Karlsruher Institut für Technologie

Institut für Technik der Informationsverarbeitung

Institut für Prozessdatenverarbaiten und Elektronik - IPE

KSETA Workshop 2013

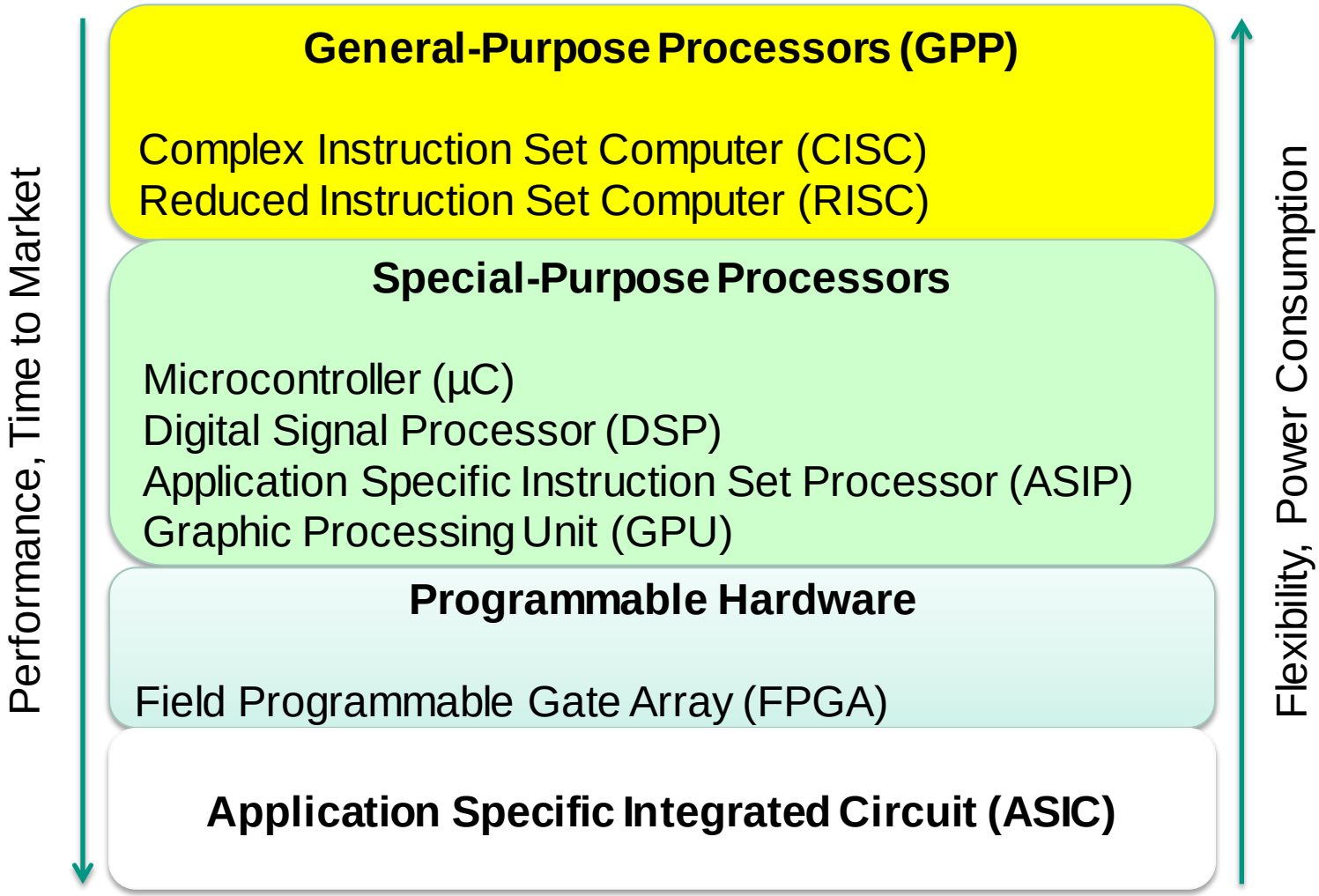
Dipl.-Inform. Steffen Bähr
Dipl.-Inform. Tanja Harbaum
Dipl.-Ing. Christian Amstutz
Dipl.-Ing. Uros Stevanovic

Data Analysis in Hardware
-
A Tutorial on VHDL and FPGAs

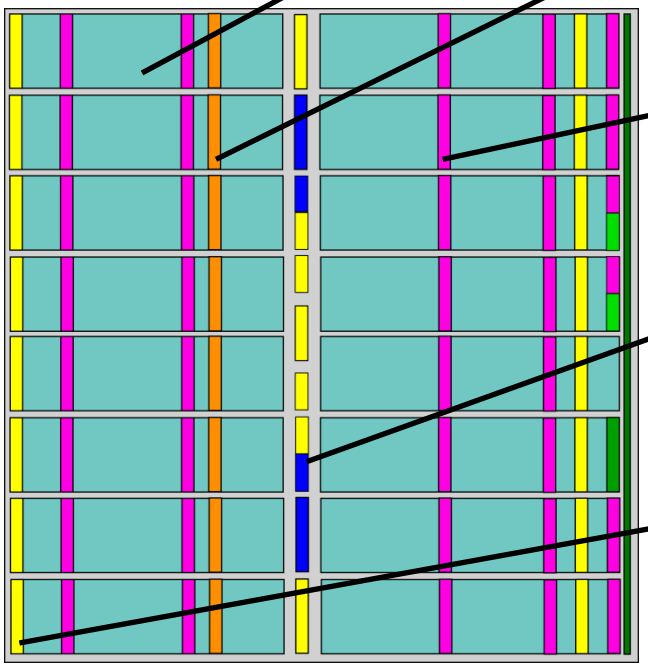
Overview

- Technologies Overview
- Introduction to Field Programmable Gate Arrays (FPGAs)
  - Technology: Xilinx Virtex5
  - Example: Full Adder
- FPGA Design Flow
  - VHDL
  - Xilinx ISE

Comparison between Technologies

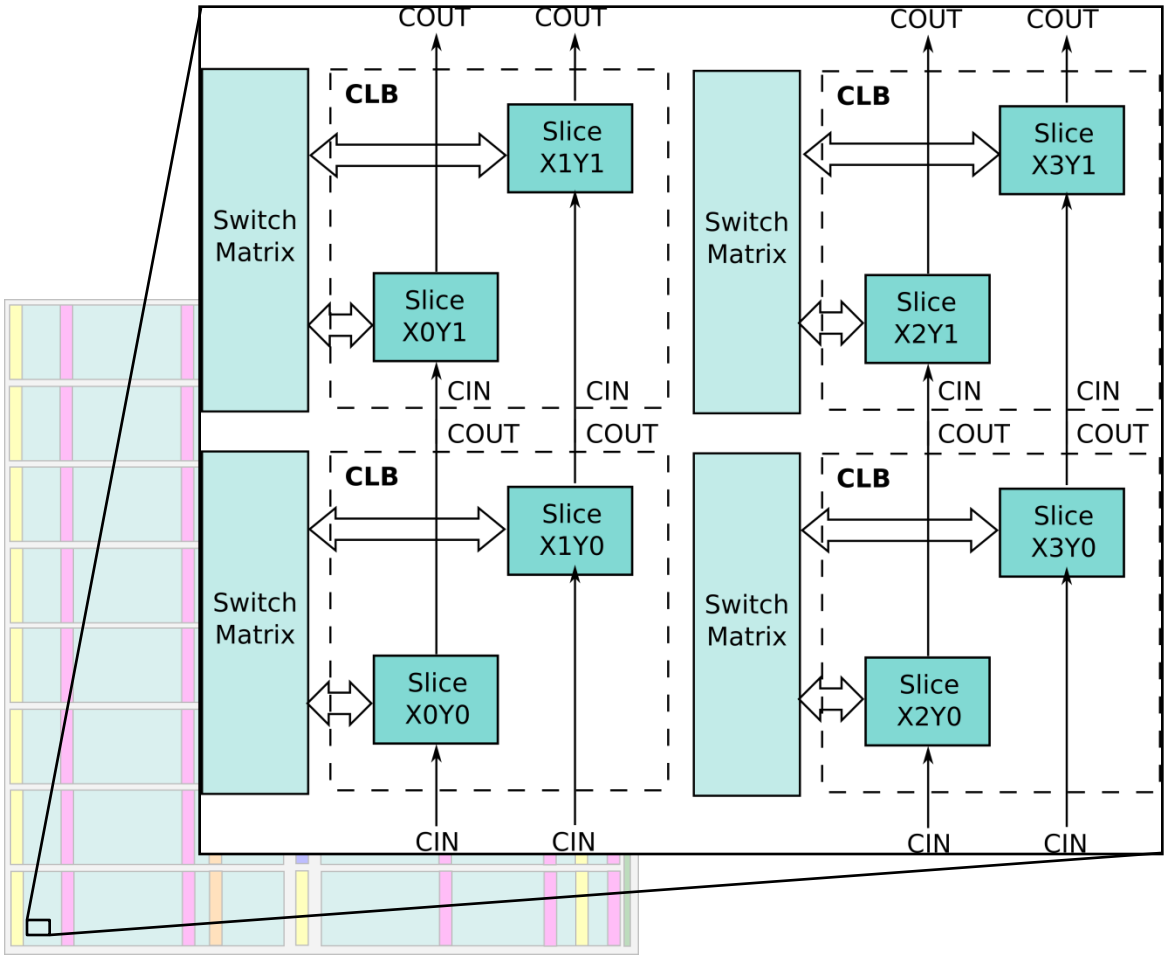


Example: Xilinx Virtex5 110T



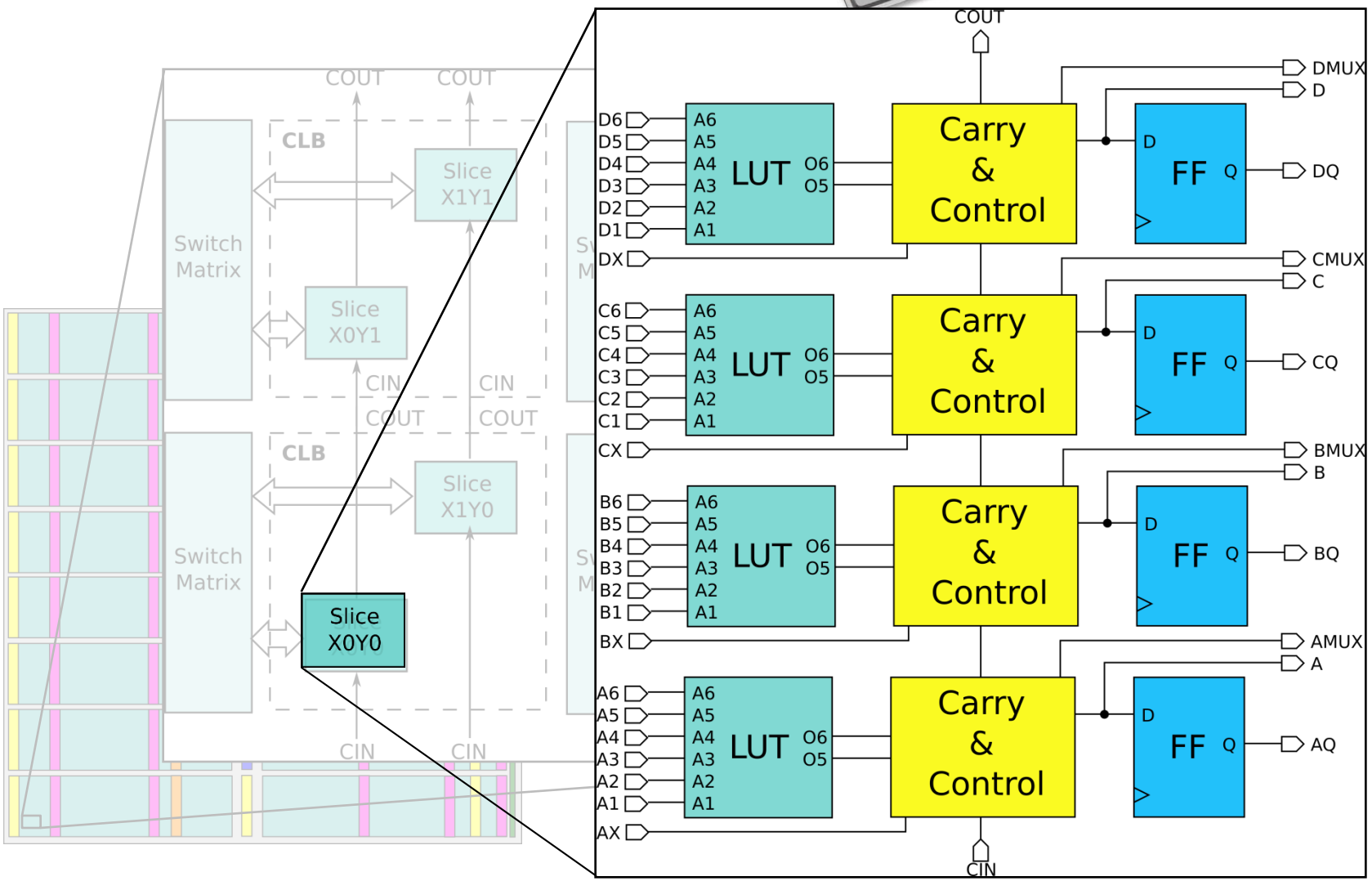
- Configurable Logic Block (CLB)
- Digital Signal Processor (DSP)
- Block RAM (BRAM)
- Digital Clock Memory (DCM)
- I/O Bank

Example: Xilinx Virtex5 110T



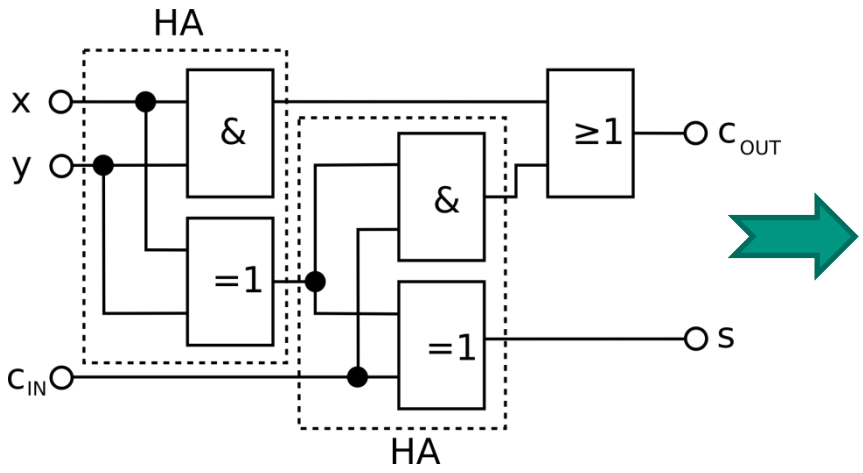
- One CLB consists of two Slices
- Programmable Switch Matrix
- Fast local routing inside a CLB
- Global routing between CLBs

Example: Xilinx Virtex5 110T



How to map a full adder into a Slice

- Step 1: Create a truth table

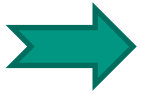


X	Y	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

How to map a full adder into a Slice

- Step 2: Produce Disjunctive Normal Form (DNF)

X	Y	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



$$s = (\overline{x} \wedge \overline{y} \wedge Cin) \vee (\overline{x} \wedge y \wedge \overline{Cin}) \vee (x \wedge \overline{y} \wedge \overline{Cin}) \vee (x \wedge y \wedge Cin)$$

$$Cout = (\overline{x} \wedge y \wedge Cin) \vee (x \wedge y) \vee (x \wedge Cin)$$

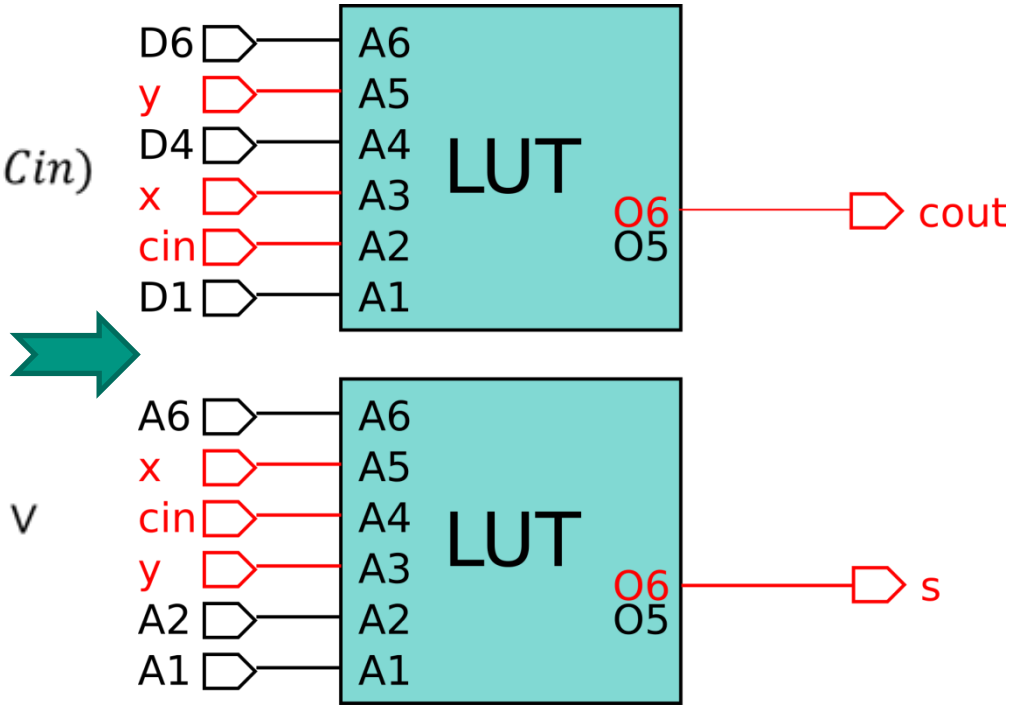
Institut für Technik der Informationsverarbeitung (ITIV)
Institut für Prozessdatenverarbeitung und Elektronik (IPE)

How to map a full adder into a Slice

- Step 2: Realization with two Lookup Tables

$$Cout = (\overline{x} \wedge y \wedge Cin) \vee (x \wedge y) \vee (x \wedge Cin)$$

