Karlsruher Institut für Technologie

Institut für Technik der Informationsverarbeitung

KSETA Workshop 2013

Institut für Prozessdatenverarbaiten und Elektronik - IPE

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

Institut für Technik der Informationsverarbeitung (ITIV)

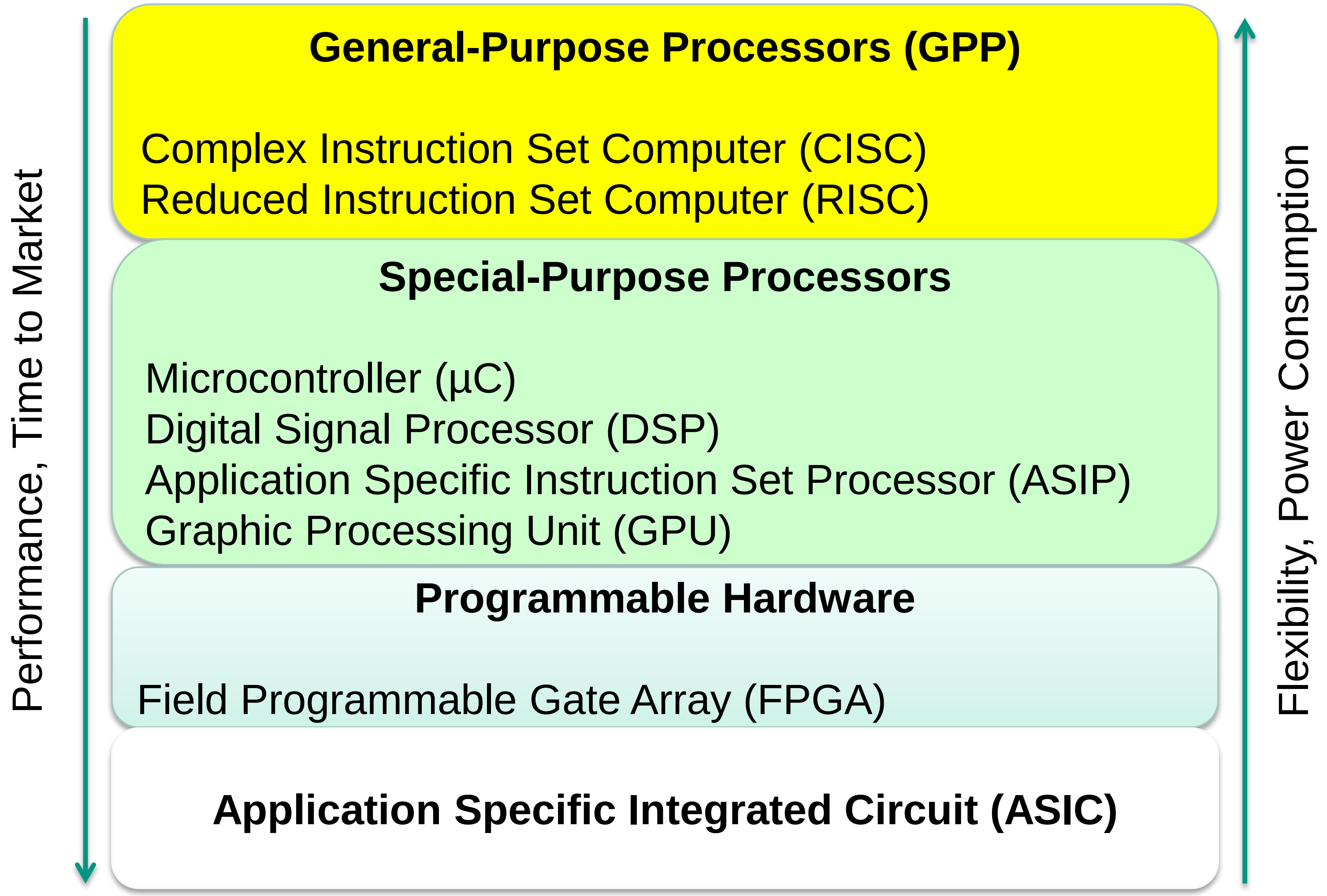
Institut für Prozessdatenverarbeitung und Elektronik (IPE)

Data Analysis in Hardware - A Tutorial on VHDL and FPGAs

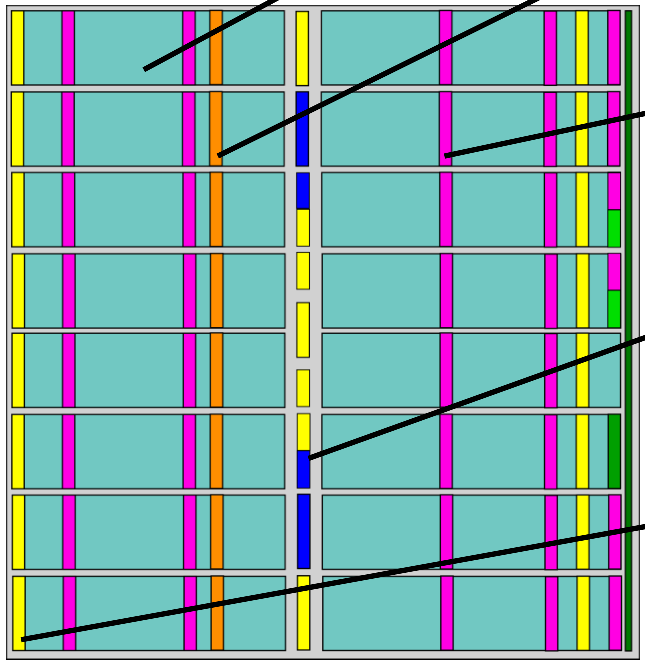
Overview

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

Comparison between Technologies

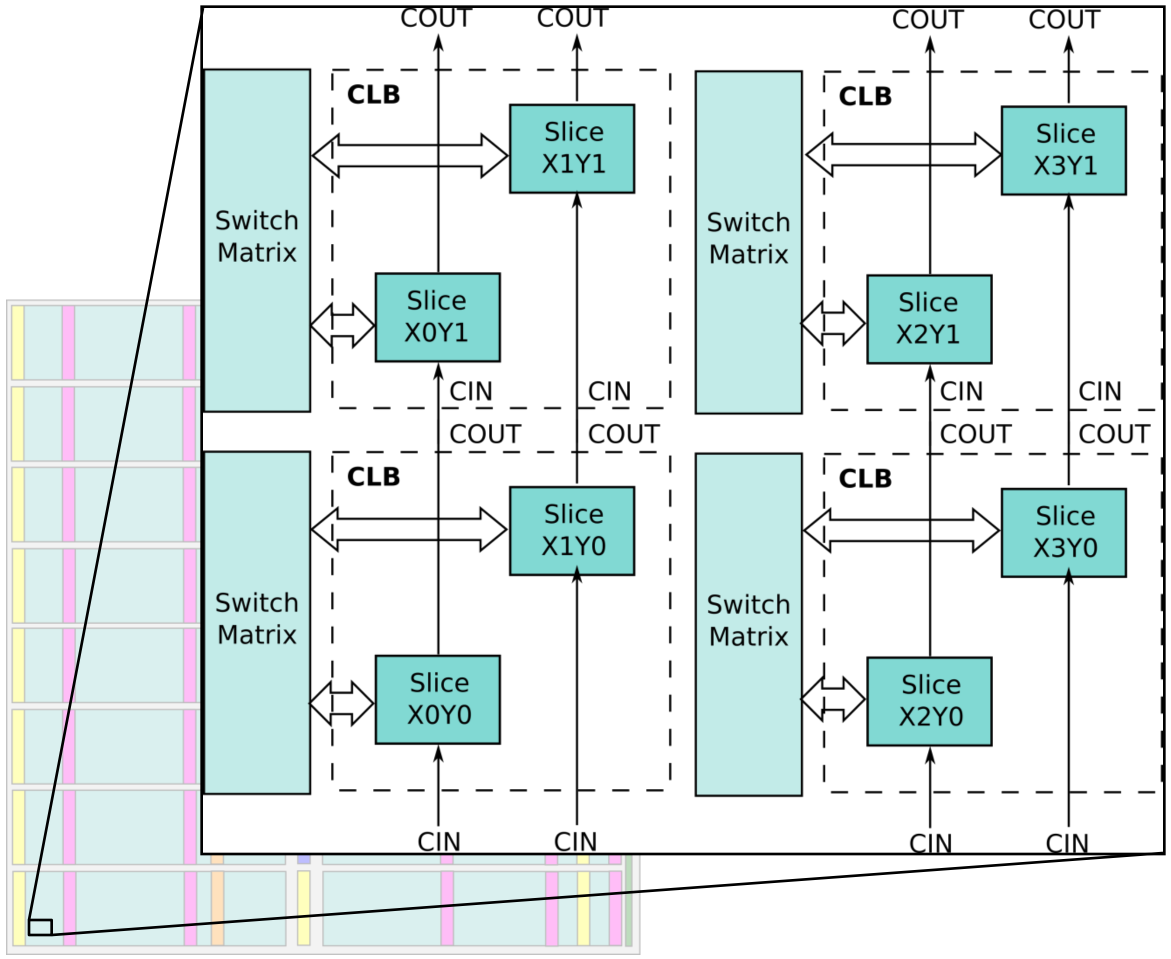

Performance, Time to Market
General-Purpose Processors (GPP)
Complex Instruction Set Computer (CISC)
Reduced Instruction Set Computer (RISC)
Special-Purpose Processors
Microcontroller (µC)
Digital Signal Processor (DSP)
Application Specific Instruction Set Processor (ASIP)
Graphic Processing Unit (GPU)
Programmable Hardware
Field Programmable Gate Array (FPGA)
Application Specific Integrated Circuit (ASIC)
Flexibility, Power Consumption

Example: Xilinx Virtex5 110T



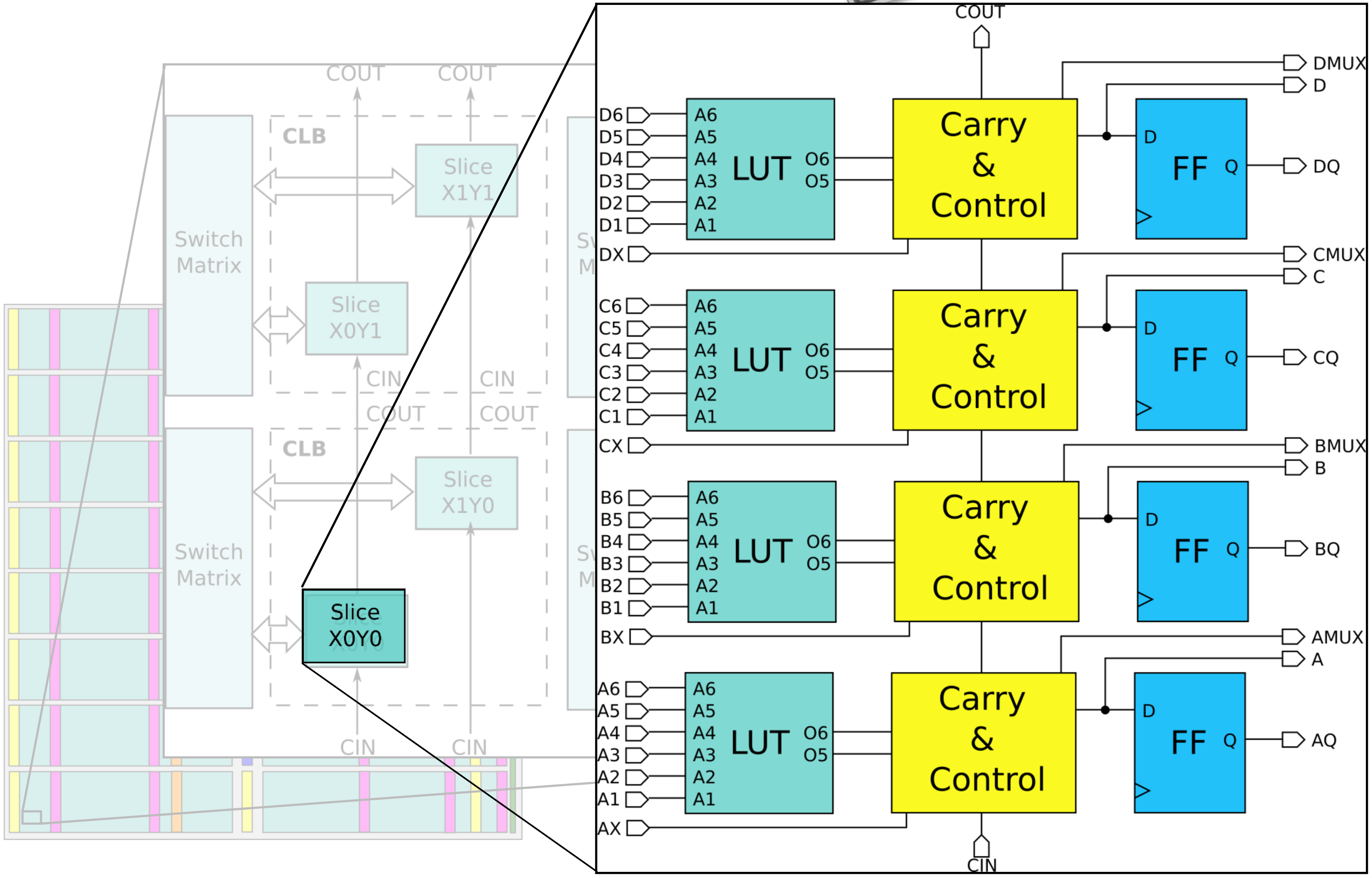
- Configurable Logic Block (CLB)
- Digital Signal Processor (DSP)
- Block RAM (BRAM)
- Digital Clock Memory (DCM)
- I/O Bank

Example: Xilinx Virtex5 110T



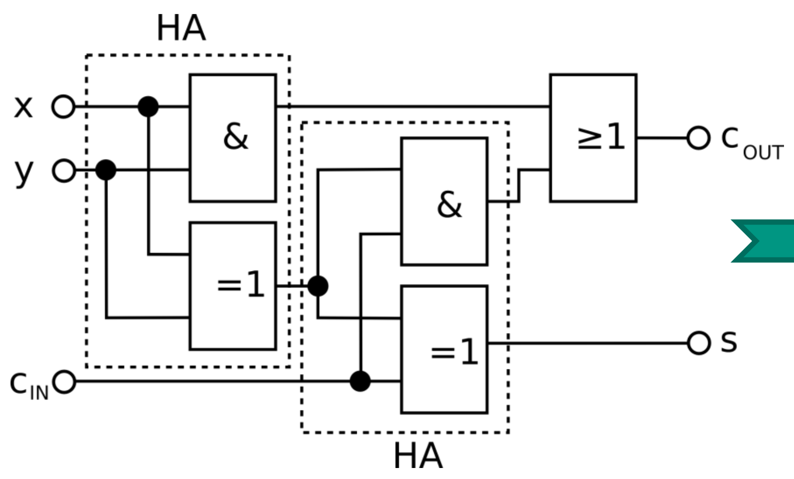
- One CLB consists of two Slices
- Programmable Switch Matrix
- Fast local routing inside a CLB
- Global routing between CLBs

