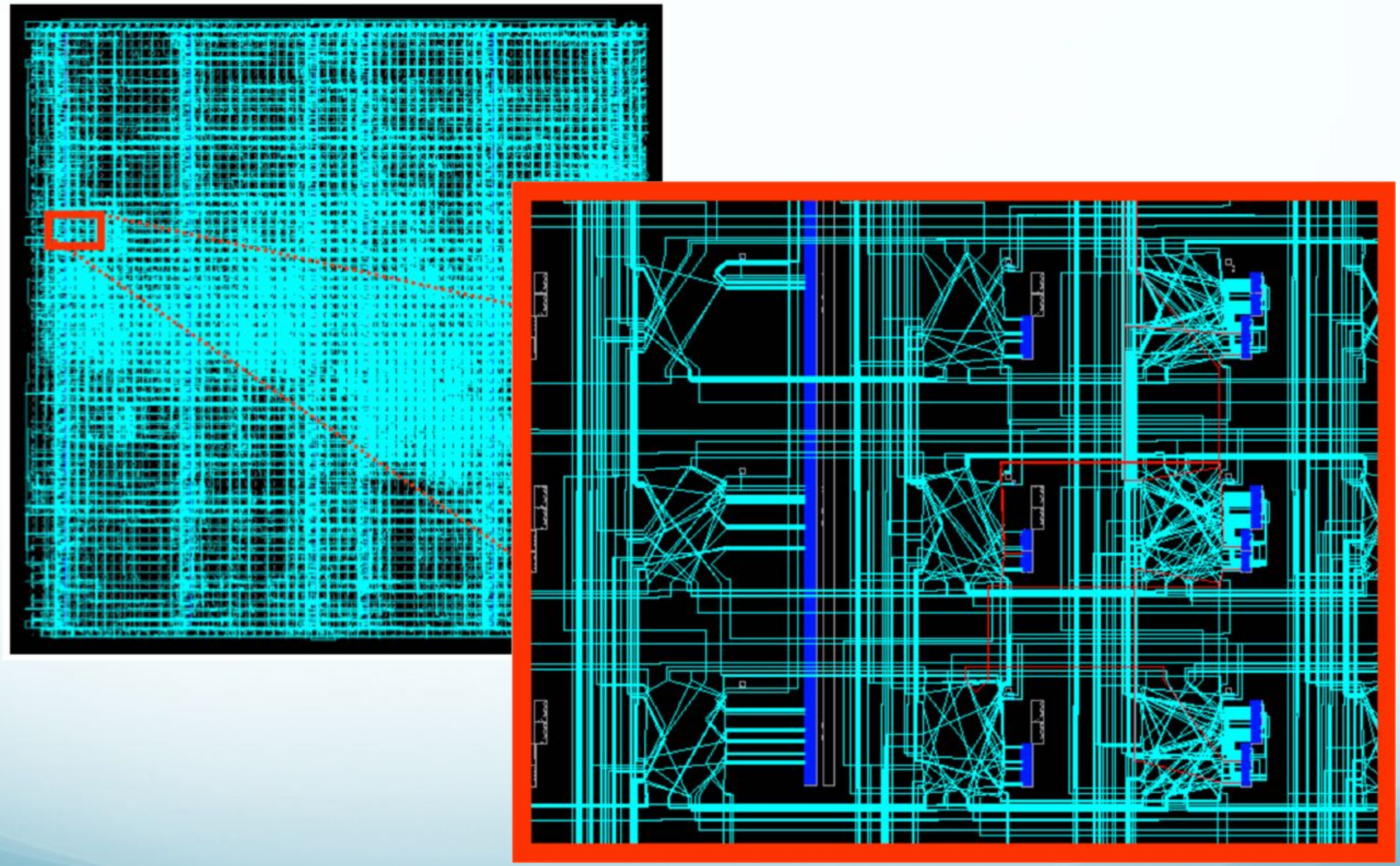


Design flow

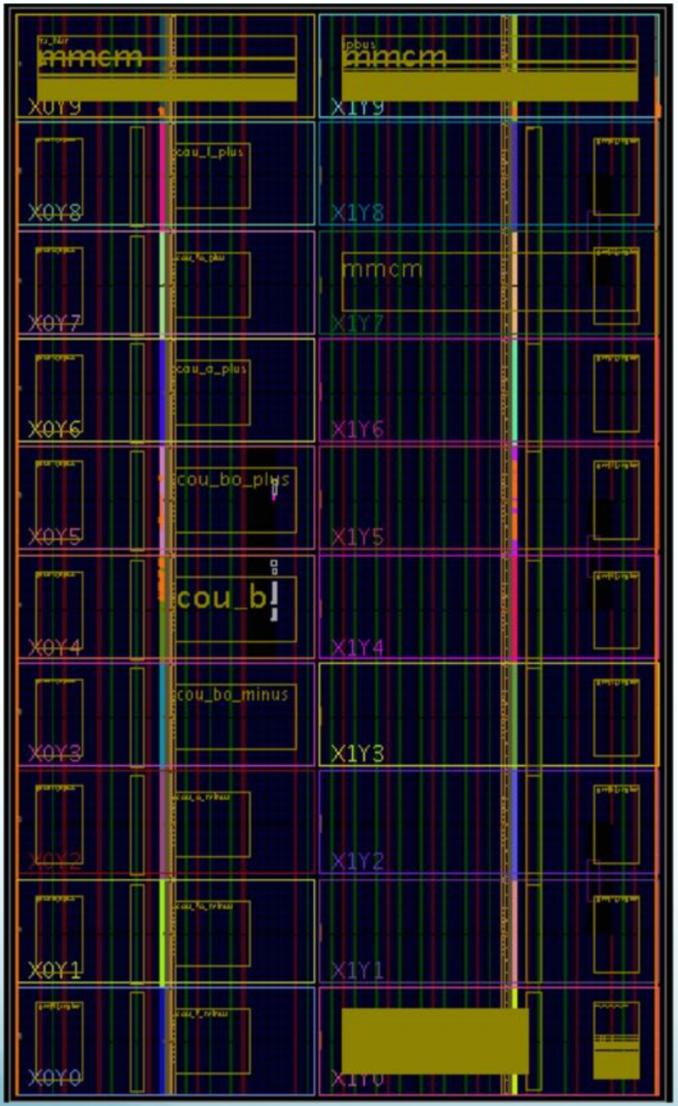


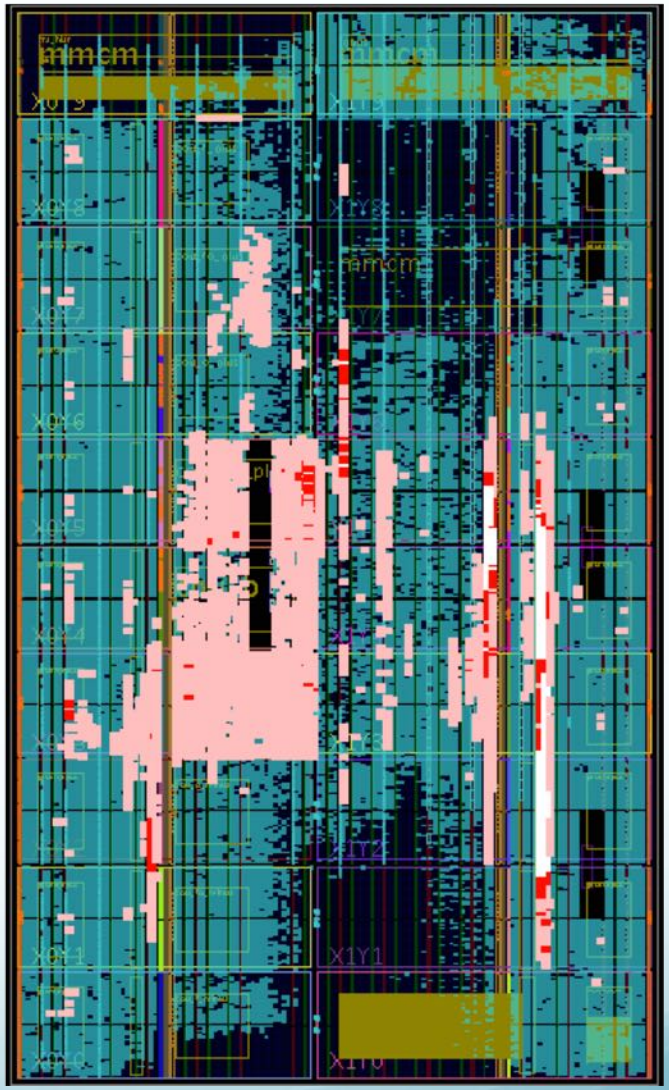
(*) asynchronous logic + registers (=flip-flops)

Floorplan



Manual Floor planning

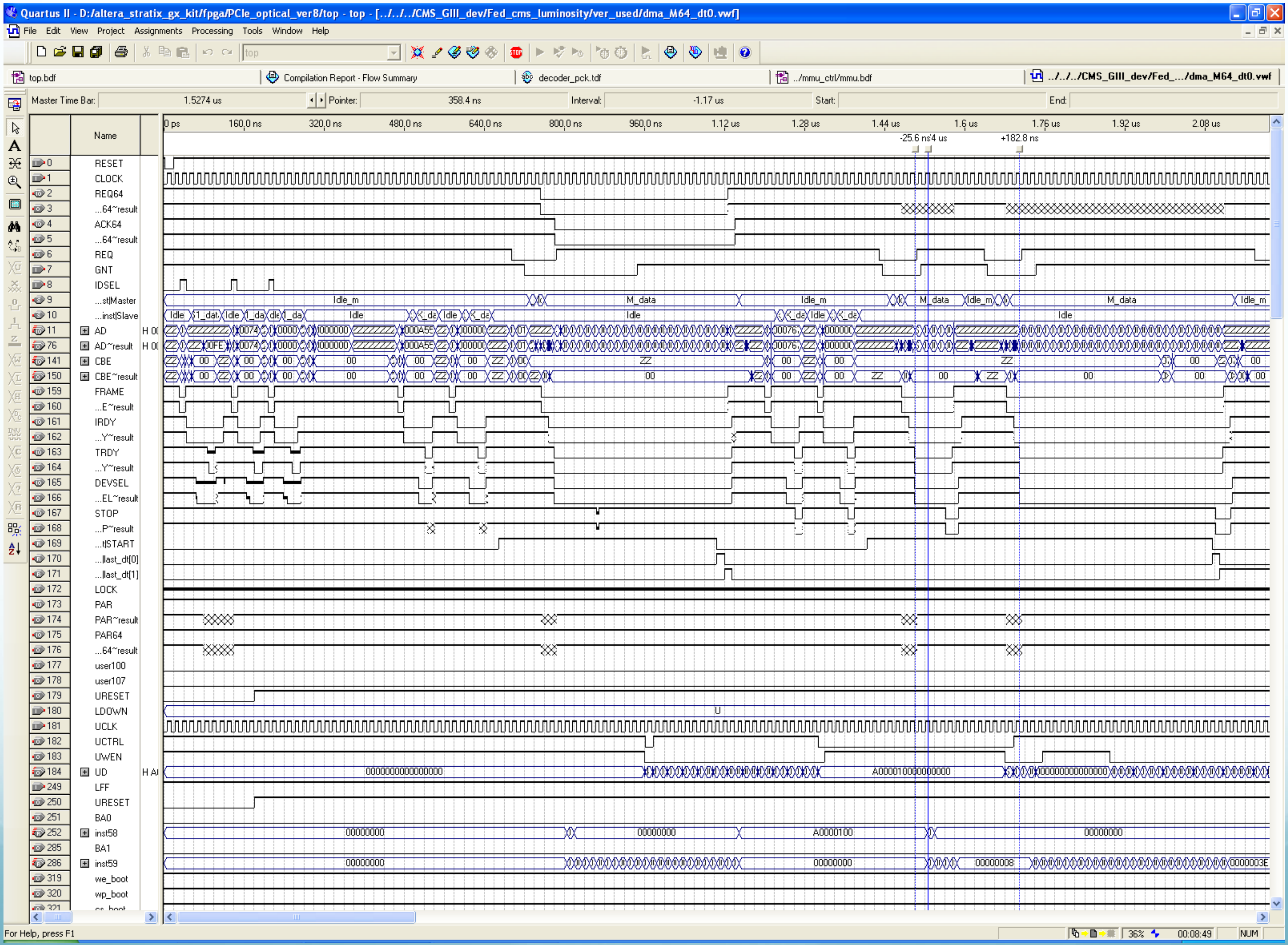




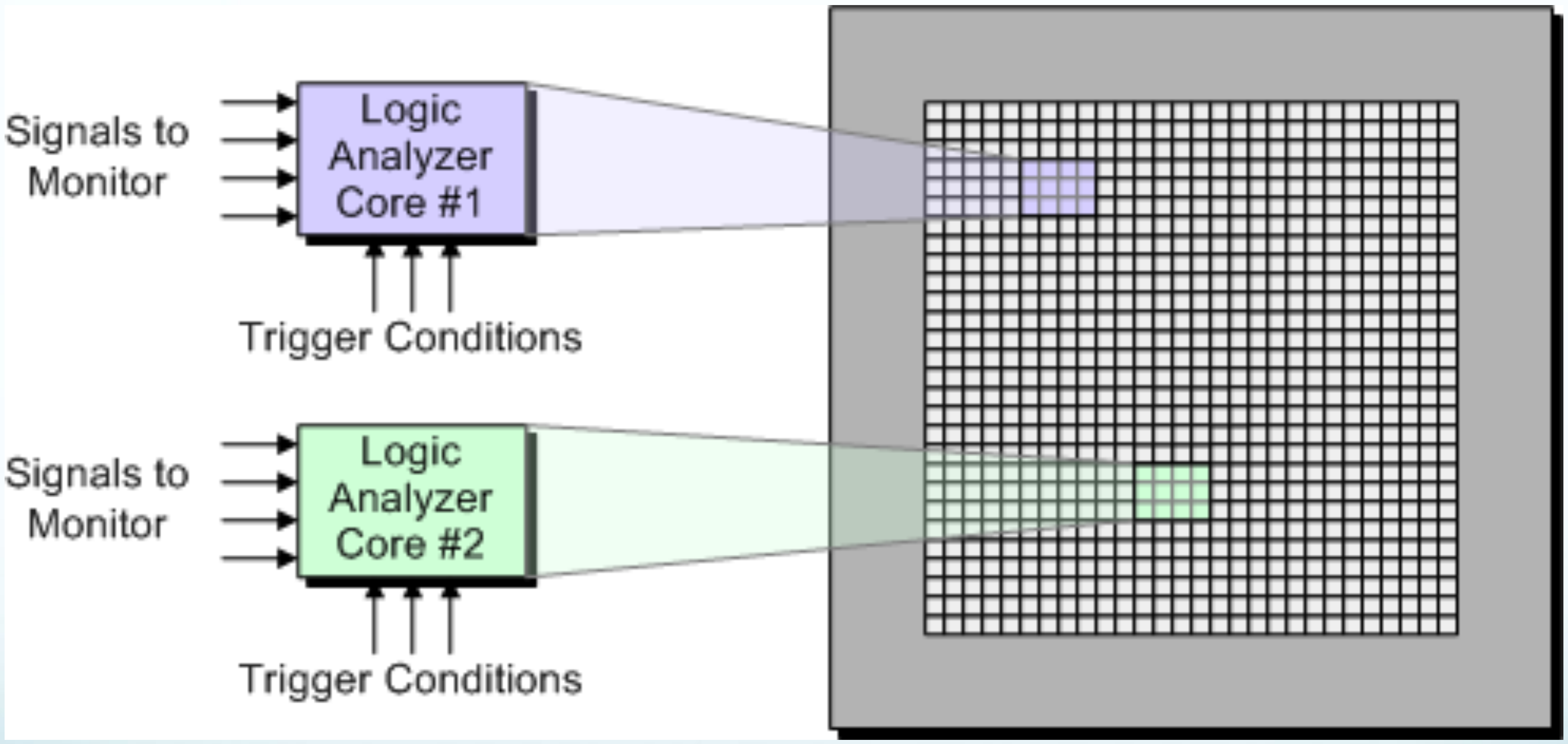
- For large designs, manual floor planning may be necessary

Routing congestion
Xilinx Virtex 7 (Vivado)

Simulation


Quartus II - D:/altera_stratix_gx_kit/fpga/PCIe_optical_ver8/top - top - [../../../CMS_GIII_dev/Fed_cms_luminosity/ver_used/dma_M64_dt0.vwf]
File Edit View Project Assignments Processing Tools Window Help
top.bdf
Compilation Report - Flow Summary
decoder_pck.tdf
../mmu_ctrl/mmu.bdf
../../../CMS_GIII_dev/Fed_.../dma_M64_dt0.vwf
Master Time Bar: 1.5274 us
Pointer: 358.4 ns
Interval: -1.17 us
Start:
End:
Name
0 ps
160.0 ns
320.0 ns
480.0 ns
640.0 ns
800.0 ns
960.0 ns
1.12 us
1.28 us
1.44 us
1.6 us
1.76 us
1.92 us
2.08 us
-25.6 ns
4 us
+182.8 ns
0 RESET
1 CLOCK
2 REQ64
3 ...64~result
4 ACK64
5 ...64~result
6 REQ
7 GNT
8 IDSEL
9 ...st|Master
10 ...inst|Slave
11 AD
76 AD~result
141 CBE
150 CBE~result
159 FRAME
160 ...E~result
161 IRDY
162 ...Y~result
163 TRDY
164 ...Y~result
165 DEVSEL
166 ...EL~result
167 STOP
168 ...P~result
169 ...t|START
170 ...|last_dt[0]
171 ...|last_dt[1]
172 LOCK
173 PAR
174 PAR~result
175 PAR64
176 ...64~result
177 user100
178 user107
179 URESET
180 LDOWN
181 UCLK
182 UCTRL
183 UWEN
184 UD
249 LFF
250 URESET
251 BA0
252 inst58
285 BA1
286 inst59
319 we_boot
320 wp_boot
321 cs_boot
Idle_m
M_data
Idle
ZZ
00
U
0000000000000000
A000010000000000
00000000
A0000100
00000008
0000003E
For Help, press F1
36%
00:08:49
NUM