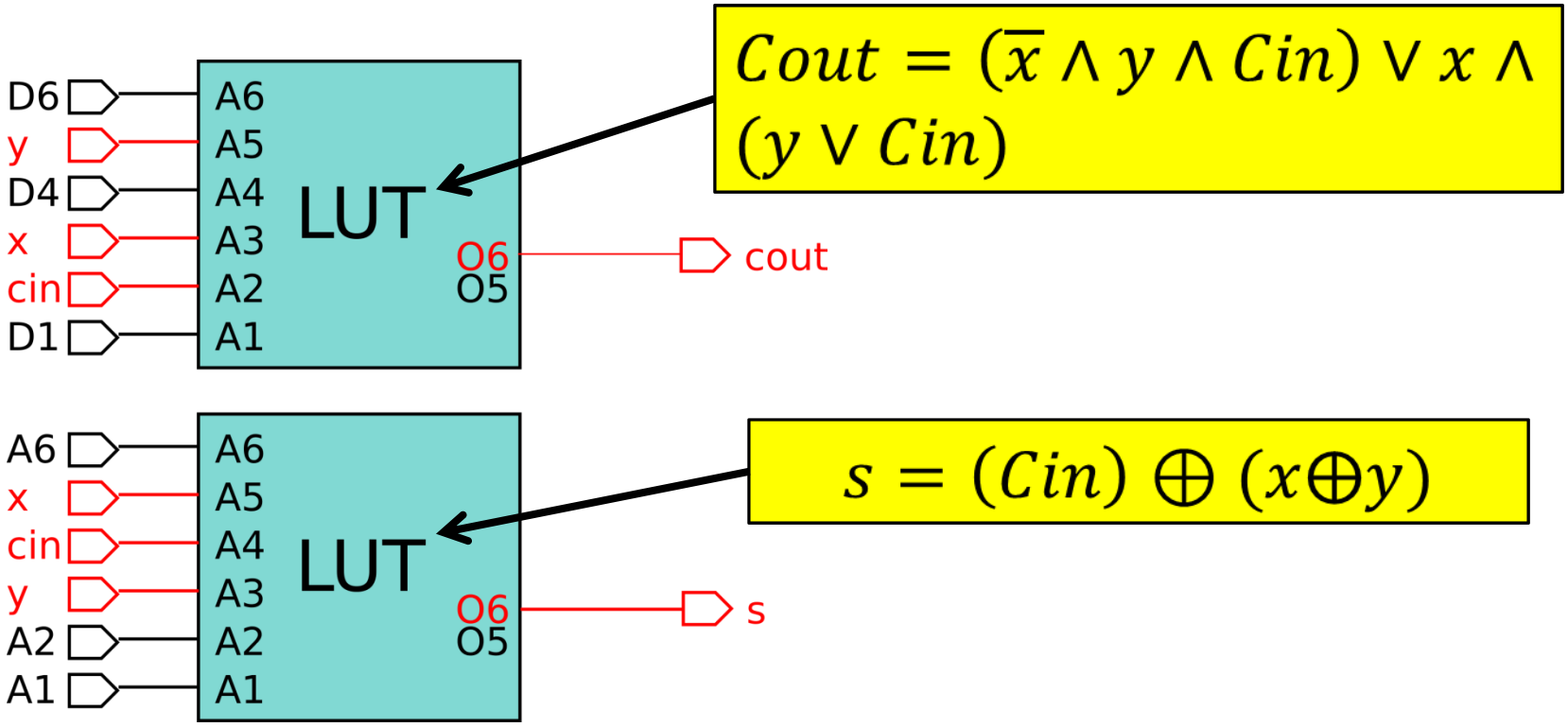
$$s = (\overline{x} \wedge \overline{y} \wedge Cin) \vee (\overline{x} \wedge y \wedge \overline{Cin}) \vee (x \wedge \overline{y} \wedge \overline{Cin}) \vee (x \wedge y \wedge Cin)$$



Institut für Technik der Informationsverarbeitung (ITIV)
Institut für Prozessdatenverarbeitung und Elektronik (IPE)

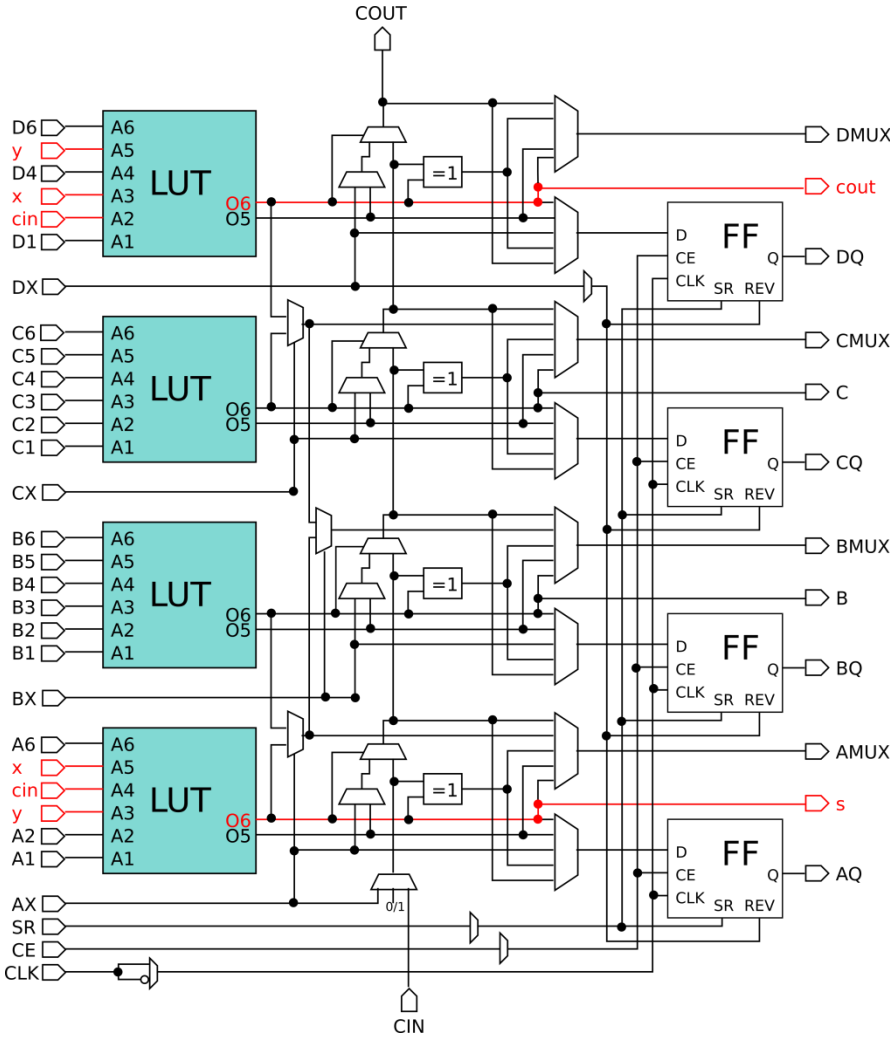
How to map a full adder into a Slice

- Step 2: Realization with two Lookup Tables



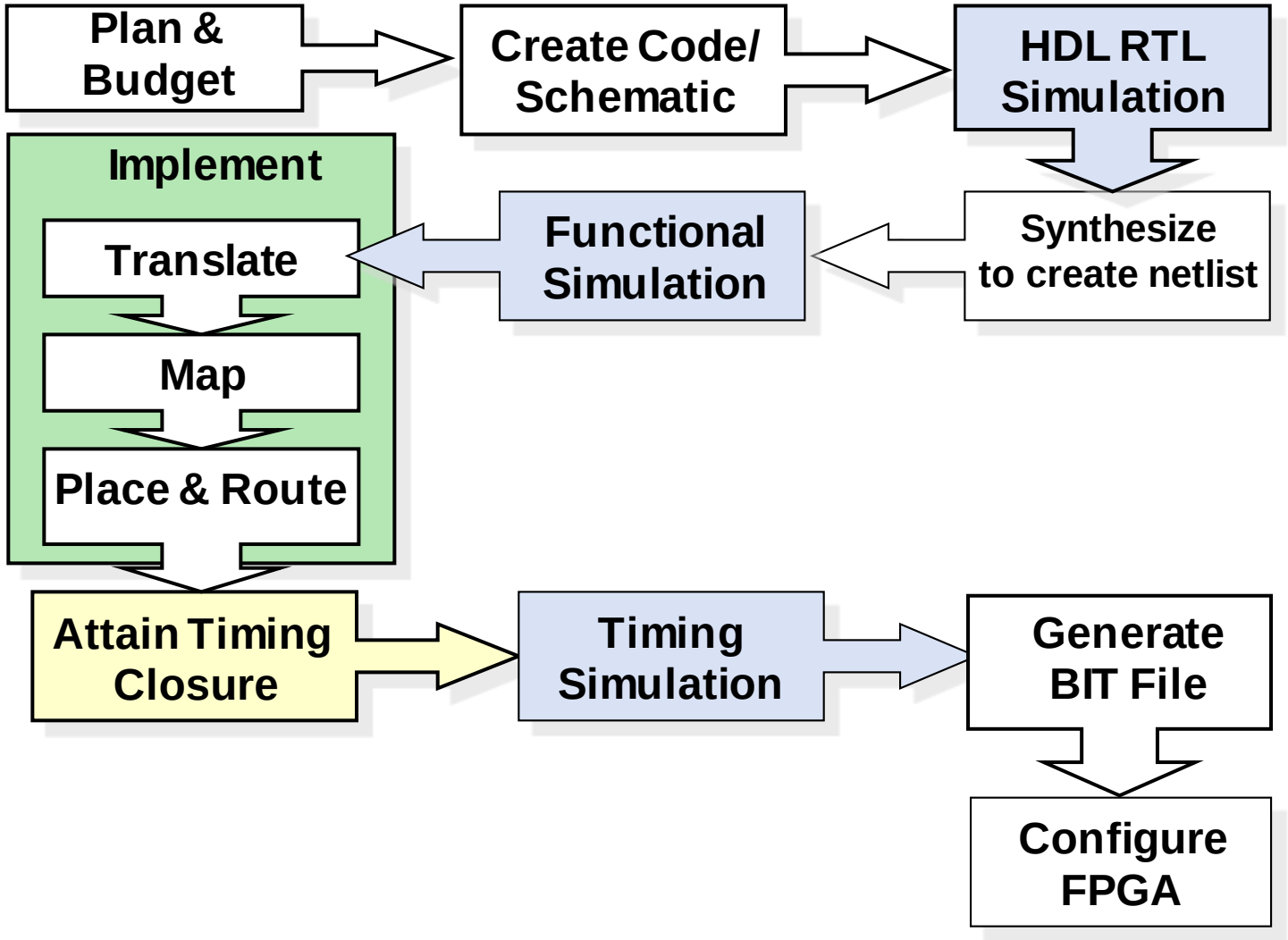
Generated by Xilinx ISE 14.2

Full adder in a Virtex5 SliceL



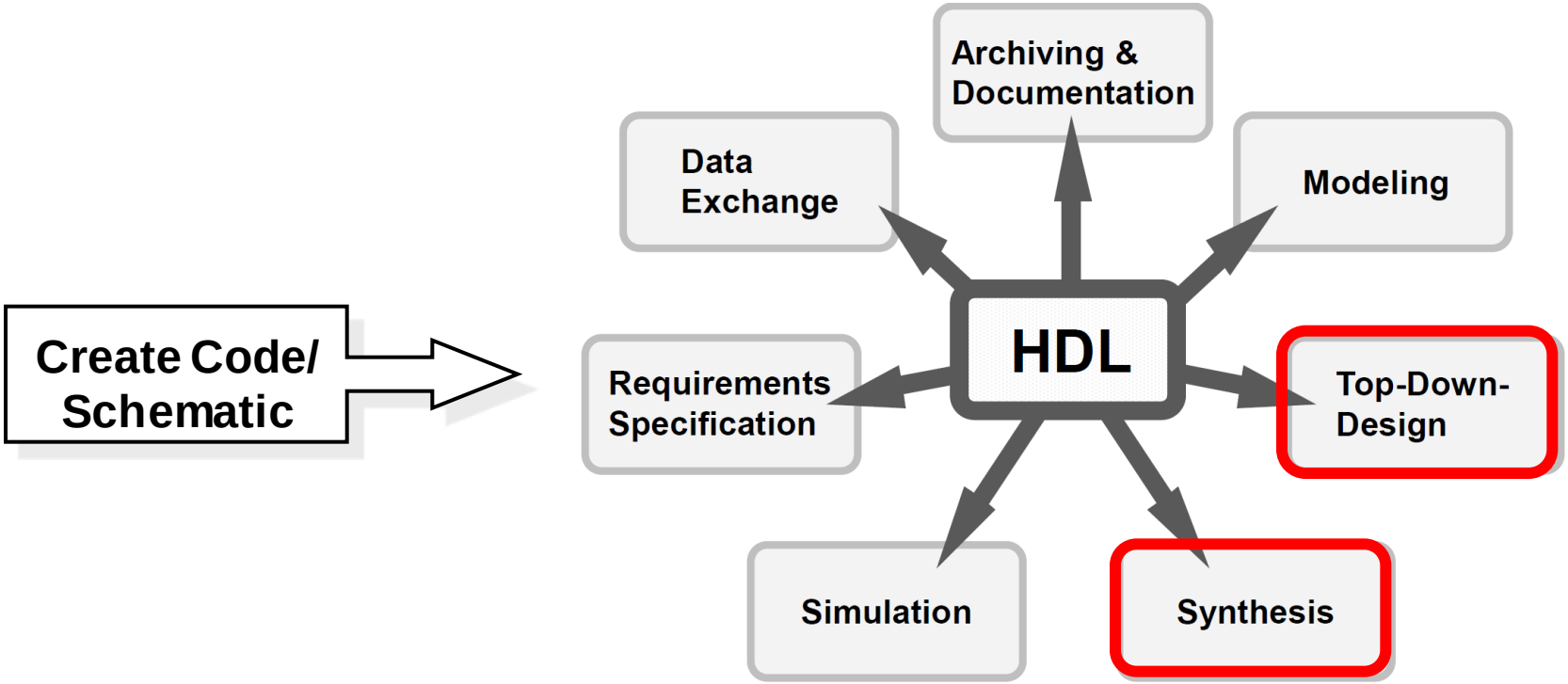
Institut für Technik der Informationsverarbeitung (ITIV)
Institut für Prozessdatenverarbeitung und Elektronik (IPE)

FPGA Design Flow



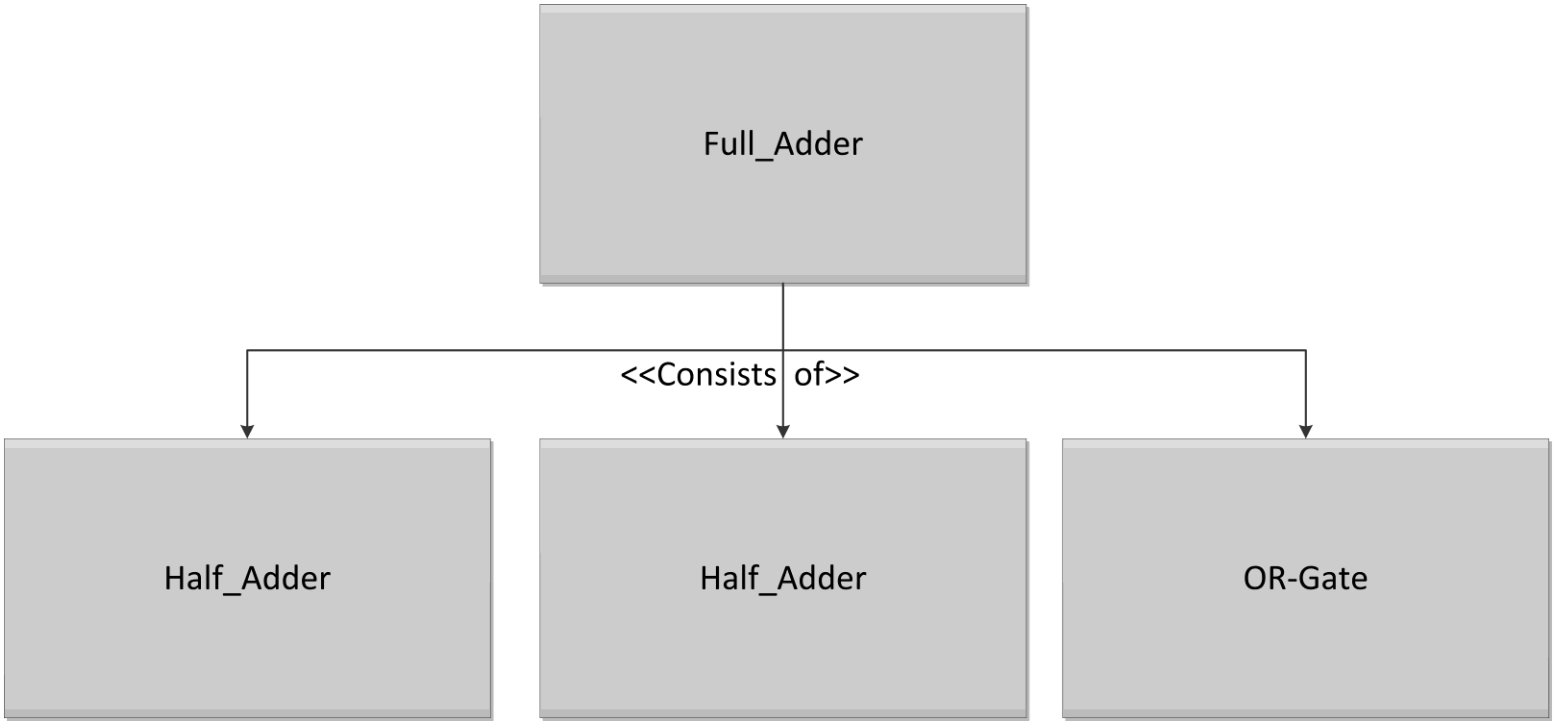
VHDL-Introduction

- VHSIC HDL – Very high Speed Integrated Circuits Hardware Descprition Language
- Enforced by the US-Department of Defense for documentation of ASICs



VHDL - Top-Down Design

- Hardware Designs are typically described in a top-down fashion



VHDL-Entity

- Description of a module‘s interface
  - Ports as mean of communication
    - Name
    - Direction
    - Type

```
1 entity full_adder
2         Port(
3                 x               : in  std_logic;
4                 y               : in  std_logic;
5                 cin             : in  std_logic;
6                 s               : out std_logic;
7                 cout            : out std_logic
8         );
9 end full_adder;
```

Entity description of full adder in VHDL

Graphical representation of full adder entity

VHDL-Signals

- Special Container for Data in VHDL
  - Parallel assignments of values
  - Connection between Modules
  - Coupling with timing information for simulations

```
1 signal name : type;
```