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

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

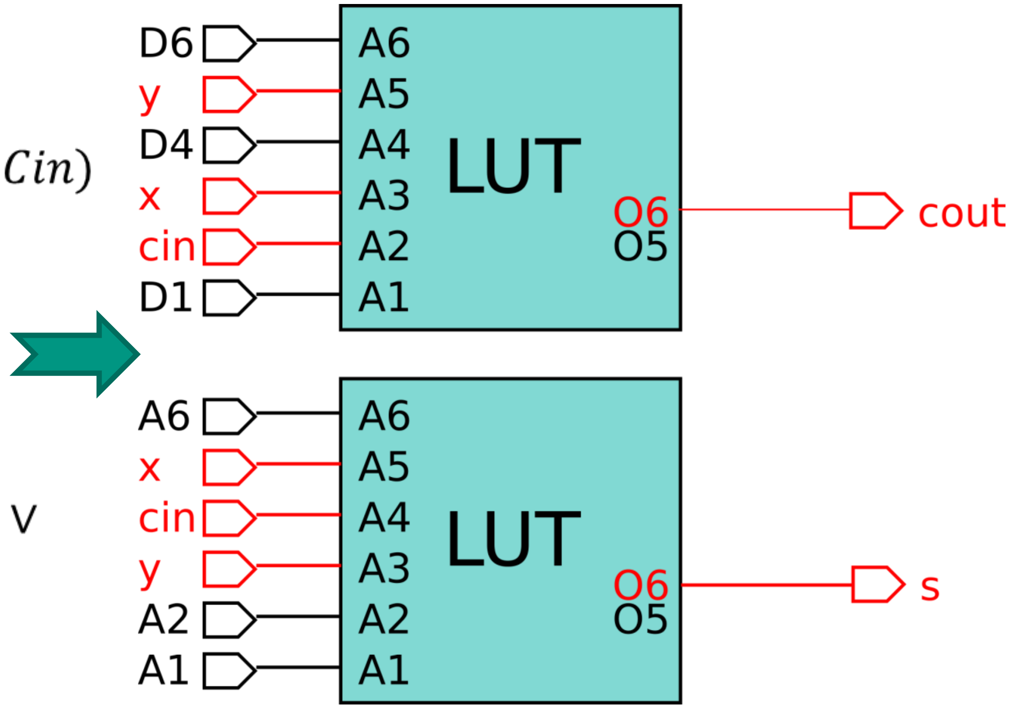
$$Cout = (\overline{x} \wedge y \wedge Cin) \vee (x \wedge y) \vee (x \wedge Cin)$$

How to map a full adder into a Slice

- Step 2: Realization with two Lookup Tables

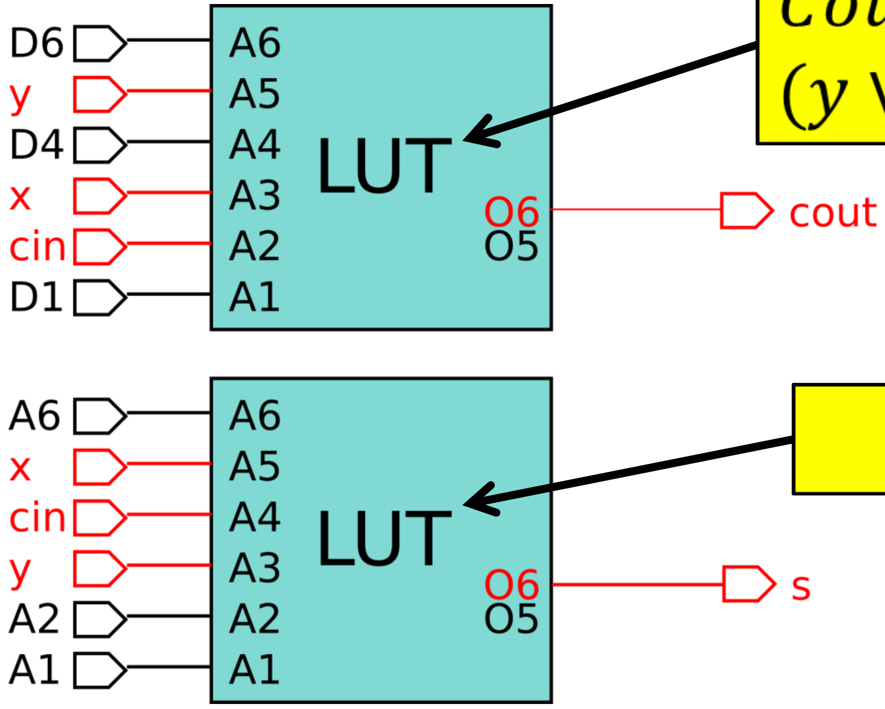
$$Cout = (\overline{x} \wedge y \wedge Cin) \vee (x \wedge y) \vee (x \wedge Cin)$$