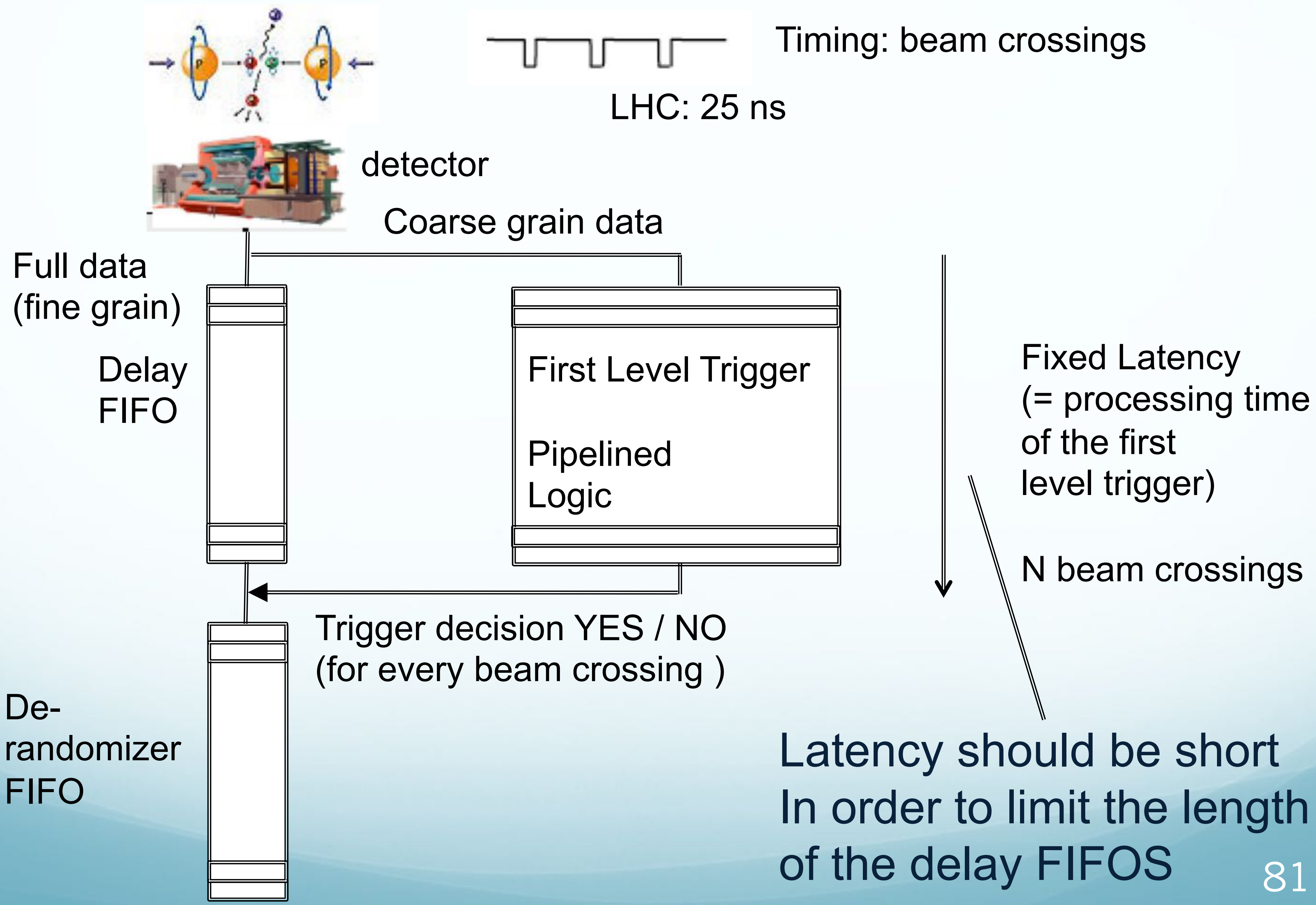
Embedded Logic Analyzers



A great tool for debugging your design

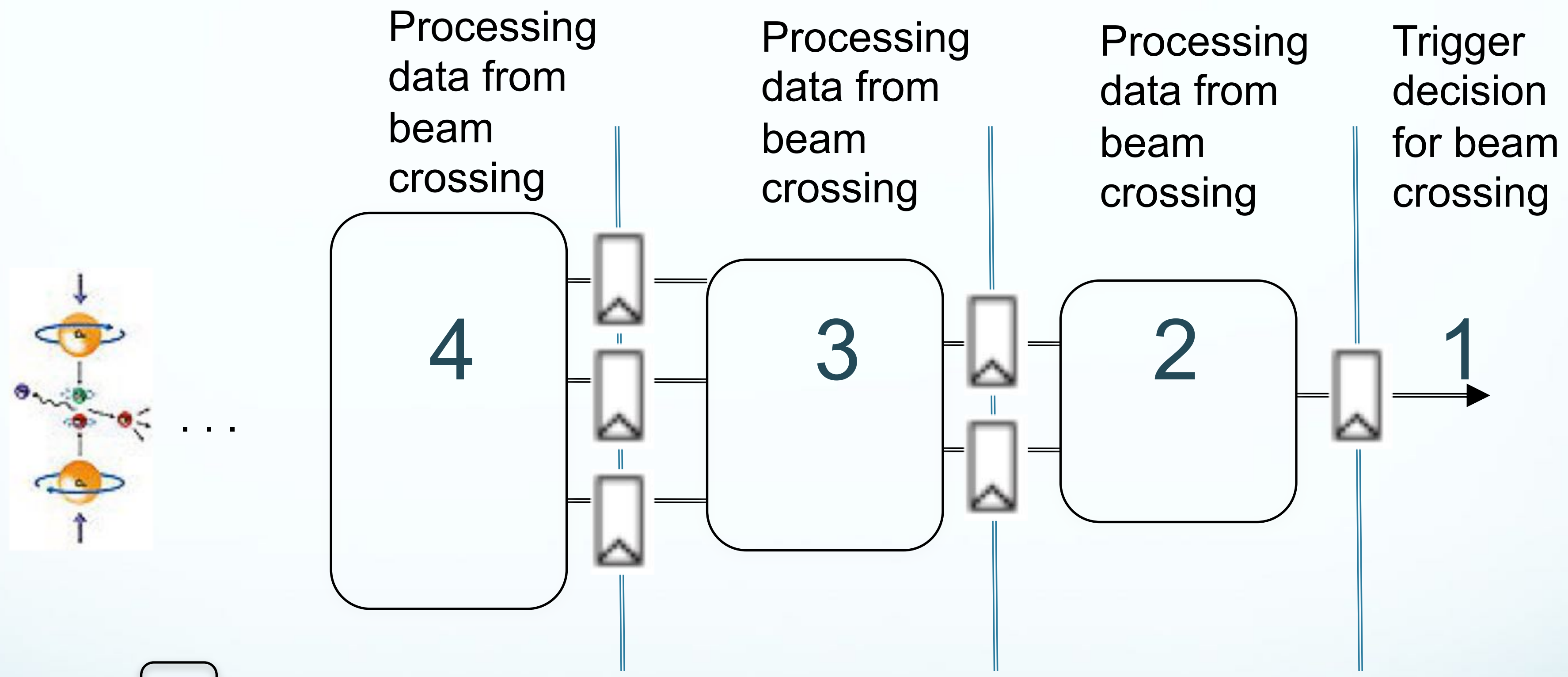
FPGA applications in the Trigger & DAQ domain

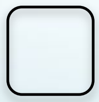
First-Level Trigger at Collider

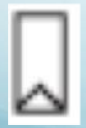


Latency should be short
In order to limit the length
of the delay FIFOS

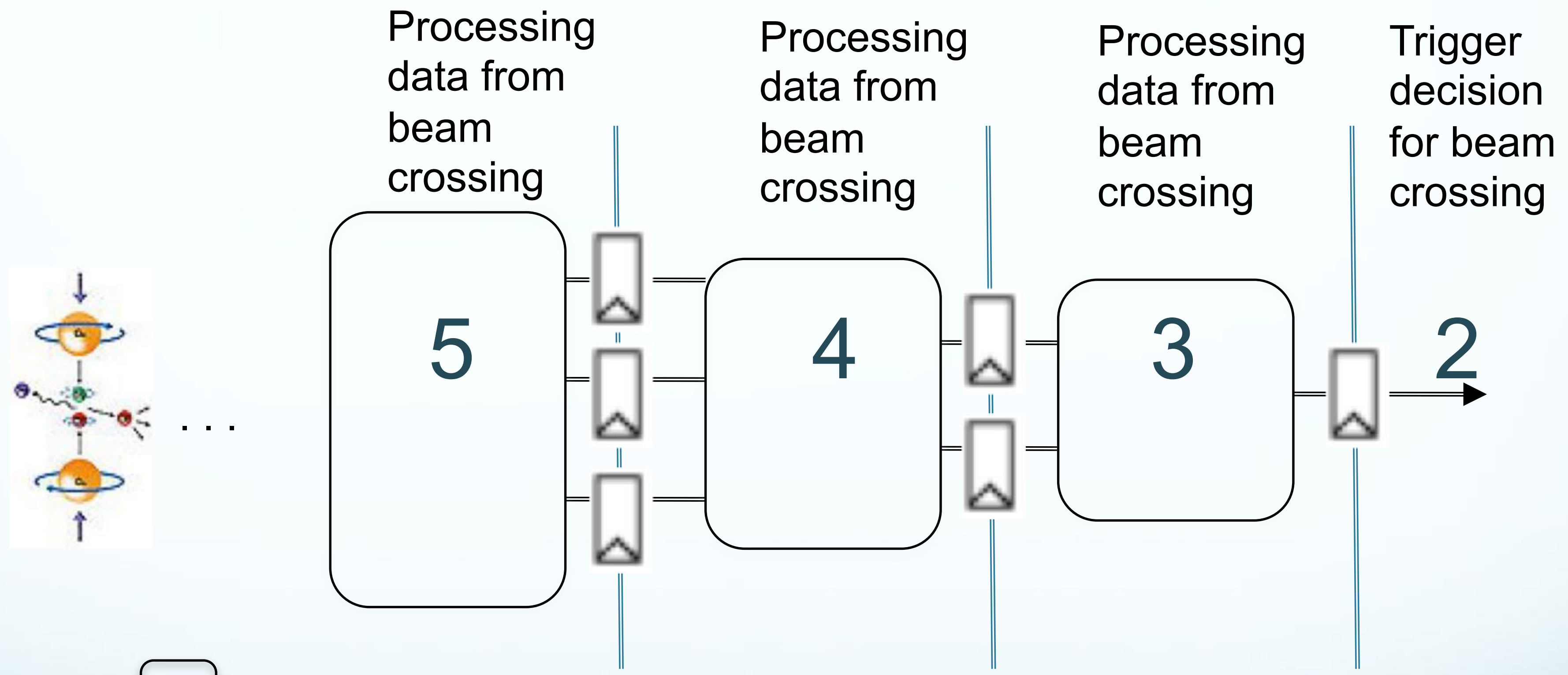
Pipelined Logic


Processing data from beam crossing
Processing data from beam crossing
Processing data from beam crossing
Trigger decision for beam crossing
. . .
4
3
2
1

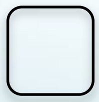

Combinatorial logic

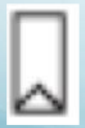

Flip flop
Clocked with same clock as collider

Pipelined Logic – a clock cycle later



Processing data from beam crossing
Processing data from beam crossing
Processing data from beam crossing
Trigger decision for beam crossing
. . .
5
4
3
2


Combinatorial logic


Flip flop
Clocked with same clock as collider

Why are FPGAs ideal for First-Level Triggers ?

- They are fast
  - Much faster than discrete electronics (shorter connections)
- Many inputs
  - Data from many parts of the detector has to be combined

Low latency

- All operations are performed in parallel
  - Can build pipelined logic
- They can be re-programmed
  - Trigger algorithms can be optimized

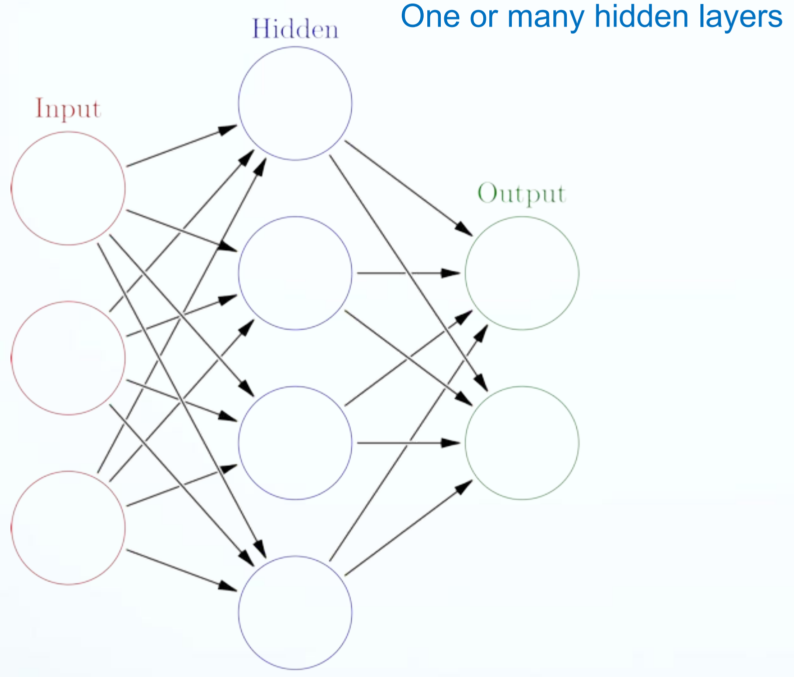
High performance

Trigger algorithms implemented in FPGAs

- Trigger
  - Peak finding
  - Pattern Recognition
  - Track Finding
  - Clustering / Energy summing
  - Topological Algorithms (invariant mass)
  - Vertex Finding
  - Particle flow (reconstruction jets, etc. from individual particle tracks)
  - Inference with Neural Networks
  - Many more …

- Trigger Control system
  - Fast (busy) signal merging & monitoring
  - Generation of random triggers
  - Generation of calibration sequences
  - Automatic recovery sequences
  - Monitoring (dead times, rates, …)

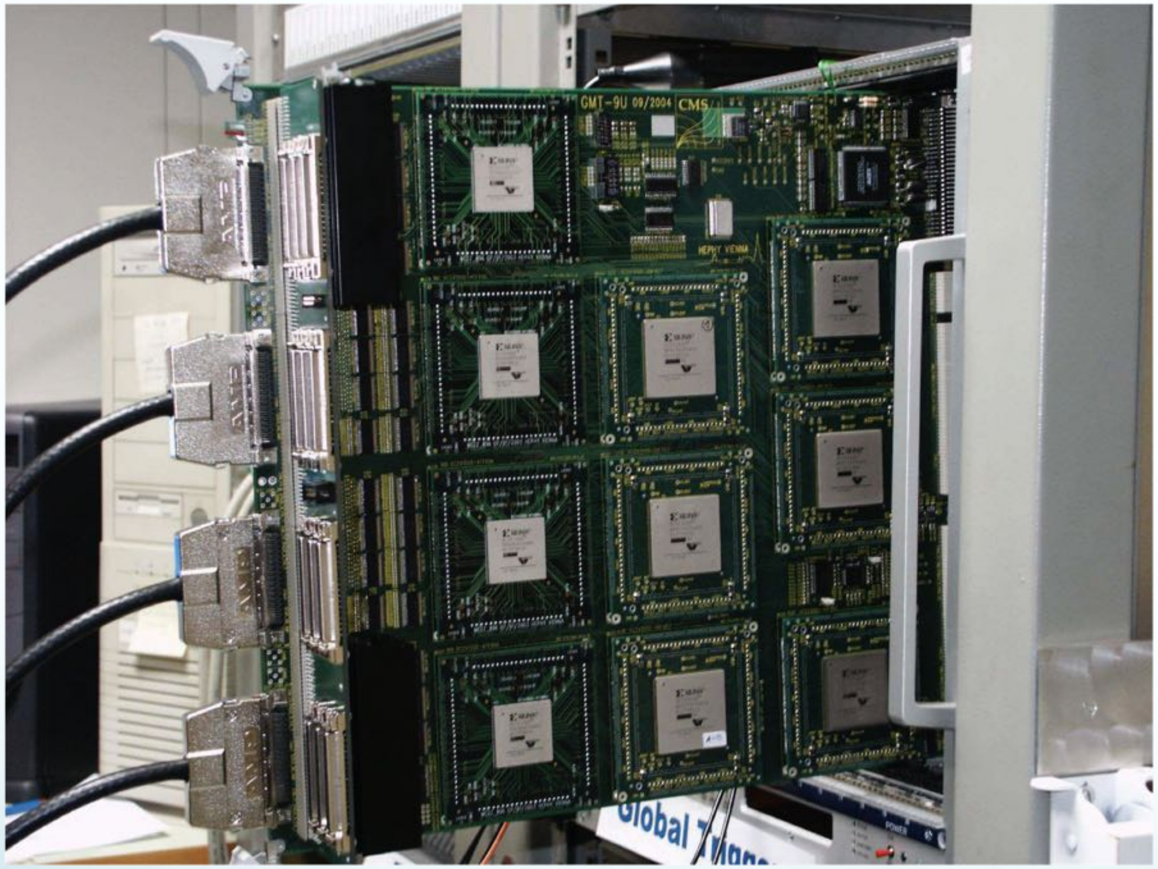
Neural Networks in Trigger



By Glosser.ca - Own work, Derivative of File:Artificial neural network.svg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=24913461

- Principle
  - Node is assigned a value based on the weighted sum of nodes in the previous layer
  - Maps well to DSP resources in FPGA (multiplier + adder)
- Applications:
  - Jet classification
  - Assignment of transverse momentum based on many measurements
  - Topological trigger
  - …
- Tools
  - Many commercial tools
  - hls4ml (optimized for latency)
    - Firmware generation from high-level model using Vivado HLS

CMS Global Muon Trigger

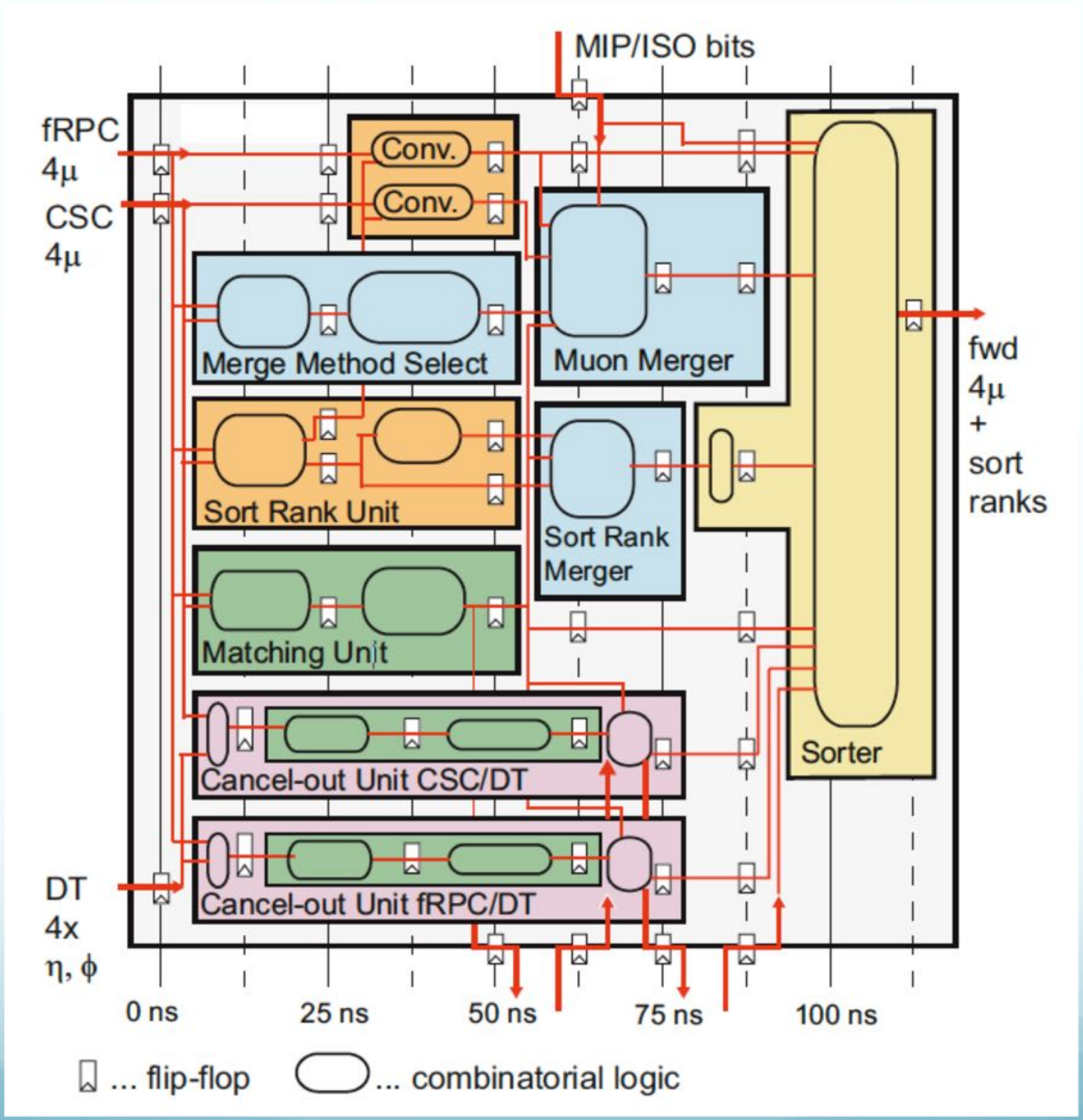


- The CMS Global Muon trigger received 16 muon candidates from the three muon systems of CMS
- It merged different measurements for the same muon and found the best 4 over-all muon candidates