VHDL signal declaration syntax

```
1 signal result : std_logic;
```

VHDL signal declaration

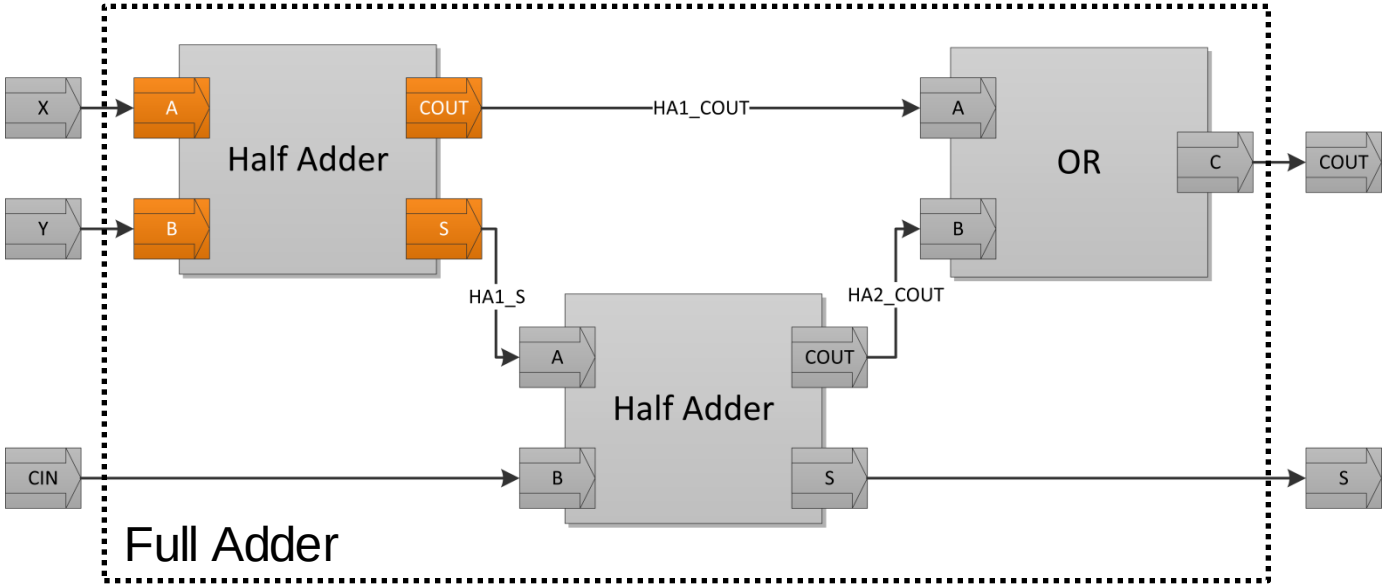
VHDL-Architecture

- Couples Entities with
  - Structural description
    - Declaration of used modules
    - Instantiation of modules
    - Connection of modules
  - Behavioural description
    - What does the module do ?

```
1 architecture structure of full_adder is
2 ..
3 end structure full_adder;
```

Architecture declaration of full adder in VHDL

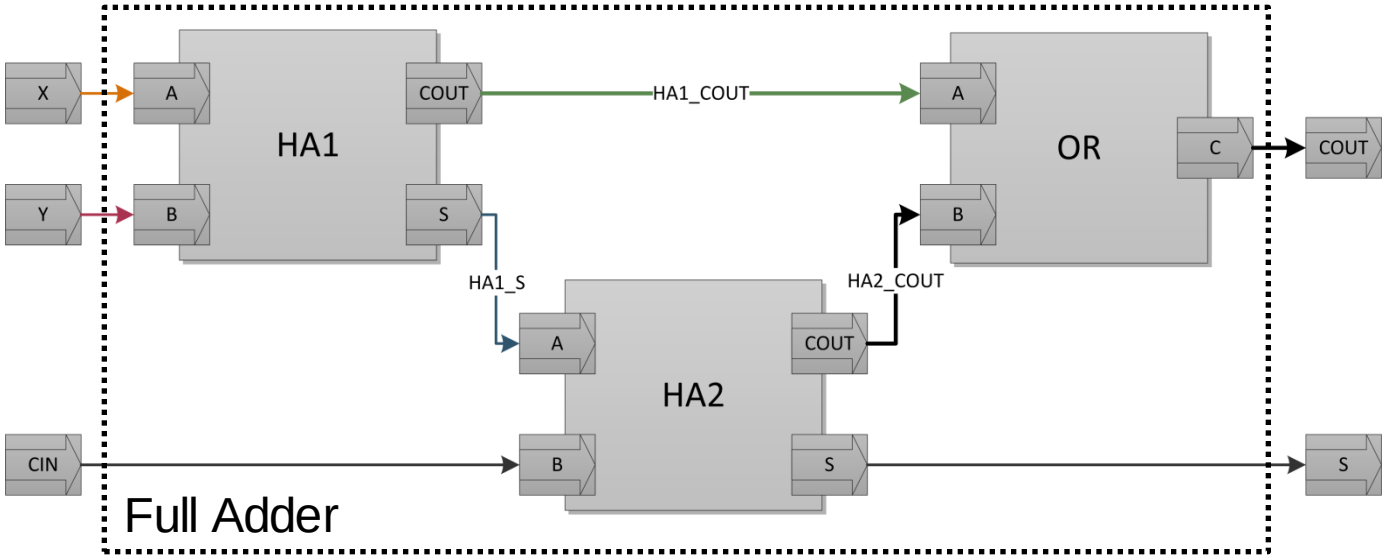
VHDL-Structural Description(1)




```
1  component half_adder
2          Port(
3                  a      : in  std_logic;
4                  b      : in  std_logic;
5                  s      : out std_logic;
6                  cout   : out std_logic
7          );
8  component half_adder;
```

Component declaration of full adder entity in VHDL

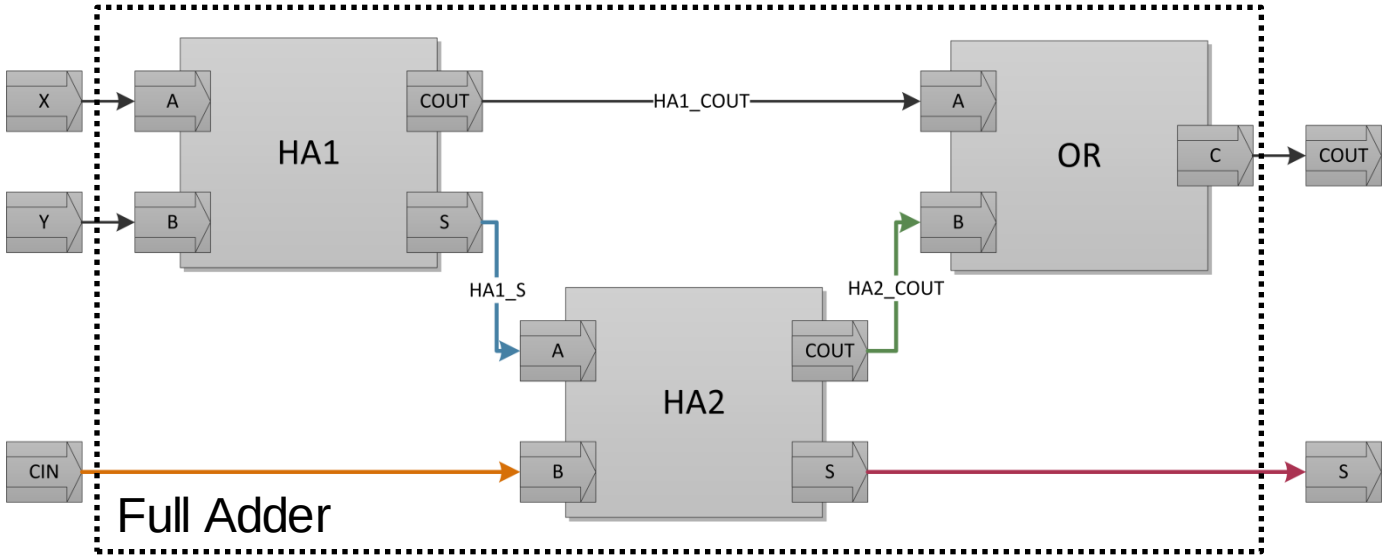
VHDL-Structural Description(2)




```
1  ha1 : half_adder
2          Port Map(
3                   a               => x,
4                   b               => y,
5                   s               => ha1_s,
6                   cout            => ha1_cout
7          );
```

Component instanstiation of half adder in VHDL

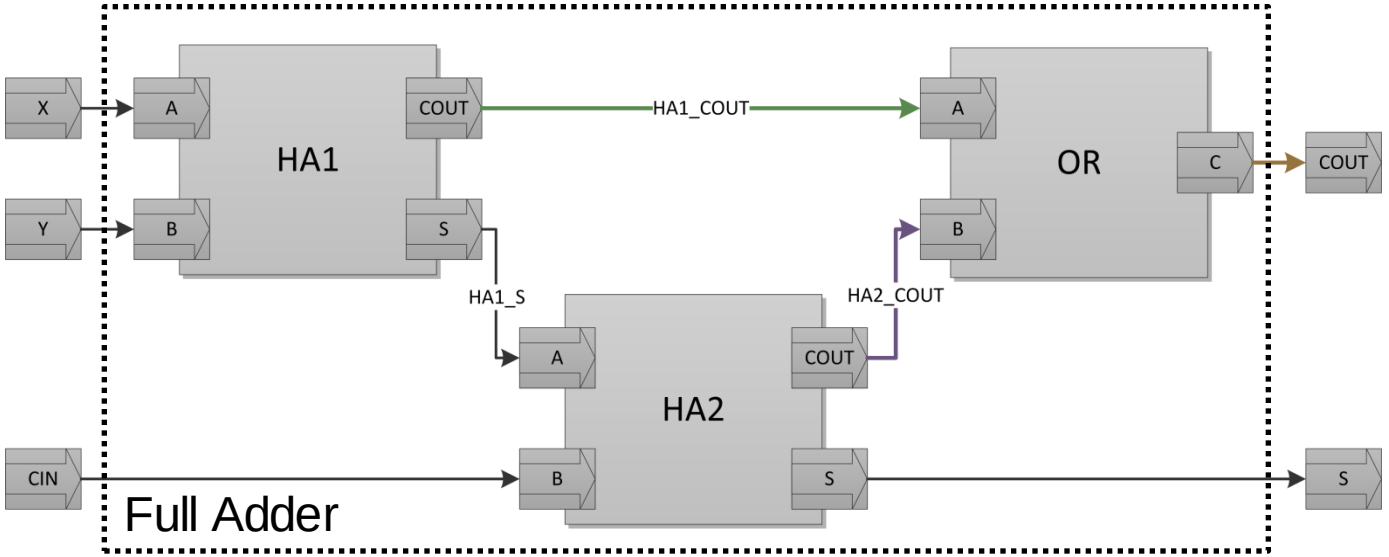
VHDL-Structural Description(3)




```
1  ha2 : half_adder
2          Port Map(
3                   a               => ha1_s,
4                   b               => cin,
5                   s               => s,
6                   cout            => ha2_cout
7          );
```

Component instanstiation of half adder in VHDL

VHDL-Structural Description(3)




```
1 cout <= ha1_cout or ha2_cout;
```

Connection of OR-Gate in VHDL

VHDL-Behavioural Description

- VHDL Process
  - Describes the behaviour of modules
  - Executed in parallel

```
1  s <= a or b; --implicit process syntax
2
3  process( a, b ) --explicit process syntax
4  begin
5
6          if( a = '0' and b = '0') then
7                  cout <= '0';
8          else
9                  cout <= '1';
10         end if;
11
12 end process;
```

Behavioural description „or“ in VHDL

Half Adder Code Example

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4
5  entity half_adder is
6  PORT (
7                  a               : in  std_logic;
8                  b               : in  std_logic;
9                  s               : out std_logic;
10                 cout  : out std_logic
11 );
12 end half_adder;
13
14 architecture Behavioral of half_adder is
15
16 begin
17
18 s               <= a and b; --implicit process syntax
19
20 process( a, b ) --explicit process syntax
21 begin
22
23         if( a = not b) then
24                 cout <= '0';
25         else
26                 cout <= '1';
27         end if;
28
29 end process;
30
31 end Behavioral;
```

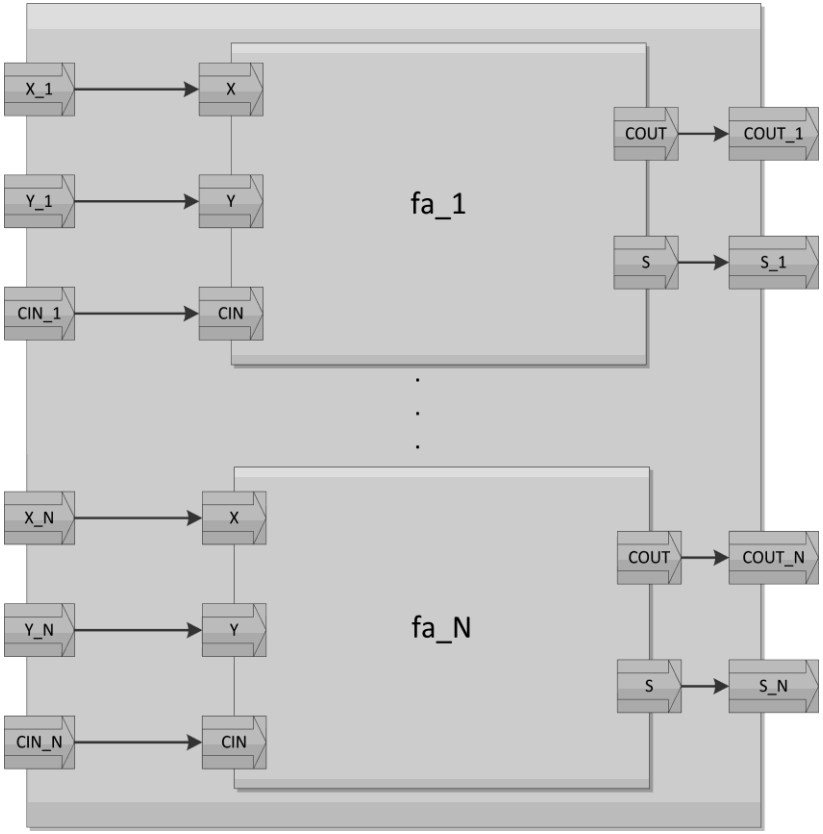
Full Adder Code Example

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4
5
6  entity full_adder is
7          Port(
8                    x              : in  std_logic;
9                    y              : in  std_logic;
10                   cin            : in  std_logic;
11                   s              : out std_logic;
12                   cout   : out std_logic
13         );
14 end full_adder;
15
16 architecture Behavioral of full_adder is
17
18 component half_adder
19         Port(
20                   a      : in  std_logic;
21                   b      : in  std_logic;
22                   s      : out std_logic;
23                   cout : out std_logic
24         );
25 end component;
26
27 signal ha1_cout : std_logic;
28 signal ha1_s     : std_logic;
```

```
29 signal ha2_cout : std_logic;
30
31 begin
32
33 ha1 : half_adder
34         Port Map(
35                   a => x,
36                   b => y,
37                   s => ha1_s,
38                   cout => ha1_cout
39         );
40
41 ha2 : half_adder
42         Port Map(
43                   a => ha1_s,
44                   b => cin,
45                   s => s,
46                   cout => ha2_cout
47         );
48
49 cout <= ha1_cout or ha2_cout;
50
51 end Behavioral;
```

VHDL-Parallelism

- Modules can be instantiated mutiple times and execute in parallel



Mulitple Full Adders executing in parallel

```
1  fa_1 : full_adder
2          Port Map(
3                    x => x_1,
4                    y => y_1,
5                    cin => cin_1,
6                    cout => cout_1,
7                    s => s_1
8          );
9
10 ..
11
12 fa_n : full_adder
13         Port Map(
14                   x => x_n,
15                   y => y_n,
16                   cin => cin_n,
17                   cout => cout_n,
18                   s => s_n
19         );
```

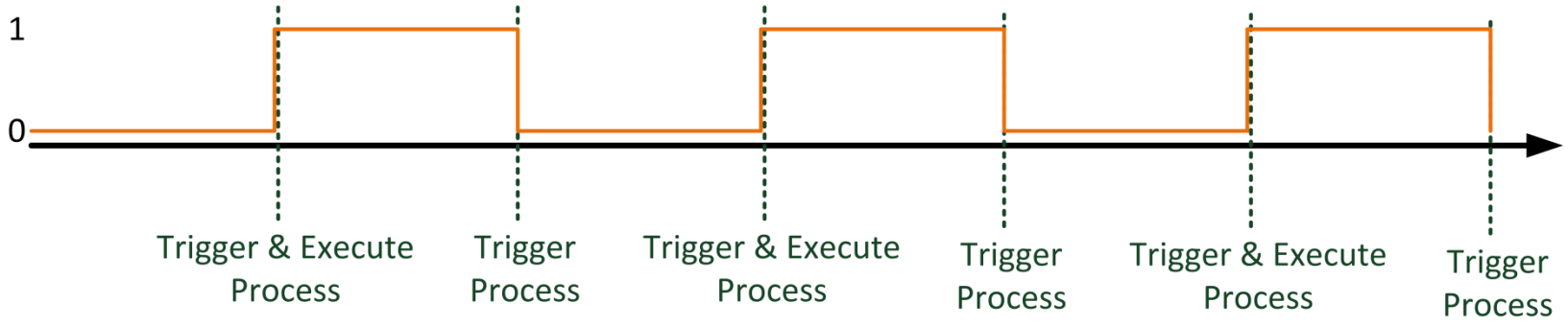
Mulitple instantiations of a Full Adder

Institut für Technik der Informationsverarbeitung (ITIV)
Institut für Prozessdatenverarbeitung und Elektronik (IPE)