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



How to map a full adder into a Slice

- Step 2: Realization with two Lookup Tables

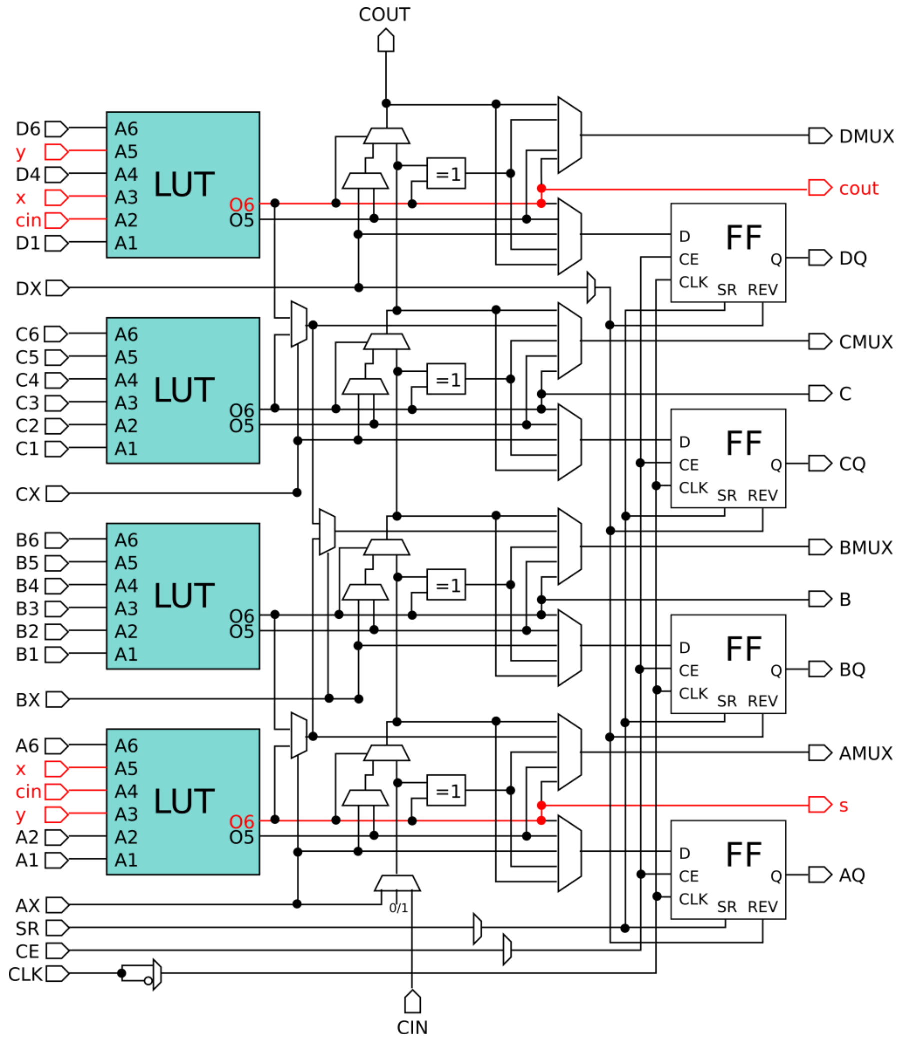


$Cout = (\overline{x} \wedge y \wedge Cin) \vee x \wedge (y \vee Cin)$

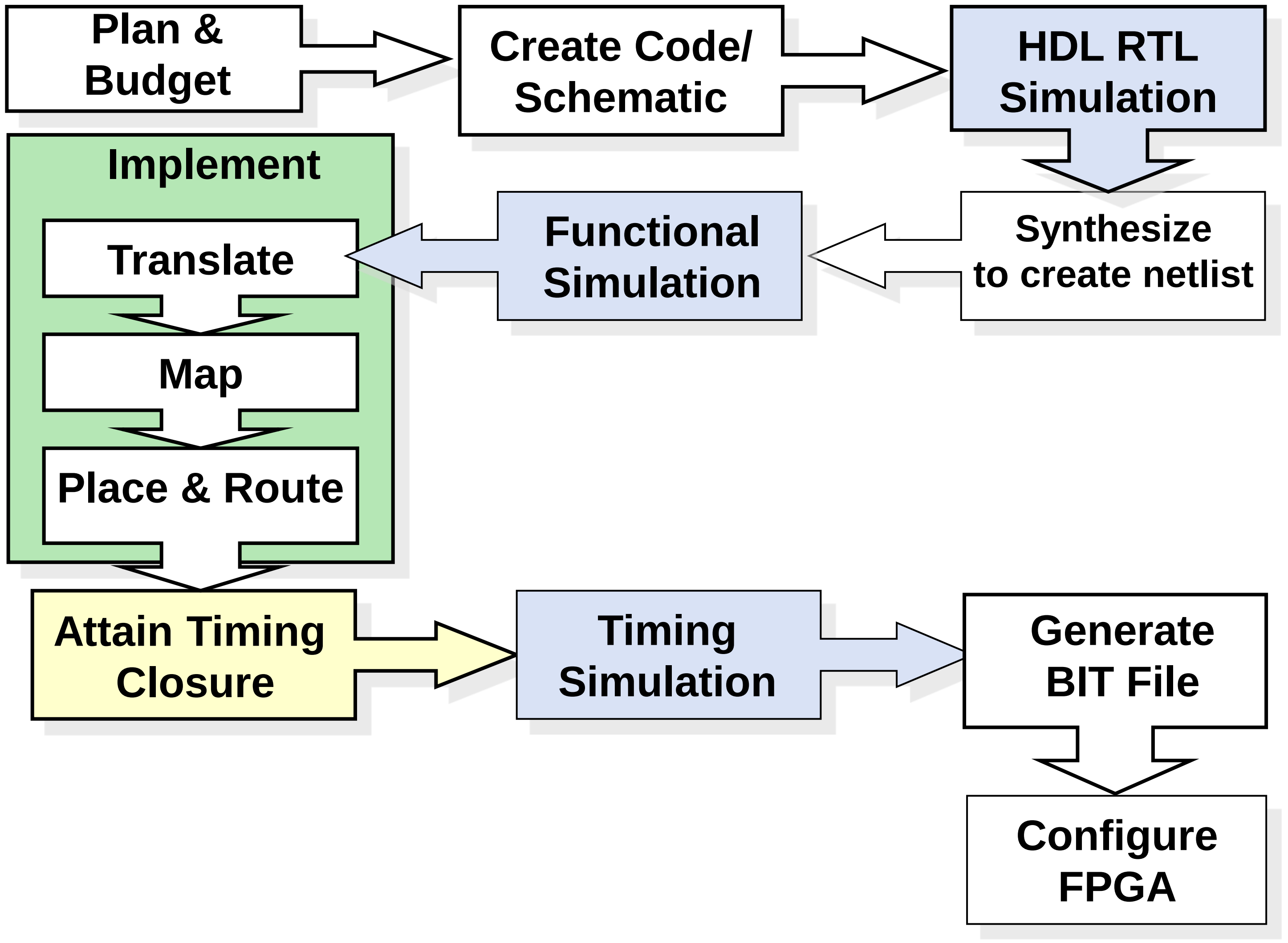
$s = (Cin) \oplus (x \oplus y)$

Generated by Xilinx ISE 14.2

Full adder in a Virtex5 ScliceL

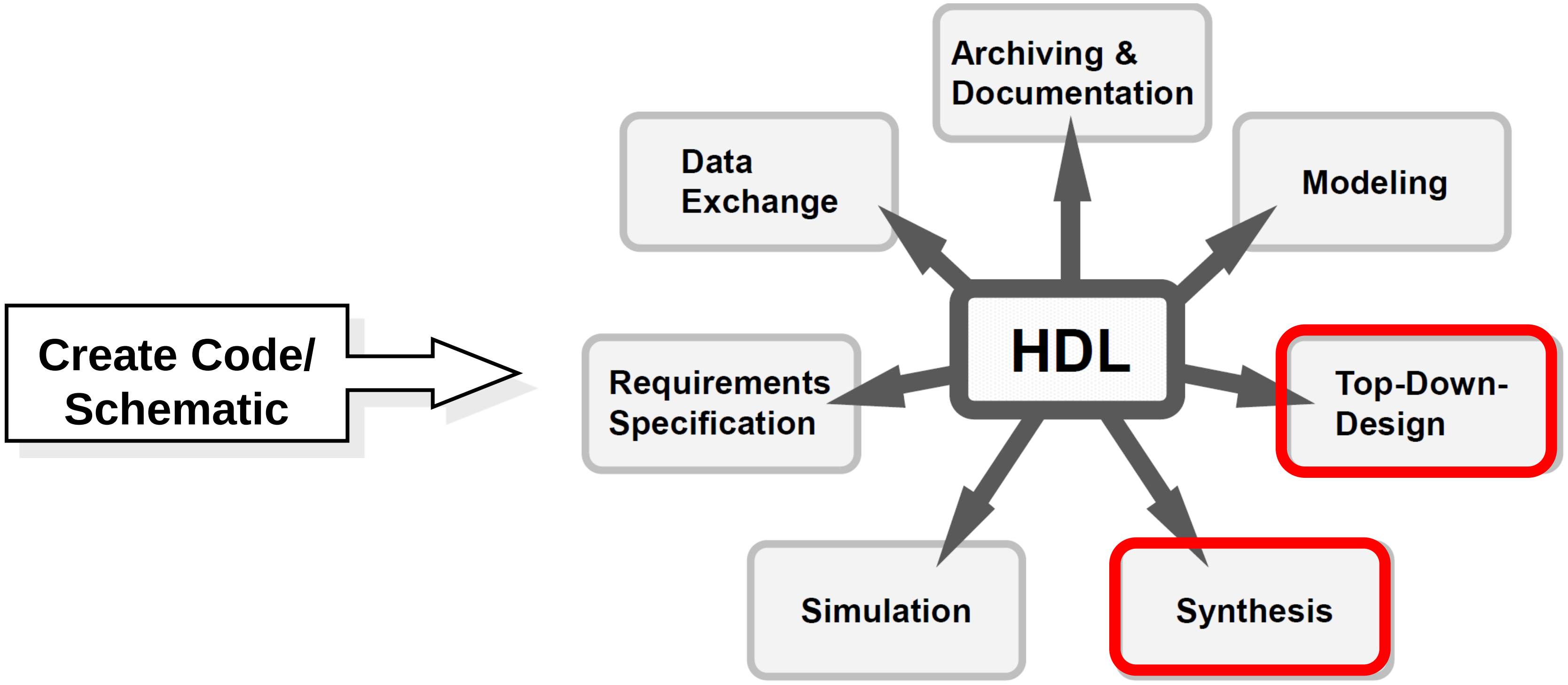


FPGA Design Flow



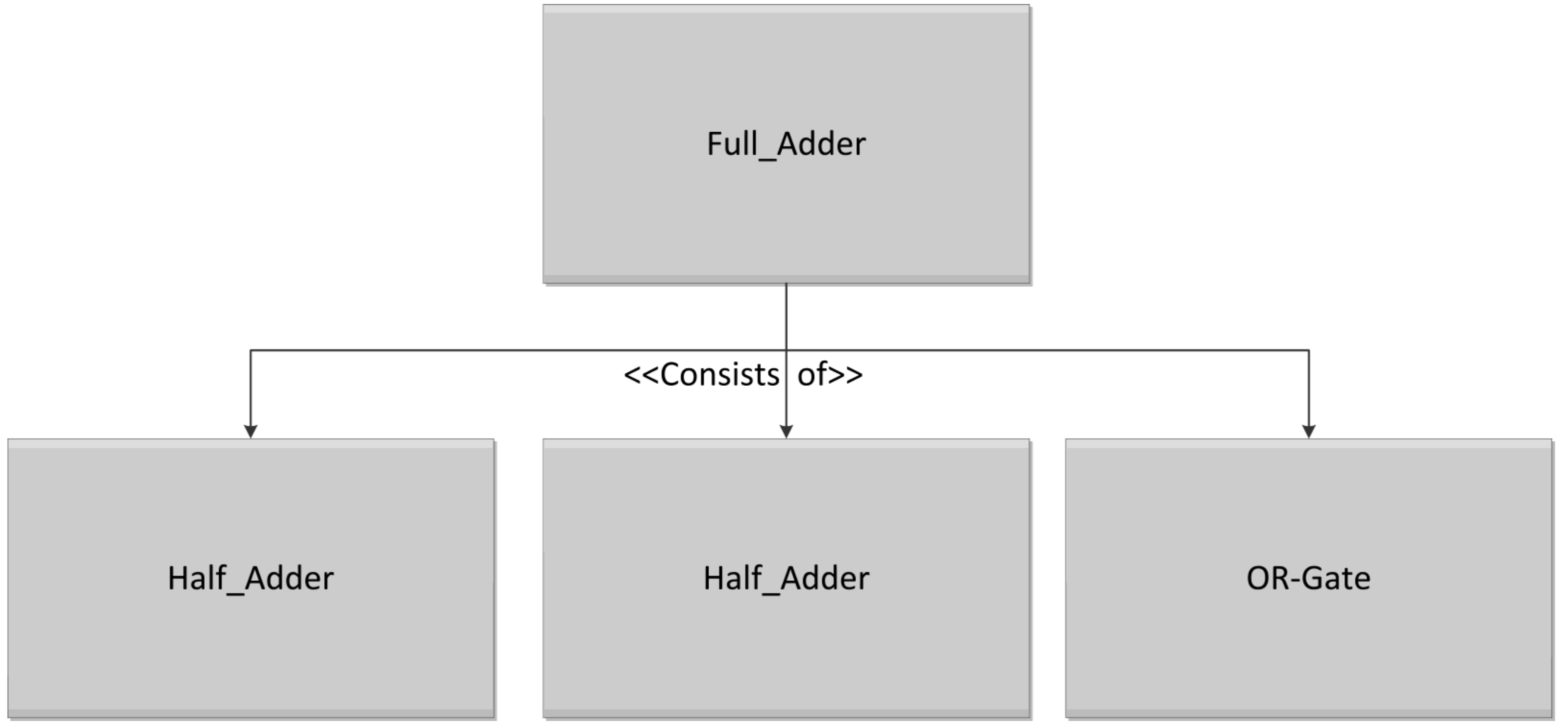
VHDL-Introduction

- 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

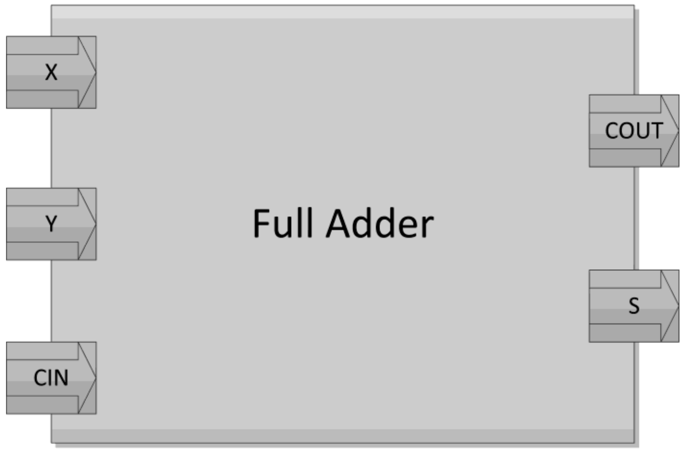


VHDL-Entity

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

```
entity full_adder
        Port(
                  x              : in   std_logic;
                  y              : in   std_logic;
                  cin            : in   std_logic;
                  s              : out  std_logic;
                  cout           : out  std_logic
        );
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

```
signal name : type;
```

VHDL signal declaration syntax

```
signal result : std_logic;
```

VHDL signal declaration

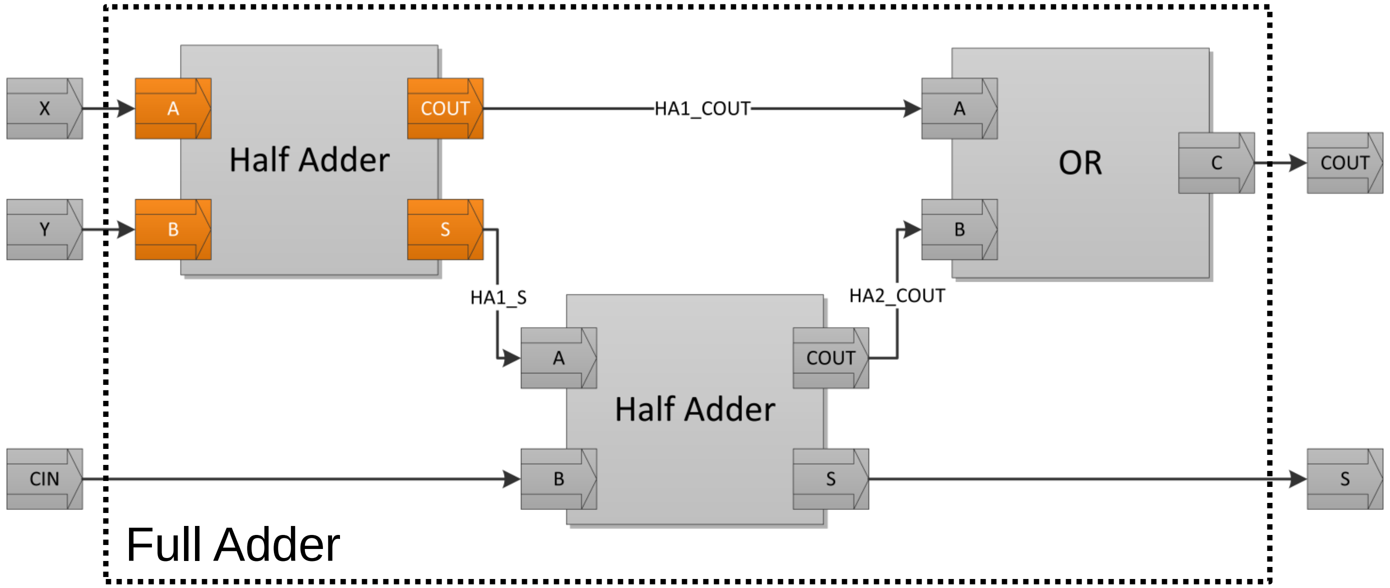
VHDL-Architecture

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

```
architecture structure of full_adder  is
..
end structure full_adder;
```

Architecture declaration of full adder in VHDL

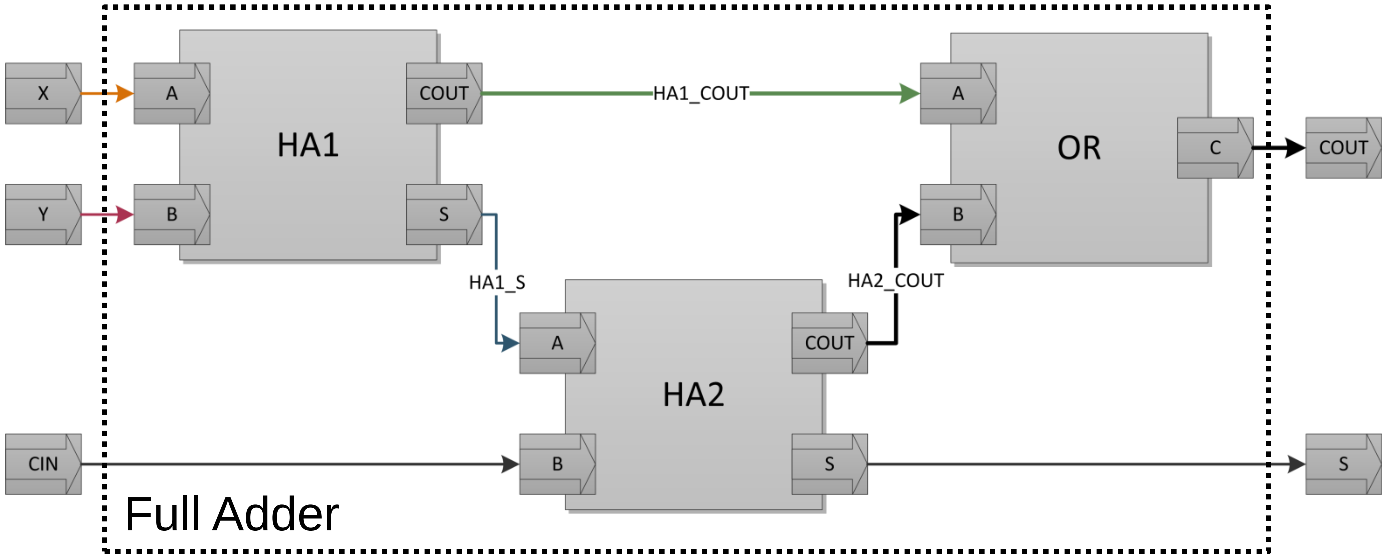
VHDL-Structural Description(1)



```vhdl
component half_adder
        Port(
                  a      : in  std_logic;
                  b      : in  std_logic;
                  s      : out std_logic;
                  cout   : out std_logic
        );
component half_adder;
```

Component declaration of full adder entity in VHDL

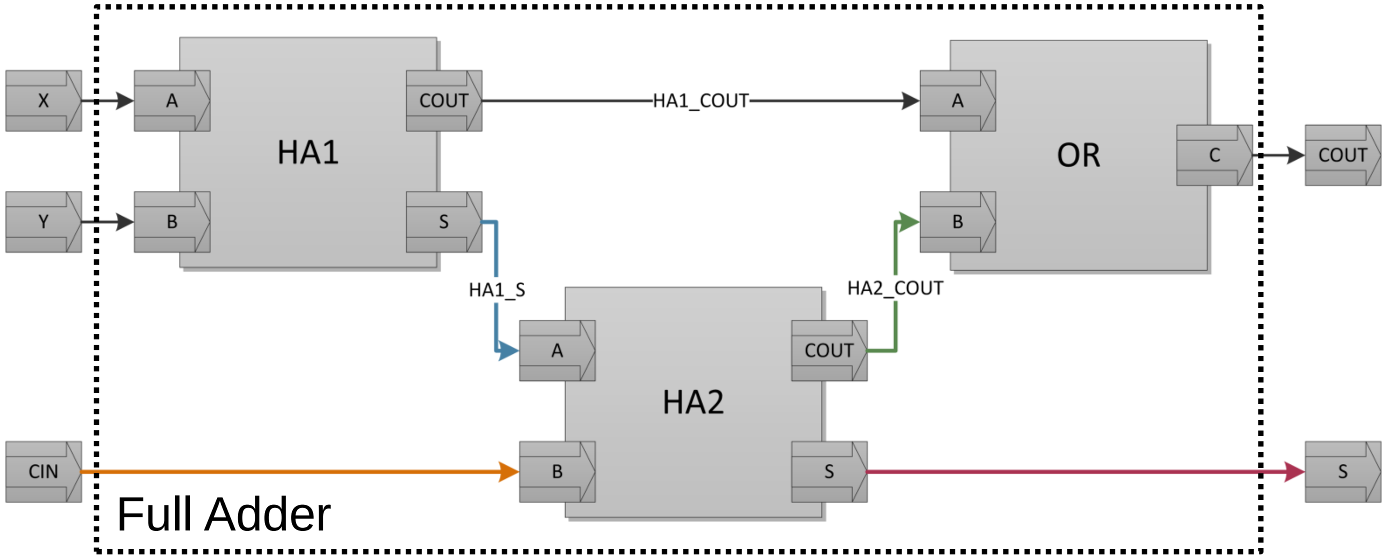
VHDL-Structural Description(2)



```
ha1 : half_adder
        Port Map(
                  a             => x,
                  b             => y,
                  s             => ha1_s,
                  cout          => ha1_cout
        );
```

Component instanstiation of half adder in VHDL

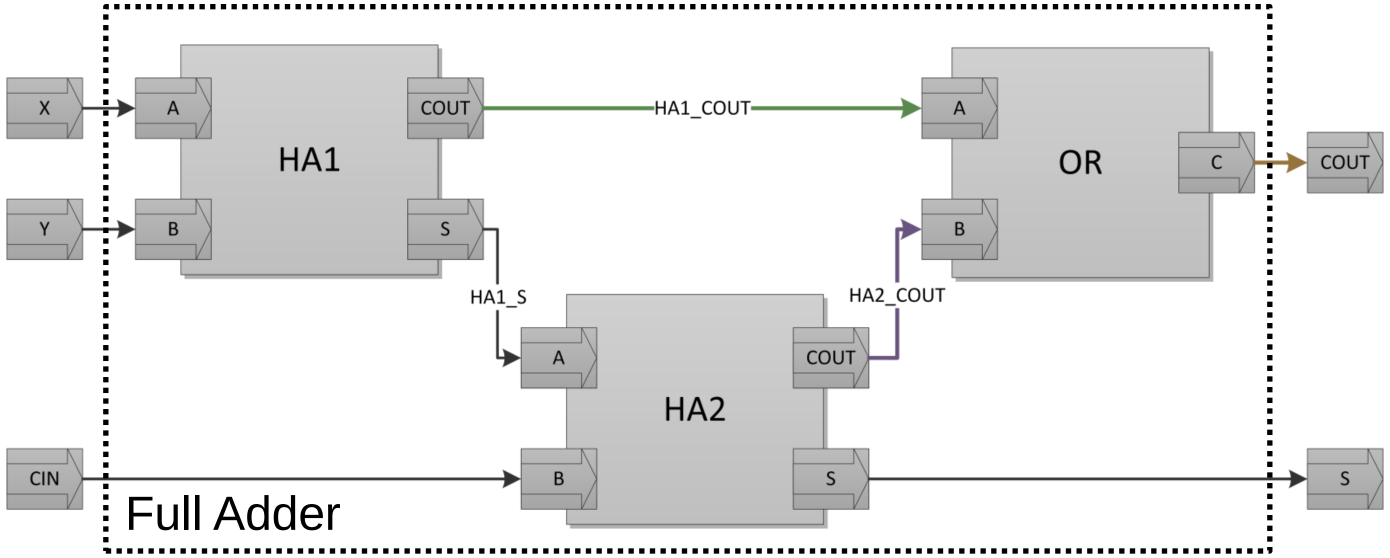
VHDL-Structural Description(3)



```vhdl
ha2 : half_adder
        Port Map(
                  a               => ha1_s,
                  b               => cin,
                  s               => s,
                  cout            => ha2_cout
        );
```

Component instanstiation of half adder in VHDL

VHDL-Structural Description(3)



```vhdl
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

```
s <= a or b; --implicit process syntax

process( a, b ) --explicit process syntax
begin

        if( a = '0' and b = '0') then
                cout <= '0';
        else
                cout <= '1';
        end if;

end process;
```