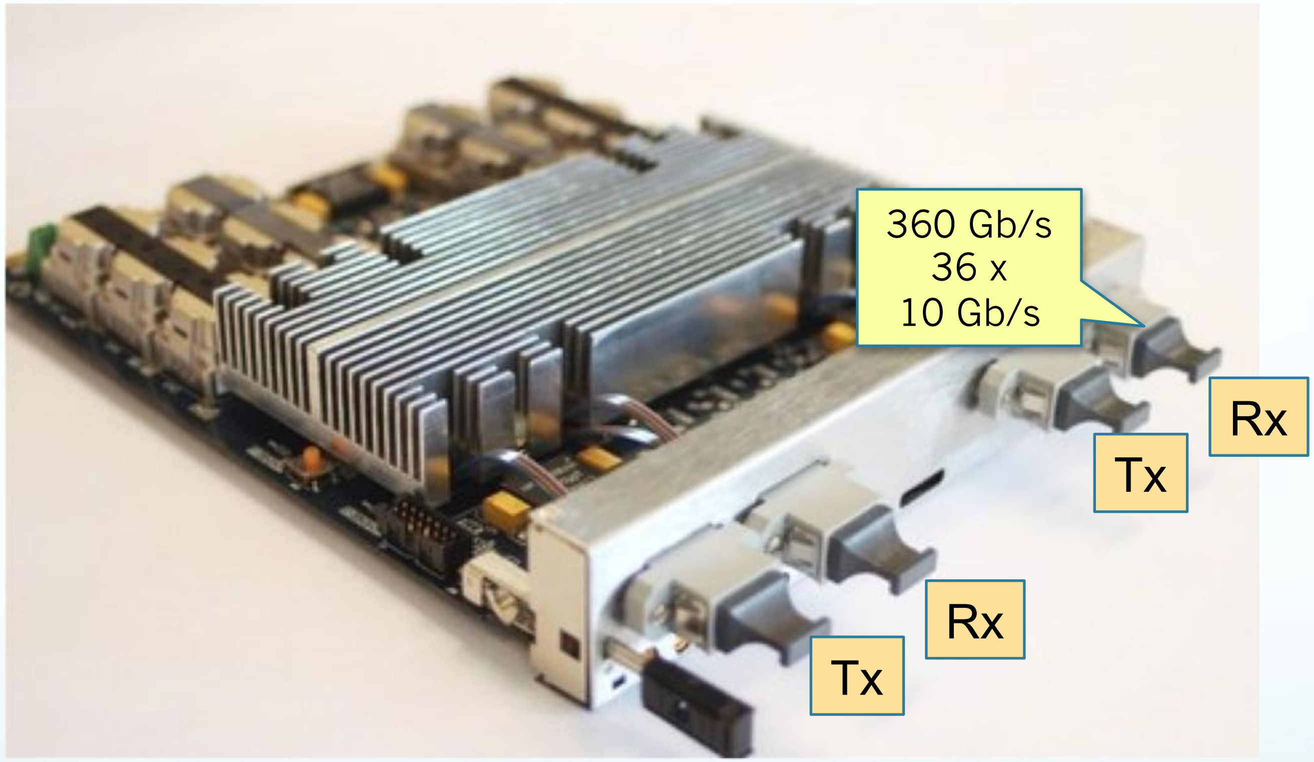
- VME card (9U)
- Input: ~1000 bits
  @ 40 and 80 MHz
- Output: ~50 bits @ 80MHz
- Processing time: 250 ns
- Pipelined logic
  one new result every 25 ns

- 10 Xilinx Virtex-II FPGAs
- up to 500 user I/Os per chip
- Up to 25000 LUTs per chip used
- Up to 96 x 18kbit RAM used
- In use in the CMS trigger 2008-2015

CMS Global Muon Trigger main FPGA



µTCA board for Run 2&3 CMS trigger based on Virtex 7



MP7, Imperial College

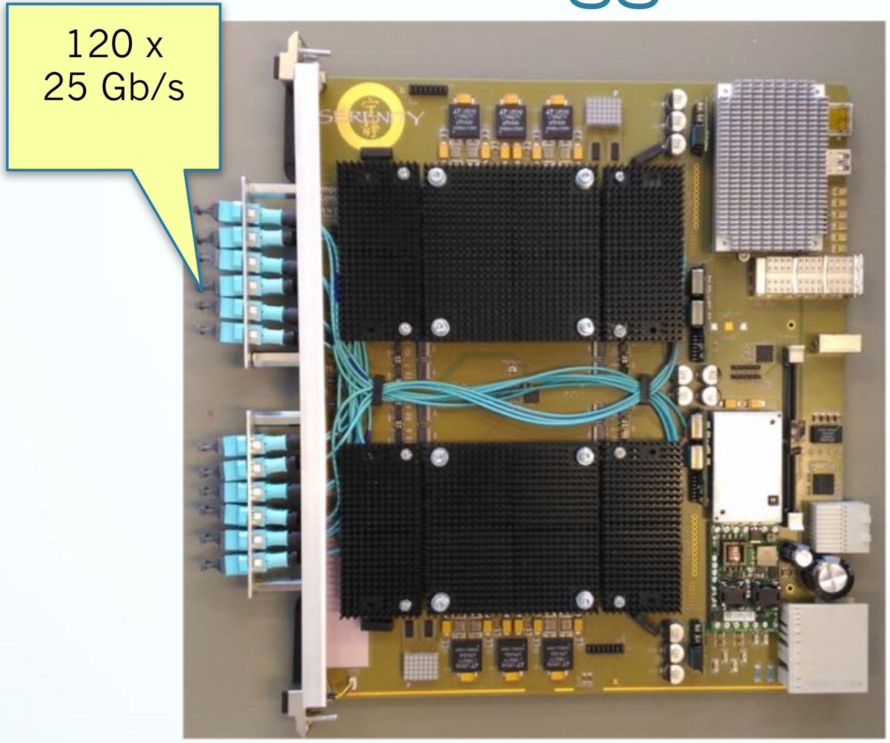
Virtex 7 with 690k logic cells
- 80 x 10 Gb/s transceivers bi-directional
- 72 of them as optical links on front panel
  - 0.75 + 0.75 Tb/s

Being used in the CMS trigger since 2015

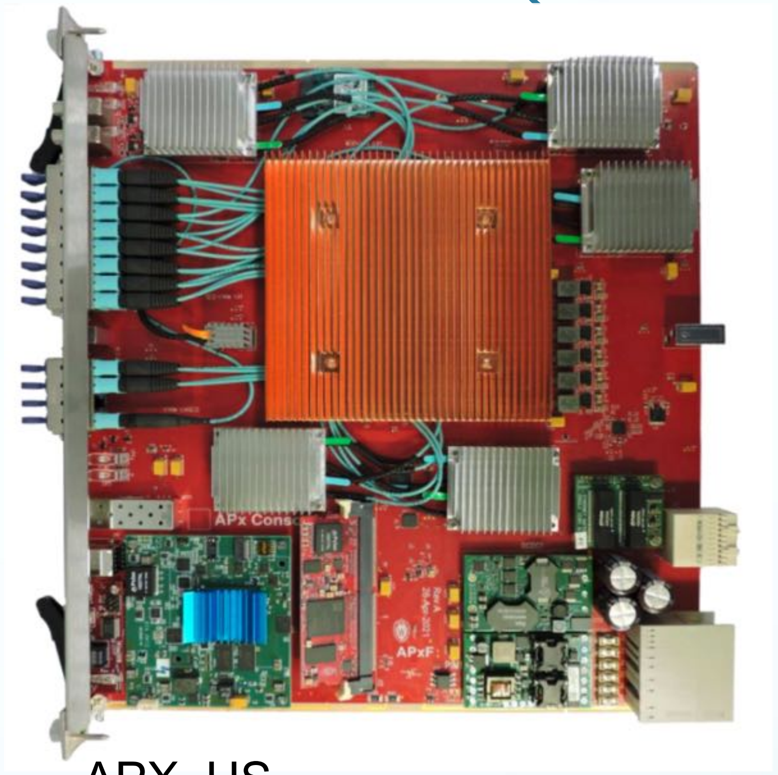
Input/output:
up to 14k bits per 40 MHz clock

Same board used for different functions (different gateware)
Separation of framework + algorithm fw

CMS ATCA Trigger boards for HL-LHC (2029+)



Serenity, UK



APX, US

- Few types of generic boards, ATCA standard
- Xilinx Virtex/Kintex Ultrascale+ FPGAs (> 3 million logic cells / FPGA)
- 25-28 Gb/s optical links
- SoC FPGAs used for board control (on some boards)
- Advanced firmware algorithms
  - Vertex finding
  - Particle flow
  - Neural network classifiers

FPGAs in Data Acquisition

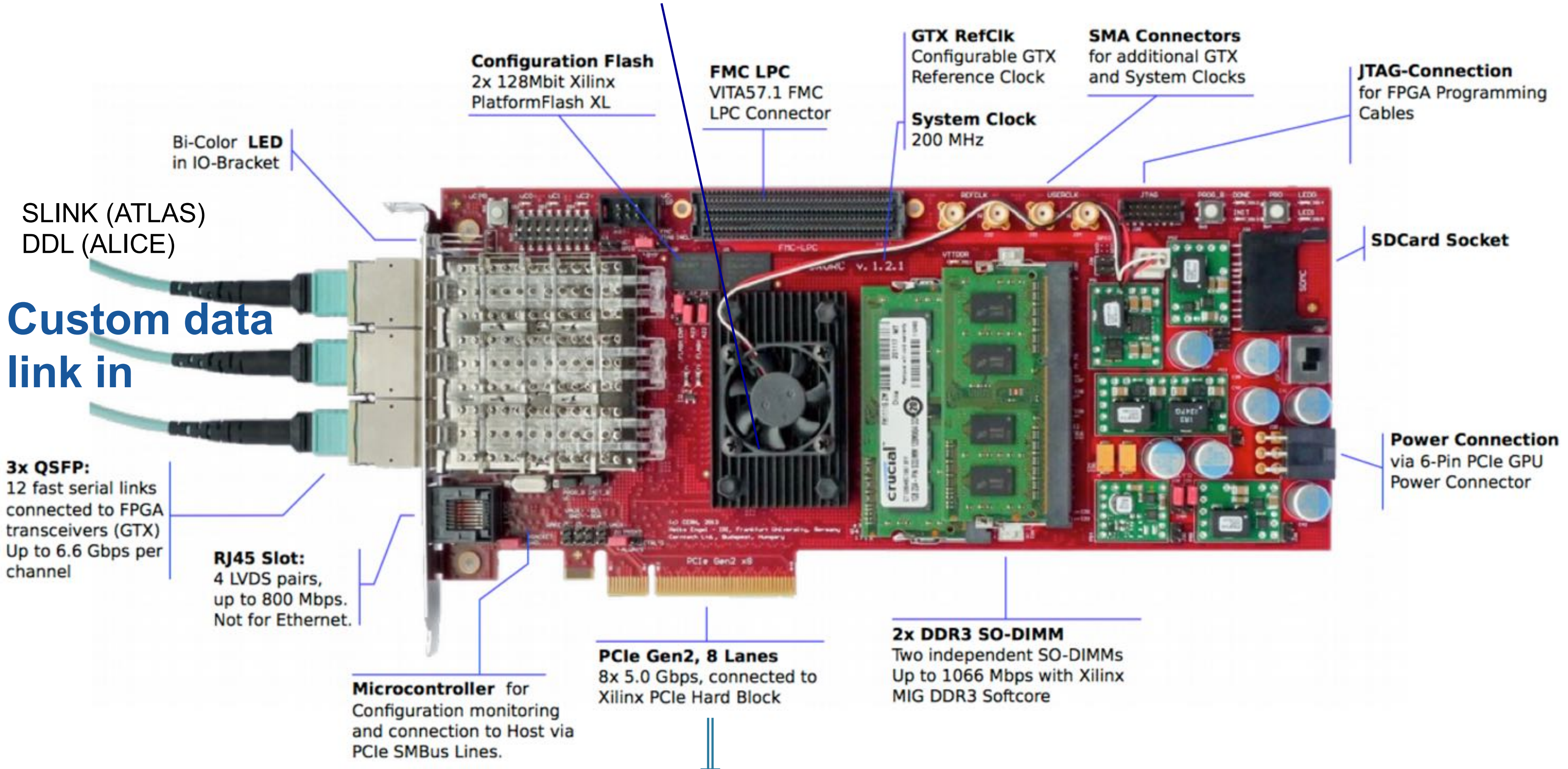
- Frontend Electronics
  - Pedestal subtraction
  - Zero suppression
  - Compression
  - Buffering …
- Custom data links
  - E.g. SLINK-64 over copper
    - Several serial LVDS links in parallel
    - Up to 400 MB/s
  - SLINK/SLINK-express over optical
- Interface from custom hardware to commercial electronics
  - PCI/PCIe, VME bus, Myrinet, 10/40/100 Gb/s Ethernet etc.

C-RORC (Alice) / Robin NP (ATLAS) for Run-2

Xilinx Virtex-6 FPGA

SLINK (ATLAS)
DDL (ALICE)

Custom data link in


Configuration Flash 2x 128Mbit Xilinx PlatformFlash XL
FMC LPC VITA57.1 FMC LPC Connector
GTX RefClk Configurable GTX Reference Clock
System Clock 200 MHz
SMA Connectors for additional GTX and System Clocks
JTAG-Connection for FPGA Programming Cables
Bi-Color **LED** in IO-Bracket
SDCard Socket
Power Connection via 6-Pin PCIe GPU Power Connector
3x QSFP: 12 fast serial links connected to FPGA transceivers (GTX) Up to 6.6 Gbps per channel
RJ45 Slot: 4 LVDS pairs, up to 800 Mbps. Not for Ethernet.
Microcontroller for Configuration monitoring and connection to Host via PCIe SMBus Lines.
PCIe Gen2, 8 Lanes 8x 5.0 Gbps, connected to Xilinx PCIe Hard Block
2x DDR3 SO-DIMM Two independent SO-DIMMs Up to 1066 Mbps with Xilinx MIG DDR3 Softcore

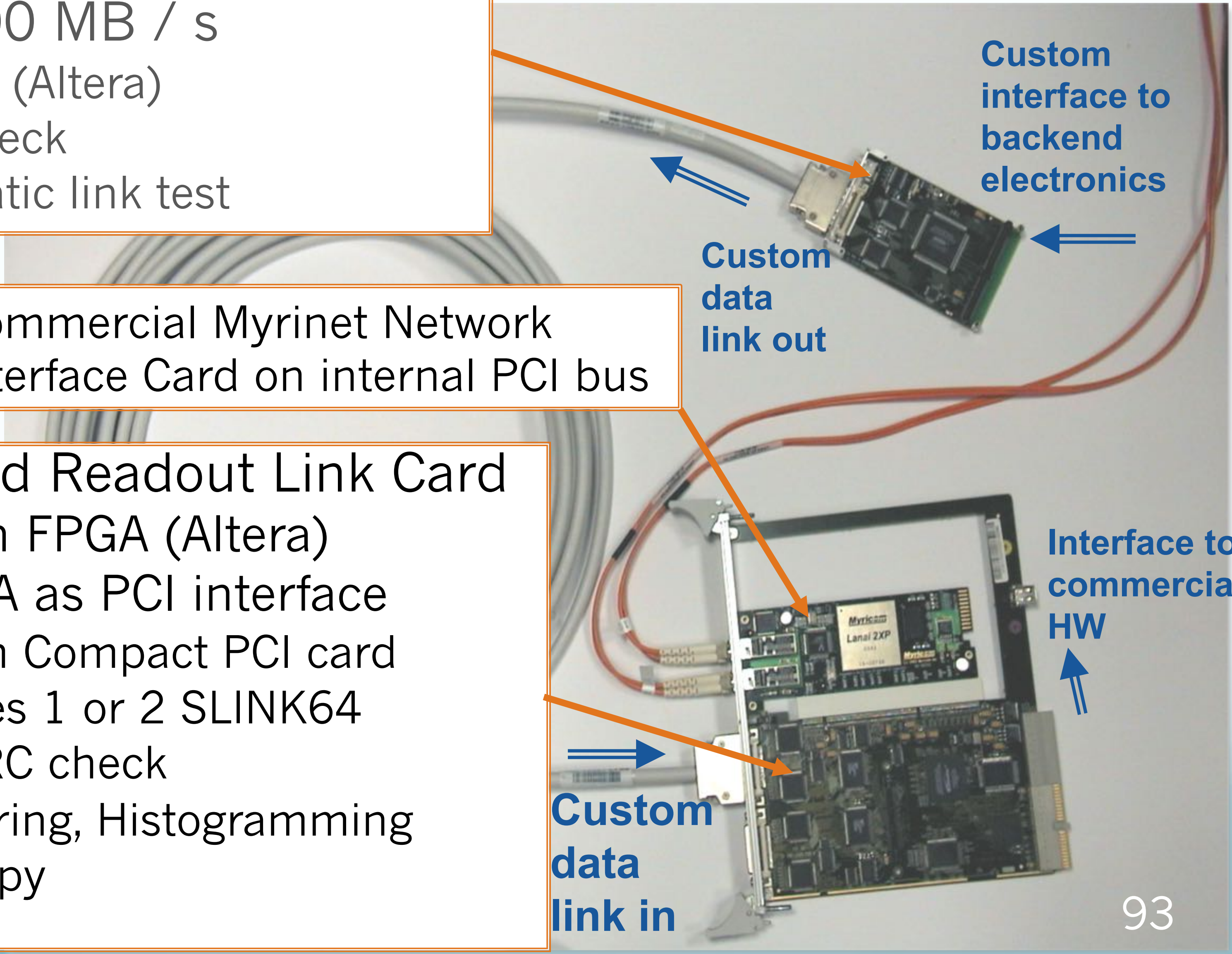
Commercial PCIe link out (DMA to host memory)

CMS Front-end Readout Link (Run-1)

- SLINK Sender Mezzanine Card: 400 MB / s
  - 1 FPGA (Altera)
  - CRC check
  - Automatic link test

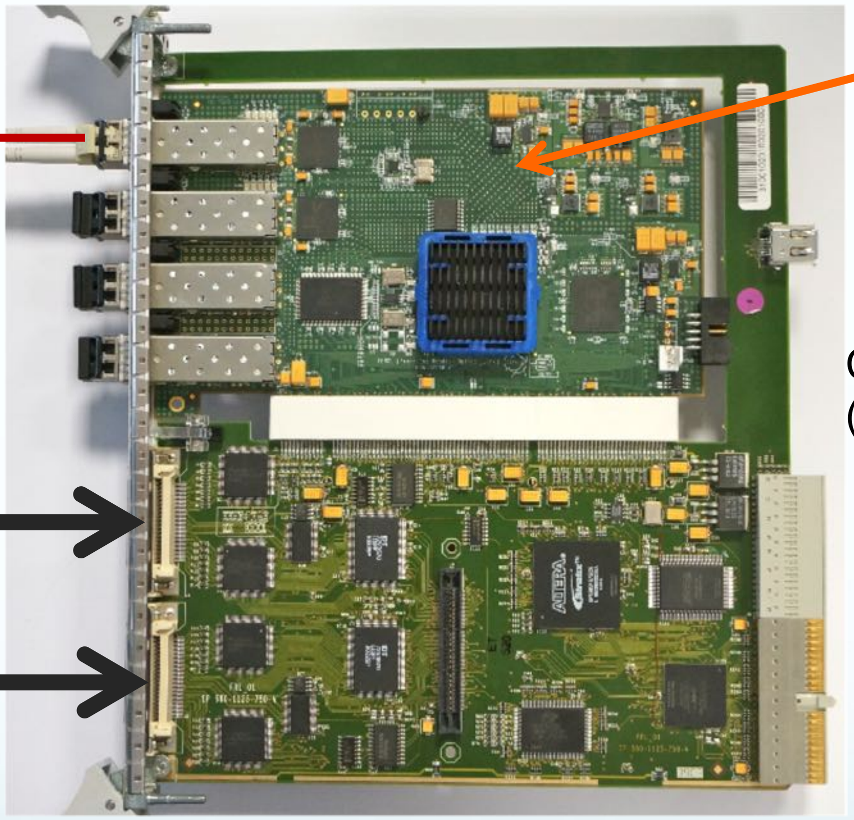
Commercial Myrinet Network Interface Card on internal PCI bus

- Front-end Readout Link Card
  - 1 main FPGA (Altera)
  - 1 FPGA as PCI interface
  - Custom Compact PCI card
  - Receives 1 or 2 SLINK64
  - 2nd CRC check
  - Monitoring, Histogramming
  - Event spy


Custom interface to backend electronics
Custom data link out
Interface to commercial HW
Custom data link in

CMS Readout Link for Run-2&3 in use since 2015

10 Gb/s TCP/IP



Myrinet NIC replaced by custom-built card ("FEROL")

Cost effective solution (need many boards)
Rather inexpensive FPGA
+ commercial chip to combine
3 Gb/s links to 10 Gb/s

SLINK-64 input
LVDS / copper

FEROL (Front End Readout Optical Link)

Input:	1x or 2x SLINK (copper)	Output:	10 Gb/s Ethernet optical
	1x or 2x 5Gb/s optical		TCP/IP sender in FPGA
	1x 10Gb/s optical		