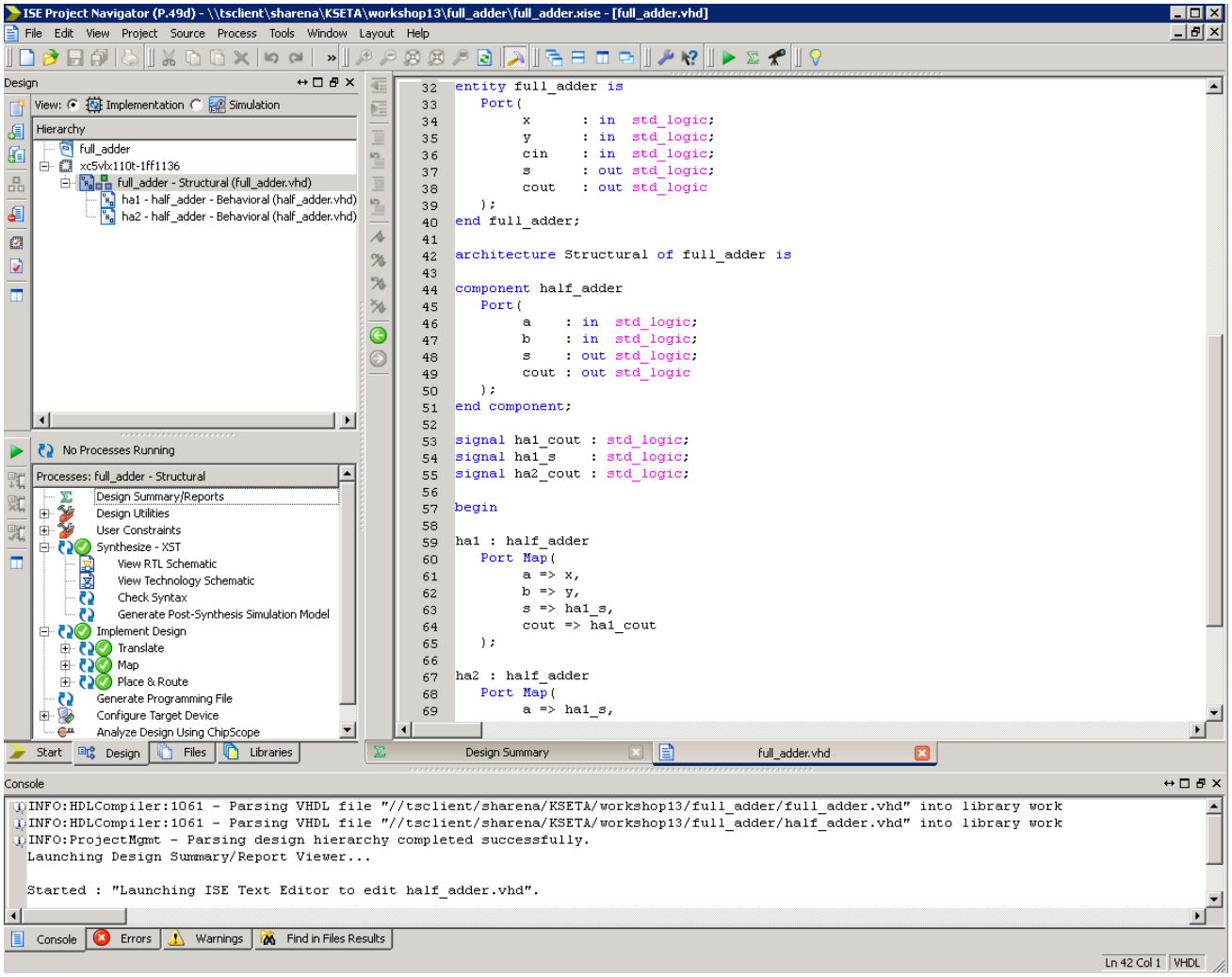
VHDL-Clocking

- Using a clock for synchronisation in VHDL Modules
  - Clock signal included in sensitivity list of process
  - Rising or falling clock edge

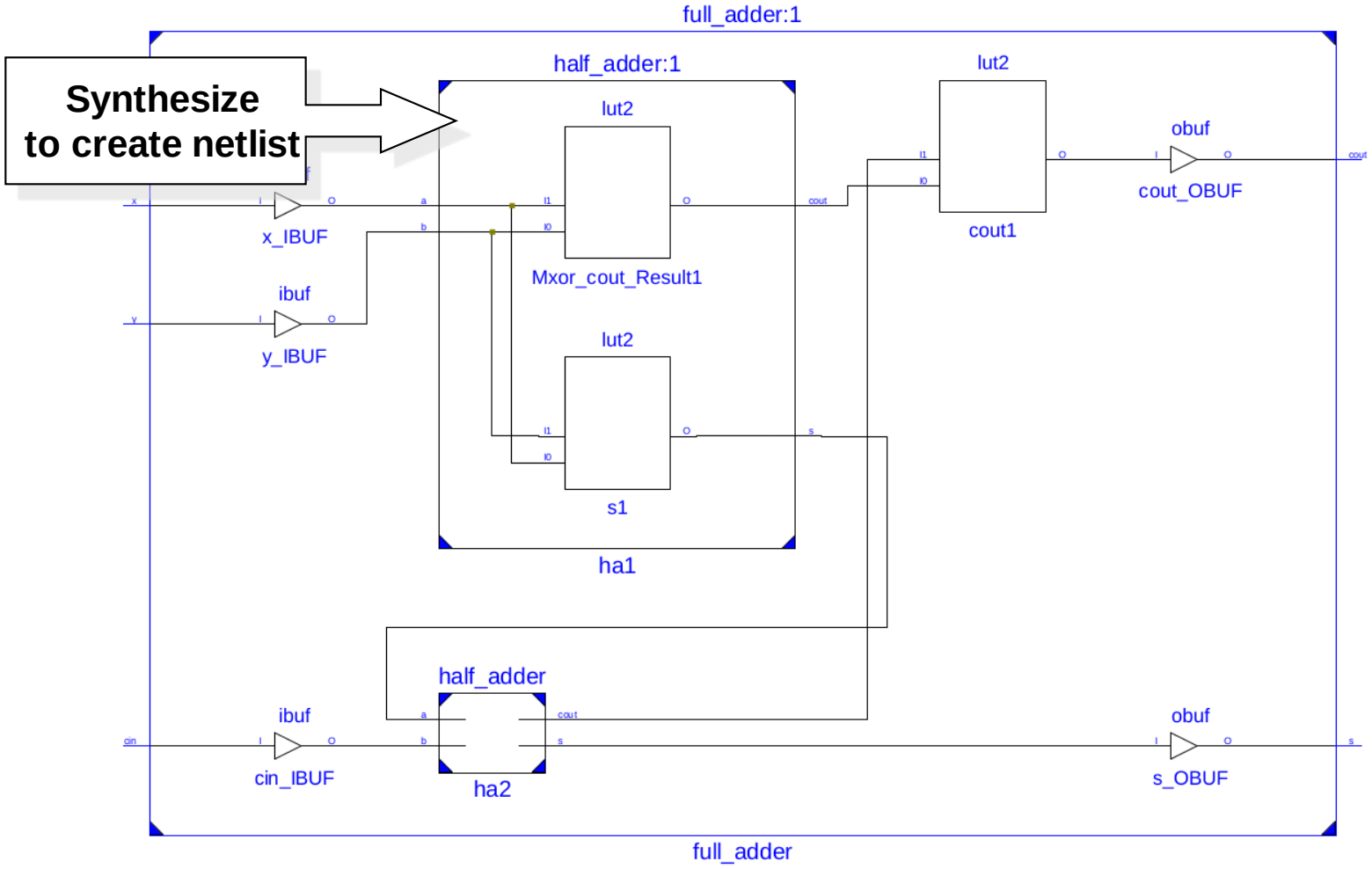
```
1 process(clk)
2 begin
3         if( clk = '1' and clk'event) then
4                 result <= 1;
5         end if;
6 end process;
```



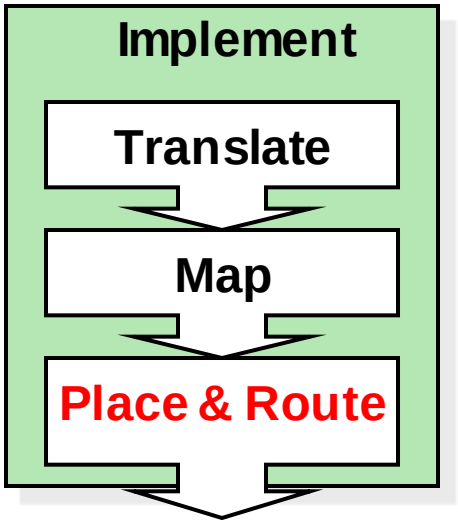
XILINX Design Tool ISE

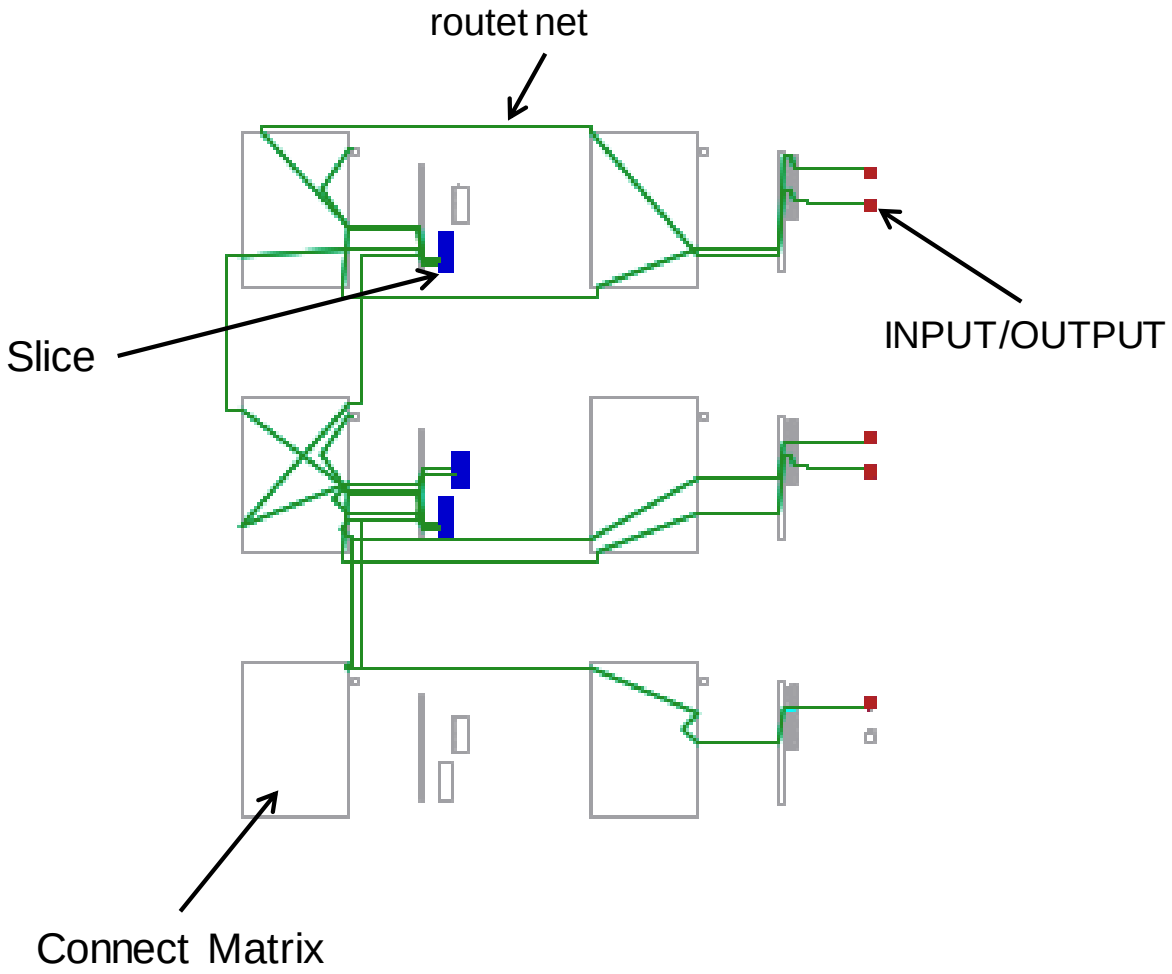


Demo Synthesize

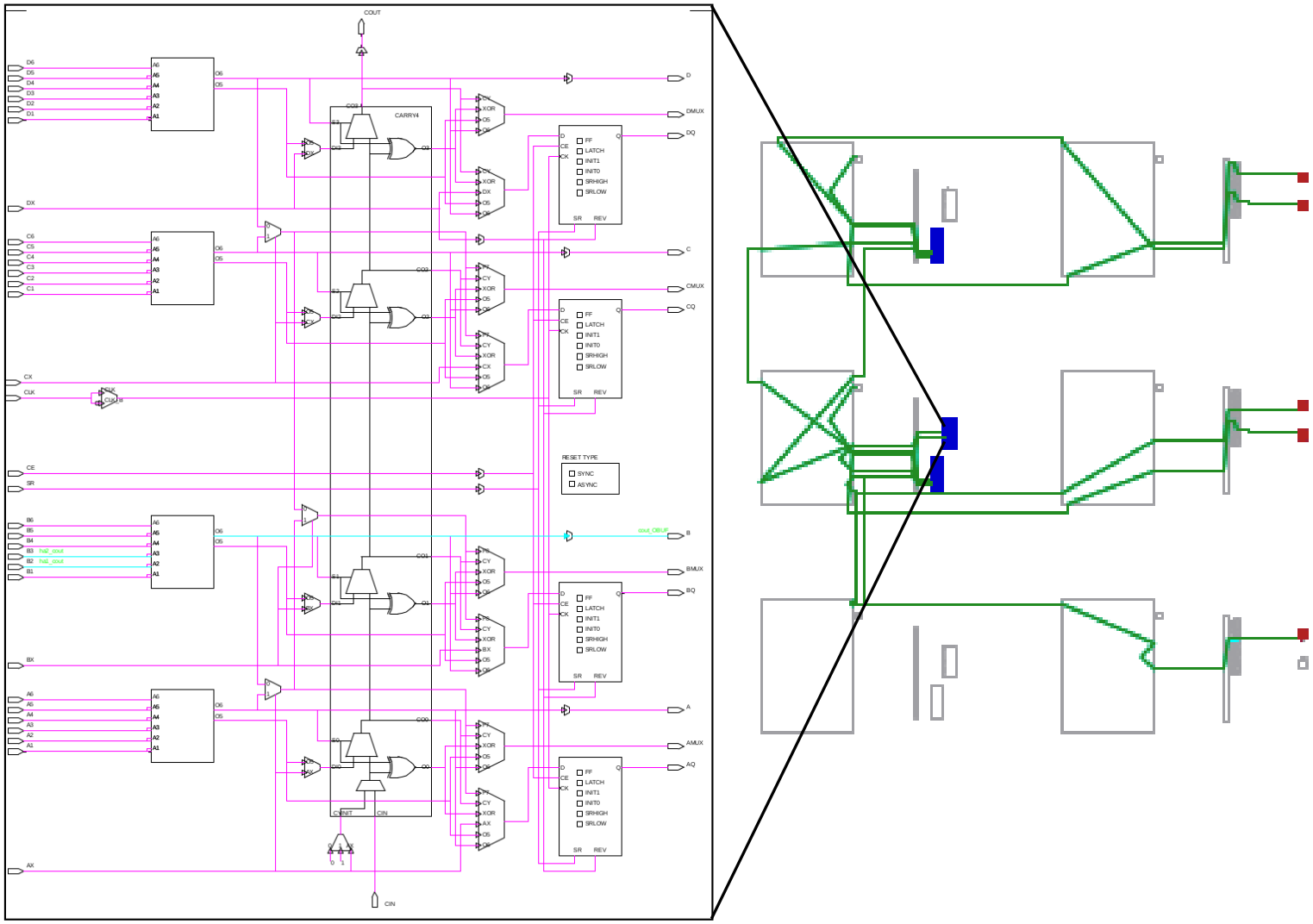


Demo P&R


Implement
Translate
Map
Place & Route


routet net
Slice
INPUT/OUTPUT
Connect Matrix

Demo P&R


COUT
CARRY4
FF
LATCH
INIT1
INIT0
SRHIGH
SRLOW
SR
REV
RESET TYPE
SYNC
ASYNC
CIN

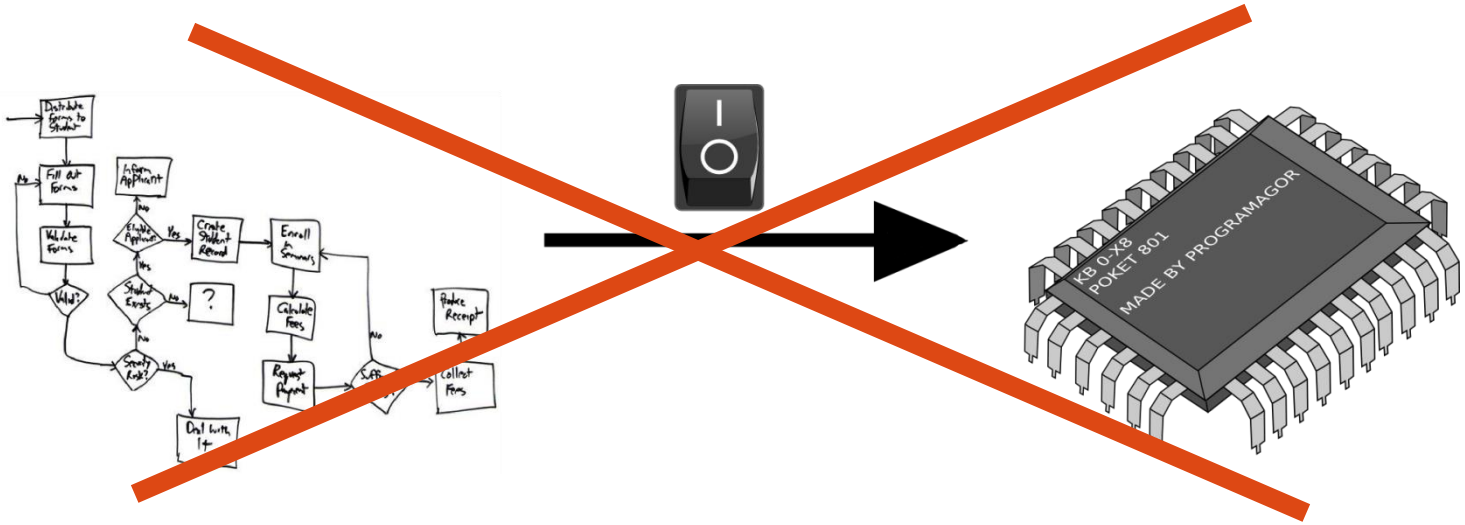
High-Level-Design Approaches

- FPGA design by HDLs:
  - Time consuming
  - Error-prone
  - Difficult verification
  - Large teams
- Algorithms are not designed in HDL

- **Is there a faster/simpler way to design FPGAs?**

Yes! But ...