Behavioural description „or“ in VHDL

Half Adder Code Example

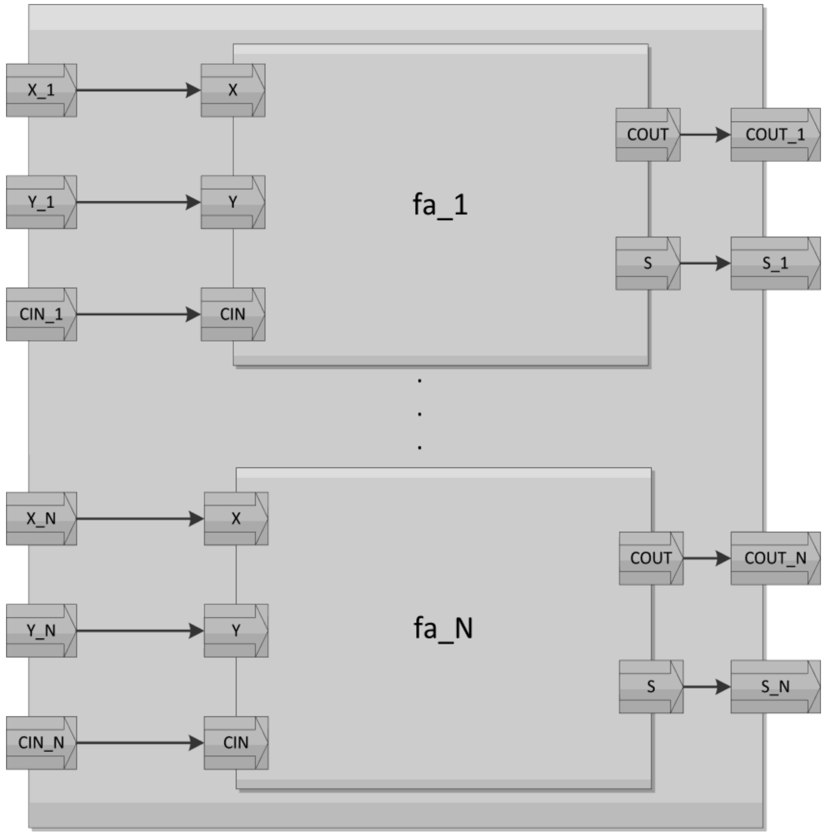
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity half_adder is
PORT (
                a               : in  std_logic;
                b               : in  std_logic;
                s               : out std_logic;
                cout   : out std_logic
);
end half_adder;

architecture Behavioral of half_adder is

begin

s               <= a and b; --implicit process syntax

process( a, b ) --explicit process syntax
begin

        if( a = not b) then
                cout <= '0';
        else
                cout <= '1';
        end if;

end process;

end Behavioral;
```

Full Adder Code Example

```vhdl

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity full_adder is
        Port(
                  x             : in  std_logic;
                  y             : in  std_logic;
                  cin           : in  std_logic;
                  s             : out std_logic;
                  cout   : out std_logic
        );
end full_adder;

architecture Behavioral of full_adder is

component half_adder
        Port(
                  a      : in  std_logic;
                  b      : in  std_logic;
                  s      : out std_logic;
                  cout : out std_logic
        );
end component;

signal ha1_cout : std_logic;
signal ha1_s     : std_logic;
signal ha2_cout : std_logic;

begin

ha1 : half_adder
        Port Map(
                  a => x,
                  b => y,
                  s => ha1_s,
                  cout => ha1_cout
        );

ha2 : half_adder
        Port Map(
                  a => ha1_s,
                  b => cin,
                  s => s,
                  cout => ha2_cout
        );

cout <= ha1_cout or ha2_cout;

end Behavioral;
```

VHDL-Parallelism

- Modules can be instantiated mutiple times and execute in parallel



Mulitple Full Adders executing in parallel

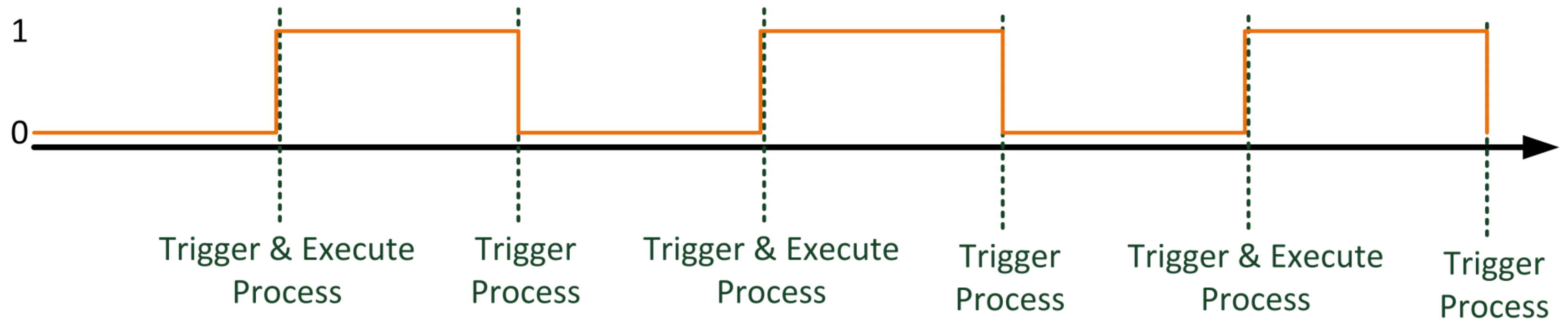
```
fa_1 : full_adder
        Port Map(
                  x => x_1,
                  y => y_1,
                  cin => cin_1,
                  cout => cout_1,
                  s => s_1
        );

..

fa_n : full_adder
        Port Map(
                  x => x_n,
                  y => y_n,
                  cin => cin_n,
                  cout => cout_n,
                  s => s_n
        );
```

Mulitple instantiations of a Full Adder

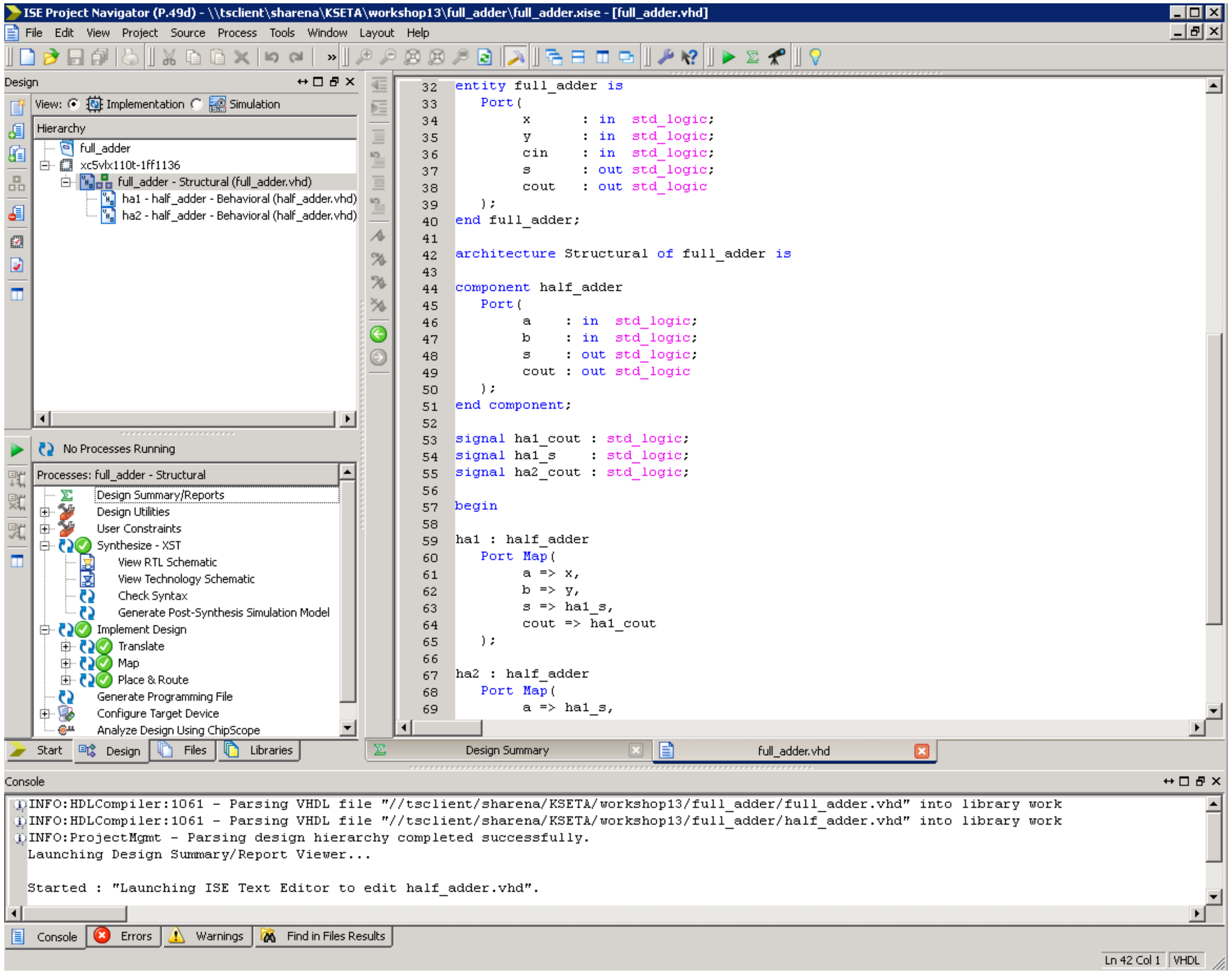
VHDL-Clocking

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

```
process(clk)
begin
        if( clk = '1' and clk'event) then
                result <= 1;
        end if;
end process;
```


1
0
Trigger & Execute Process
Trigger Process
Trigger & Execute Process
Trigger Process
Trigger & Execute Process
Trigger Process

XILINX Design Tool ISE


ISE Project Navigator (P.49d) - \\tsclient\sharena\KSETA\workshop13\full_adder\full_adder.xise - [full_adder.vhd]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
No Processes Running
Processes: full_adder - Structural
Start Design Files Libraries
Design Summary full_adder.vhd
Console
Console Errors Warnings Find in Files Results
Ln 42 Col 1 VHDL

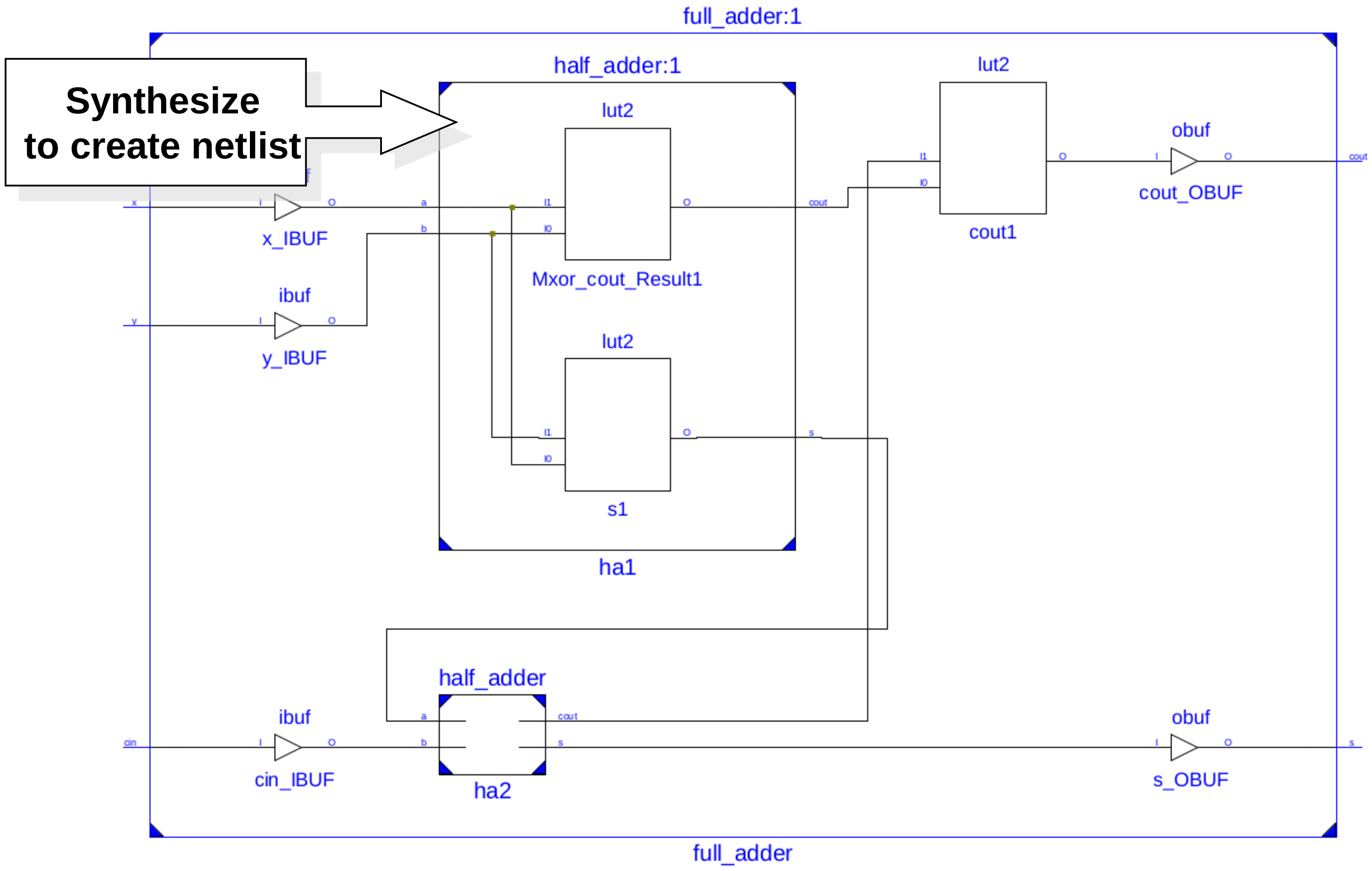
```
full_adder
xc5vlx110t-1ff1136
  full_adder - Structural (full_adder.vhd)
    ha1 - half_adder - Behavioral (half_adder.vhd)
    ha2 - half_adder - Behavioral (half_adder.vhd)
```

```
Design Summary/Reports
Design Utilities
User Constraints
Synthesize - XST
    View RTL Schematic
    View Technology Schematic
    Check Syntax
    Generate Post-Synthesis Simulation Model
Implement Design
    Translate
    Map
    Place & Route
Generate Programming File
Configure Target Device
Analyze Design Using ChipScope
```

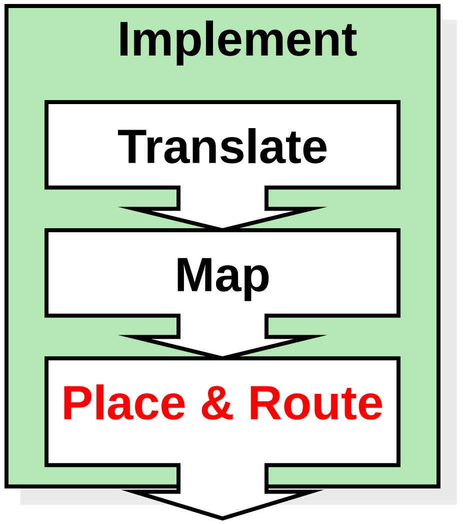
```vhdl
entity full_adder is
    Port(
            x      : in  std_logic;
            y      : in  std_logic;
            cin    : in  std_logic;
            s      : out std_logic;
            cout   : out std_logic
    );
end full_adder;

architecture Structural of full_adder is

component half_adder
    Port(
            a     : in  std_logic;
            b     : in  std_logic;
            s     : out std_logic;
            cout : out std_logic
    );
end component;

signal ha1_cout : std_logic;
signal ha1_s    : std_logic;
signal ha2_cout : std_logic;

begin

ha1 : half_adder
    Port Map(
            a => x,
            b => y,
            s => ha1_s,
            cout => ha1_cout
    );

ha2 : half_adder
    Port Map(
            a => ha1_s,
```

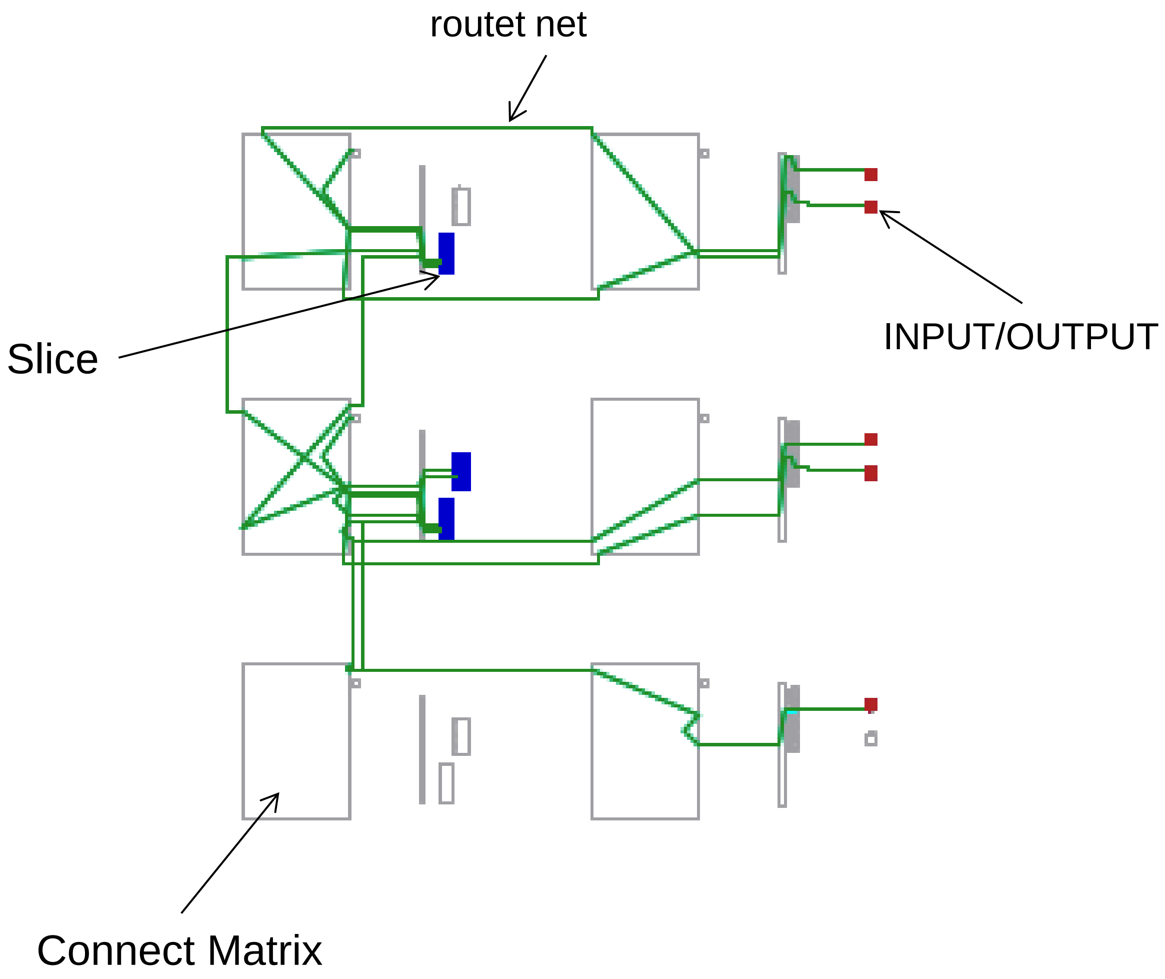
```
INFO:HDLCompiler:1061 - Parsing VHDL file "//tsclient/sharena/KSETA/workshop13/full_adder/full_adder.vhd" into library work
INFO:HDLCompiler:1061 - Parsing VHDL file "//tsclient/sharena/KSETA/workshop13/full_adder/half_adder.vhd" into library work
INFO:ProjectMgmt - Parsing design hierarchy completed successfully.
Launching Design Summary/Report Viewer...