CMS Readout Link for Run-2&3 in use since 2015

Commercial data link out

10 Gb/s TCP/IP

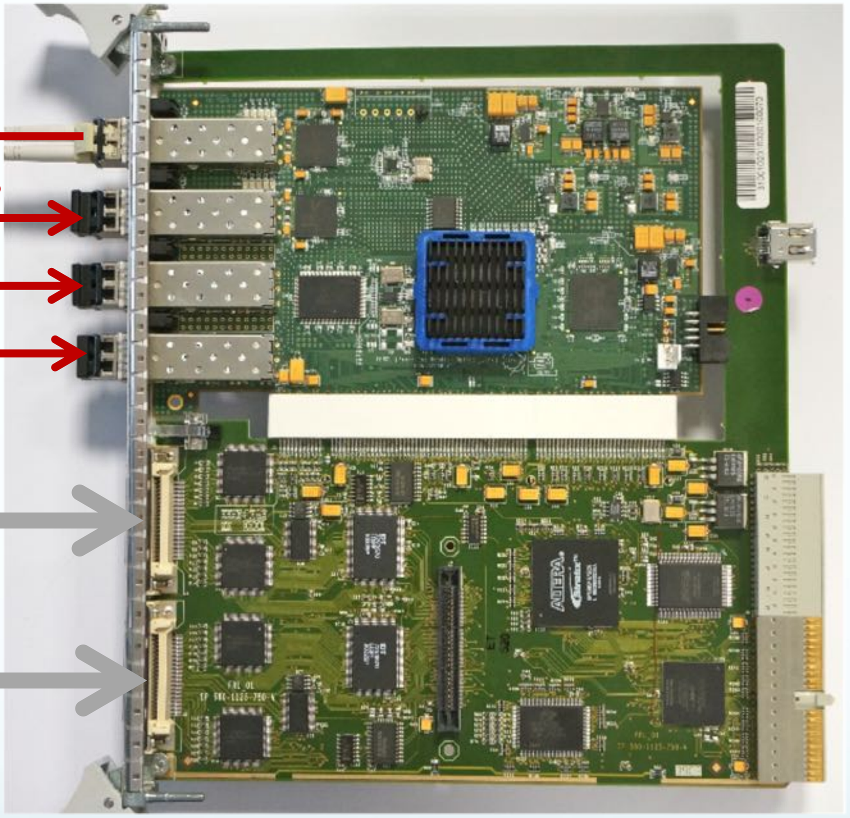
10 Gb/s SLINK Express

5 Gb/s SLINK Express

5 Gb/s SLINK Express

Custom data link in

SLINK-64 input
LVDS / copper



FEROL (Front End Readout Optical Link)

Input: 1x or 2x SLINK (copper)
1x or 2x 5Gb/s optical
1x 10Gb/s optical

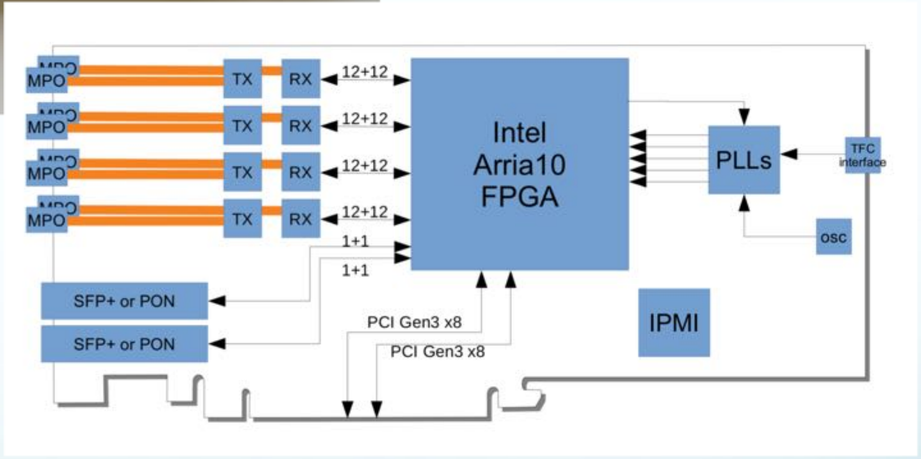
Output: 10 Gb/s Ethernet optical
TCP/IP sender in FPGA

PCIe40 – LHCb and ALICE Run-3

Custom data link in

Clock, control

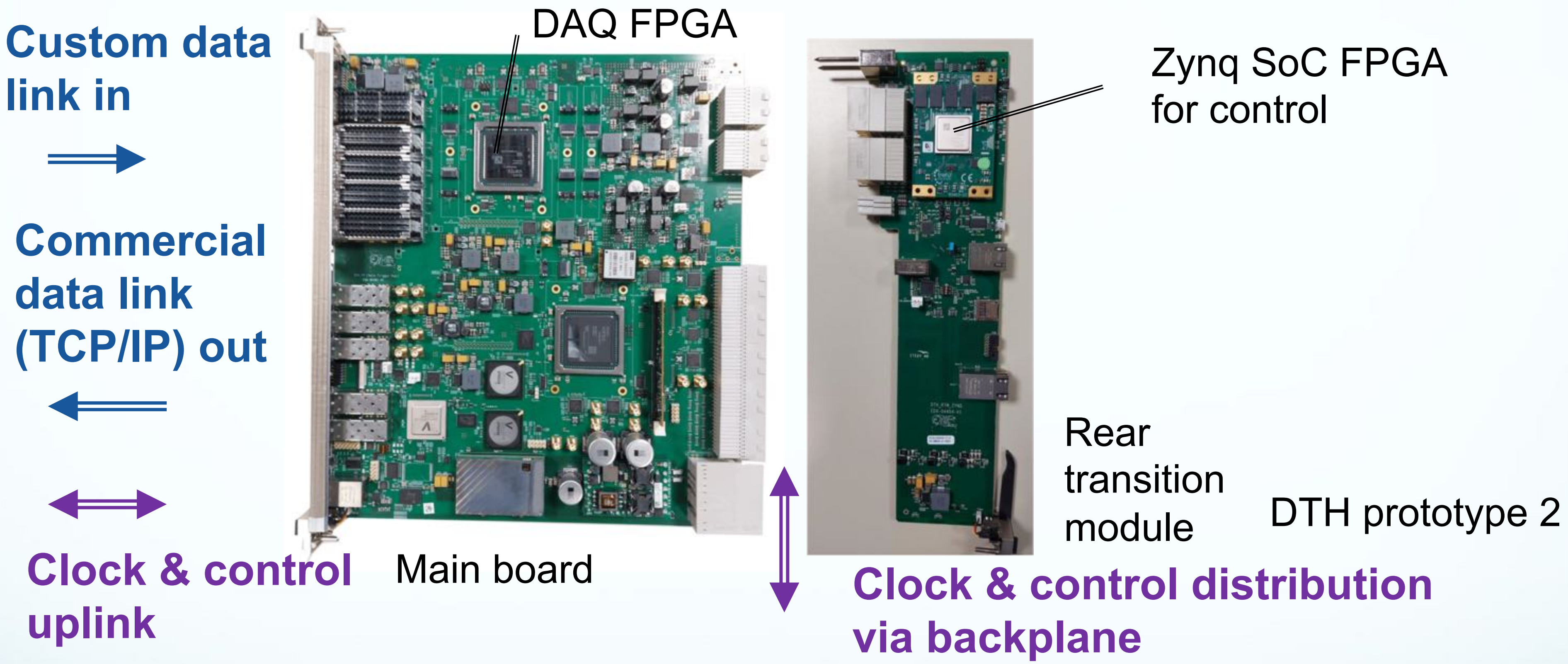
Direct Memory Access transfer to host memory



- 48 bidirectional links running at up to 10 Gbits/s each (minipods)
- 2 bidirectional links running at up to 10 Gbits/s devoted to time distribution (can use SFP+ or 10G PON devices)
- Sustained 112 Gbits/s interface with CPU through PCIe

J.P. Cachemiche, ACES 2018

CMS DTH (DAQ and Timing Hub) for HL-LHC (2029+)


Custom data link in
Commercial data link (TCP/IP) out
Clock & control uplink
DAQ FPGA
Main board
Zynq SoC FPGA for control
Rear transition module
DTH prototype 2
Clock & control distribution via backplane

- **ATCA** board using Xilinx Virtex Ultrascale + FPGAs
- One or two DAQ units per board
  - Up to 24 inputs at 25 Gb/s
  - **5x 100 Gb/s** Ethernet to commercial network
  - TCP/IP in FPGA
- Board contains switch for control network

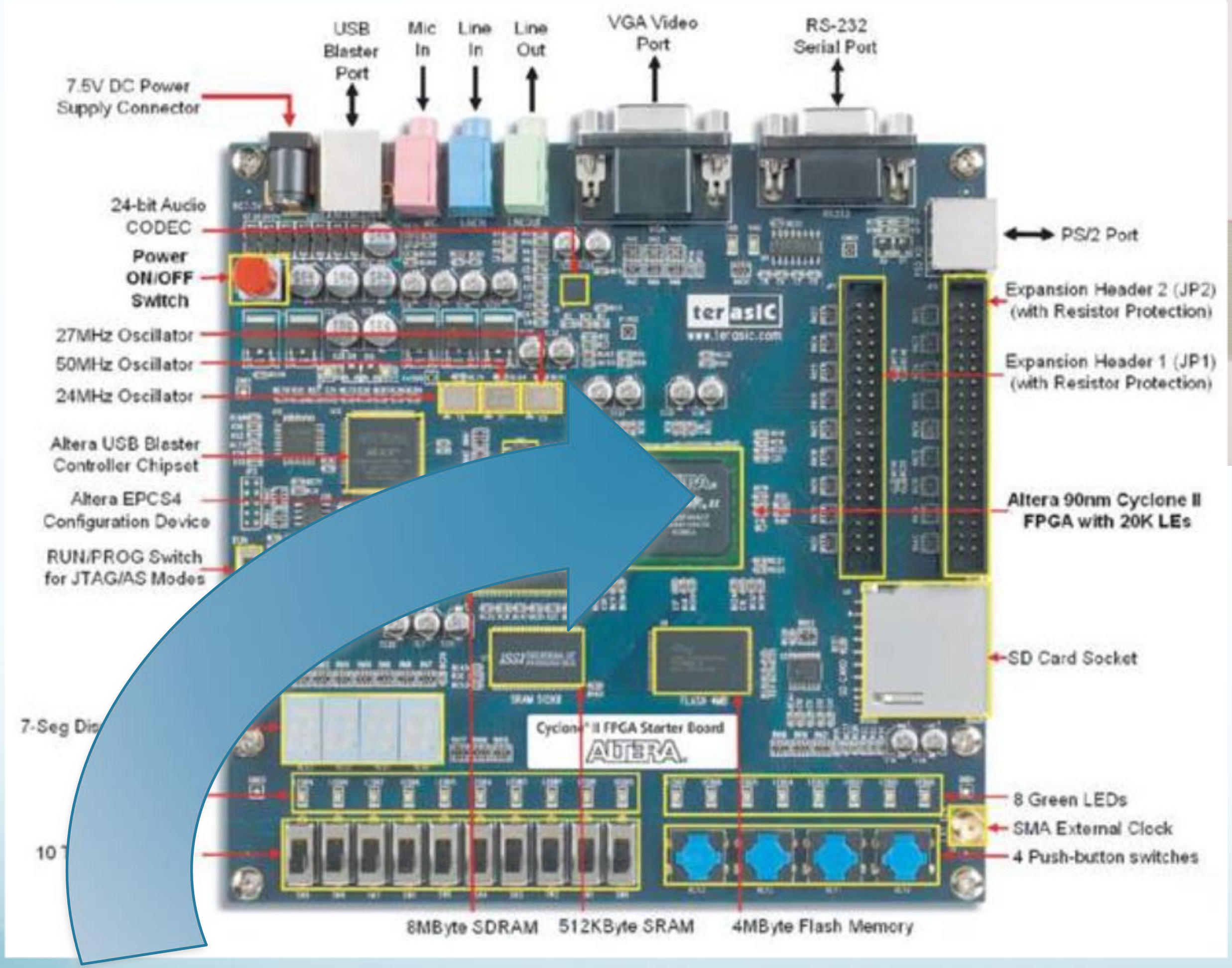
FPGAs in other domains

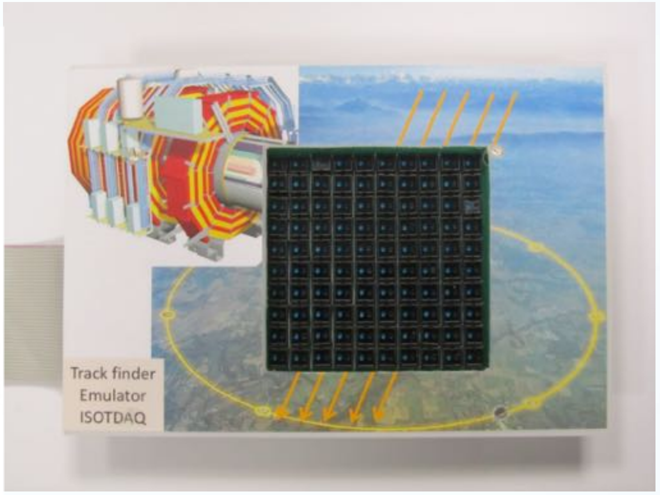
- Machine Learning Inferencing
- Automotive Driver Assist (Image Processing)
- 5G Wireless
- Medical imaging
- Speech recognition
- Cryptography
- Bioinformatics (Genome sequencing)
- Aerospace / Defense
- (Bitcoin mining)

- ASIC Prototyping
- Compute accelerators
  - Accelerator cards

  - Server processors w. FPGA
  - Financial
  - Video transcoding
  - …

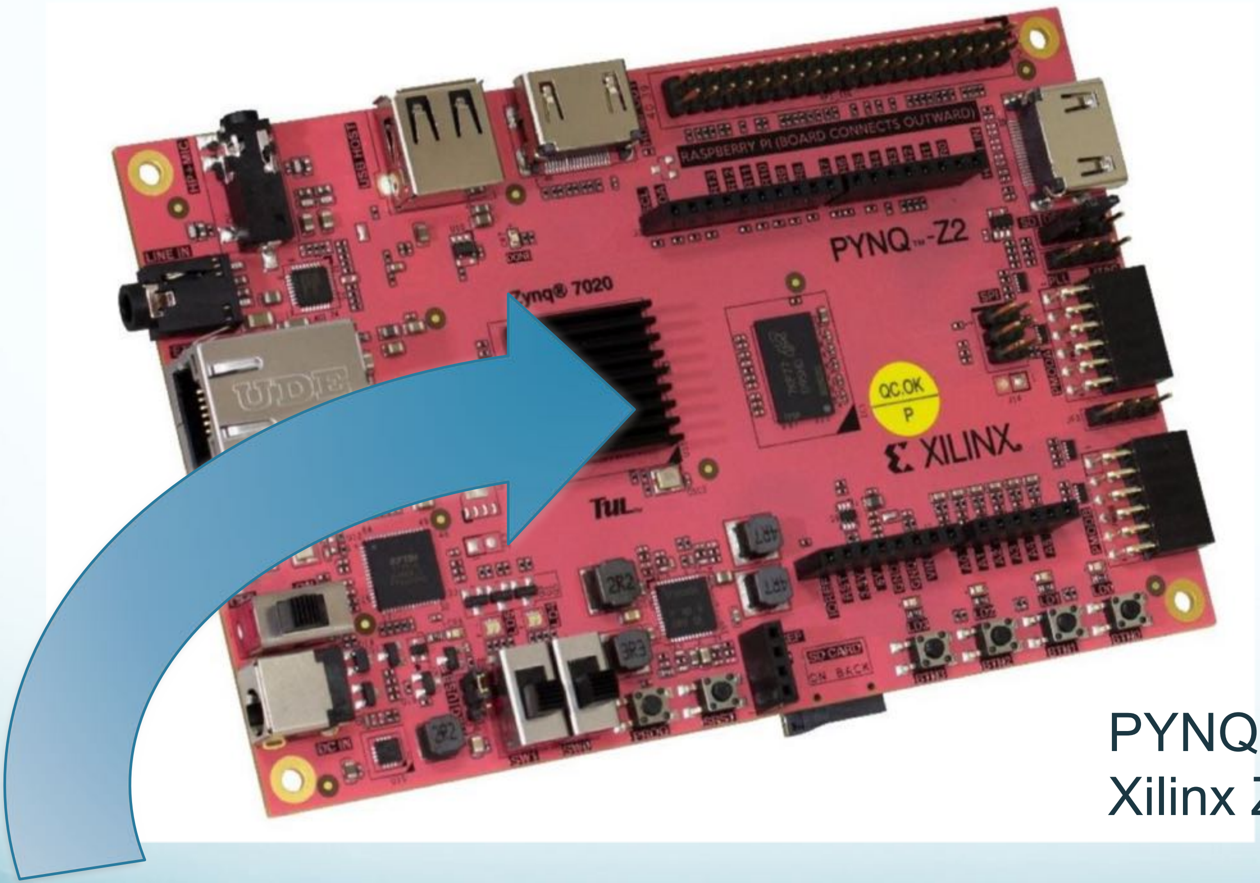
Lab Session 5: Programming an FPGA





You are going to design the digital electronics inside this FPGA !

Lab Session 13: System-on-a-chip FPGA



PYNQ-Z2 board
Xilinx Zynq w. dual-core ARM

Design the digital electronics and software in this SoC FPGA!

Thank you

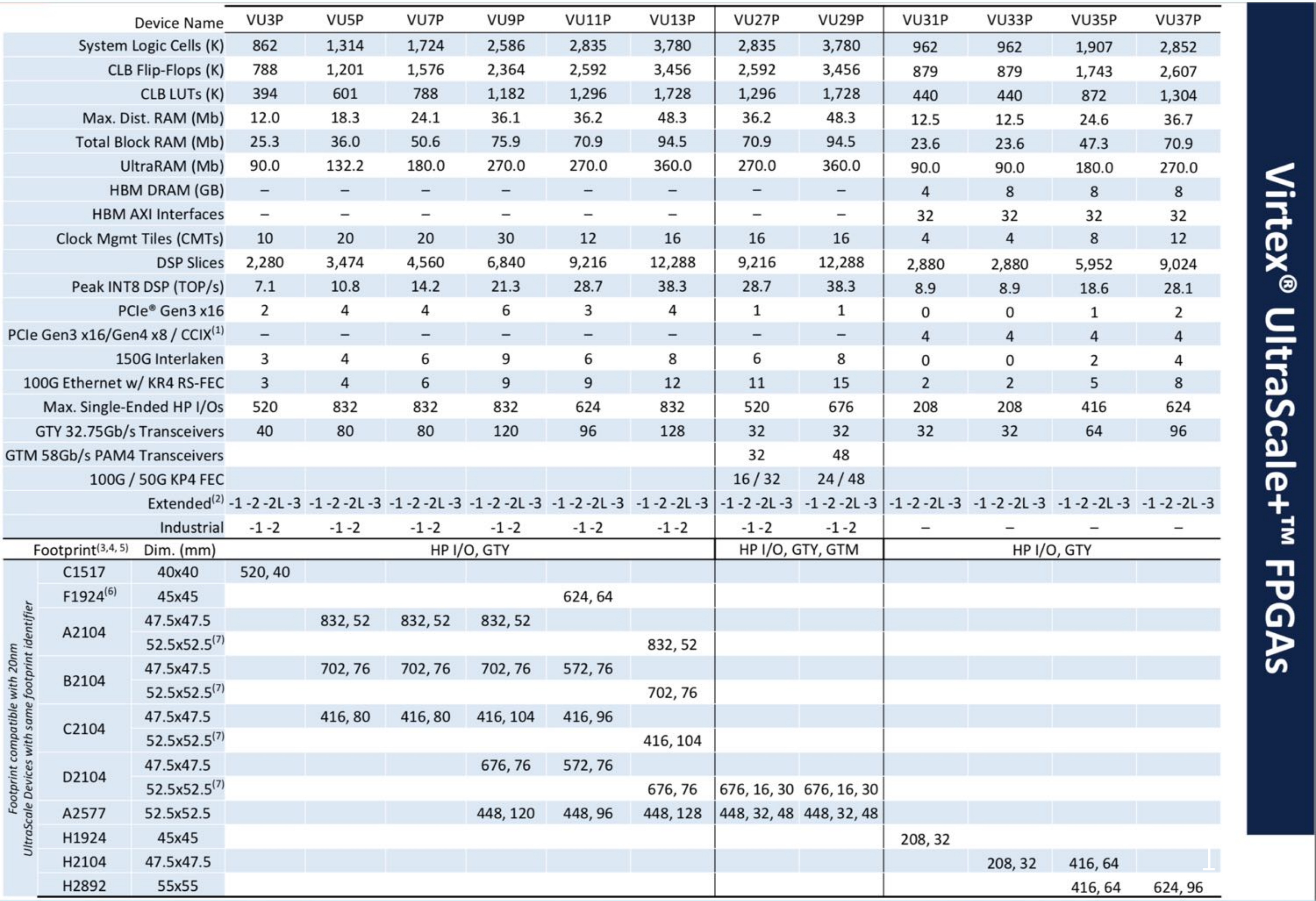
Acknowledgement

- Parts of this lecture are based on material by Clive Maxfield, author of several books on FPGAs. Many thanks for his kind permission to use his material!