Considerations for High-Level Approaches

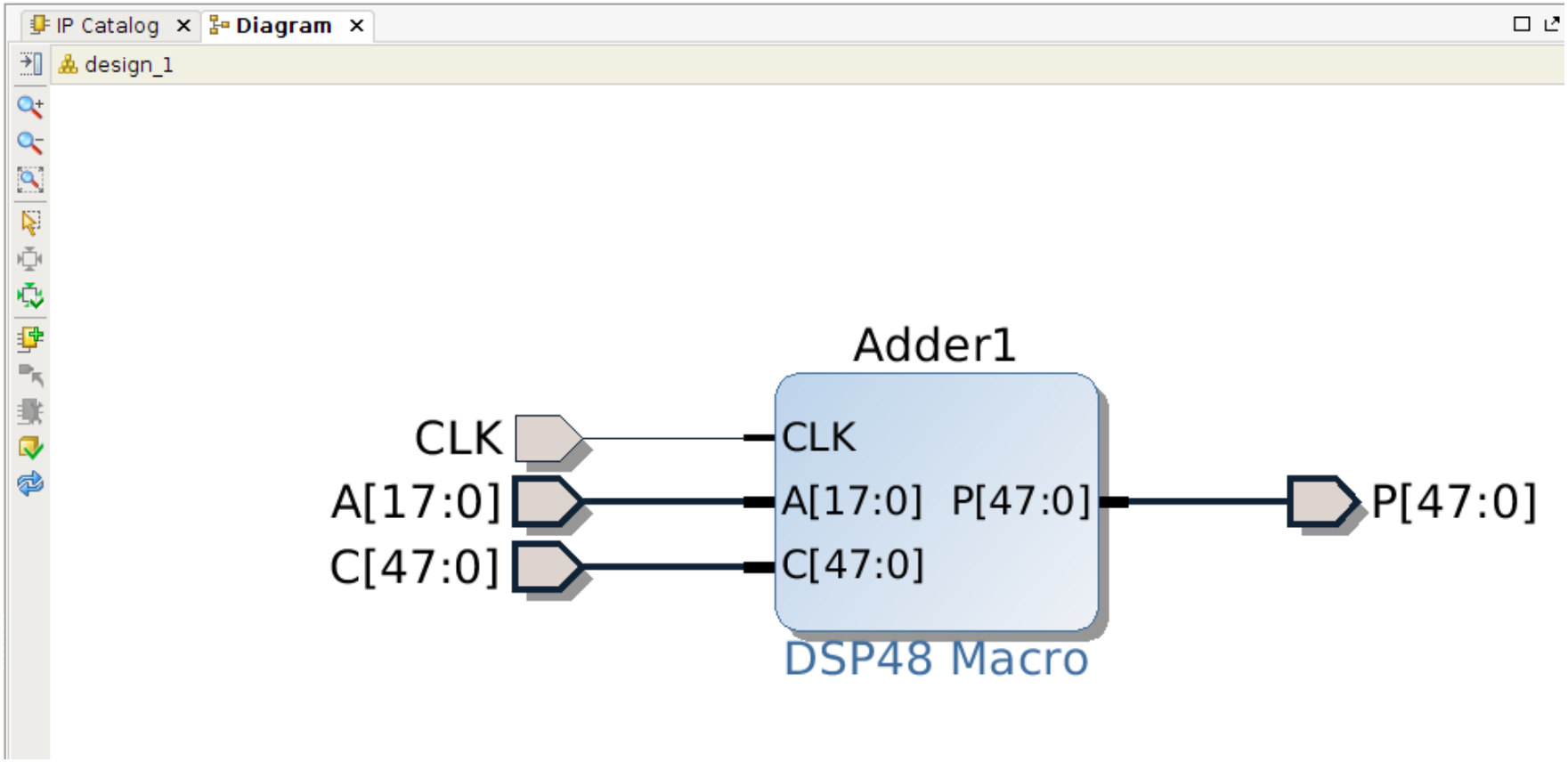


- There is no “Push-Button” approach
- Unlike for software compilation
- Designer must keep hardware in mind

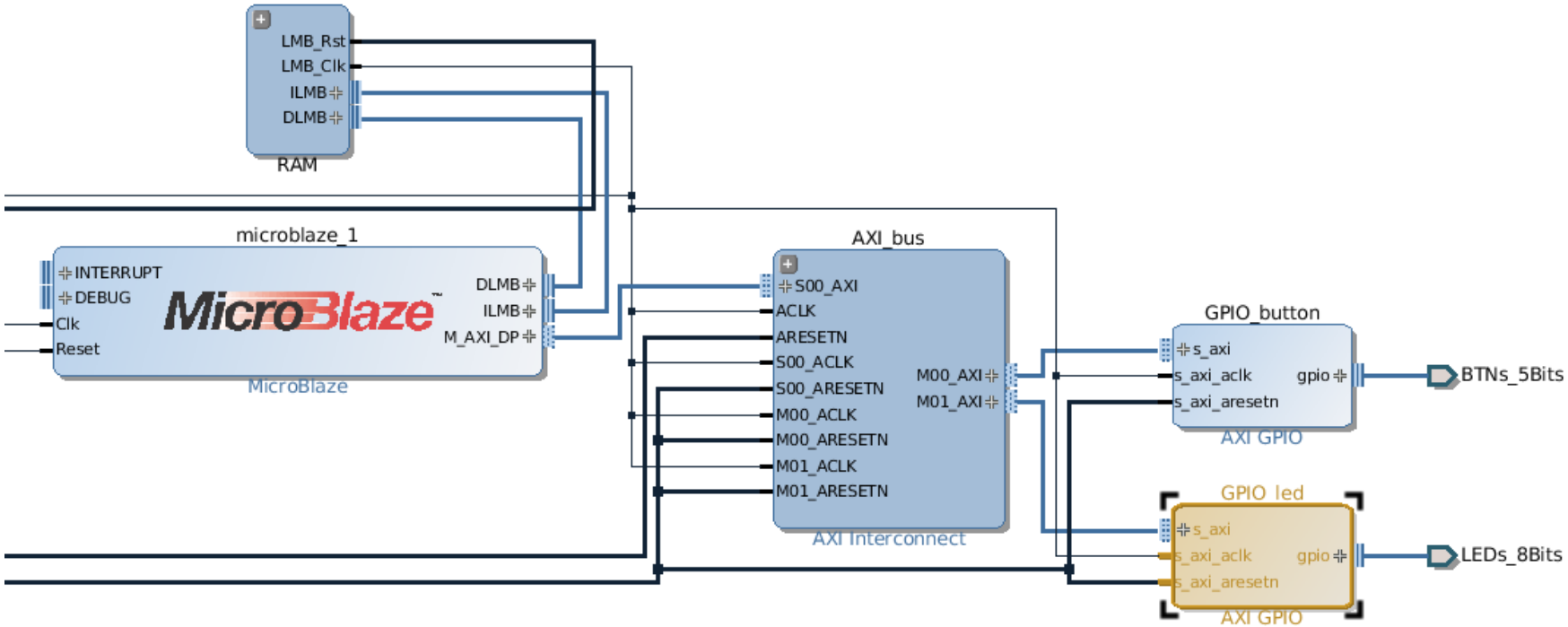
System Designer

- Design of (Embedded) Systems based on building blocks
- IP-blocks connected by a standard bus
- Available blocks:
  - Arithmetics: Adder, Mulitplier, sqrt, sin/cos, ...
  - Digital Signal Processing: FIR filters, FFT, ...
  - Processors, RAMs, ROMs, Peripherial controller, ...
- Limitations
  - Limited to the available blocks (included, purchased)
  - Describing a complex algorithm soly by basic blocks is cumbersome
- Applications: Altera Qsys / Xilinx EDK

Xilinx EDK: Simple Adder



Xilinx EDK: Complete Processor System



High-Level-Synthesis (HLS)

- Converts an algorithm written in C/C++/SystemC to HDL
- C is used as most of its constructs could be supported
- Everything great?
  - Algorithm must be parallelizable
  - Data types should fit hardware
    - •Minimal bit width which ensures the accuracy of algorithm
    - •Floating point is expensive in hardware
  - Different optimization goals → Design Space Exploration
    - •Throughput
    - •Latency
    - •Chip area
    - •Power
- Applications: Xilinx Vivado HLS, Calypto Catapult

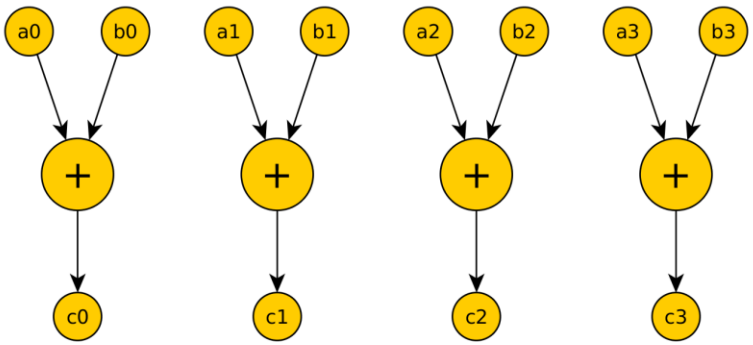
Design Space Exploration (Example)

- Vector Addition:

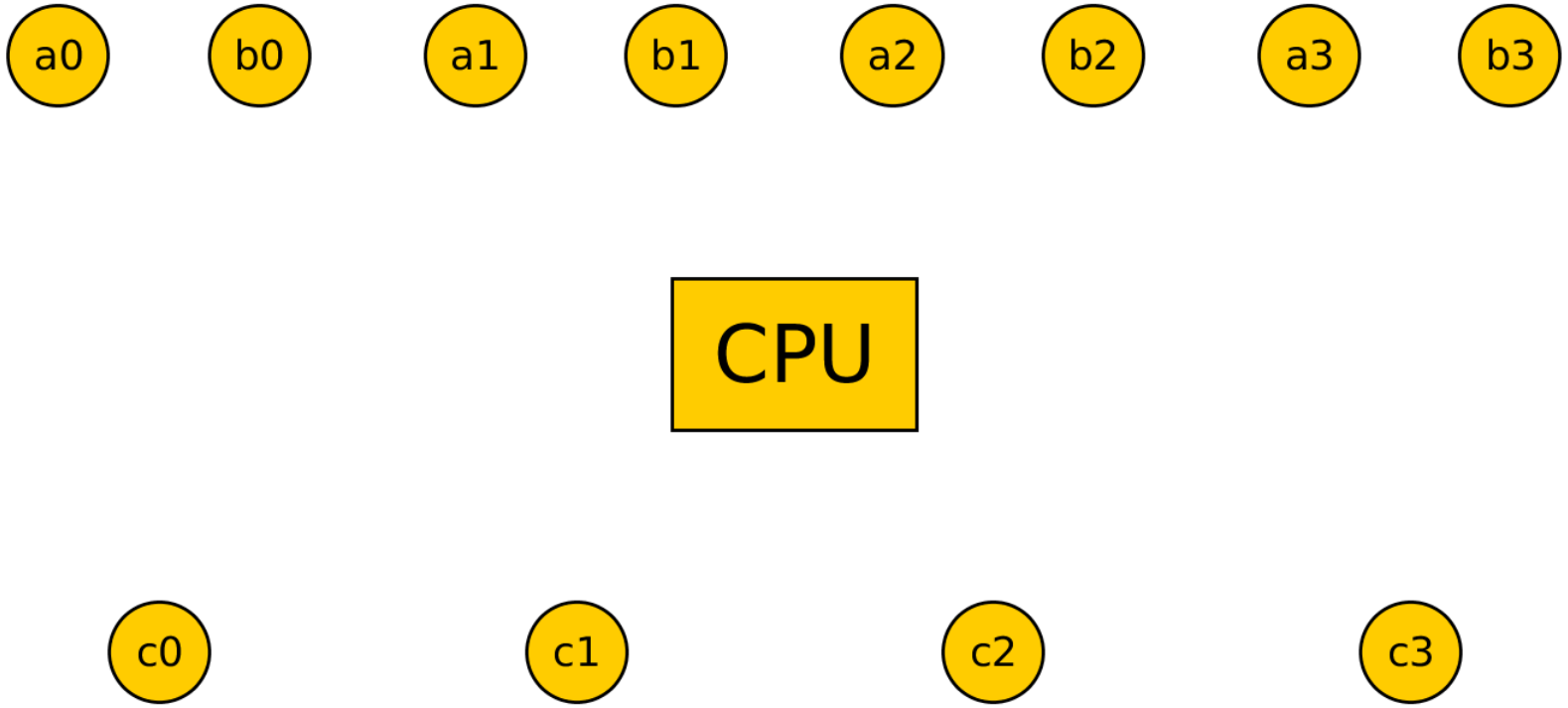
$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} + \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

```
1 void addVectors(int a[], int b[], int c[], int n) {
2
3     for (int i=0; i<n; i++) {
4         c[i] = a[i] + b[i];
5     }
6
7 }
```

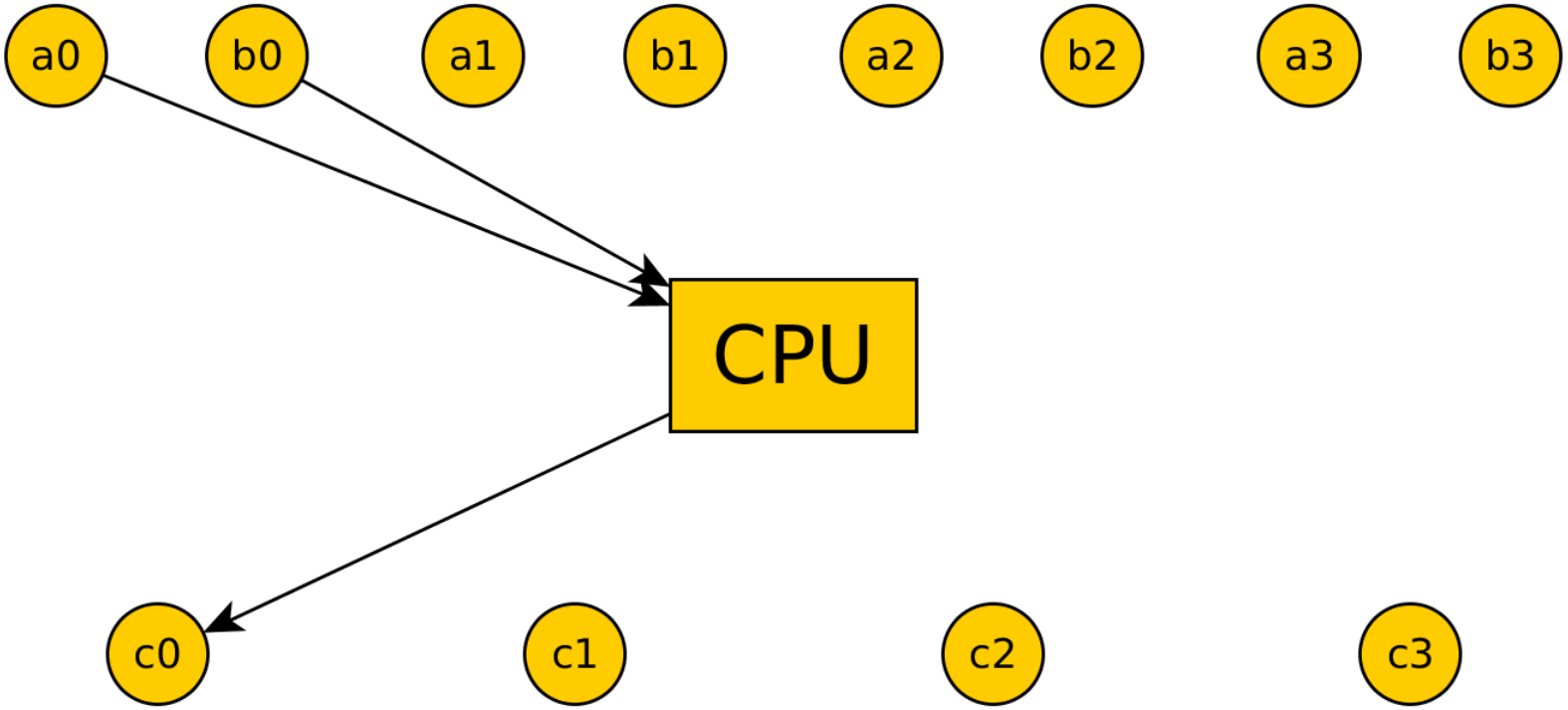
n=4:



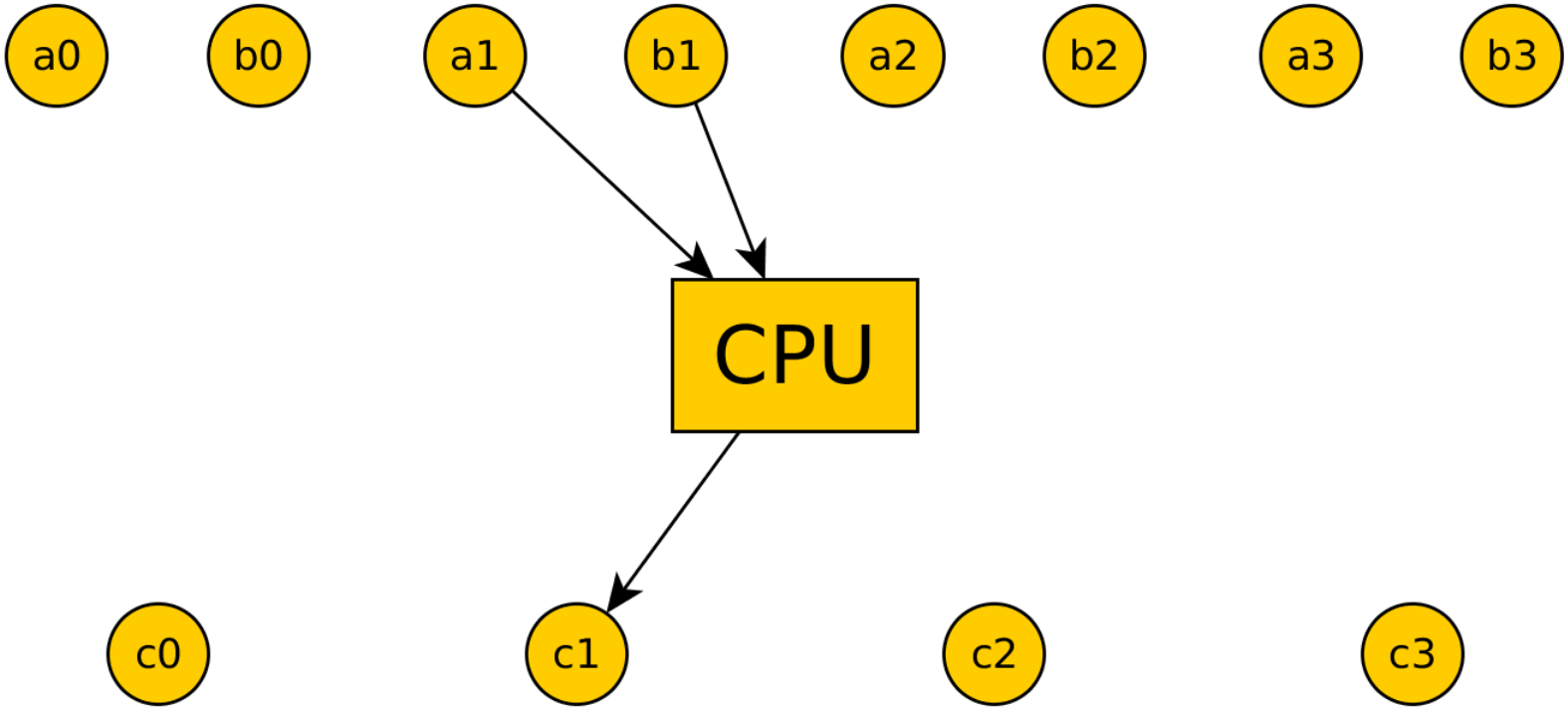
Design Space Exploration – CPU / 1 Adder