Started : "Launching ISE Text Editor to edit half_adder.vhd".
```

Demo Synthesize

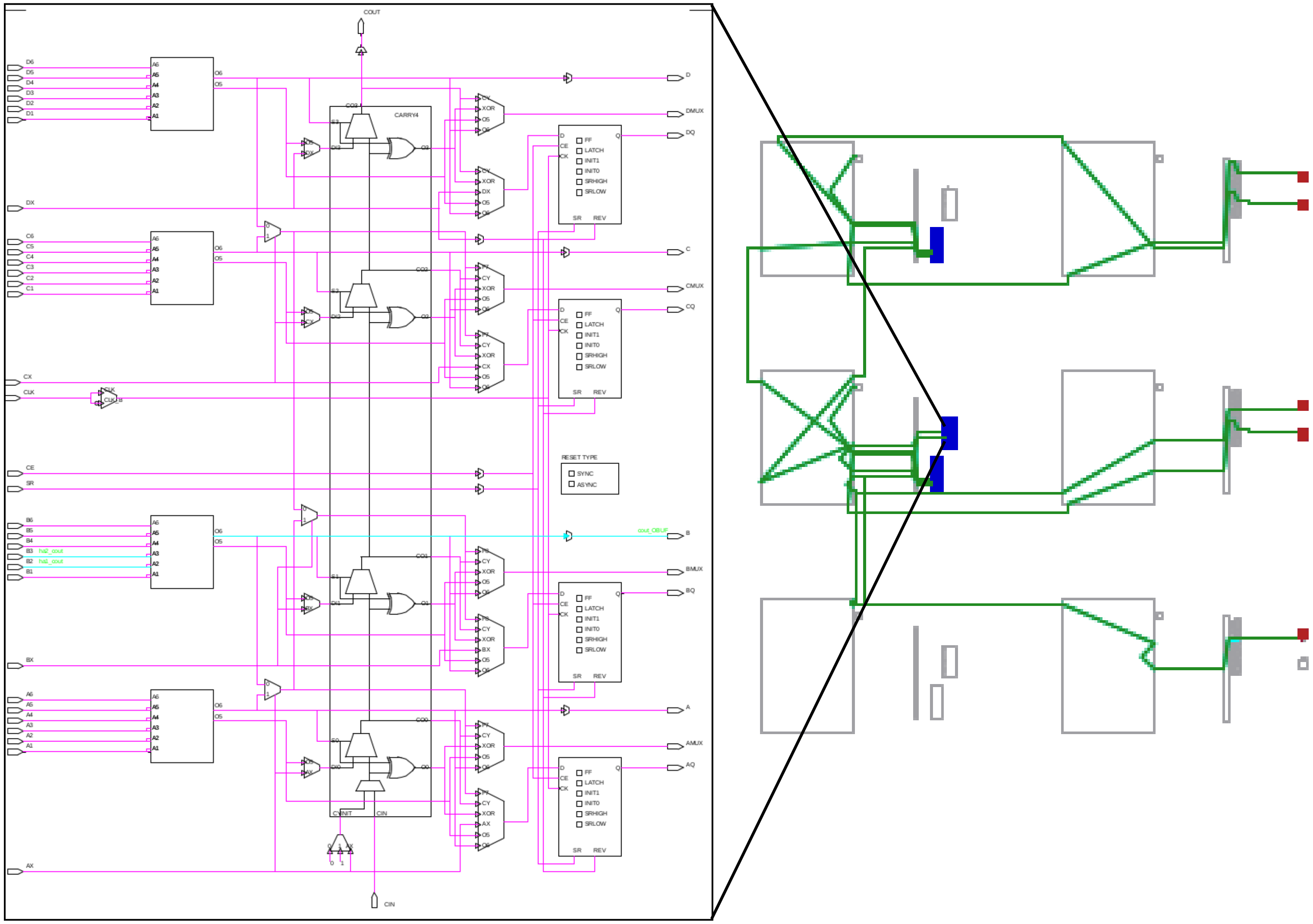


Demo P&R


Implement
Translate
Map
Place & Route


routet net
Slice
INPUT/OUTPUT
Connect Matrix

Demo P&R



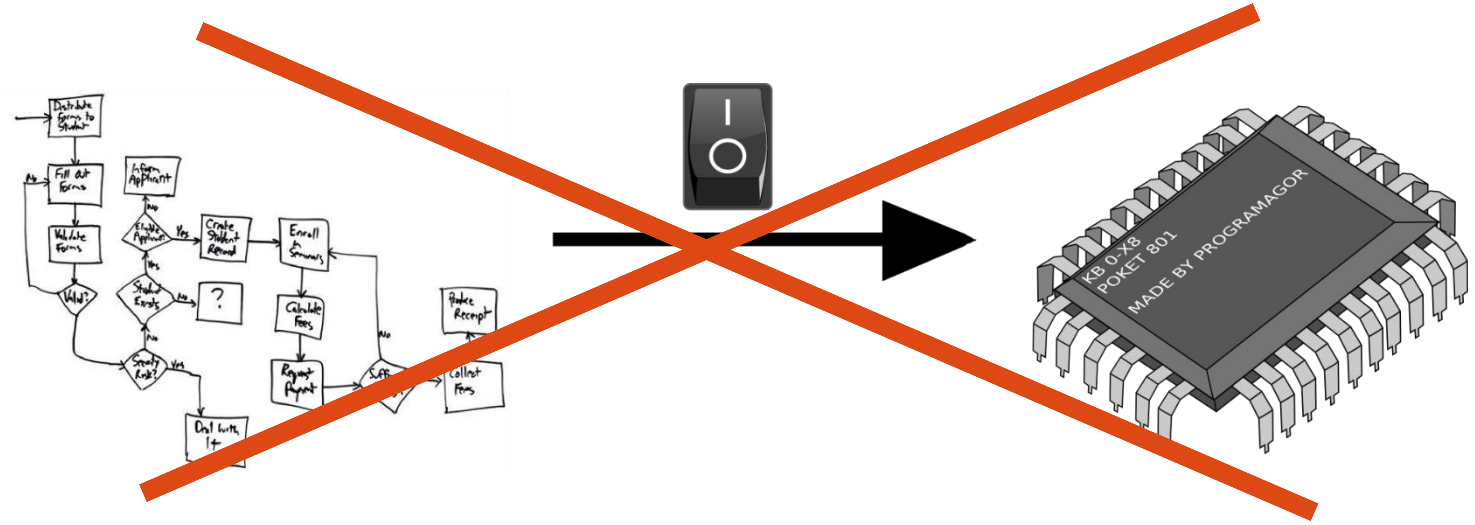
High-Level-Design Approaches

- 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



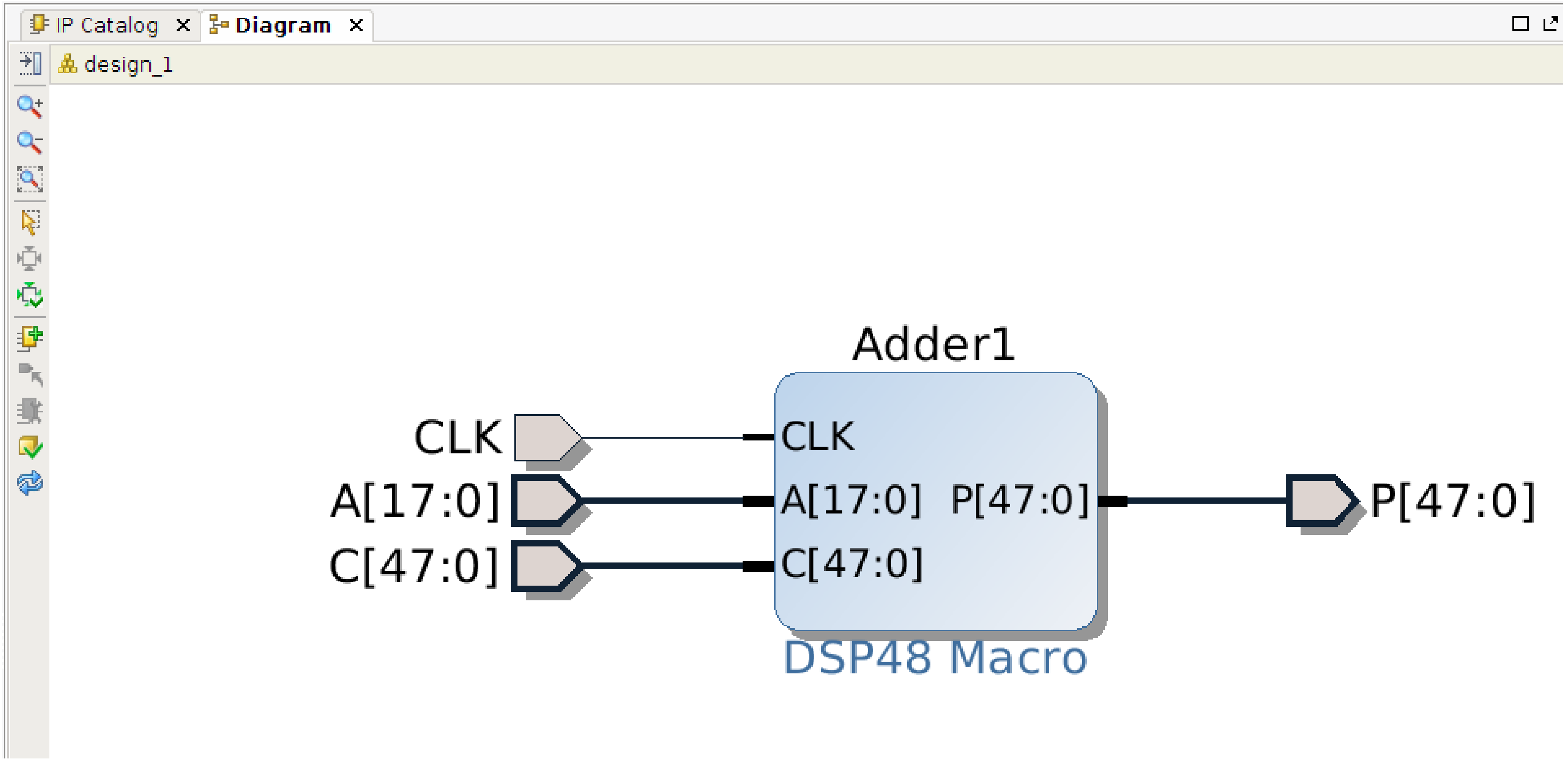
- There is no “Push-Button” approach
- Unlike for software compilation
- Designer must keep hardware in mind

System Designer

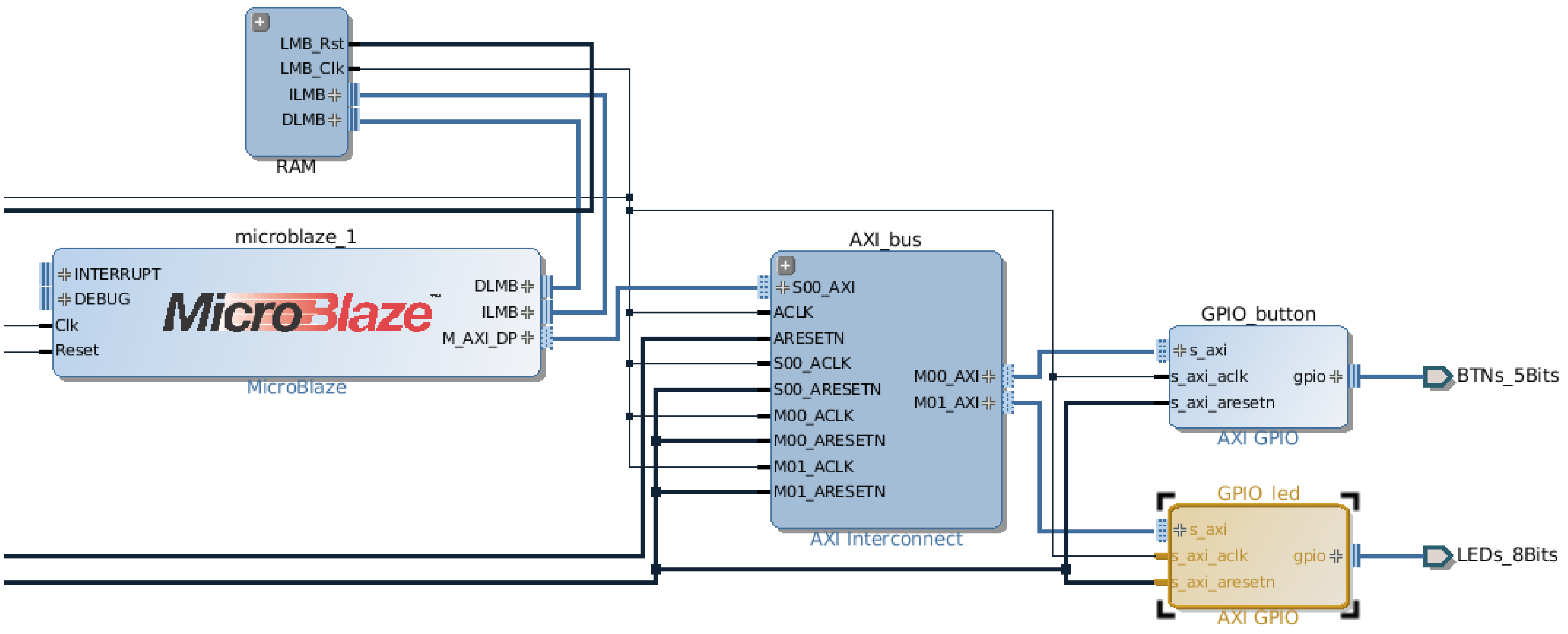
- Design of (Embedded) Systems based on building blocks
- IP-blocks connected by a standard bus
- Available blocks:
  - Arithmetics: Adder, Mulitplier, sqrt, sin/cos, ...
  - Digital Signal Processing: FIR filters, FFT, ...
  - Processors, RAMs, ROMs, Peripherial controller, ...
- Limitations
  - Limited 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

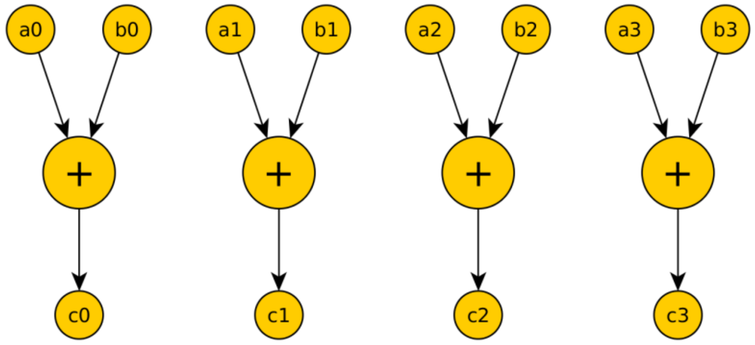
Design Space Exploration (Example)

- Vector Addition:

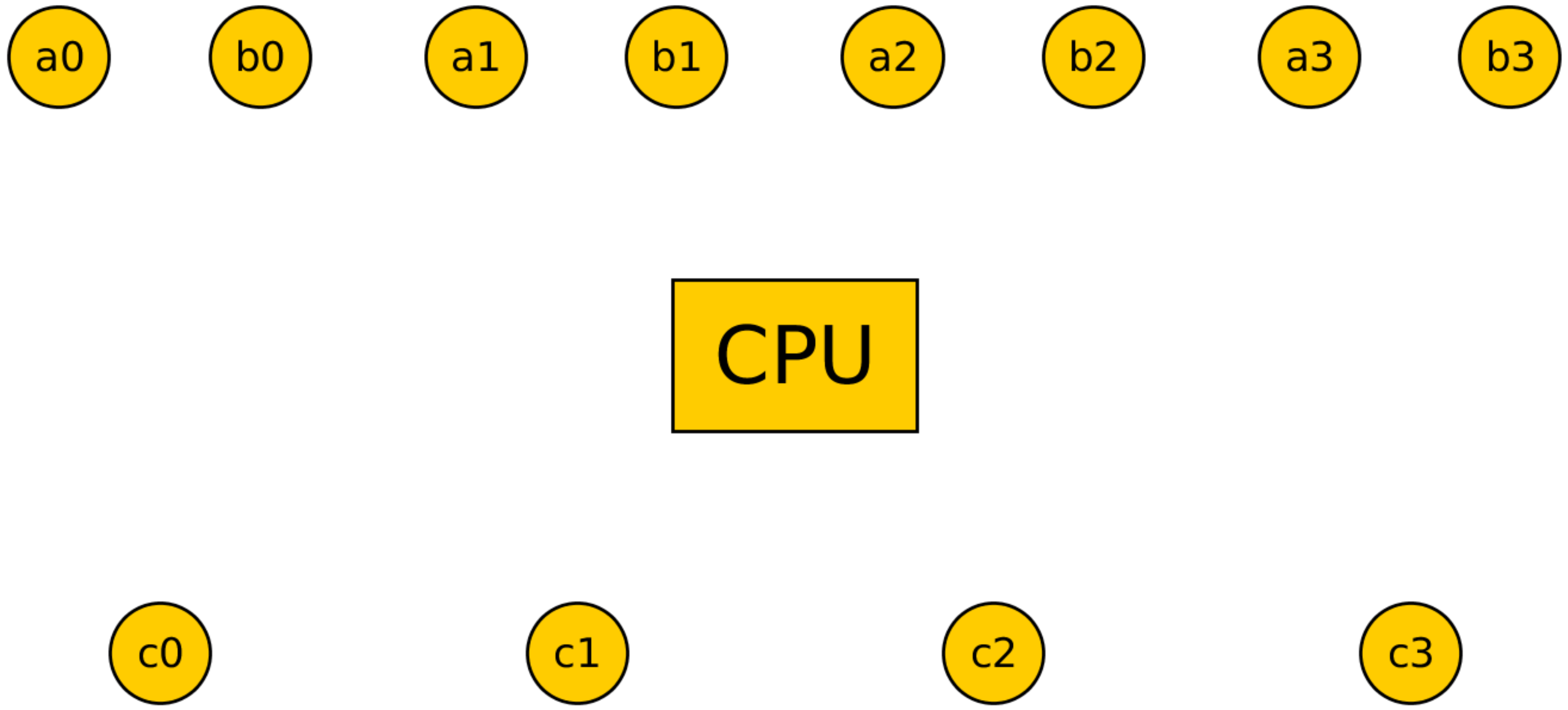
$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} + \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