Re-use

Reference Material

Top-of-the-line Xilinx devices



Virtex® UltraScale+™ FPGAs

Device Name	VU3P	VU5P	VU7P	VU9P	VU11P	VU13P	VU27P	VU29P	VU31P	VU33P	VU35P	VU37P
System Logic Cells (K)	862	1,314	1,724	2,586	2,835	3,780	2,835	3,780	962	962	1,907	2,852
CLB Flip-Flops (K)	788	1,201	1,576	2,364	2,592	3,456	2,592	3,456	879	879	1,743	2,607
CLB LUTs (K)	394	601	788	1,182	1,296	1,728	1,296	1,728	440	440	872	1,304
Max. Dist. RAM (Mb)	12.0	18.3	24.1	36.1	36.2	48.3	36.2	48.3	12.5	12.5	24.6	36.7
Total Block RAM (Mb)	25.3	36.0	50.6	75.9	70.9	94.5	70.9	94.5	23.6	23.6	47.3	70.9
UltraRAM (Mb)	90.0	132.2	180.0	270.0	270.0	360.0	270.0	360.0	90.0	90.0	180.0	270.0
HBM DRAM (GB)	–	–	–	–	–	–	–	–	4	8	8	8
HBM AXI Interfaces	–	–	–	–	–	–	–	–	32	32	32	32
Clock Mgmt Tiles (CMTs)	10	20	20	30	12	16	16	16	4	4	8	12
DSP Slices	2,280	3,474	4,560	6,840	9,216	12,288	9,216	12,288	2,880	2,880	5,952	9,024
Peak INT8 DSP (TOP/s)	7.1	10.8	14.2	21.3	28.7	38.3	28.7	38.3	8.9	8.9	18.6	28.1
PCIe® Gen3 x16	2	4	4	6	3	4	1	1	0	0	1	2
PCIe Gen3 x16/Gen4 x8 / CCIX(1)	–	–	–	–	–	–	–	–	4	4	4	4
150G Interlaken	3	4	6	9	6	8	6	8	0	0	2	4
100G Ethernet w/ KR4 RS-FEC	3	4	6	9	9	12	11	15	2	2	5	8
Max. Single-Ended HP I/Os	520	832	832	832	624	832	520	676	208	208	416	624
GTY 32.75Gb/s Transceivers	40	80	80	120	96	128	32	32	32	32	64	96
GTM 58Gb/s PAM4 Transceivers							32	48				
100G / 50G KP4 FEC							16 / 32	24 / 48				
Extended(2)	-1 -2 -2L -3	-1 -2 -2L -3	-1 -2 -2L -3	-1 -2 -2L -3	-1 -2 -2L -3	-1 -2 -2L -3	-1 -2 -2L -3	-1 -2 -2L -3	-1 -2 -2L -3	-1 -2 -2L -3	-1 -2 -2L -3	-1 -2 -2L -3
Industrial	-1 -2	-1 -2	-1 -2	-1 -2	-1 -2	-1 -2	-1 -2	-1 -2	–	–	–	–

Footprint compatible with 20nm UltraScale Devices with same footprint identifier

Footprint(3,4,5)	Dim. (mm)	VU3P	VU5P	VU7P	VU9P	VU11P	VU13P	VU27P	VU29P	VU31P	VU33P	VU35P	VU37P
		HP I/O, GTY						HP I/O, GTY, GTM		HP I/O, GTY			
C1517	40x40	520, 40											
F1924(6)	45x45					624, 64							
A2104	47.5x47.5		832, 52	832, 52	832, 52								
	52.5x52.5(7)						832, 52						
B2104	47.5x47.5		702, 76	702, 76	702, 76	572, 76							
	52.5x52.5(7)						702, 76						
C2104	47.5x47.5		416, 80	416, 80	416, 104	416, 96							
	52.5x52.5(7)						416, 104						
D2104	47.5x47.5				676, 76	572, 76							
	52.5x52.5(7)						676, 76	676, 16, 30	676, 16, 30				
A2577	52.5x52.5				448, 120	448, 96	448, 128	448, 32, 48	448, 32, 48				
H1924	45x45									208, 32			
H2104	47.5x47.5										208, 32	416, 64	
H2892	55x55											416, 64	624, 96

intel

Intel® Stratix® 10

INTEL® STRATIX® 10 GX/SX PRODUCT TABLE

	PRODUCT LINE	GX 400 SX 400	GX 650 SX 650	GX 850 SX 850	GX 1100 SX 1100	GX 1650 SX 1650	GX 2100 SX 2100	GX 2500 SX 2500	GX 2800 SX 2800	GX 4500 SX 4500	GX 5500 SX 5500
Resources	Logic elements (LEs)[1]	378,000	612,000	841,000	1,092,000	1,624,000	2,005,000	2,422,000	2,753,000	4,463,000	5,510,000
	Adaptive logic modules (ALMs)	128,160	207,360	284,960	370,080	550,540	679,680	821,150	933,120	1,512,820	1,867,680
	ALM registers	512,640	829,440	1,139,840	1,480,320	2,202,160	2,718,720	3,284,600	3,732,480	6,051,280	7,470,720
	Hyper-Registers from Intel® HyperFlex™ FPGA architecture	Millions of Hyper-Registers distributed throughout the monolithic FPGA fabric									
	Programmable clock trees synthesizable	Hundreds of synthesizable clock trees									
	M20K memory blocks	1,537	2,489	3,477	4,401	5,851	6,501	9,963	11,721	7,033	7,033
	M20K memory size (Mb)	30	49	68	86	114	127	195	229	137	137
	MLAB memory size (Mb)	2	3	4	6	8	11	13	15	23	29
	Variable-precision digital signal processing (DSP) blocks	648	1,152	2,016	2,520	3,145	3,744	5,011	5,760	1,980	1,980
	18 x 19 multipliers	1,296	2,304	4,032	5,040	6,290	7,488	10,022	11,520	3,960	3,960
	Peak fixed-point performance (TMACS)[2]	2.6	4.6	8.1	10.1	12.6	15.0	20.0	23.0	7.9	7.9
	Peak floating-point performance (TFLOPS)[3]	1.0	1.8	3.2	4.0	5.0	6.0	8.0	9.2	3.2	3.2
I/O and Architectural Features	Secure device manager	AES-256/SHA-256 bitsream encryption/authentication, physically unclonable function (PUF), ECDSA 256/384 boot code authentication, side channel attack protection									
	Hard processor system[4]	Quad-core 64 bit ARM* Cortex*-A53 up to 1.5 GHz with 32 KB I/D cache, NEON* coprocessor, 1 MB L2 cache, direct memory access (DMA), system memory management unit, cache coherency unit, hard memory controllers, USB 2.0 x2, 1G EMAC x3, UART x2, SPI x4, I^2C x5, general-purpose timers x7, watchdog timer x4									
	Maximum user I/O pins	392	400	736	736	704	704	1160	1160	1640	1640
	Maximum LVDS pairs 1.6 Gbps (RX or TX)	192	192	360	360	336	336	576	576	816	816
	Total full duplex transceiver count	24	48	48	48	96	96	96	96	24	24
	GXT full duplex transceiver count (up to 30 Gbps)	16	32	32	32	64	64	64	64	16	16
	GX full duplex transceiver count (up to 17.4 Gbps)	8	16	16	16	32	32	32	32	8	8
	PCI Express* (PCIe*) hard intellectual property (IP) blocks (Gen3 x16)	1	2	2	2	4	4	4	4	1	1
	Memory devices supported	DDR4, DDR3, DDR2, DDR, QDR II, QDR II+, RLDRAM II, RLDRAM 3, HMC, MoSys									
Package Options and I/O Pins: General-Purpose I/O (GPIO) Count, High-Voltage I/O Count, LVDS Pairs, and Transceiver Count[5]											
	F1152 pin (35 mm x 35 mm, 1.0 mm pitch)	392,8,192,24	392,8,192,24	–	–	–	–	–	–	–	–
	F1760 pin (42.5 mm x 42.5 mm, 1.0 mm pitch)	–	400,16,192,48	–	–	–	–	–	–	–	–
	F1760 pin (42.5 mm x 42.5 mm, 1.0 mm pitch)	–	–	688,16,336,48	688,16,336,48	688,16,336,48	688,16,336,48	688,16,336,48	688,16,336,48	–	–
	F2112 pin (47.5 mm x 47.5 mm, 1.0 mm pitch)	–	–	736,16,360,48	736,16,360,48	–	–	–	–	–	–
	F2397 pin (50 mm x 50 mm, 1.0 mm pitch)	–	–	–	–	704,32,336,96	704,32,336,96	704,32,336,96	704,32,336,96	–	–
	F2912 pin (55 mm x 55 mm, 1.0 mm pitch)	–	–	–	–	–	–	1160,8,576,24	1160,8,576,24	1640,8,816,24	1640,8,816,24

Intel® Stratix® 10

INTEL® STRATIX® 10 TX PRODUCT TABLE

	PRODUCT LINE	TX 1650		TX 2100		TX 2500			TX 2800		
Resources	Logic elements (LEs)[1]	1,679,000		2,073,000		2,422,000			2,753,000		
	Adaptive logic modules (ALMs)	569,200		702,720		821,150			933,120		
	ALM registers	2,276,800		2,810,880		3,284,600			3,732,480		
	Hyper-Registers from Intel® Hyperflex™ FPGA architecture	Millions of Hyper-Registers distributed throughout the monolithic FPGA fabric									
	Programmable clock trees synthesizable	Hundreds of synthesizable clock trees									
	eSRAM memory blocks	2		2		–			–		
	eSRAM memory size (Mb)	90		90		–			–		
	M20K memory blocks	6,162		6,847		9,963			11,721		
	M20K memory size (Mb)	120		134		195			229		
	MLAB memory size (Mb)	9		11		13			15		
	Variable-precision digital signal processing (DSP) blocks	3,326		3,960		5,011			5,760		
	18 x 19 multipliers	6,652		7,920		10,022			11,520		
	Peak fixed-point performance (TMACS)[2]	13.3		15.8		20.0			23.0		
	Peak floating-point performance (TFLOPS)[3]	5.3		6.3		8.0			9.2		
I/O and Architectural Features	Secure device manager	AES-256/SHA-256 bitsream encryption/authentication, physically unclonable function (PUF), ECDSA 256/384 boot code authentication, side channel attack protection									
	Hard processor system	Quad-core 64-bit ARM* Cortex*-A53 up to 1.5 GHz with 32KB I/D cache, NEON* coprocessor, 1 MB L2 Cache, direct memory access (DMA), system memory management unit, cache coherency unit, hard memory controllers, USB 2.0 x2, 1G EMAC x3, UART x2, SPI x4, I^{2}C x5, general purpose timers x7, watchdog timer x4									
		–		–		Yes			Yes		
	Maximum user I/O pins	544	440	544	440	544	440	296	544	440	296
	Maximum LVDS pairs 1.6 Gbps (RX or TX)	264	216	264	216	264	216	144	264	216	144
	Total full duplex transceiver count	72	96	72	96	72	96	144	72	96	144
	GXE transceiver count - PAM-4 (up to 58 Gbps) or NRZ (up to 30 Gbps)	12 PAM-4 24 NRZ	36 PAM-4 72 NRZ	12 PAM-4 24 NRZ	36 PAM-4 72 NRZ	12 PAM-4 24 NRZ	36 PAM-4 72 NRZ	60 PAM-4 120 NRZ	12 PAM-4 24 NRZ	36 PAM-4 72 NRZ	60 PAM-4 120 NRZ
	GXT transceiver count - NRZ (up to 28.3 Gbps)	32	16	32	16	32	16	16	32	16	16
	GX transceiver count - NRZ (up to 17.4 Gbps)	16	8	16	8	16	8	8	16	8	8
	PCI Express* (PCIe*) hard intellectual property (IP) blocks (Gen3 x16)	2	1	2	1	2	1	1	2	1	1
	100G Ethernet MAC (no FEC) hard IP blocks	2	1	2	1	2	1	1	2	1	1
	100G Ethernet MAC + FEC hard IP blocks	4	12	4	12	4	12	20	4	12	20
	Memory devices supported	DDR4, DDR3, DDR2, DDR, QDR II, QDR II+, RLDRAM II, RLDRAM 3, HMC, MoSys									

Package Options and I/O Pins: General-Purpose I/O (GPIO) Count, High-Voltage I/O count, LVDS pairs, GXE (E-Tile) Transceiver Count, and GXT+GX (H-Tile) Transceiver Count[4]

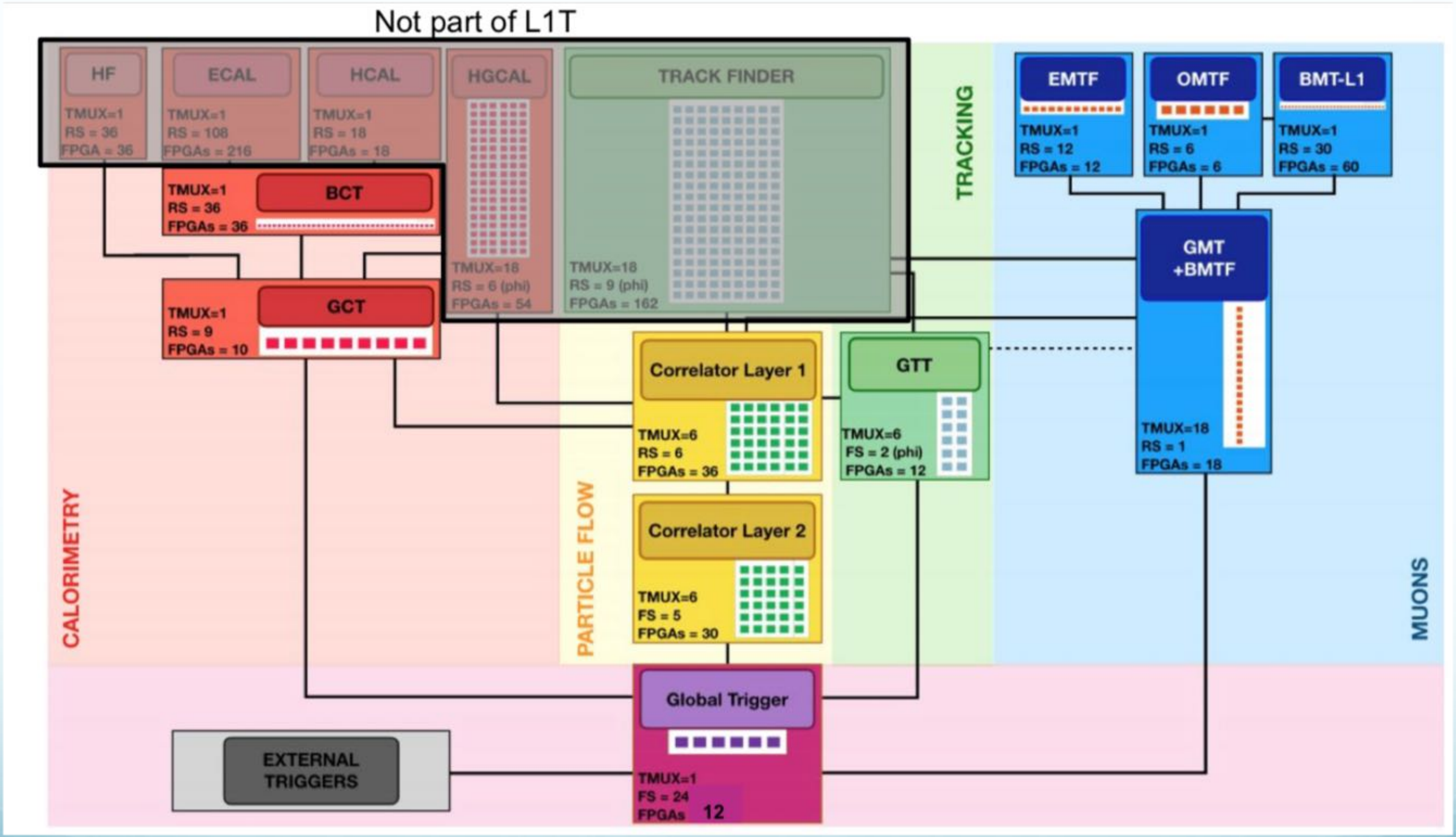
	TX 1650	TX 2100	TX 2500	TX 2800
F2112 pin (47.5 mm x 47.5 mm, 1.0 mm pitch)	544,16,264,24,48	544,16,264,24,48	544,16,264,24,48	544,16,264,24,48
F2397 pin (50 mm x 50 mm, 1.0 mm pitch)	440,8,216,72,24	440,8,216,72,24	440,8,216,72,24	440,8,216,72,24
F2912 pin (55 mm x 55 mm, 1.0 mm pitch)	–	–	296,8,144,120,24	296,8,144,120,24