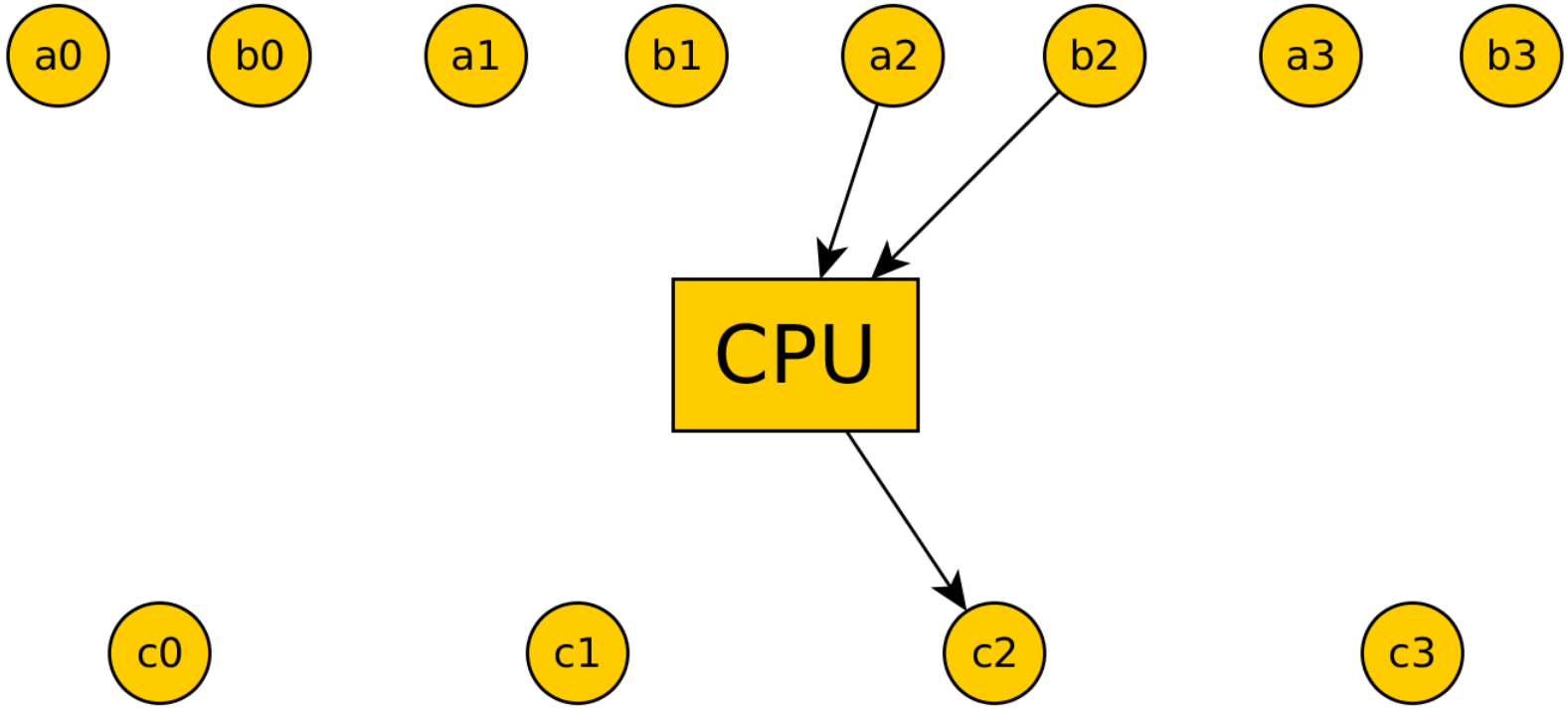
Design Space Exploration – CPU / 1 Adder



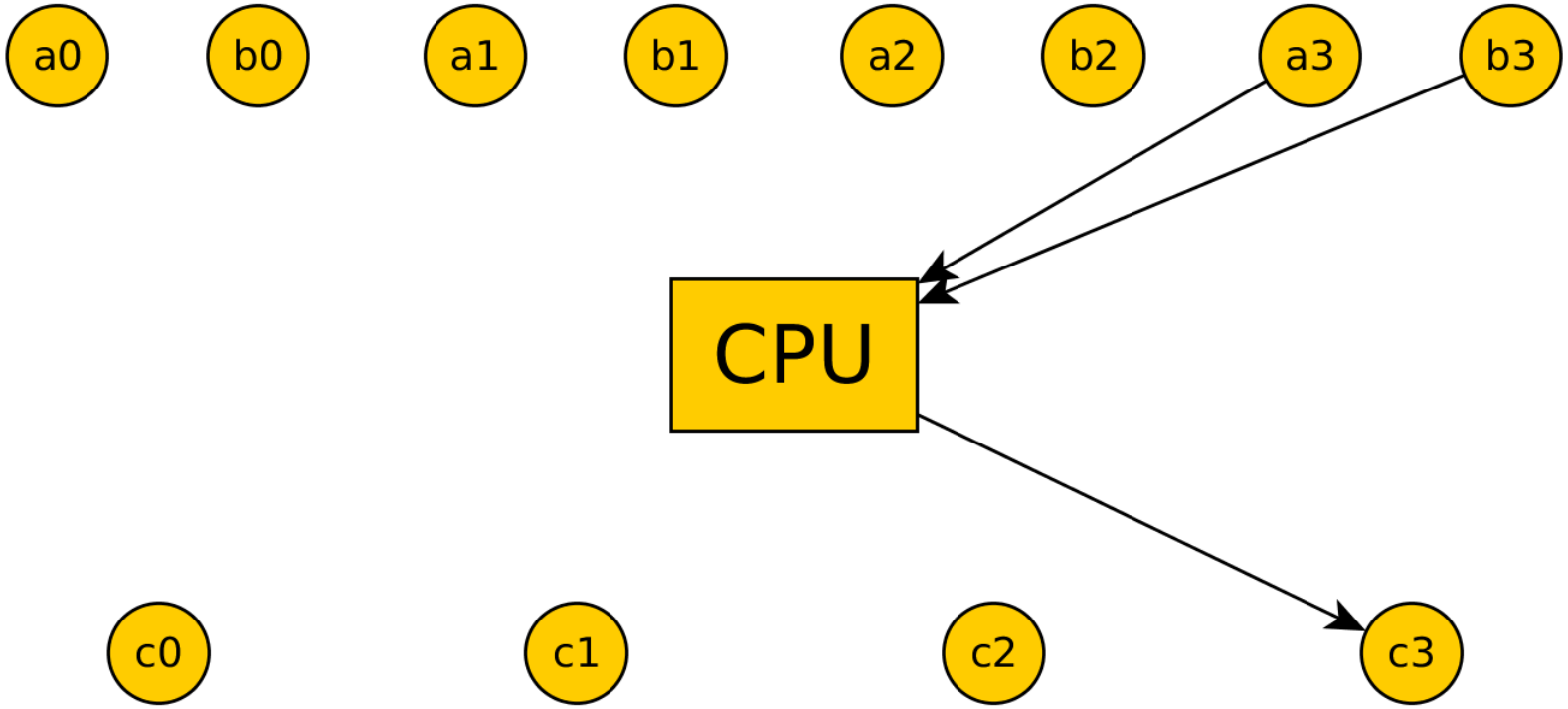
Design Space Exploration – CPU / 1 Adder



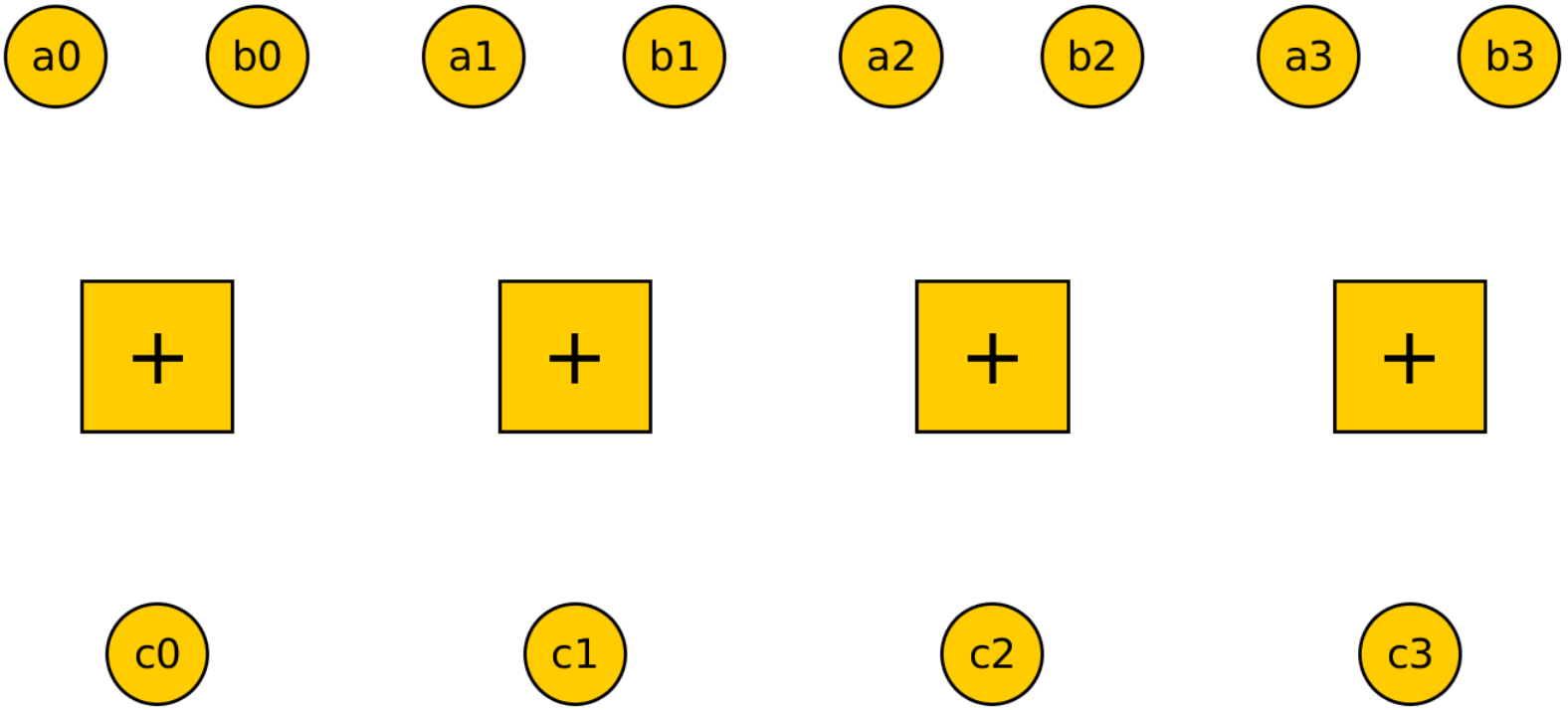
Design Space Exploration – CPU / 1 Adder



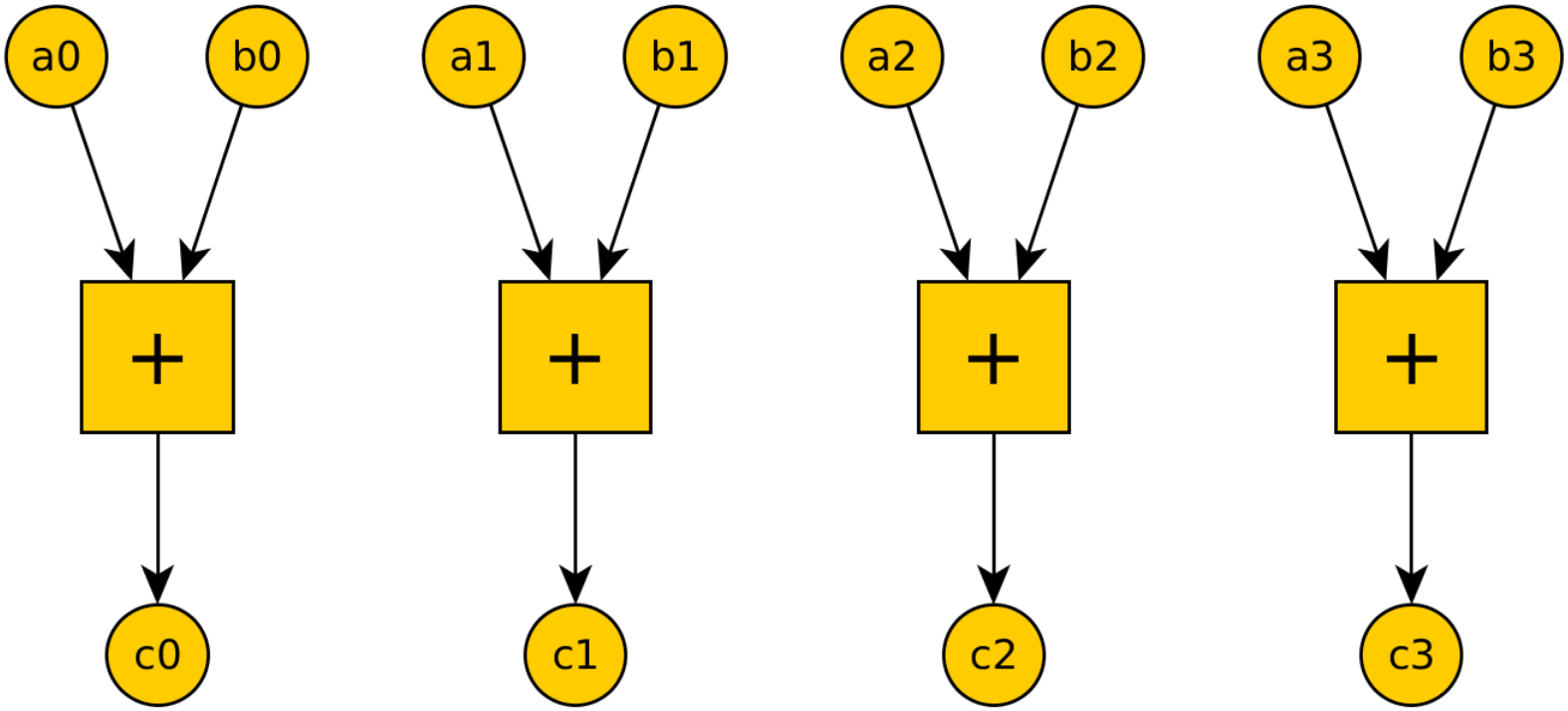
Design Space Exploration – CPU / 1 Adder