```
void addVectors(int a[], int b[], int c[], int n) {

    for (int i=0; i<n; i++) {
       c[i] = a[i] + b[i];
    }

}
```

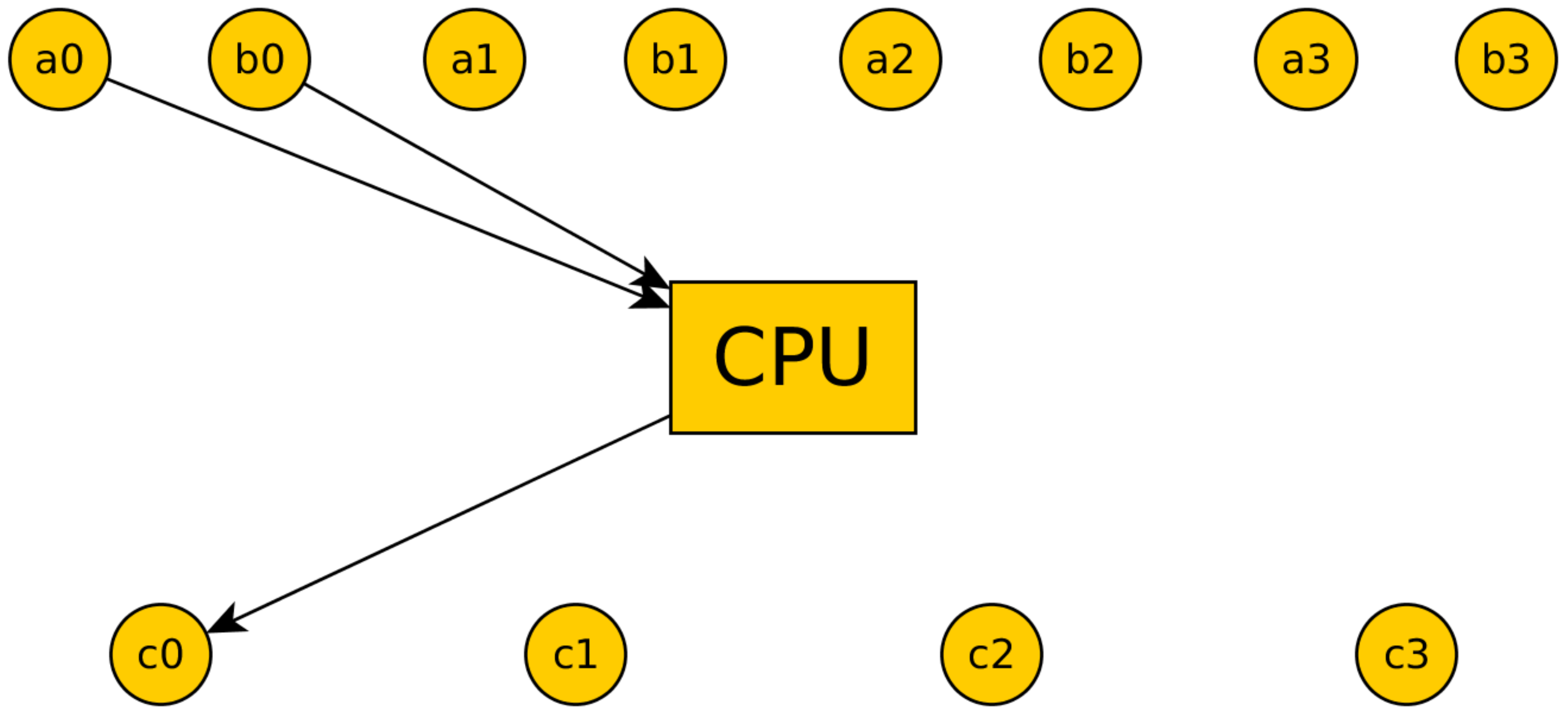
n=4:



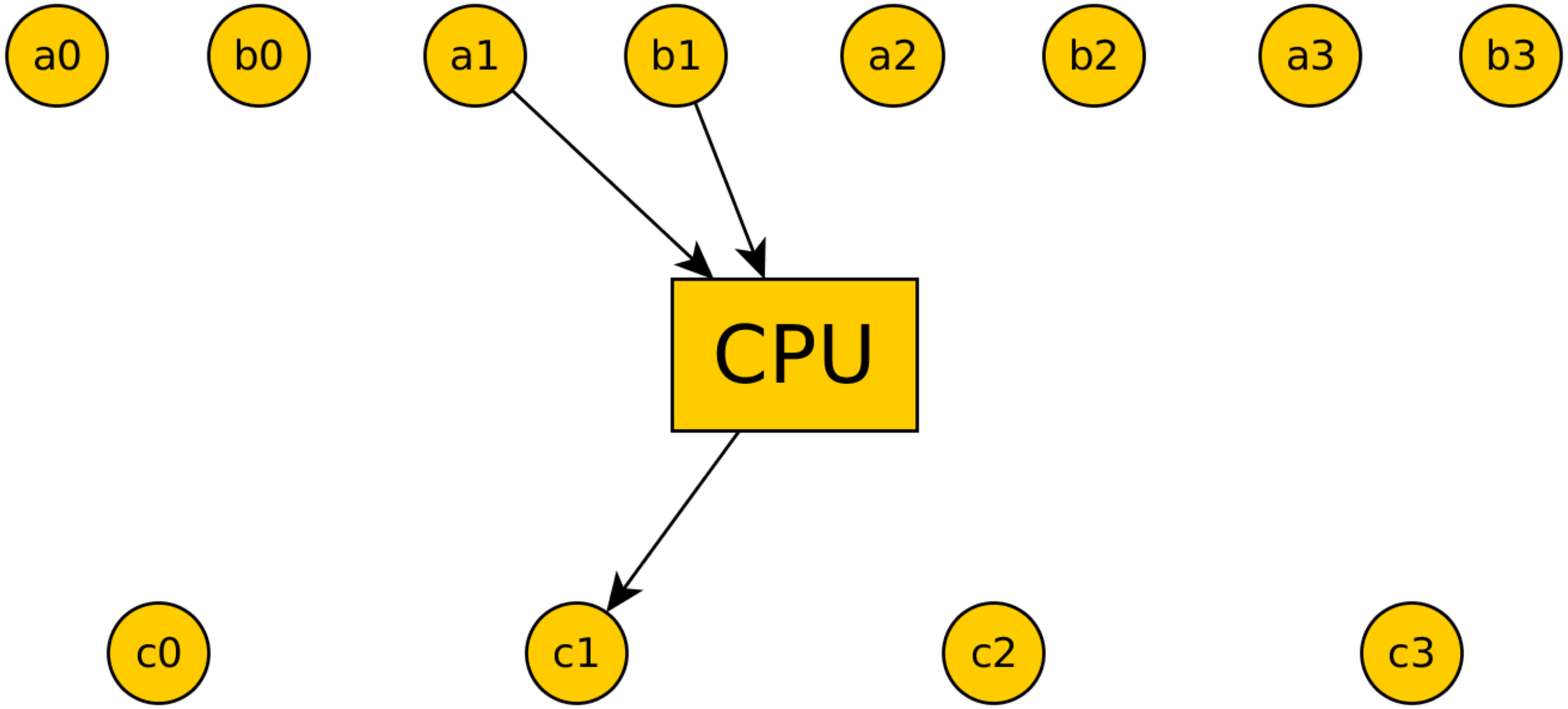
Design Space Exploration – CPU / 1 Adder



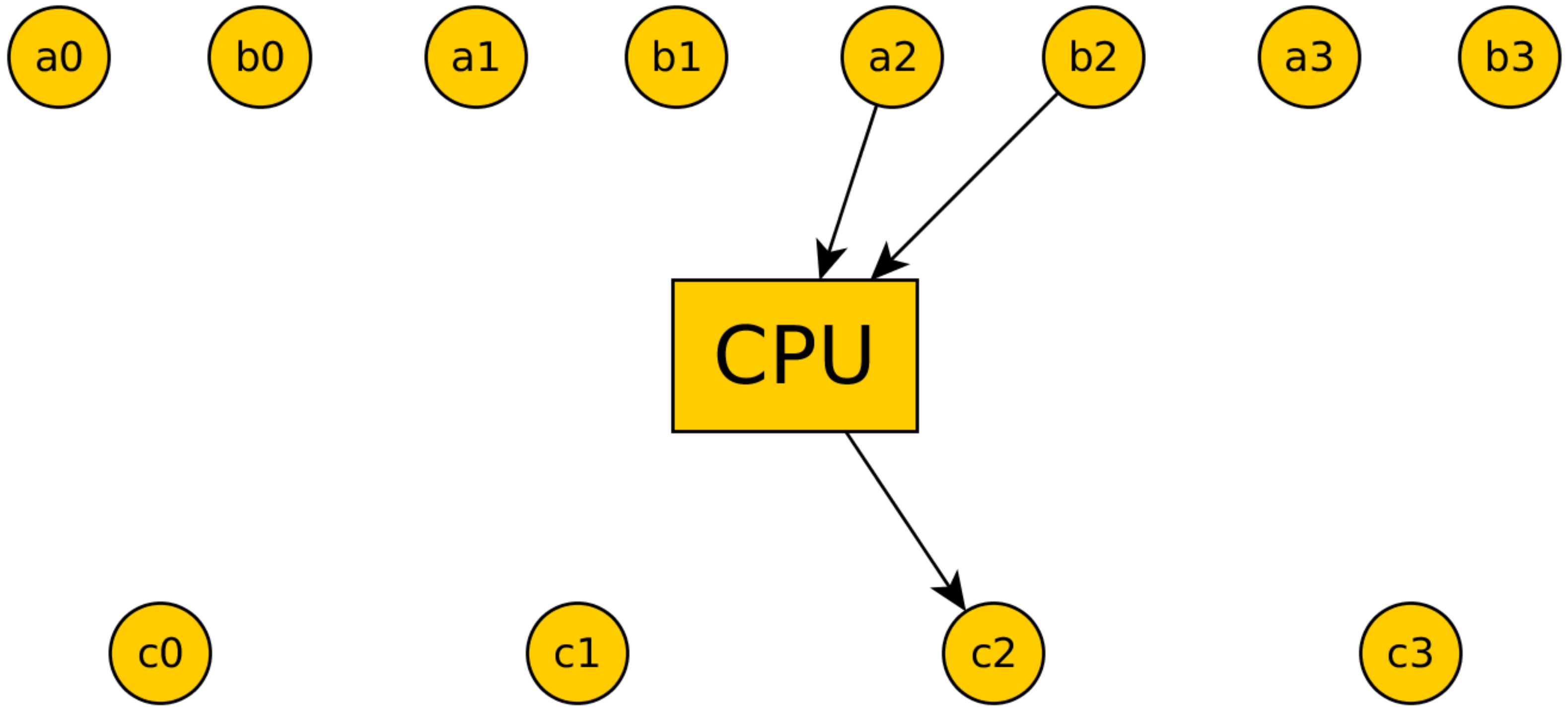
Design Space Exploration – CPU / 1 Adder



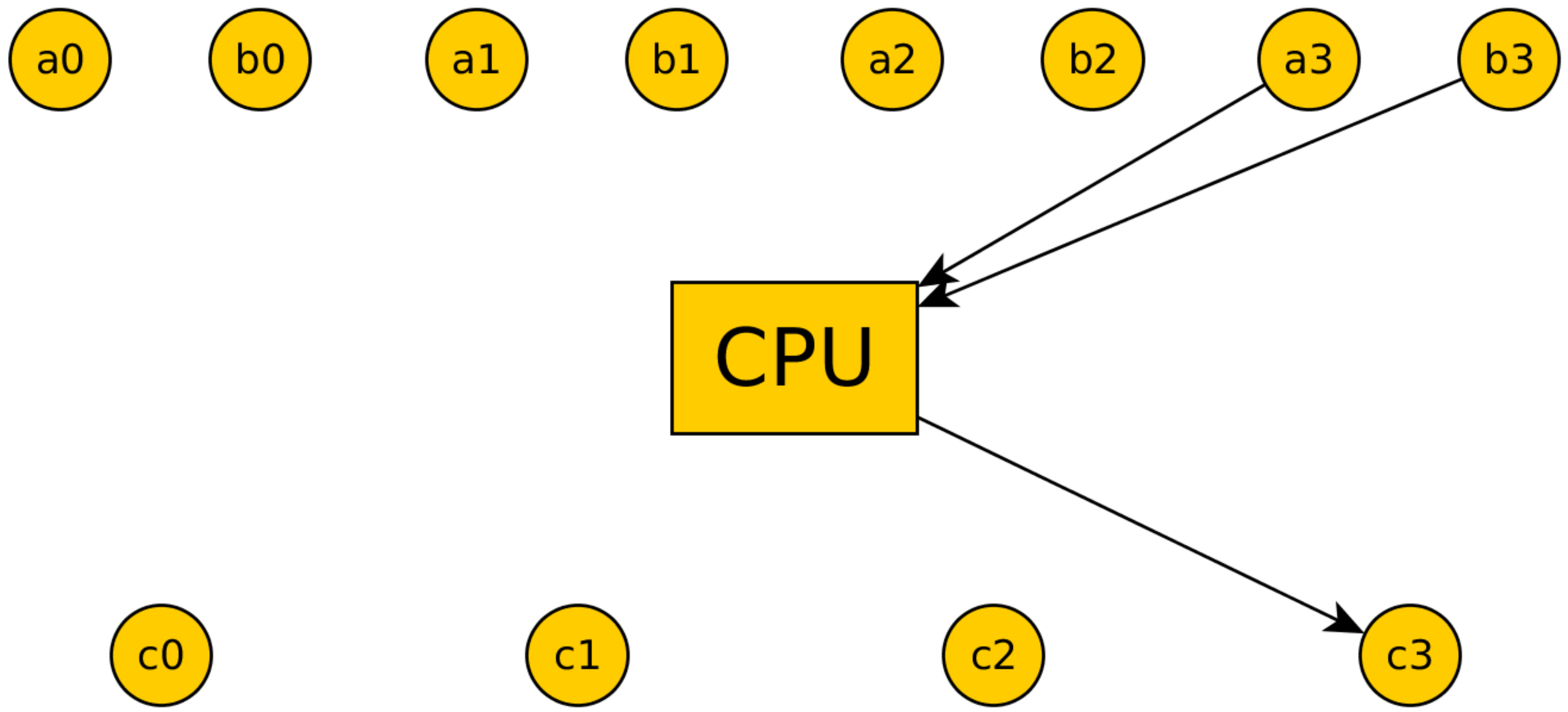
Design Space Exploration – CPU / 1 Adder



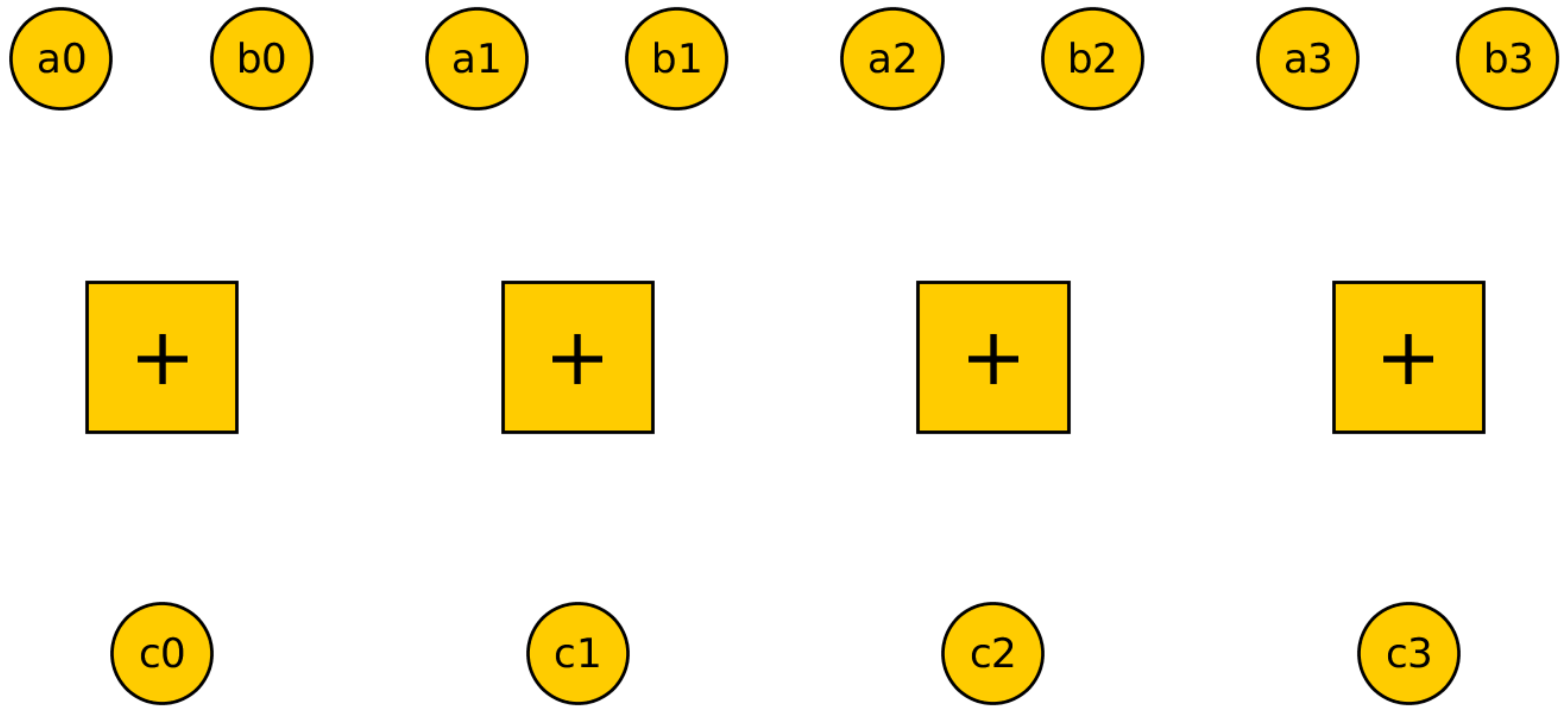
Design Space Exploration – CPU / 1 Adder



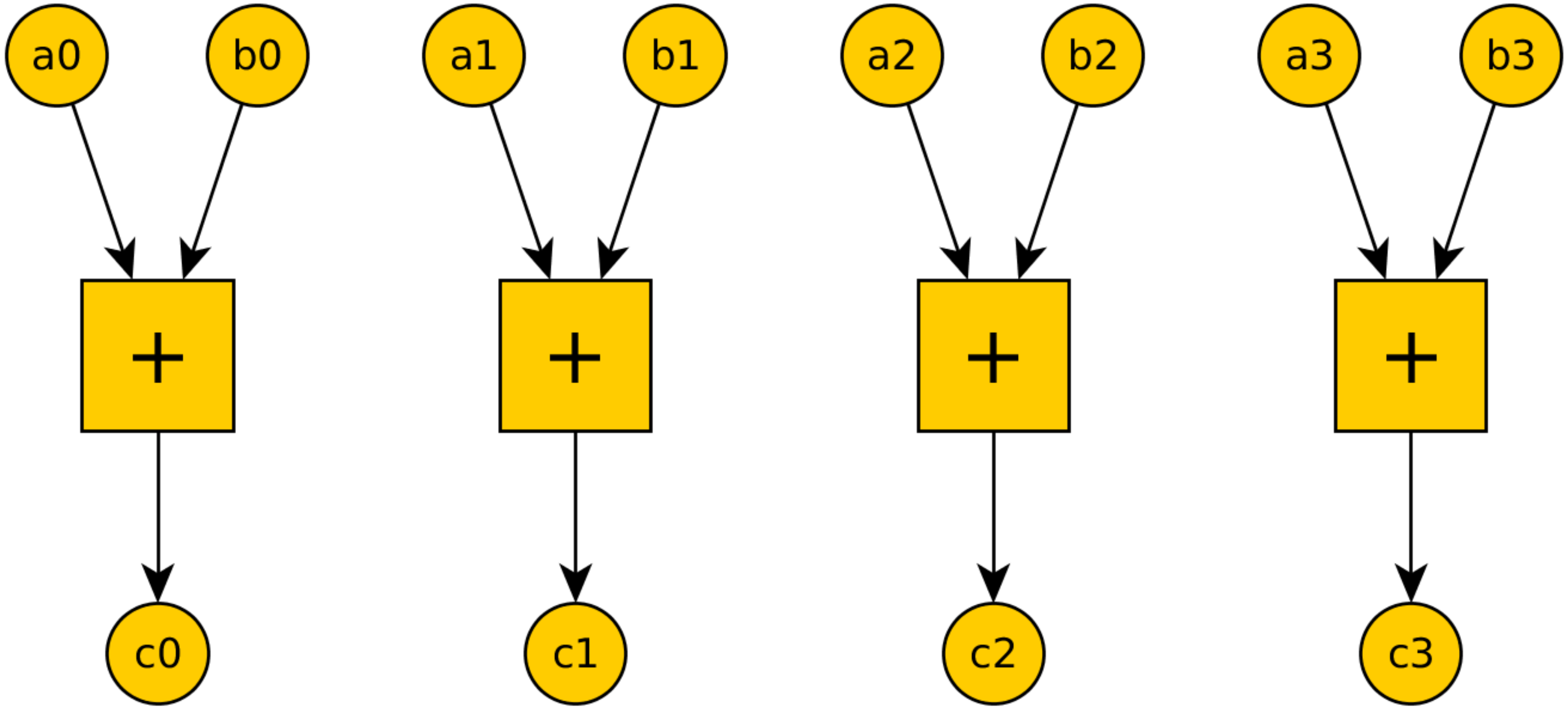
Design Space Exploration – CPU / 1 Adder



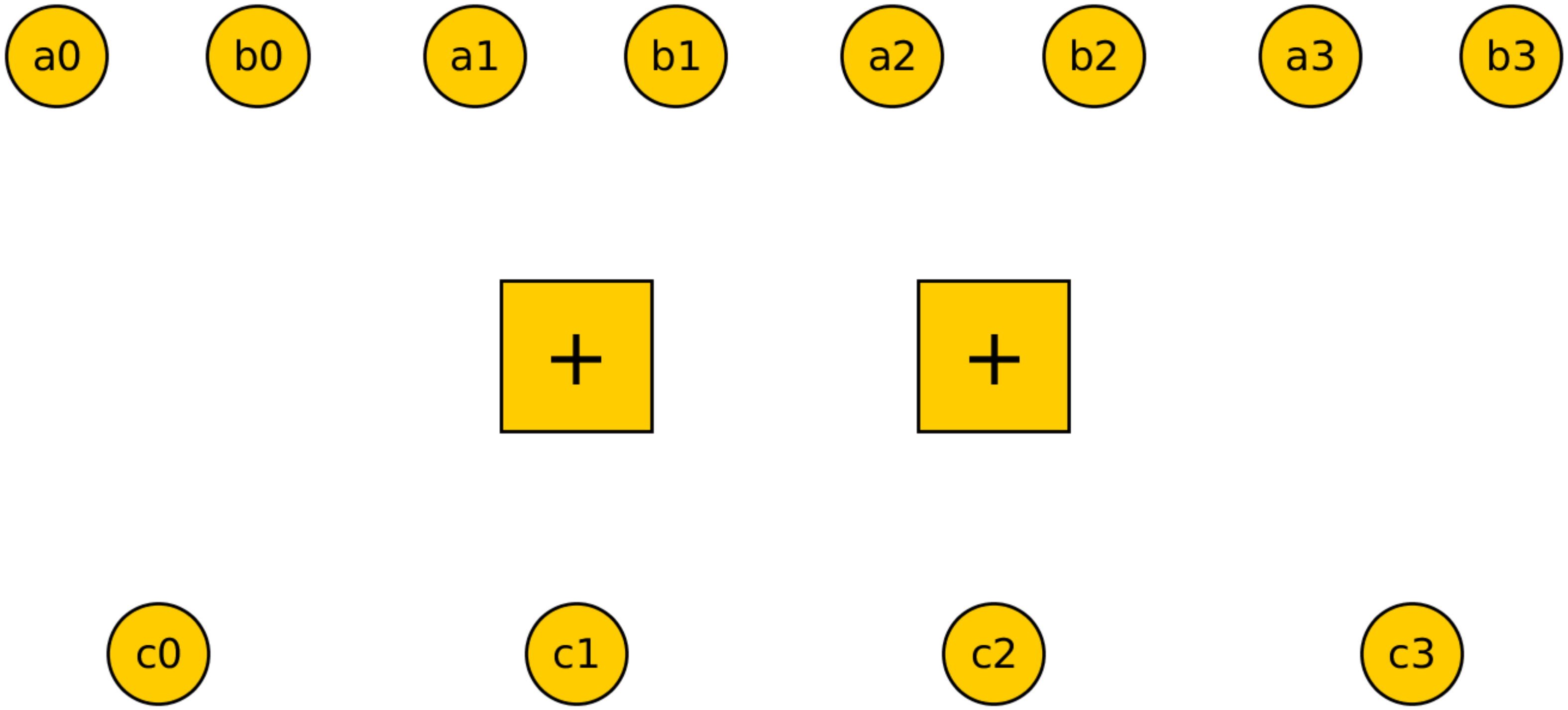
Design Space Exploration – 4 Adder



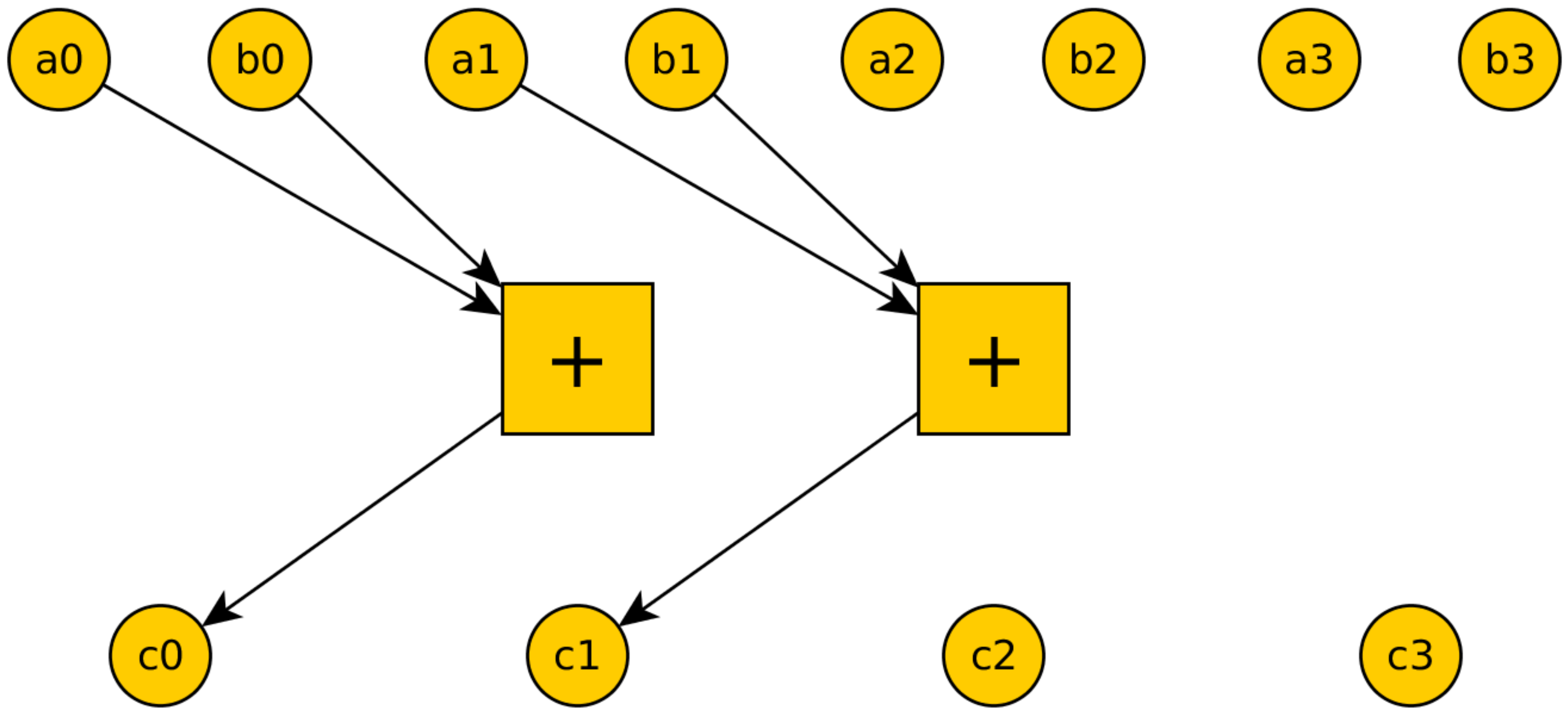
Design Space Exploration – 4 Adder