Intel® Stratix® 10

INTEL® STRATIX® 10 MX (DRAM SYSTEM-IN-PACKAGE) PRODUCT TABLE

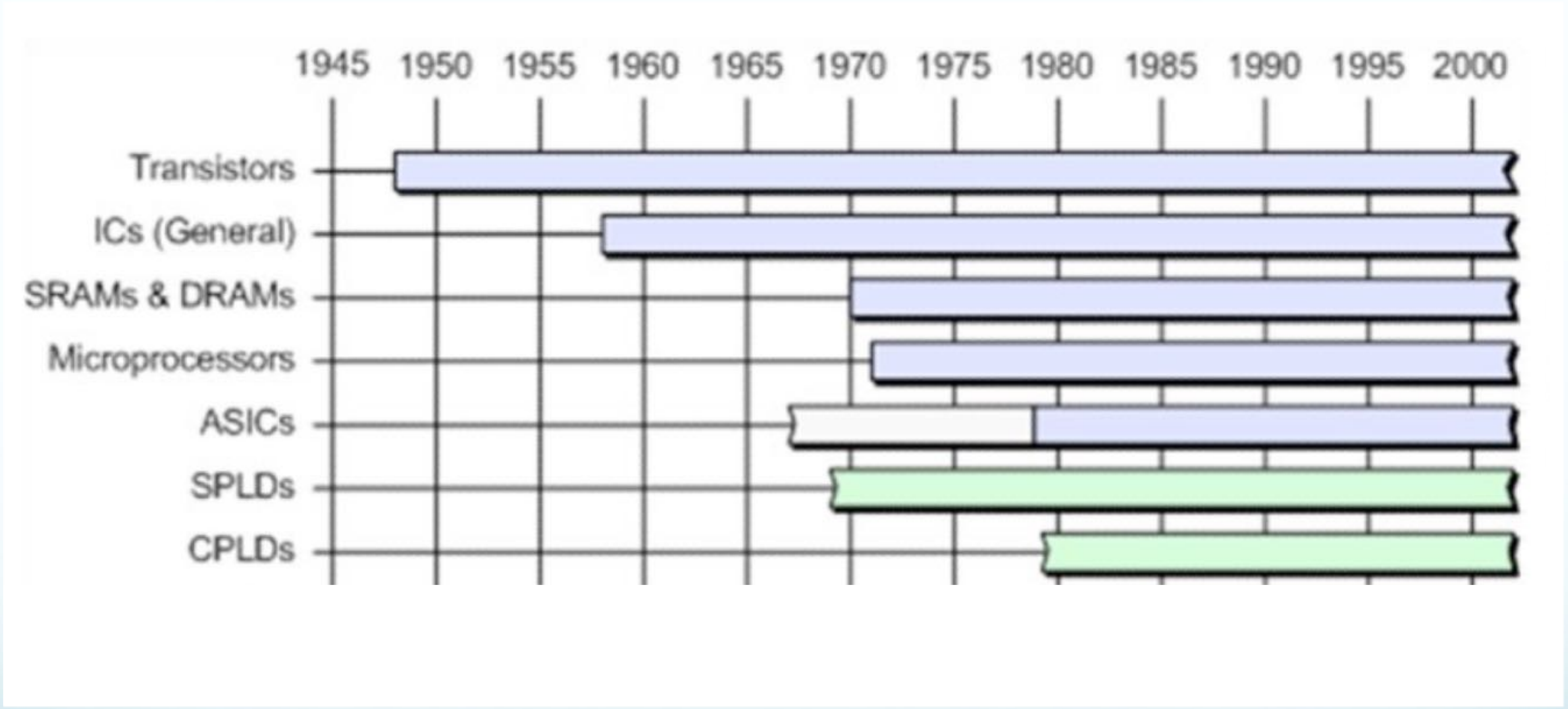
	PRODUCT LINE	MX 1100	MX 1650	MX 1650	MX 1650	MX 2100	MX 2100	MX 2100	MX 2100
Resources	Logic elements (LEs)[1]	1,092,000	1,679,000	1,679,000	1,679,000	2,073,000	2,073,000	2,073,000	2,073,000
	Adaptive logic modules (ALMs)	370,080	569,200	569,200	569,200	702,720	702,720	702,720	702,720
	ALM registers	1,480,320	2,276,800	2,276,800	2,276,800	2,810,880	2,810,880	2,810,880	2,810,880
	Hyper-Registers from Intel® Hyperflex™ FPGA architecture	Millions of Hyper-Registers distributed throughout the monolithic FPGA fabric							
	Programmable clock trees synthesizable	Hundreds of synthesizable clock trees							
	HBM2 high-bandwidth DRAM memory (GBytes)	3.25	8	16	8	8	8	16	8
	eSRAM memory blocks	1	2	2	2	2	2	2	2
	eSRAM memory size (Mb)	45	90	90	90	90	90	90	90
	M20K memory blocks	4,401	6,162	6,162	6,162	6,847	6,847	6,847	6,847
	M20K memory size (Mb)	86	120	120	120	134	134	134	134
	MLAB memory size (Mb)	6	9	9	9	11	11	11	11
	Variable-precision digital signal processing (DSP) blocks	2,520	3,326	3,326	3,326	3,960	3,960	3,960	3,960
	18 x 19 multipliers	5,040	6,652	6,652	6,652	7,920	7,920	7,920	7,920
	Peak fixed-point performance (TMACS)[2]	10.1	13.3	13.3	13.3	15.8	15.8	15.8	15.8
	Peak floating-point performance (TFLOPS)[3]	4.0	5.3	5.3	5.3	6.3	6.3	6.3	6.3
I/O and Architectural Features	Secure device manager	AES-256/SHA-256 bitstream encryption/authentication, physically unclonable function (PUF), ECDSA 256/384 boot code authentication, side channel attack protection							
	Hard processor system	Quad-core 64 bit ARM* Cortex*-A53 up to 1.5 GHz with 32 KB I/D cache, NEON* coprocessor, 1 MB L2 cache, direct memory access (DMA), system memory management unit, cache coherency unit, hard memory controllers, USB 2.0 x2, 1G EMAC x3, UART x2, serial peripheral interface (SPI) x4, I²C x5, general-purpose timers x7, watchdog timer x4							
		Yes	–	–	–	–	–	–	–
	Maximum user I/O pins	448	656	656	584	640	656	656	584
	LVDS pairs 1.6 Gbps (RX or TX)	216	312	312	288	312	312	312	288
	Total full duplex transceiver count	48	96	96	96	48	96	96	96
	GXE transceiver count - PAM4 (up to 58 Gbps) or NRZ (up to 30 Gbps)	0	0	0	72	0	0	0	72
	GXT transceiver count - NRZ (up to 28.3 Gbps)	32	64	64	16	32	64	64	16
	GX transceiver count - NRZ (up to 17.4 Gbps)	16	32	32	8	16	32	32	8
	PCI Express* (PCIe*) hard intellectual property (IP) blocks (Gen3 x16)	2	4	4	1	2	4	4	1
	100G Ethernet MAC (no FEC) hard IP blocks	2	4	4	1	2	4	4	1
	100G Ethernet MAC + FEC hard IP blocks	0	0	0	12	0	0	0	12
	Memory devices supported	DDR4, DDR3, DDR2, DDR, QDR II, QDR II+, RLDRAM II, RLDRAM 3, HMC, MoSys							
Package Options and I/O Pins: General-Purpose I/O (GPIO) Count, High-Voltage I/O Count, LVDS Pairs, and Transceiver Count[4, 5]									
F1760 pin (42.5 mm x 42.5 mm, 1.0 mm pitch)		448,16,216,48	–	–	–	–	–	–	–
F2597 pin (52.5 mm x 52.5 mm, 1.0mm pitch)		–	656, 32, 312, 96	656, 32, 312, 96	–	640, 16, 312, 48	656, 32, 312, 96	656, 32, 312, 96	–
F2912 pin (55 mm x 55 mm, 1.0 mm pitch)		–	–	–	584, 8, 288, 96	–	–	–	584, 8, 288, 96

FPGA count in CMS trigger for HL-LHC

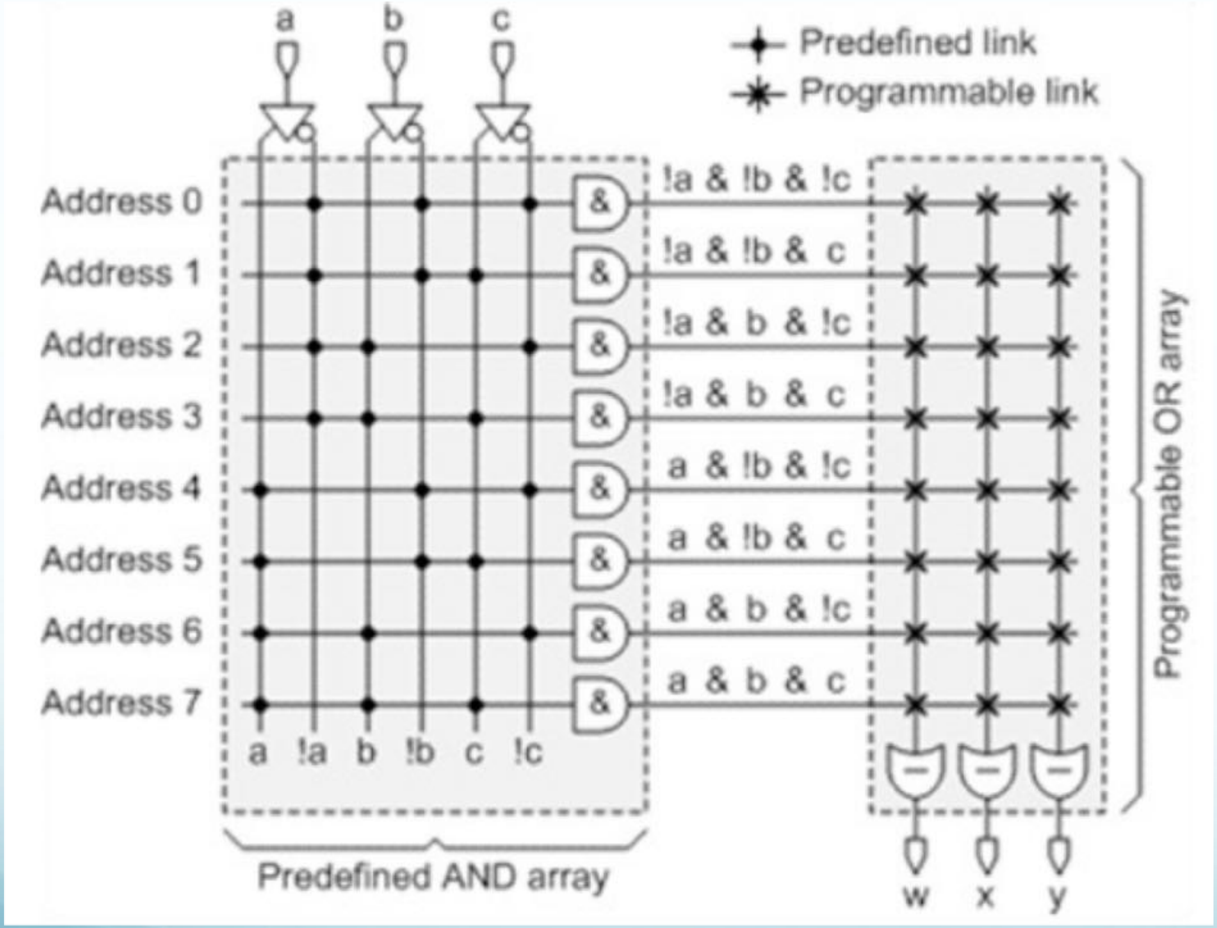


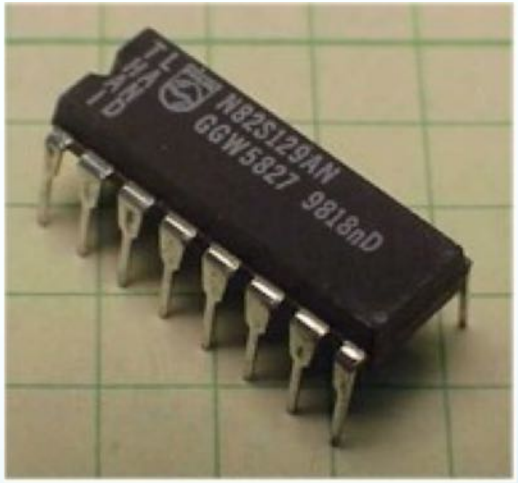
History

Long long time ago …



Simple Programmable Logic Devices (sPLDs)
a) Programmable Read Only Memory (PROMs)

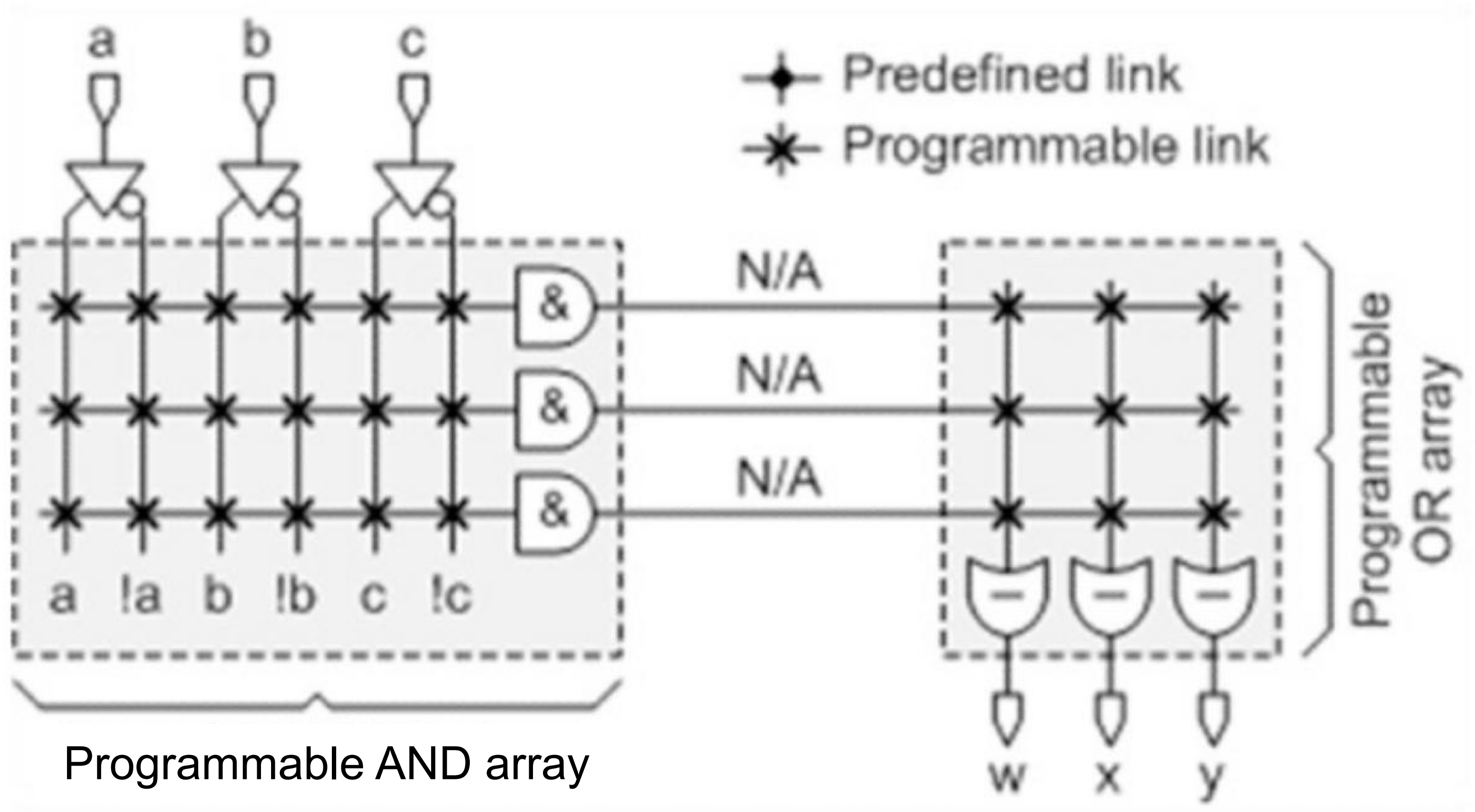




Late 60's

Unprogrammed PROM (Fixed AND Array, Programmable OR Array)

Simple Programmable Logic Devices (sPLDs)
b) Programmable Logic Arrays (PLAs)

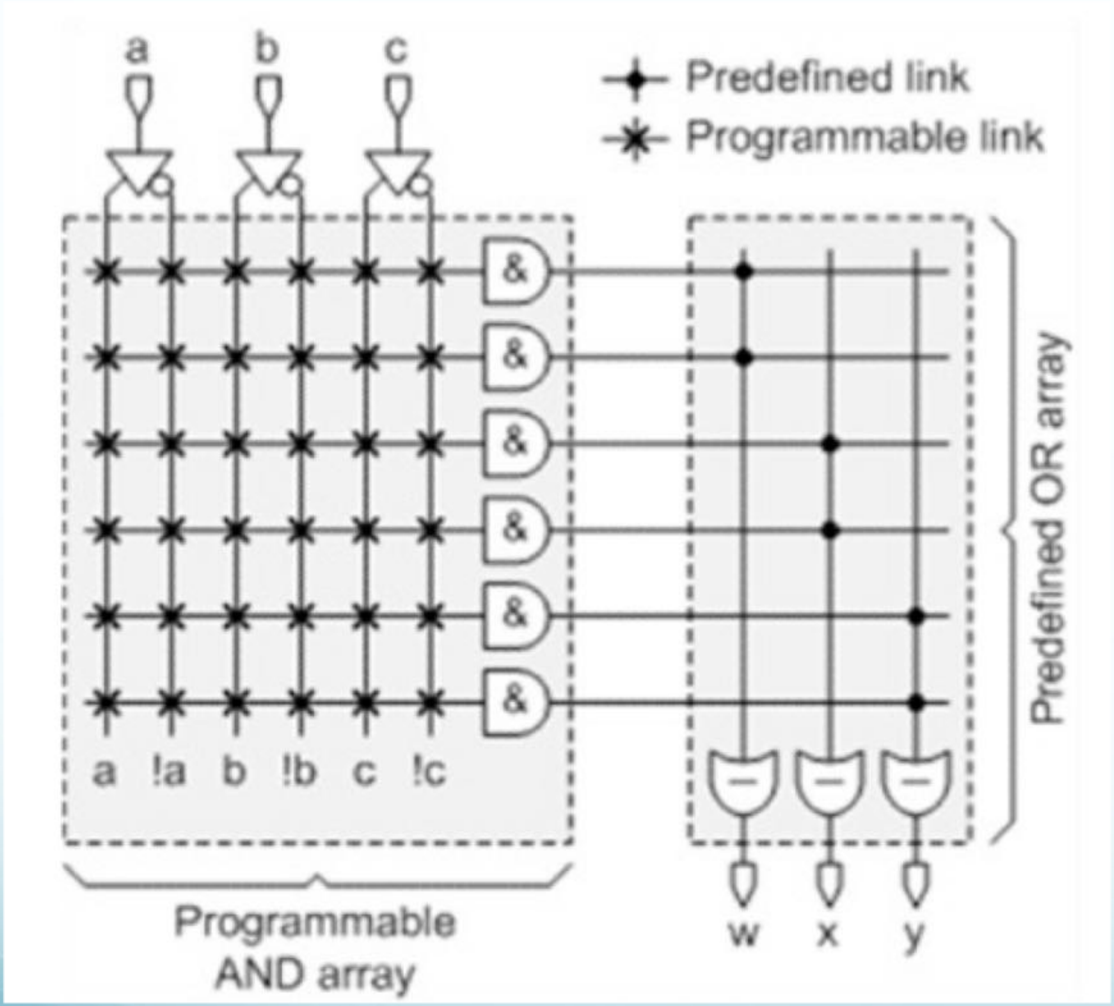


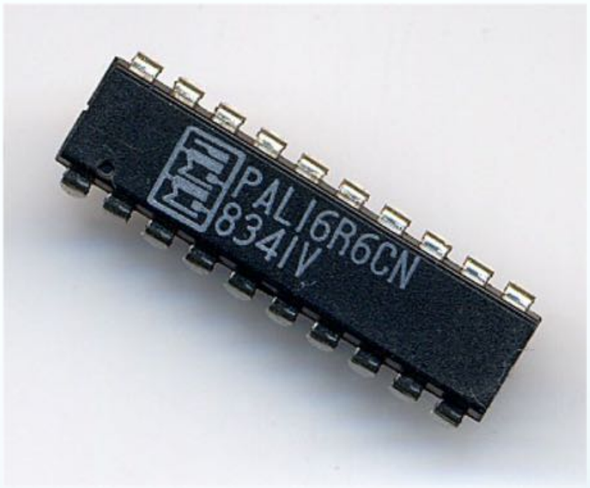
Unprogrammed PLA (Programmable AND and OR Arrays)

Most flexible
but slower

1975

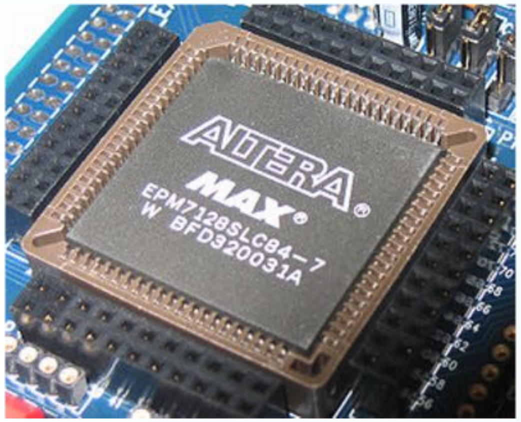
Simple Programmable Logic Devices (sPLDs)
c) Programmable Array Logic (PAL)

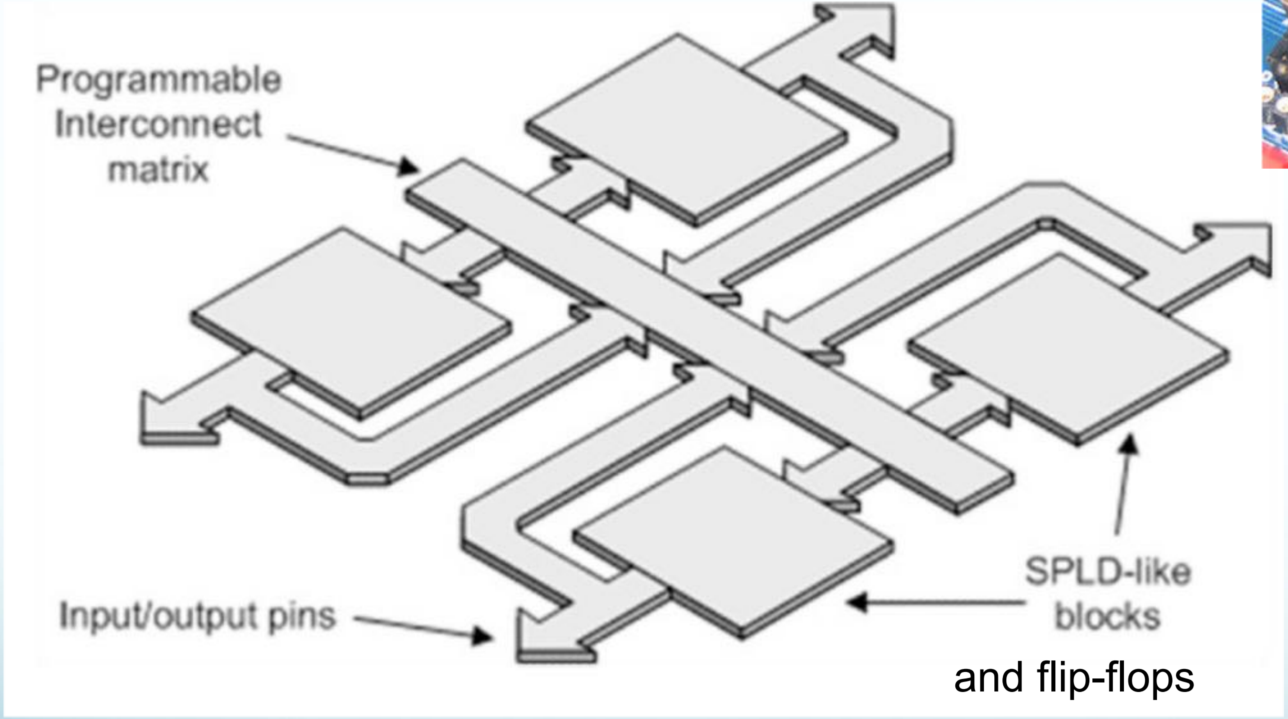




Unprogrammed PAL (Programmable AND Array, Fixed OR Array)