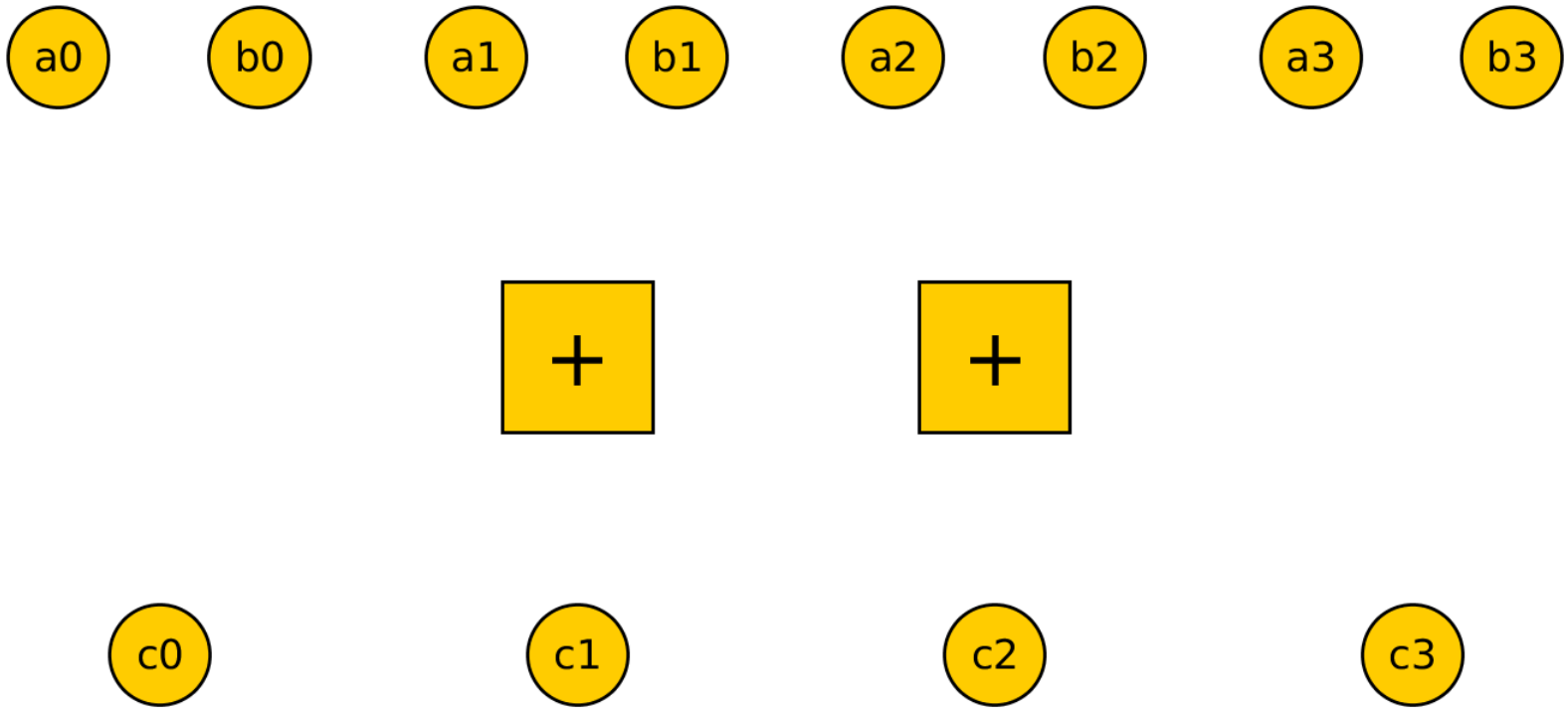
Design Space Exploration – 4 Adder



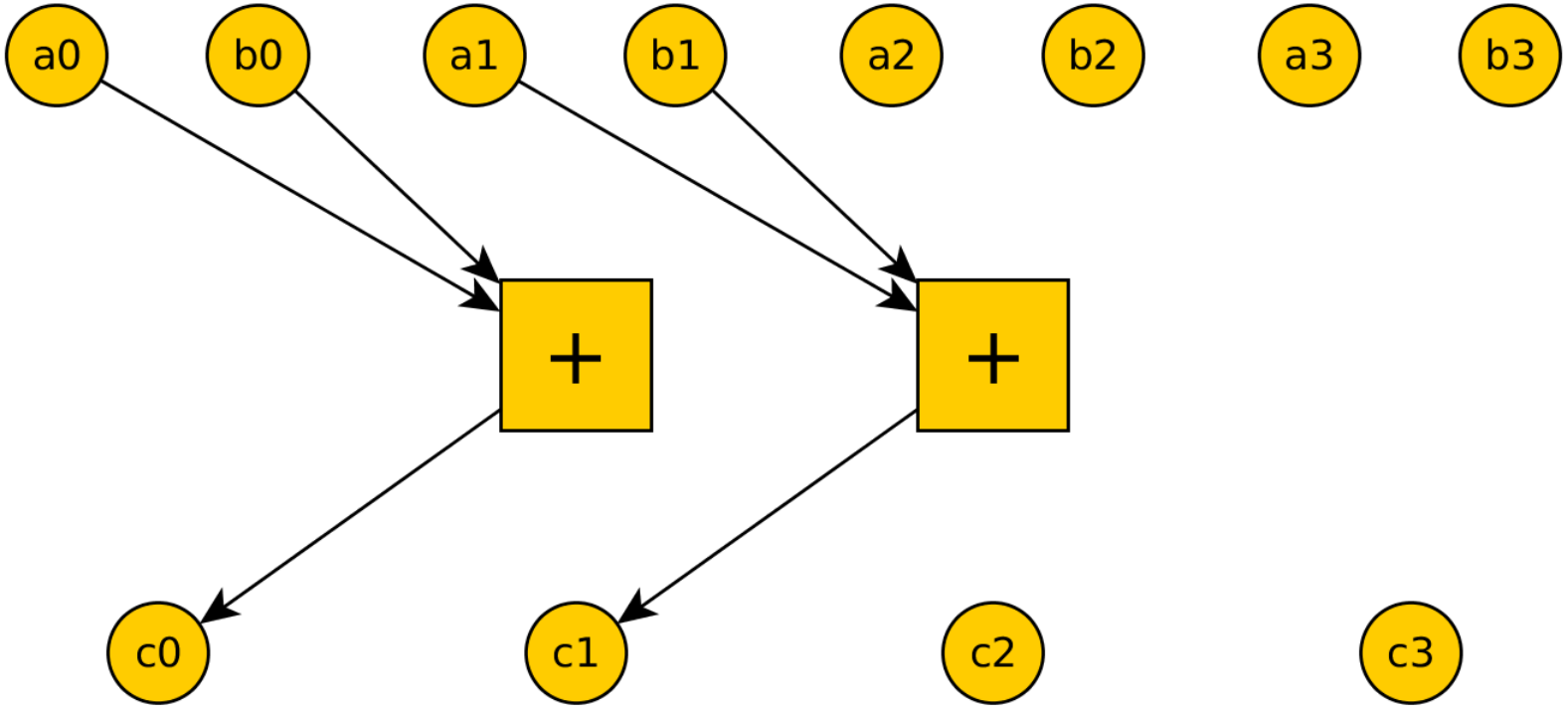
Design Space Exploration – 4 Adder



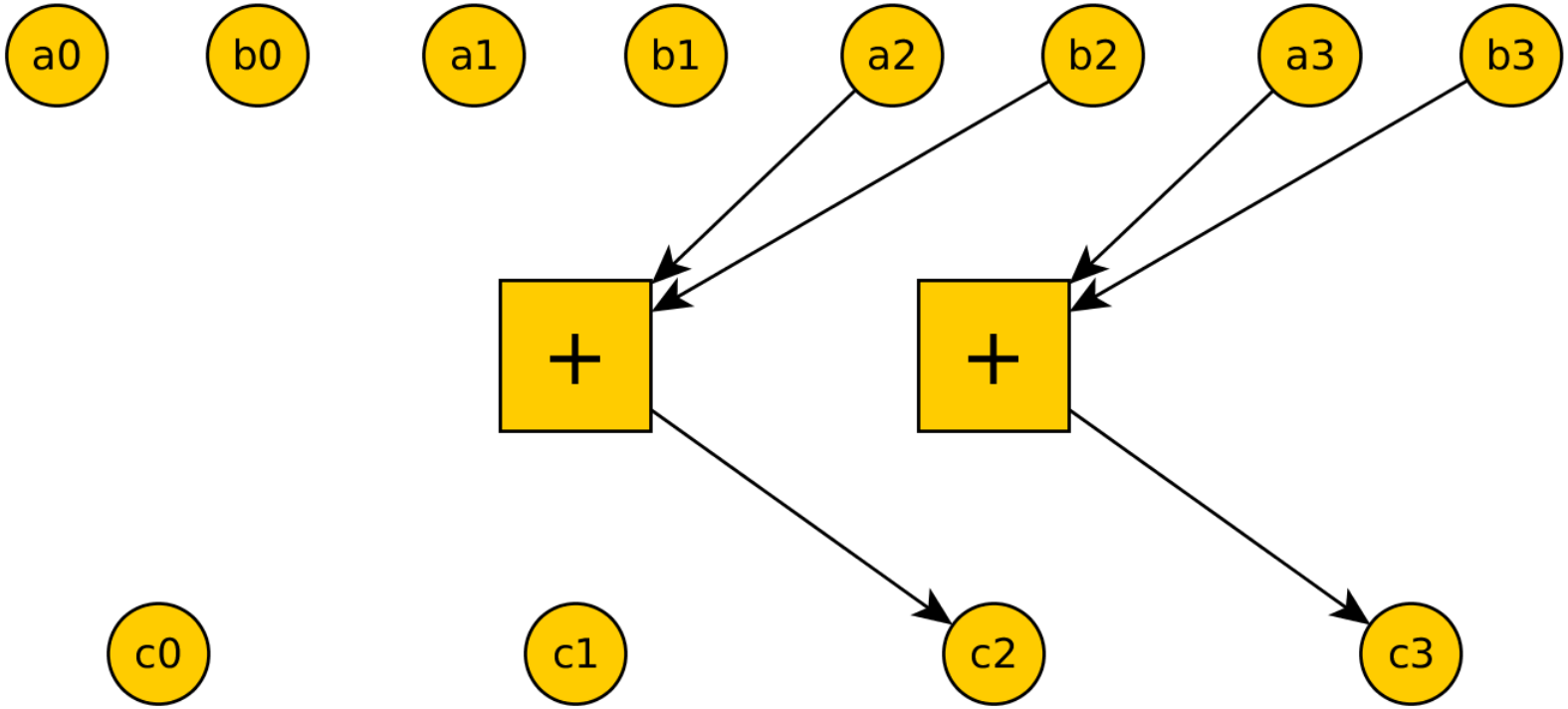
Design Space Exploration – 2 Adder



Design Space Exploration – 2 Adder



Design Space Exploration – 2 Adder



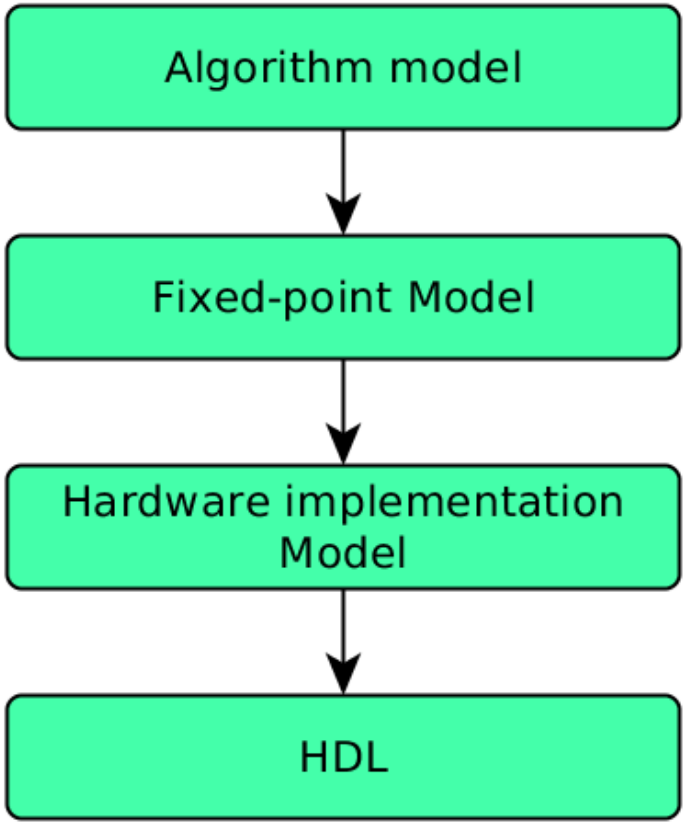
Design Space Exploration – Comparison

	CPU	1 Adder	2 Adder	4 Adder
Area	1.2	1	2	4
Results/Cycle	0.25	0.25	0.5	1
Clock [MHz]	600	400	400	400
Throughput [MOPs]	150	100	200	400

→ There is not a single optimal solution

→ Many of these optimizations needs to be done manually

Simulink / Matlab



- Graphical FPGA Design
- Many Simulink blocks supported
- Verification is integrated

- Examples:
  - Mathworks HDL Coder
  - Altera DSP Compiler
  - Xilinx System Generator

Open Computing Laguage (OpenCL)

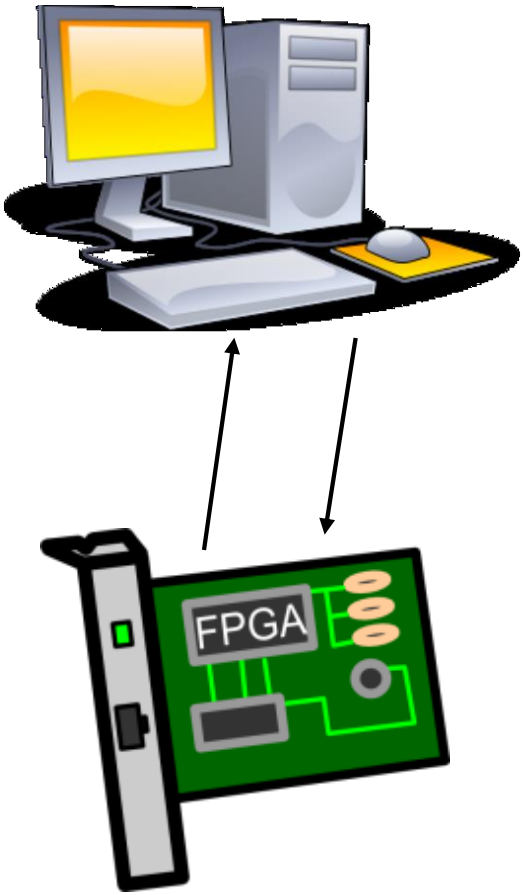
- Extension to standard C
- Description of massively parallel algorithms
- Kernels describe parallel parts of algorithms
- Kernel could execute on different computation units
  - CPU
  - GPGPU
  - FPGA
- Limitations:
  - No stand alone FPGAs – Kernels called by host program
  - (so far) only 1 FPGA board supported

- Application: Altera SDK for OpenCL

OpenCL – (very simplified) Example

```
1  int main() {
2    a = readMatrixFromFile();
3    b = readMatrixFromFile();
4
5    ConfigureAndCompileKernel();
6    CopyDataToDevice();
7
8    RunKernel(addVectors,a,b,c,vector_length)
9
10   CopyDataToHost();
11   PrintMatrix(c);
12 }
```

```
1  __kernel void addVectors(a, b, c, n) {
2
3    i = get_global_id(0);
4
5    c[i] = a[i] + b[i];
6  }
```

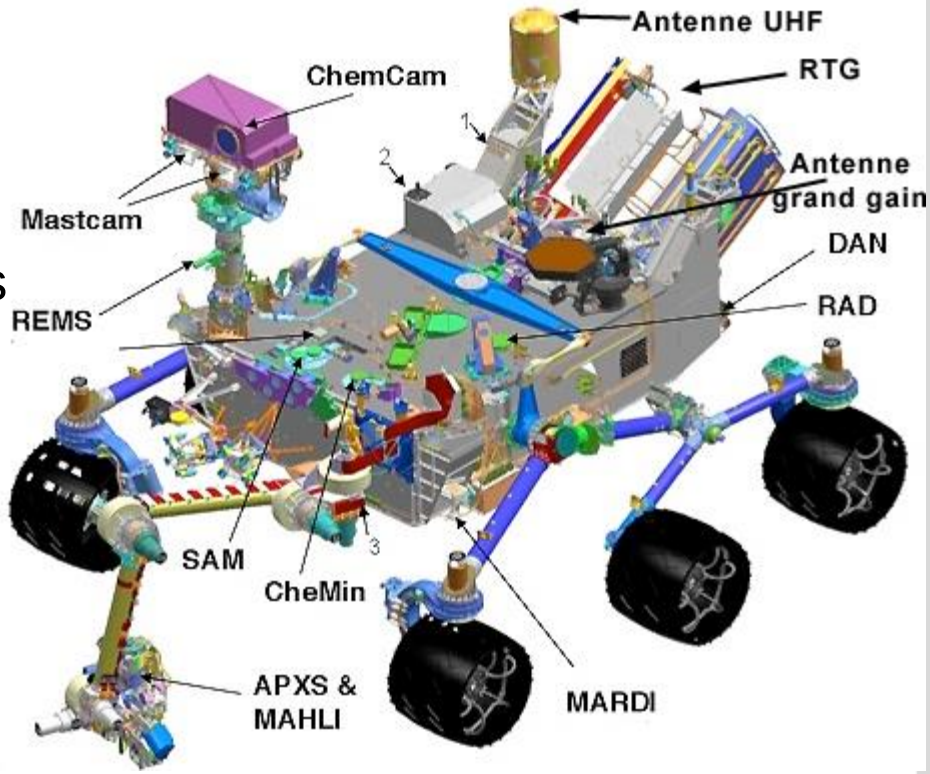


FPGA - Applications

- Digital signal processing
- ASIC prototyping
- Computer vision
- Military applications
- Medical applications
- Automotive applications
- Consumer electronics
- Industrial applications
- High performance computing
- Space and aeronautics

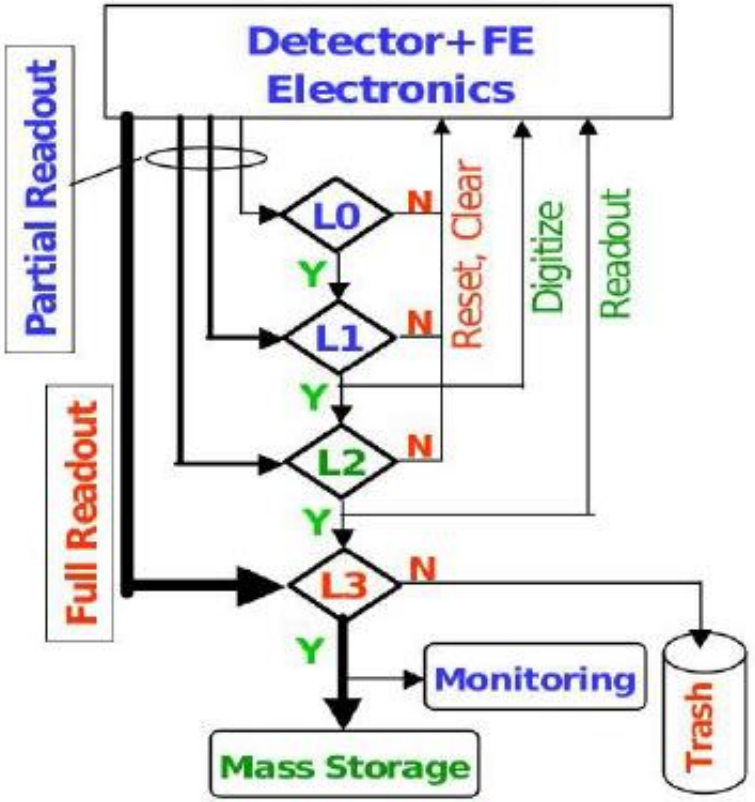
Mars exploration rovers (MER)

- Rad tolerant Xilinx XQVR FPGA
- Control of the pyrotechnic operations during descent and landing procedure
- Control of the motors for the wheels, steering, arms, cameras, instrumentation
- On-board re-programmability allowed design changes and updates even after the rover has landed