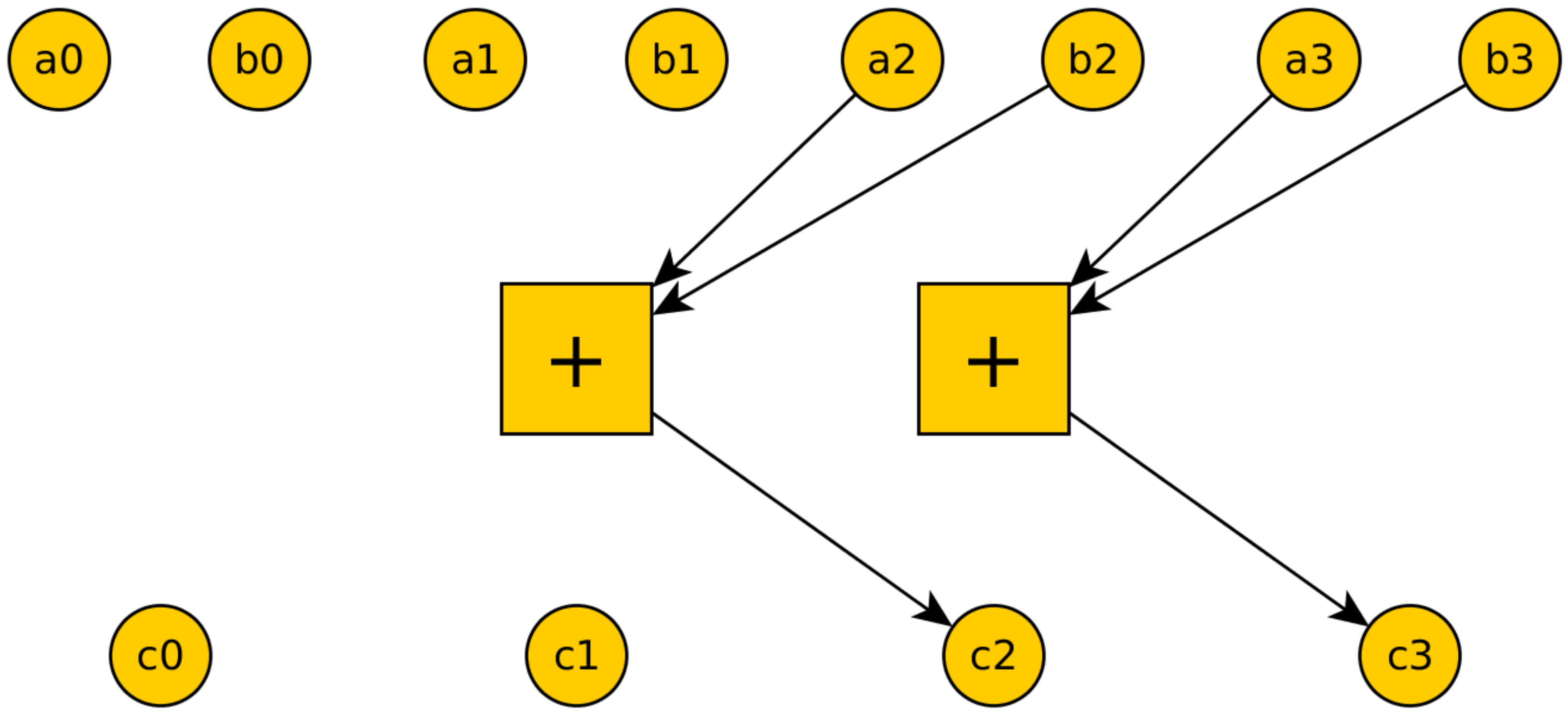
Design Space Exploration – 2 Adder



Design Space Exploration – 2 Adder



Design Space Exploration – 2 Adder



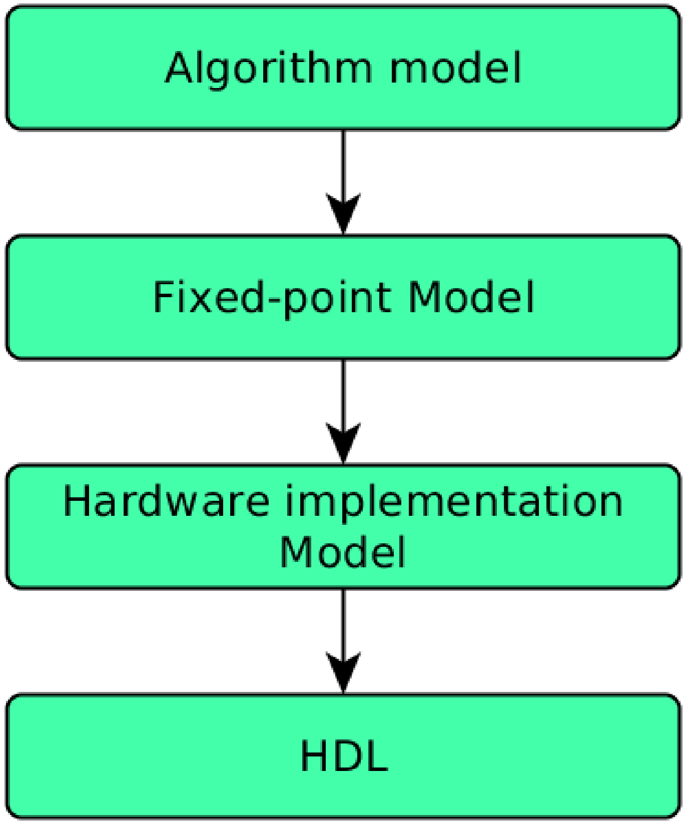
Design Space Exploration – Comparison

	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)

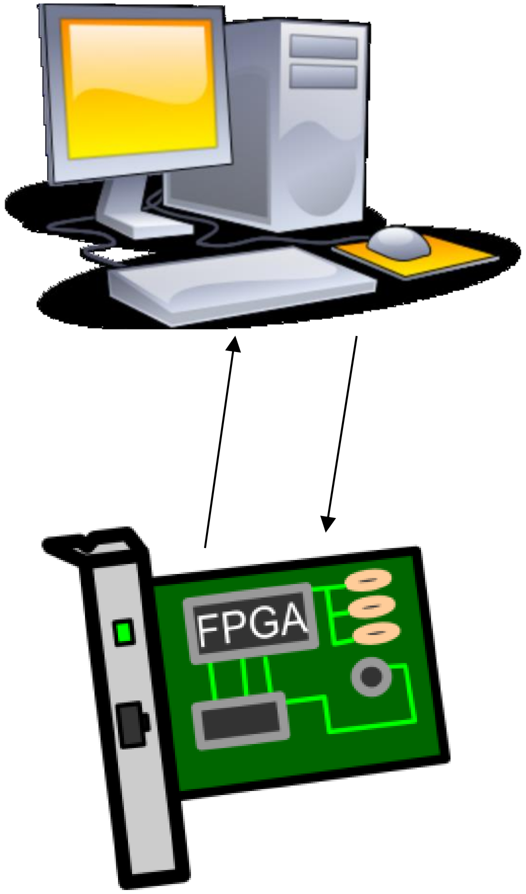
- Extension to standard C
- Description of massively parallel algorithms
- Kernels describe parallel parts of algorithms
- Kernel could execute on different computation units
  - CPU
  - GPGPU
  - FPGA
- Limitations:
  - No stand alone FPGAs – Kernels called by host program
  - (so far) only 1 FPGA board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

- Application: Altera SDK for OpenCL

OpenCL – (very simplified) Example