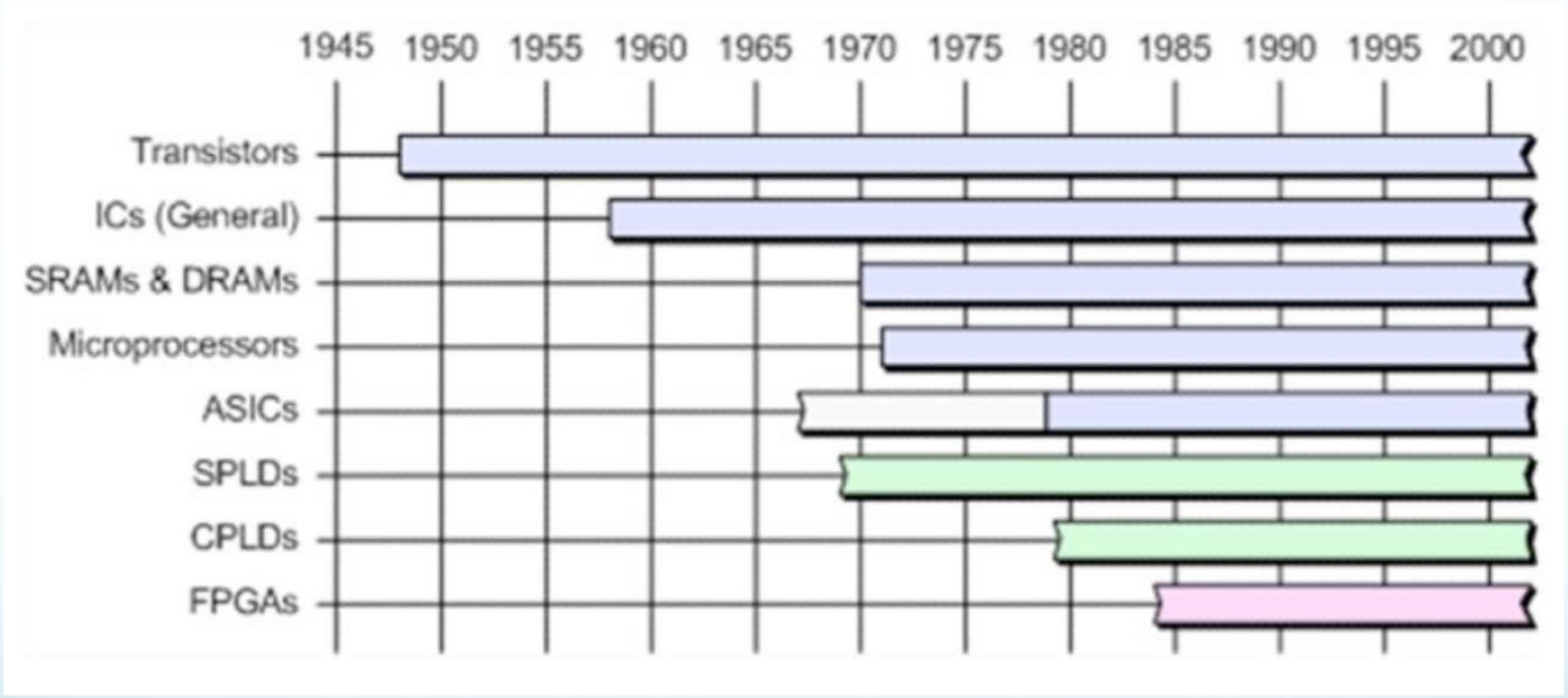
Complex PLDs (CPLDs)





Coarse grained
100’s of blocks, restrictive structure
(EE)PROM based

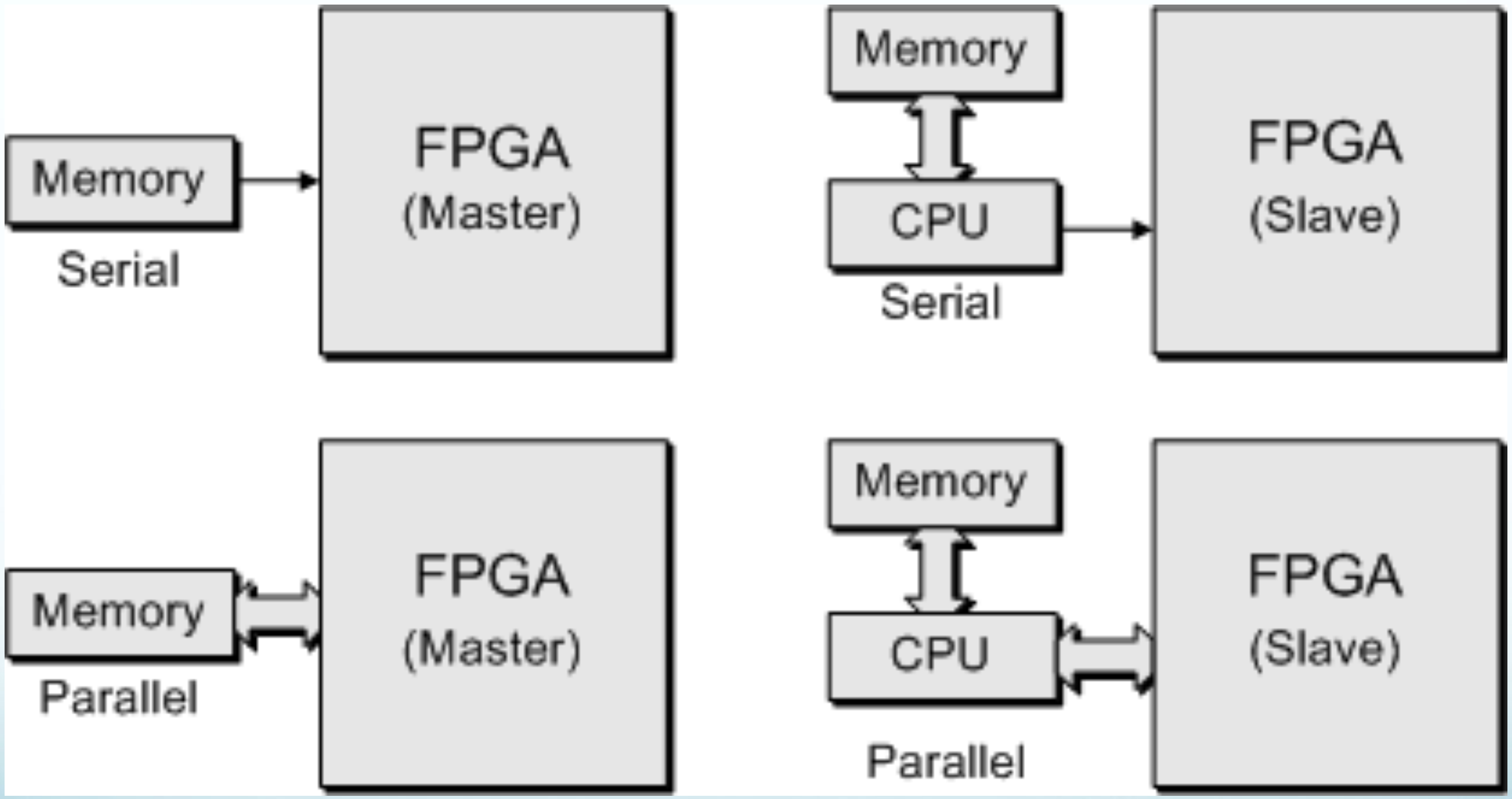
FPGAs …



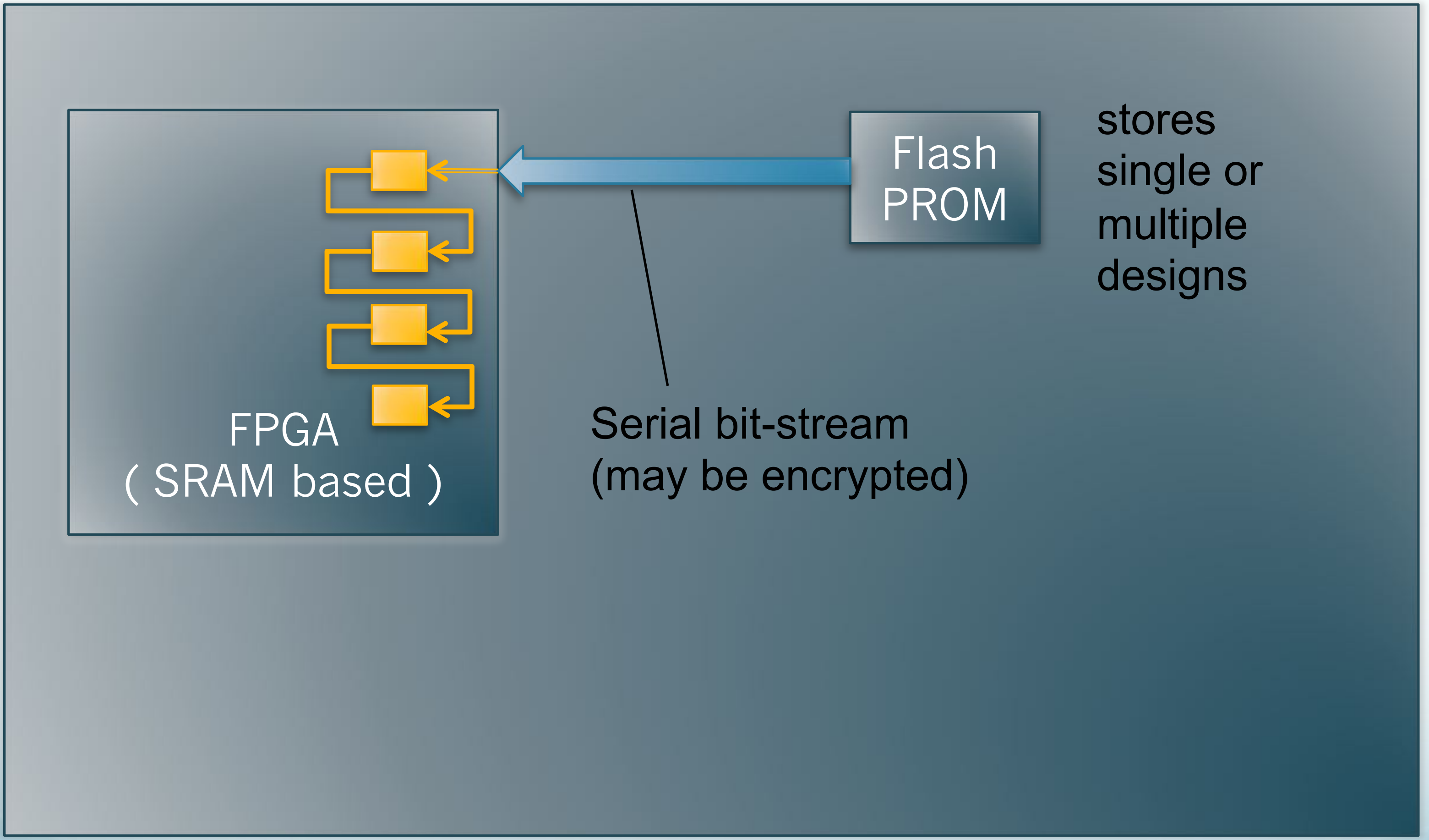
1945
1950
1955
1960
1965
1970
1975
1980
1985
1990
1995
2000
Transistors
ICs (General)
SRAMs & DRAMs
Microprocessors
ASICs
SPLDs
CPLDs
FPGAs

Timing

Design Considerations (SRAM Config.)



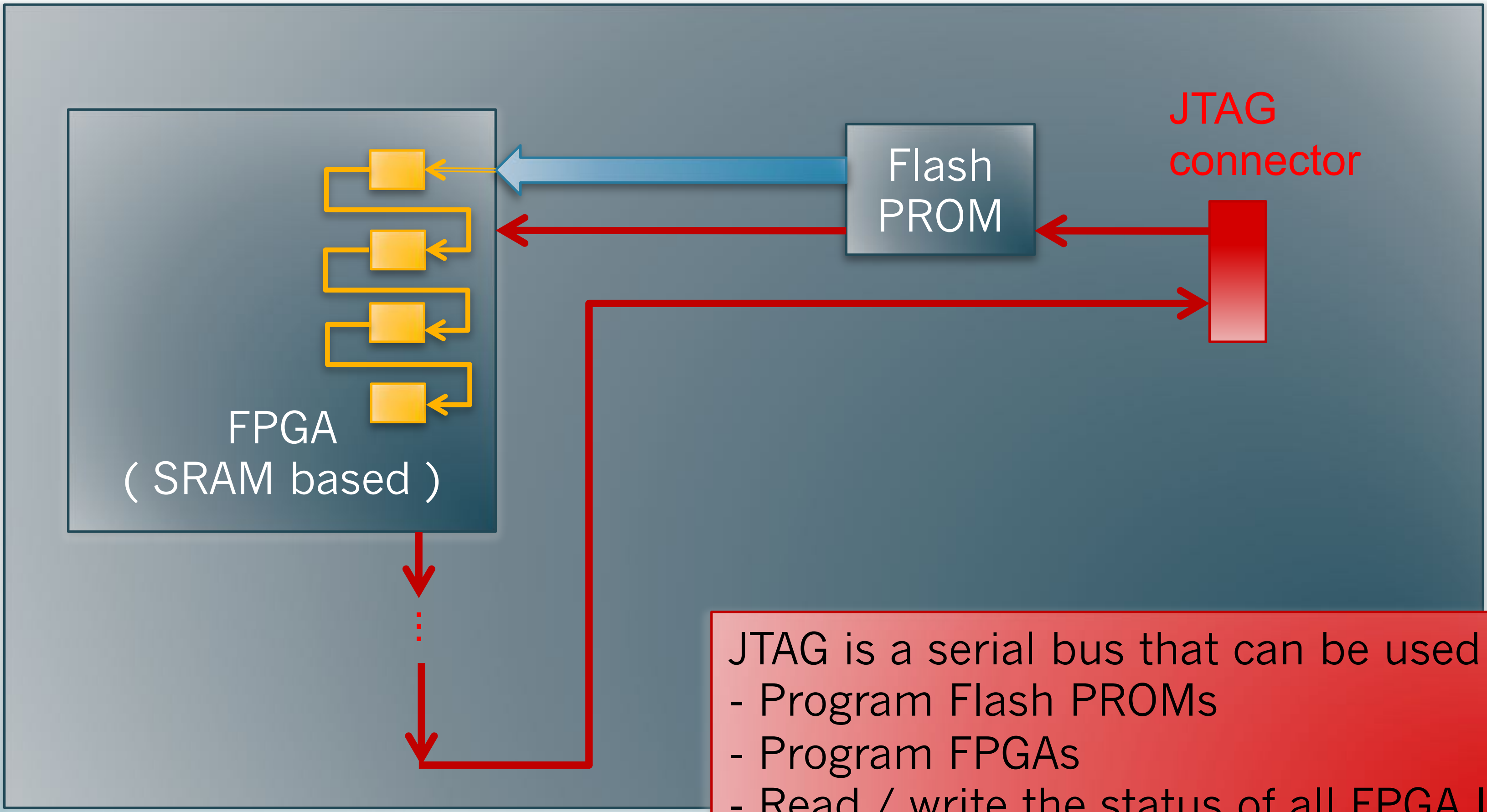
Configuration at power-up



Typical FPGA configuration time: milliseconds

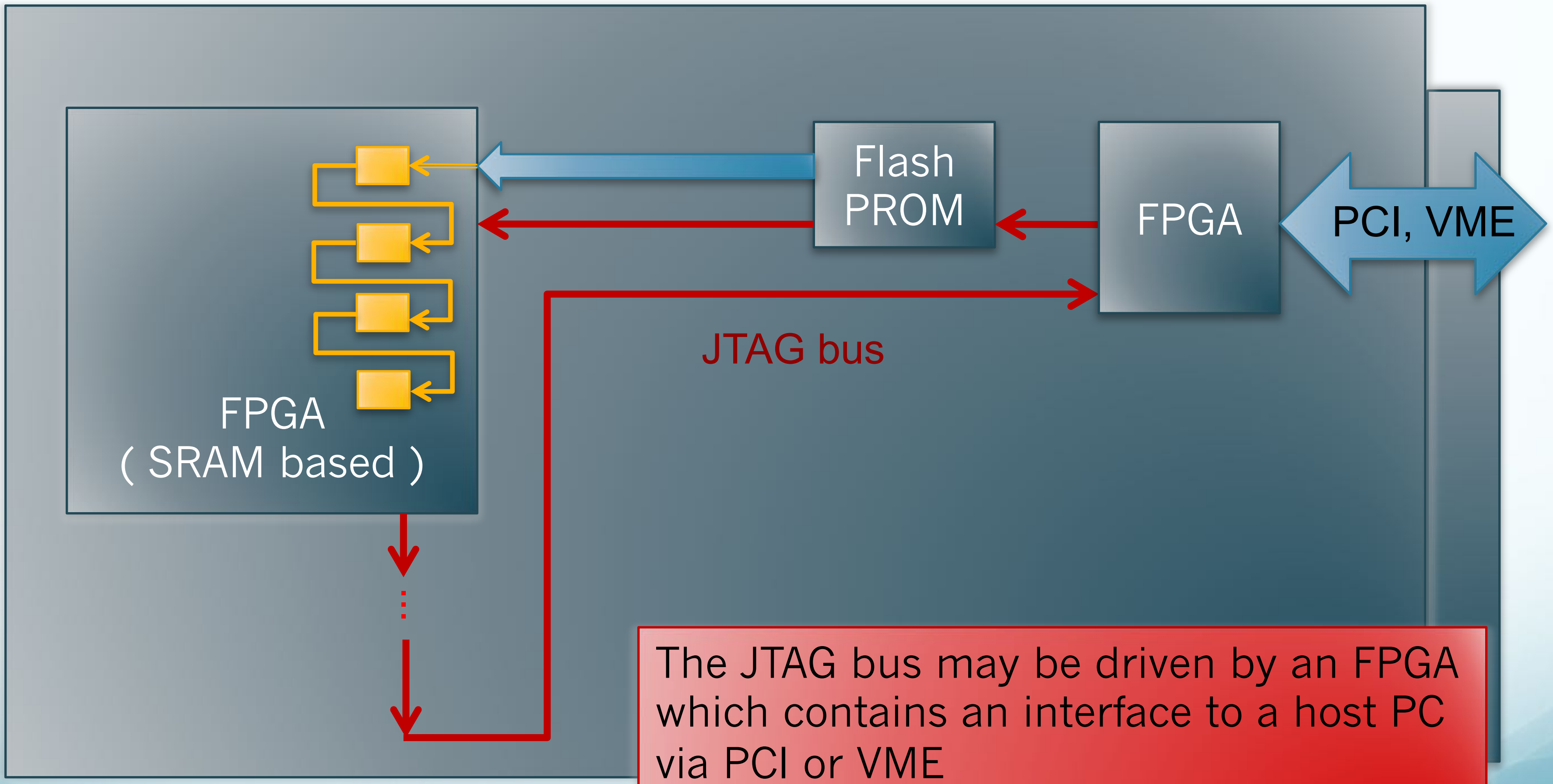
Programming via JTAG

Joint Test Action Group



JTAG is a serial bus that can be used to
- Program Flash PROMs
- Program FPGAs
- Read / write the status of all FPGA I/Os
  (= Boundary scan)