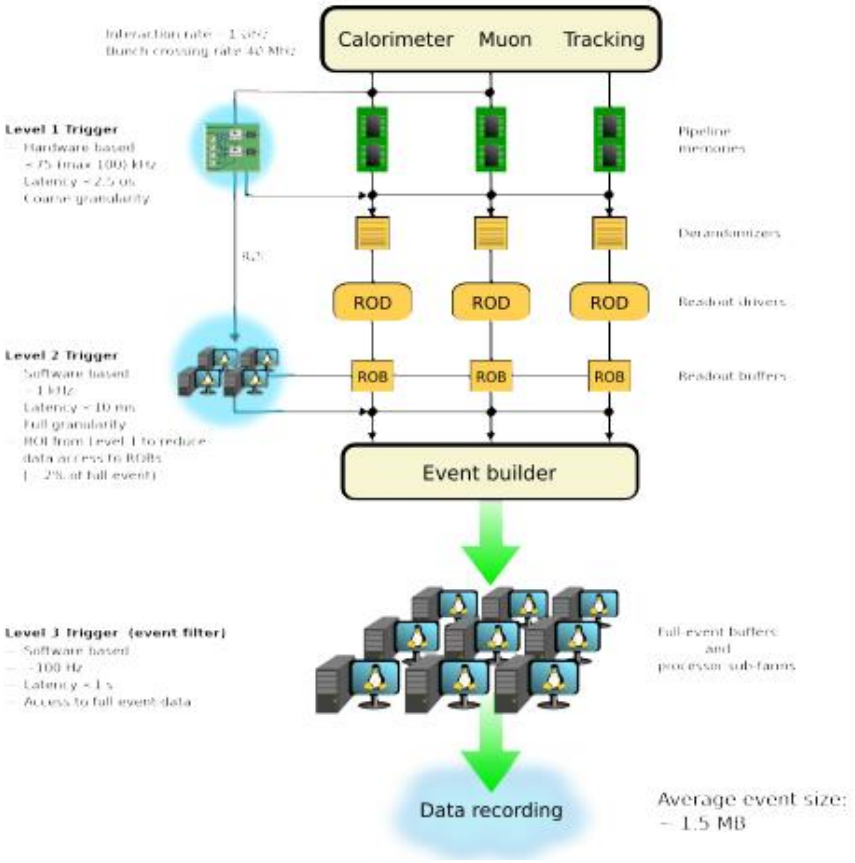


Curiosity Rover

Trigger and DAQ systems in particle physics



FPGA-based trigger for NA62 Experiment



ATLAS Trigger system
L1 is hardware based

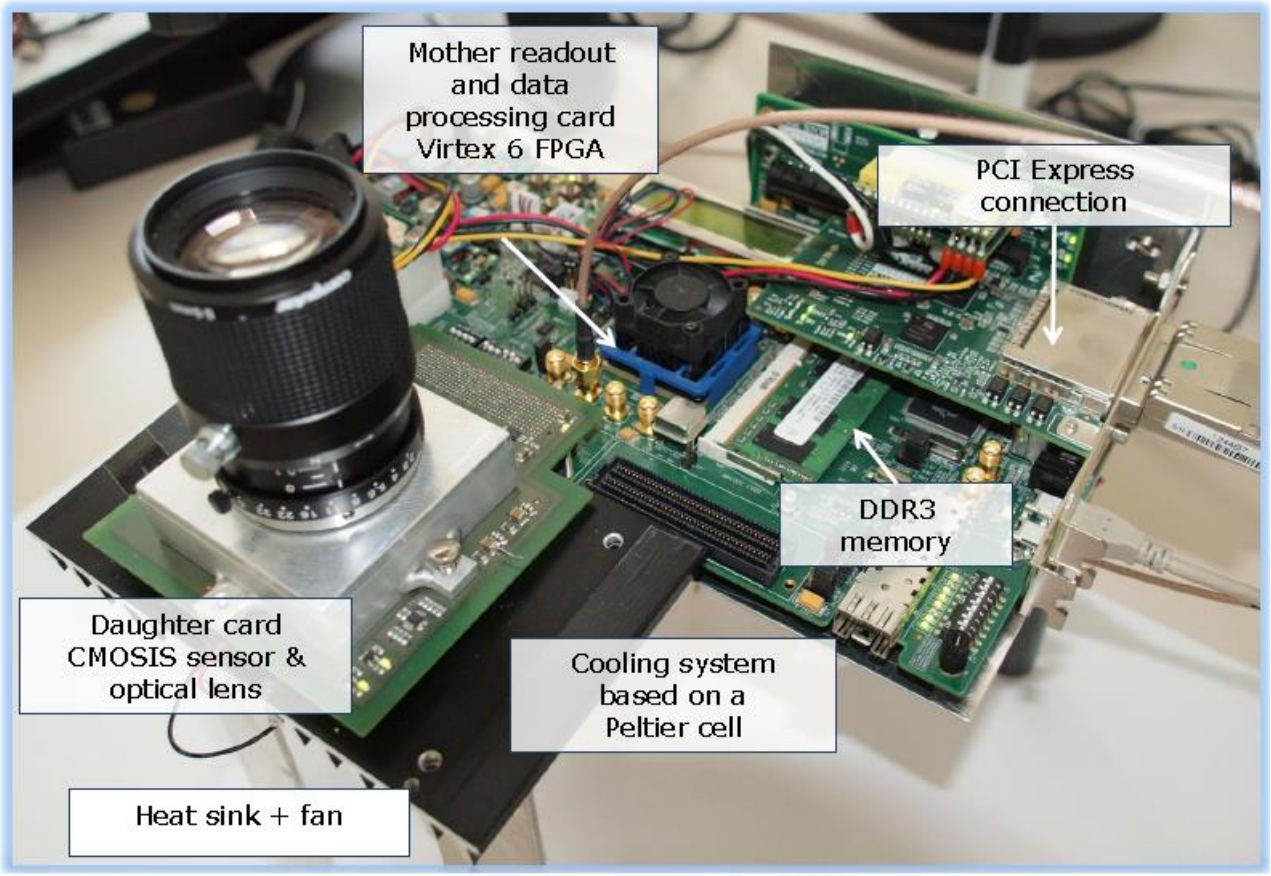
Banking applications

- Calculating company's collateral debt obligation (CDO) in near real-time
- Prior to the FPGA solution, main risk model for analyzing CDO portfolio took 8-12 hrs, based on x1000 x86 cores → in case of error, no time to resubmit for the day!
- With the speedup, same risk model took 4 minutes → multiple scenarios throughout the day
- Final hardware system is 40-node hybrid cluster
- Each cluster contains 8 Xeon (24GB) cores with 2 FPGA (Xilinx Virtex 5) (12GB)
- Time-critical, compute-intensive pieces of C++ risk model was ported to FPGA
- Advantage of the FPGA → exploit of the fine-grained parallelism and pipelines → many more calculations per watt vs. the CPU

KIT UFO camera

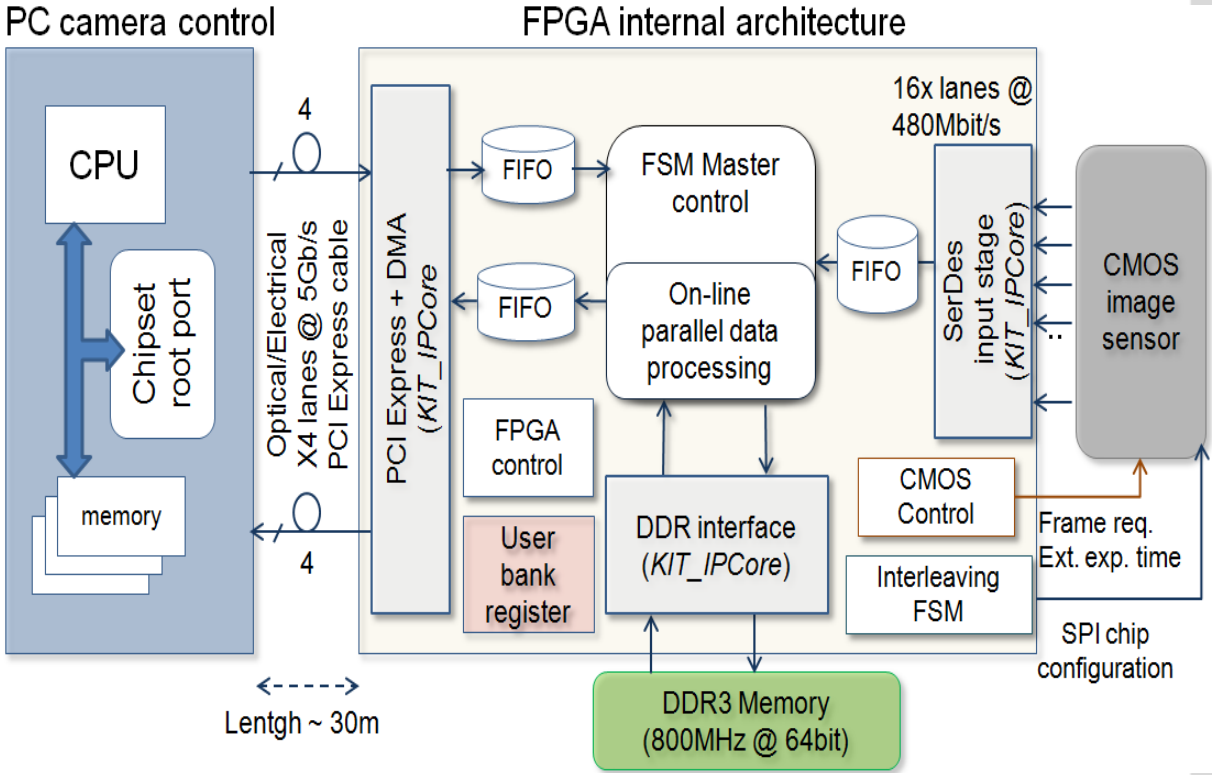
- Ultra Fast X-ray imaging of scientific processes with On-line assessment and data-driven process control

- High-speed data transfer
- Image-based process control
- Programmable camera
- On-camera image processing



Readout chain and FPGA architecture

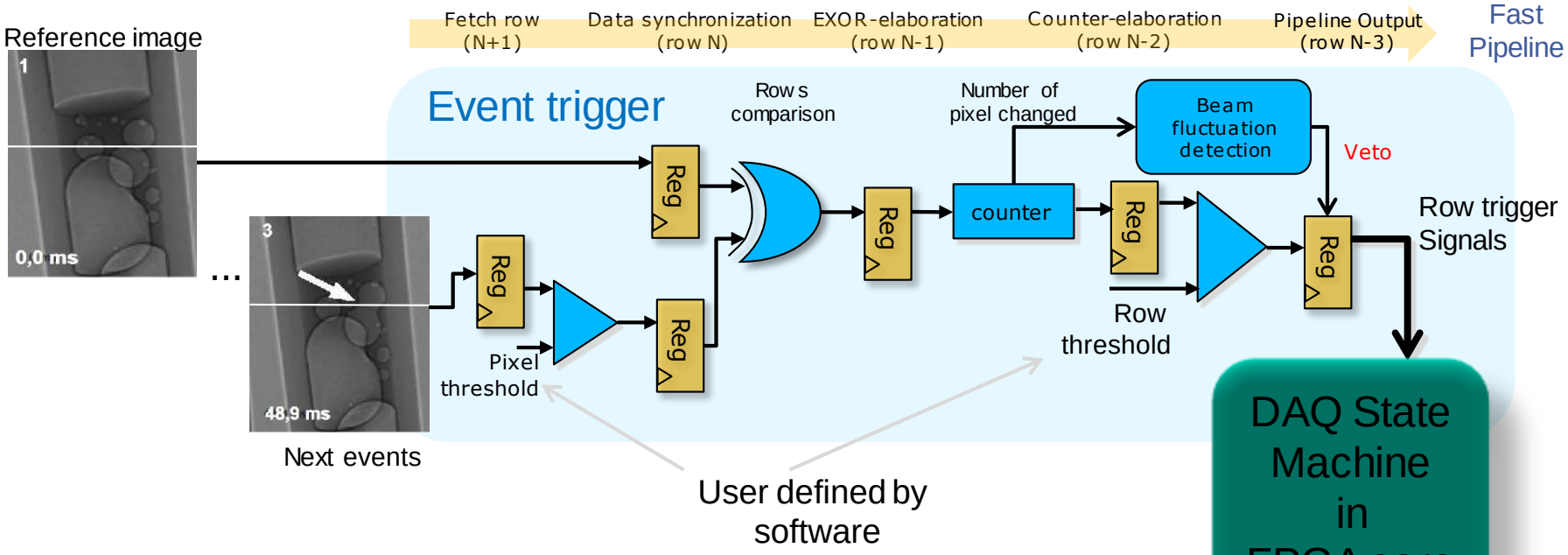
- 2.2 MP CMOS sensor
- 340 fps @ full view
- 14 Gb/s streaming with Bus Master DMA
- Virtex-6 Xilinx FPGA



Sub-sampling strategy for rows 'fast reject'

Generating fast reject signals (triggers) @ rows level → starting from a reference image

Subsampling strategy is used to allow a fast readout frame @ kHz range

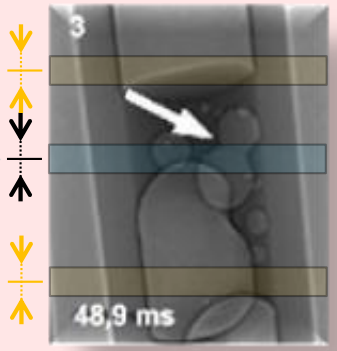


Event Readout

To limit the amount of data and increase the frame rate from sensor, windowing in Y direction is possible. The number of lines and start address can be set by SPI

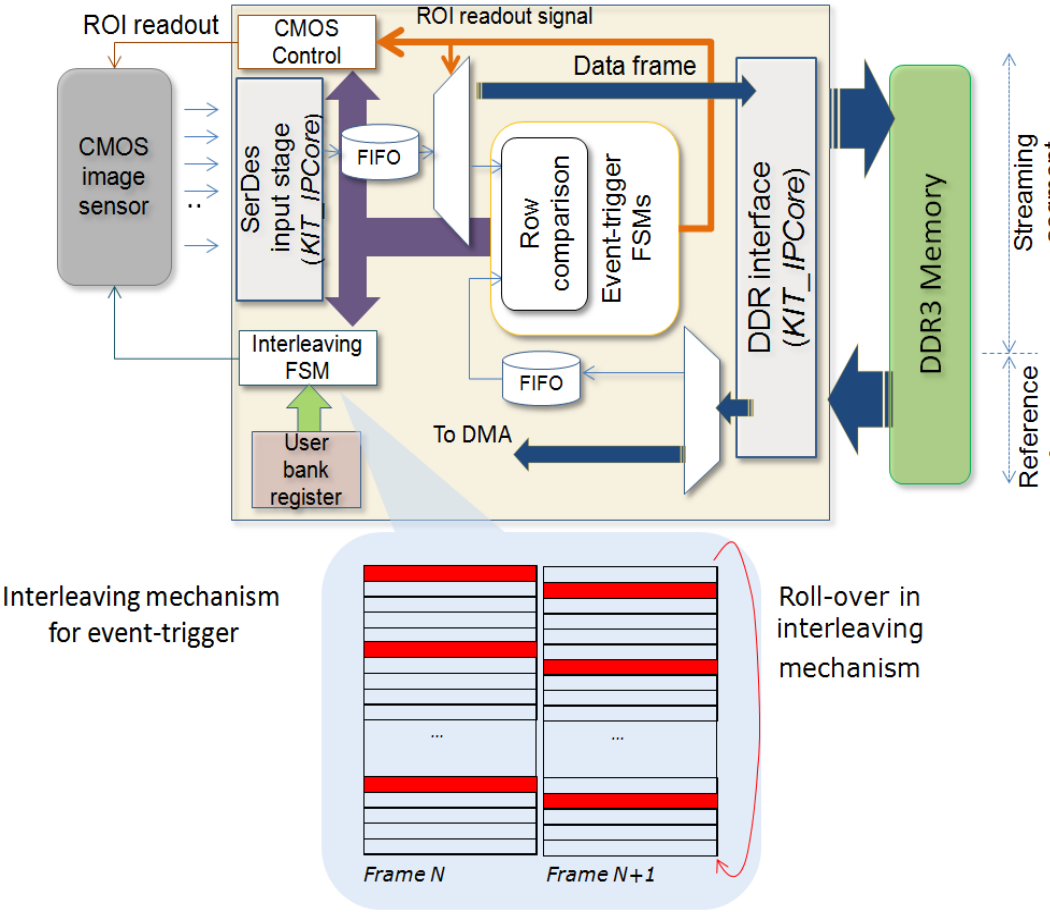
Readout of a window around the row trigger signal or full frame can be requested

Optionally the multiple windows can be defined when a multiple rows trigger are presents

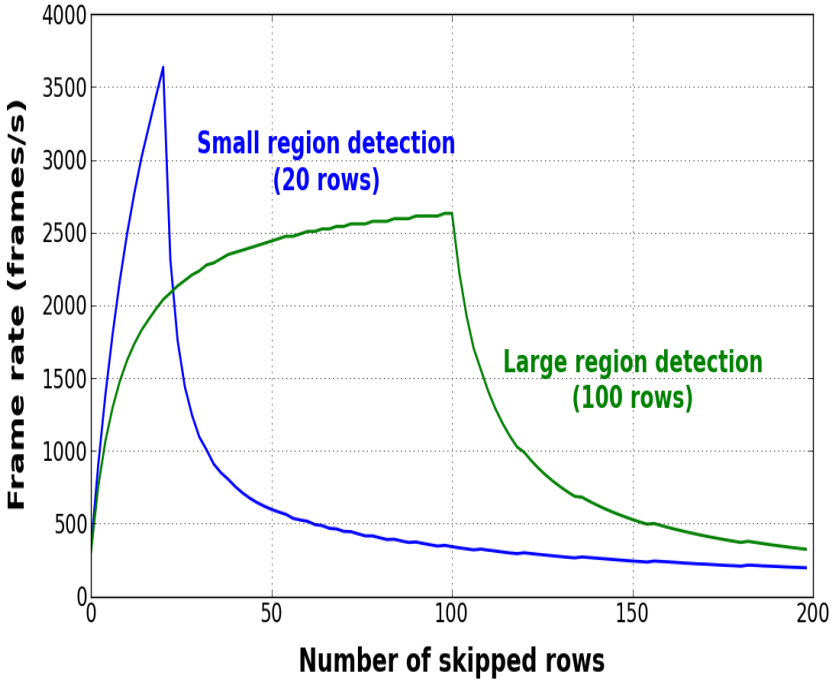


SPI chip configuration signals

Image based trigger – architecture and performance



FPGA architecture



Performance

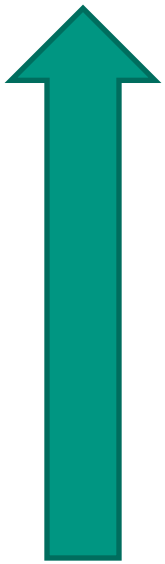
CPU, GPU vs. FPGA

Often FPGAs and CPU/GPU are complementary: they co-exist in the same system and perform different tasks.

FPGA Advantages	CPU/GPU Advantages
• more flexible processing • more flexible input/outpt • parallel processing • multi-clock • timing operations • higly customizable • “real-time” applications	• programming a CPU in normally easier than programming an FPGA (no understanding of the digital electronics) • faster compilation • easier code portability • lower unit costs - for any volume • GPU offers massive parallel execution resources and high memory bandwith • “fast” applications

Applications suitability for GPU and FPGA

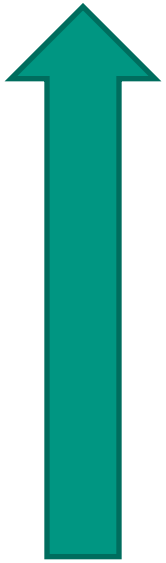
Good fit



Poor fit

FPGAs	GPUs
Computations involves lots of detailed low-level hardware control operations, with no efficient implementation in high level language	No independence in the data flow and computation can be done in parallel
A certain degree of complexity is required and the implementation take advantage of data streaming and pipelining	Applications contain a lot of parallelism but involve computations which cannot be efficiently implemented on GPUs
Applications that require a lot of complexity in the logic and data flow design	Applications have a lot of memory accesses and have limited parallelism

Good fit



Poor fit

Karlsruher Institut für Technologie

Q & A