Remote programming



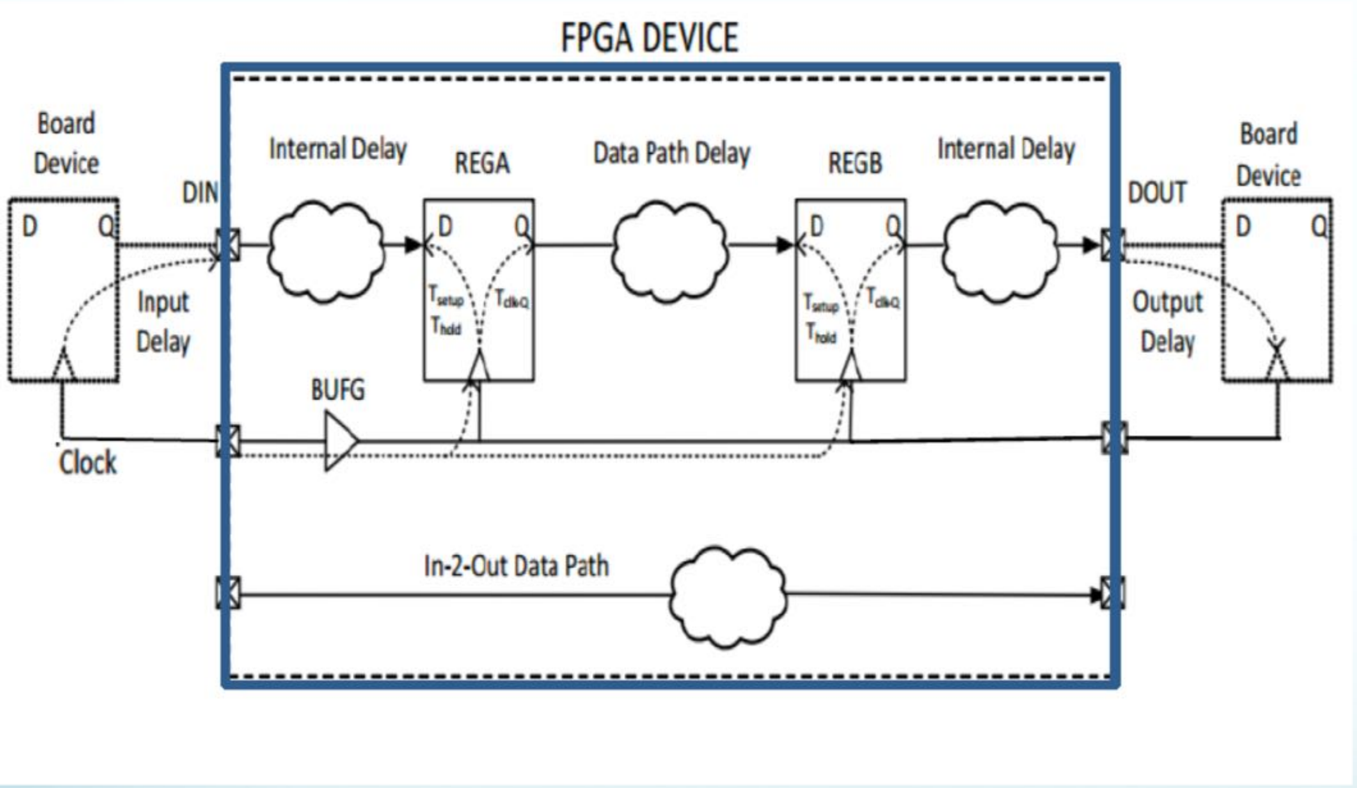
The JTAG bus may be driven by an FPGA which contains an interface to a host PC via PCI or VME

gateware can then be updated remotely

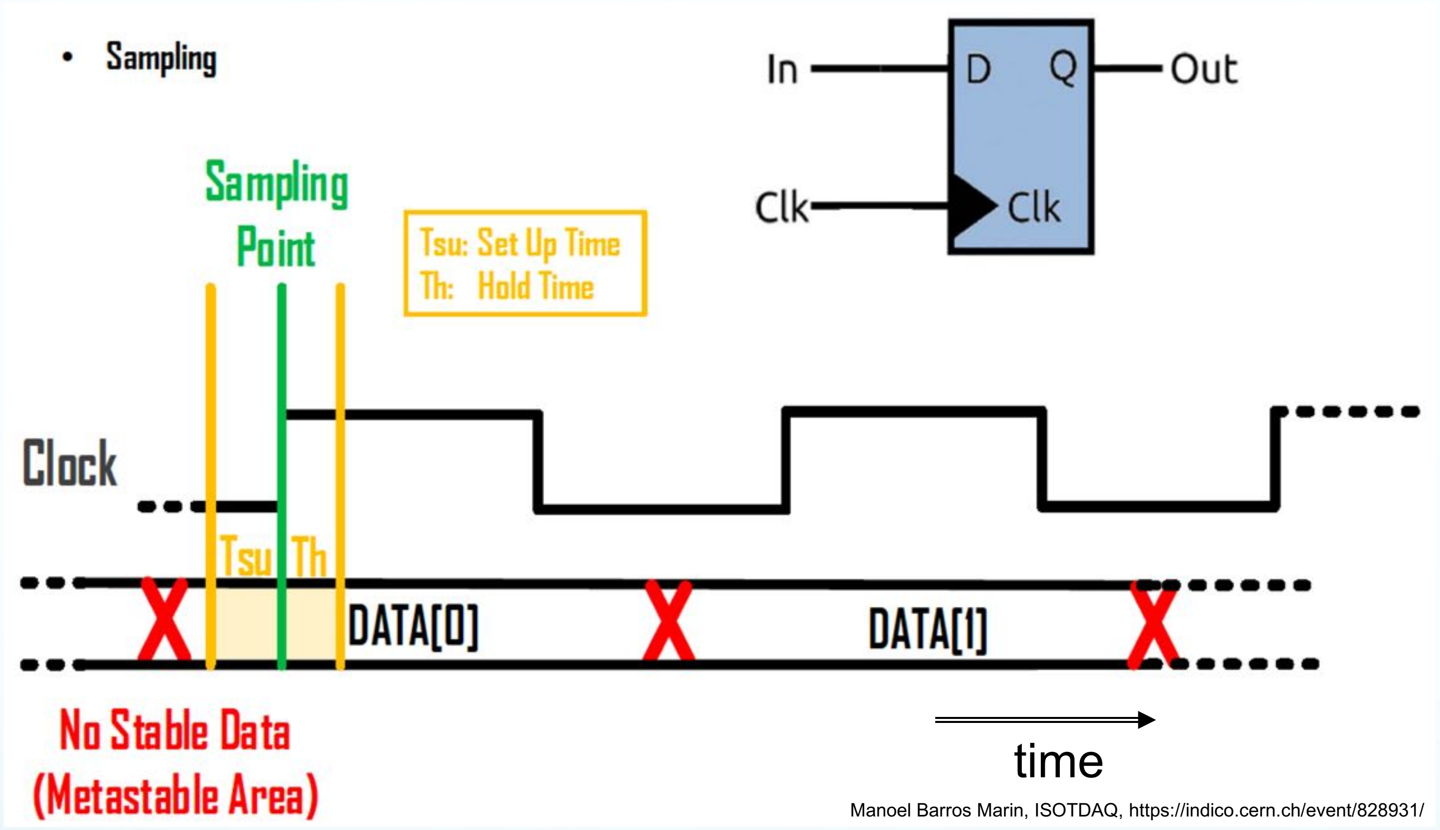
Timing

Timing

- Timing in FPGA design is critical



Data paths must respect setup and hold times

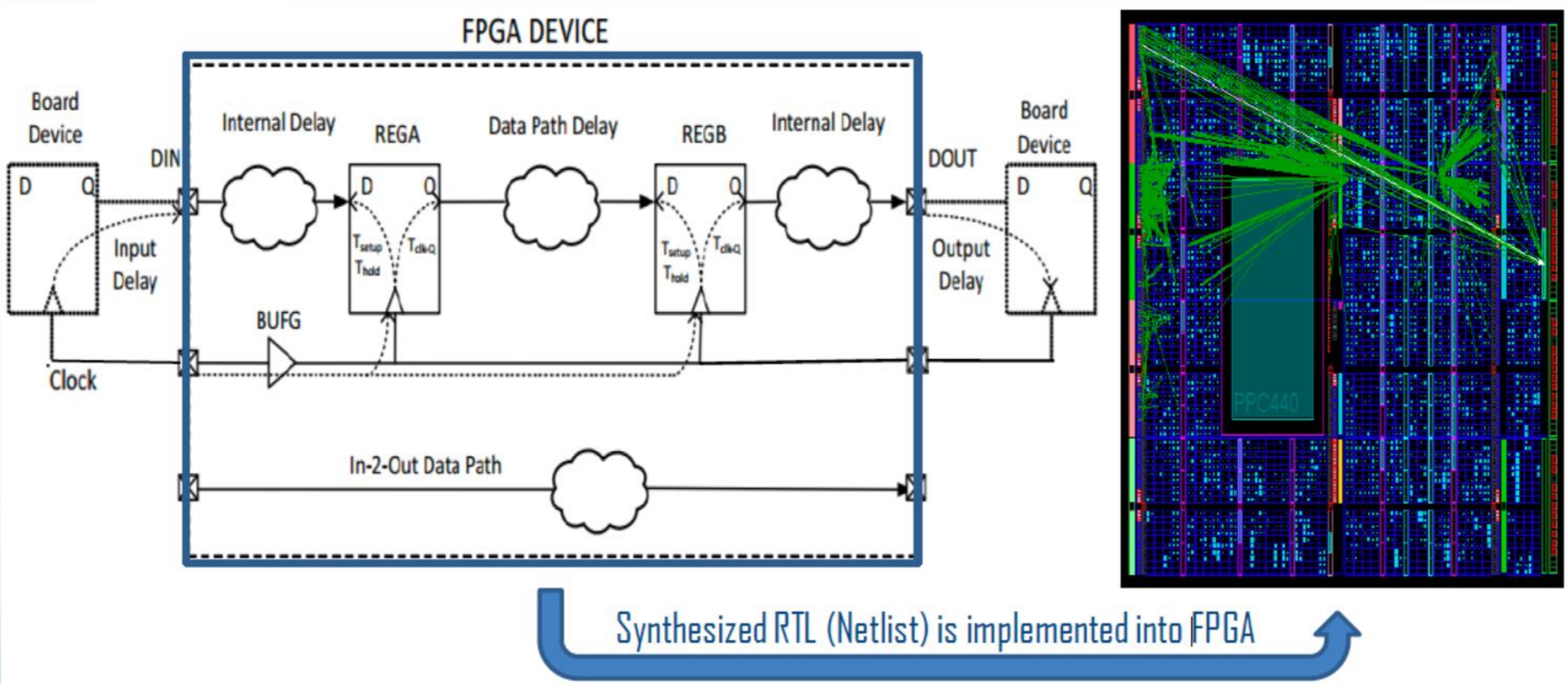


• Sampling
In
D
Q
Out
Clk
Clk
Sampling Point
Tsu: Set Up Time
Th: Hold Time
Clock
Tsu
Th
DATA[0]
DATA[1]
No Stable Data (Metastable Area)
time
Manoel Barros Marin, ISOTDAQ, https://indico.cern.ch/event/828931/

- **Setup time** is the amount of **time** required for the input to a Flip-Flop to be stable before a clock edge. **Hold time** is similar to **setup time**, but it deals with events after a clock edge occurs.

Timing

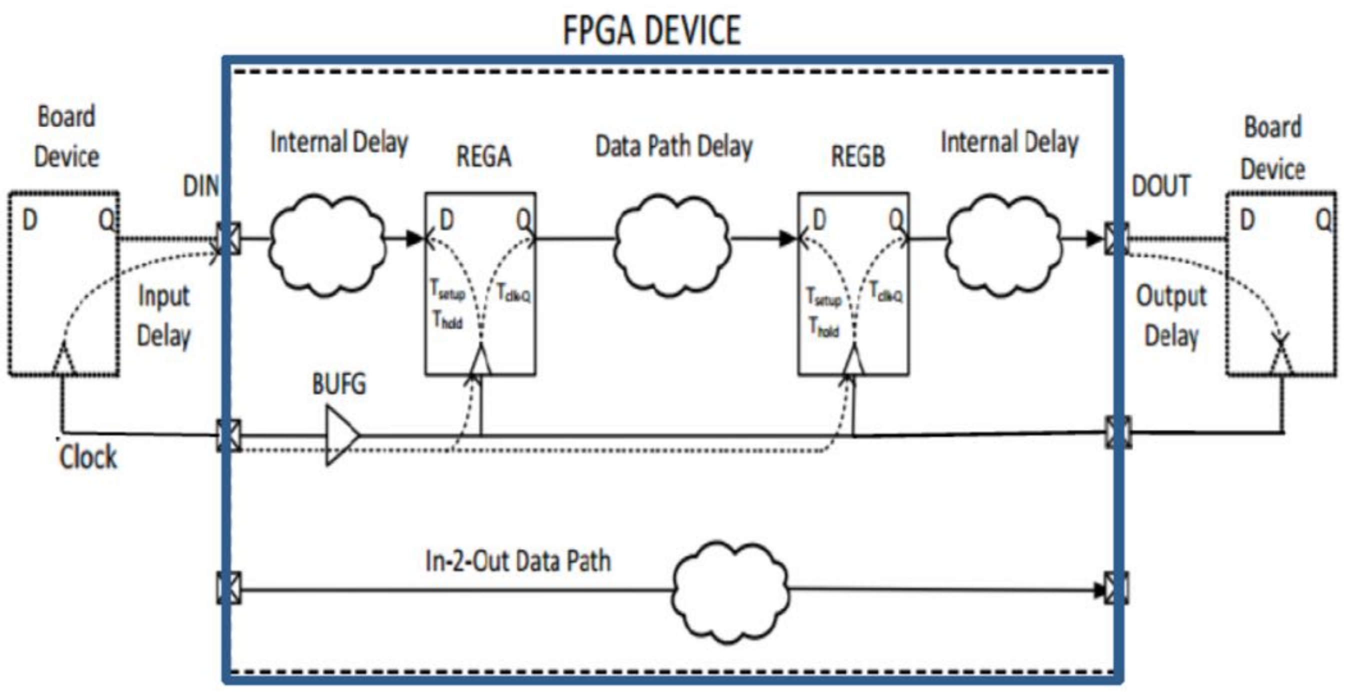
- Timing in FPGA design is critical

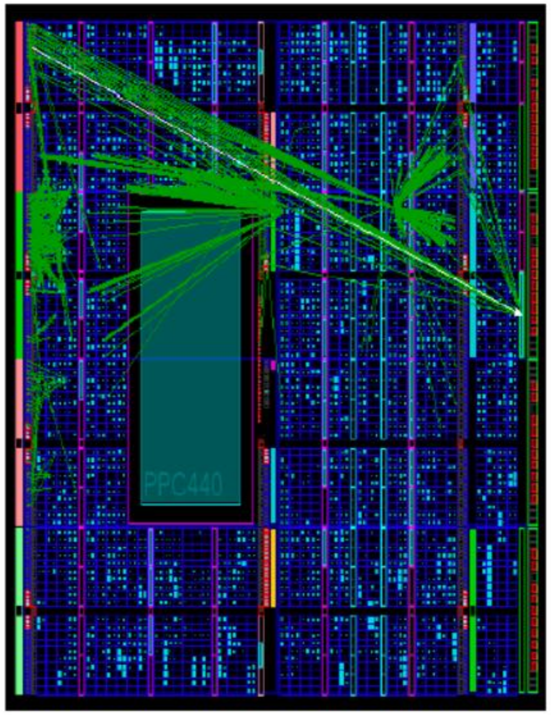


Manoel Barros Marin, ISOTDAQ, https://indico.cern.ch/event/828931/

Timing

- Timing in FPGA design is critical



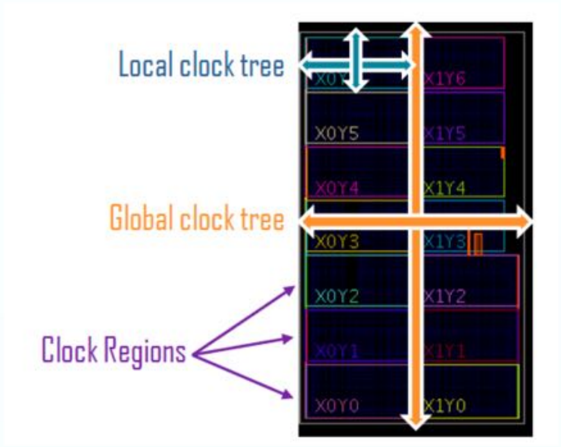


Synthesized RTL (Netlist) is implemented into FPGA

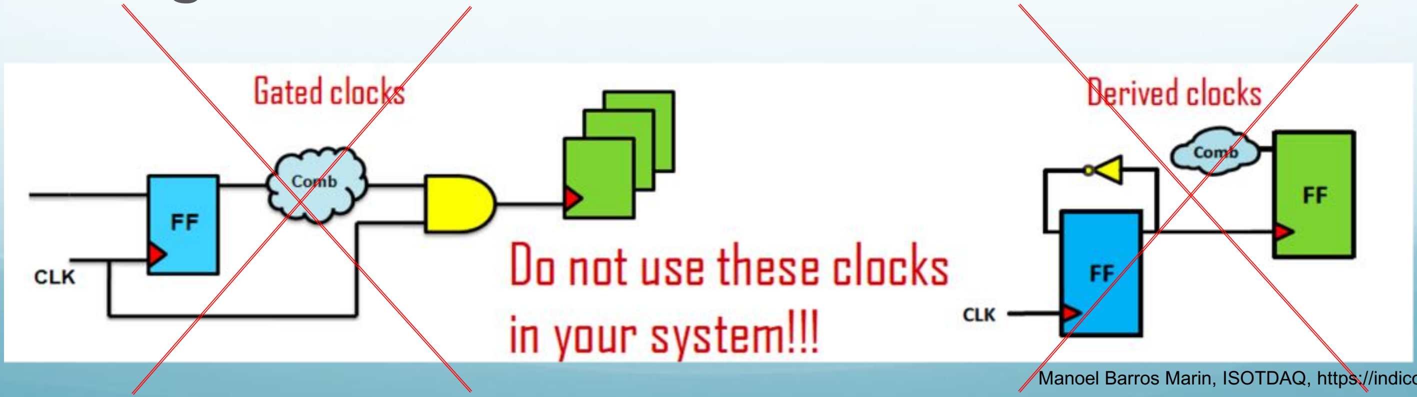
- If signals do not arrive at destination on time
    - Catastrophic consequences

Timing

- Always use dedicated clock networks to distribute clocks
  - Assures that clock is seen at all FFs at same time
  - Other clocking resources
    - Clock capable pins
    - Clock buffers
    - Clock Multiplexers
    - Phase Locked Loops
    - Digital Clock Managers

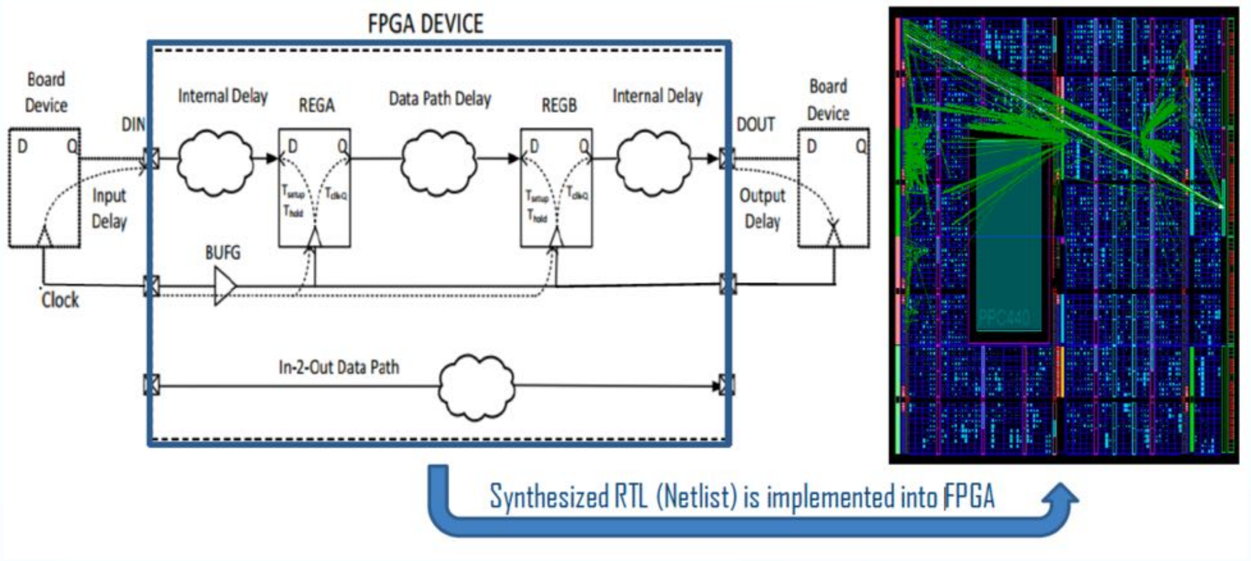


  - Do not gate or derive clocks



Do not use these clocks in your system!!!

Meeting timing closure



- Place & route step will try to position registers (flip-flops) and logic so that data path delays respect setup and hold times

- Options to meet timing
  - Instruct Place & route to use higher effort level
  - Add register stages & reduce amount of logic in data path (increases latency)
  - Choose location of inputs and outputs (at board design, or through optical patch panel)
  - Placement (area) constraints (give hints to the place & route step)

- Good practice
  - Whenever possible use I/O flip –flops (i.e FFs inside input/output cells)
    - Ensures timing with respect to external components is respected