```
int main() {
  a = readMatrixFromFile();
  b = readMatrixFromFile();

  ConfigureAndCompileKernel();
  CopyDataToDevice();

  RunKernel(addVectors,a,b,c,vector_length)

  CopyDataToHost();
  PrintMatrix(c);
}
```

```
__kernel void addVectors(a, b, c, n) {

  i = get_global_id(0);

  c[i] = a[i] + b[i];
}
```

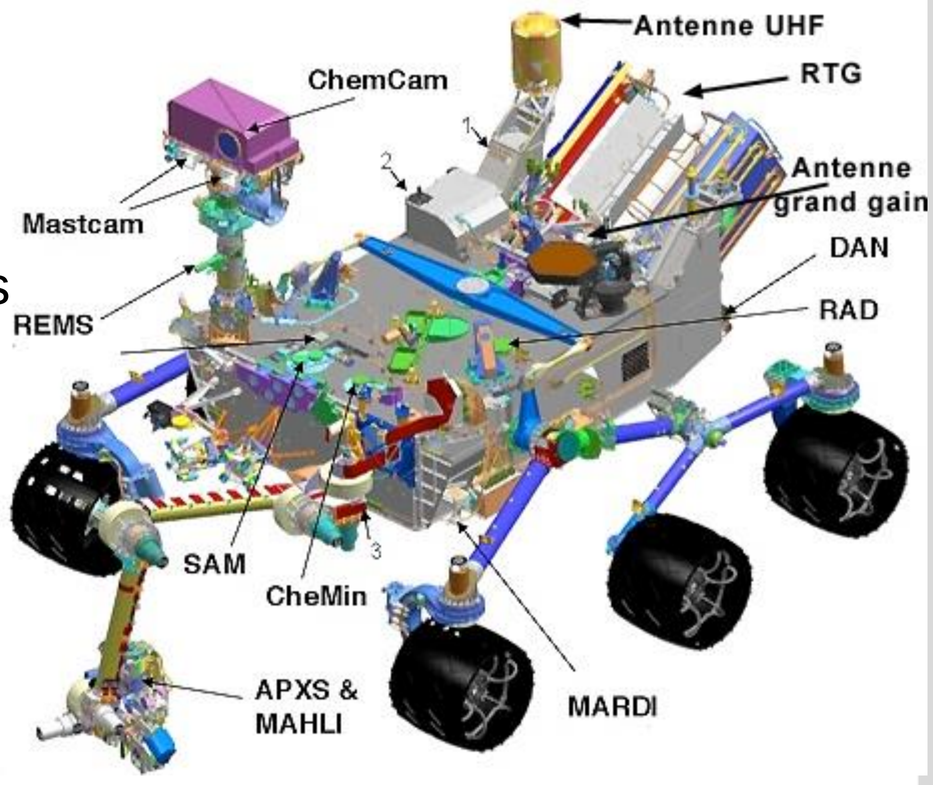


FPGA - Applications

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

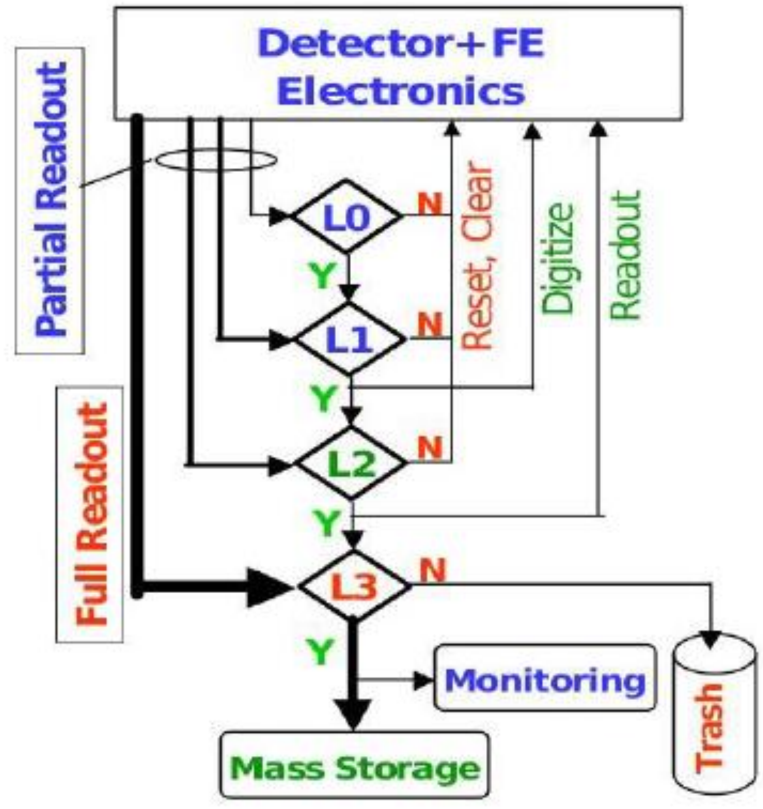
Mars exploration rovers (MER)

- 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

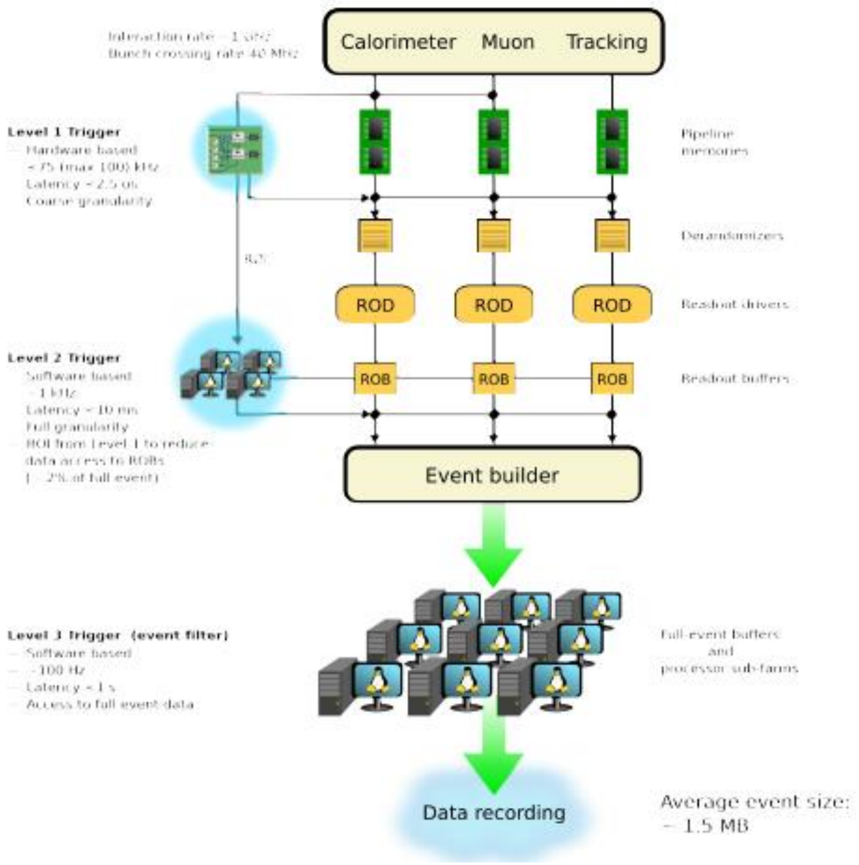


Curiosity Rover

Trigger and DAQ systems in particle physics



FPGA-based trigger for NA62 Experiment



ATLAS Trigger system
L1 is hardware based

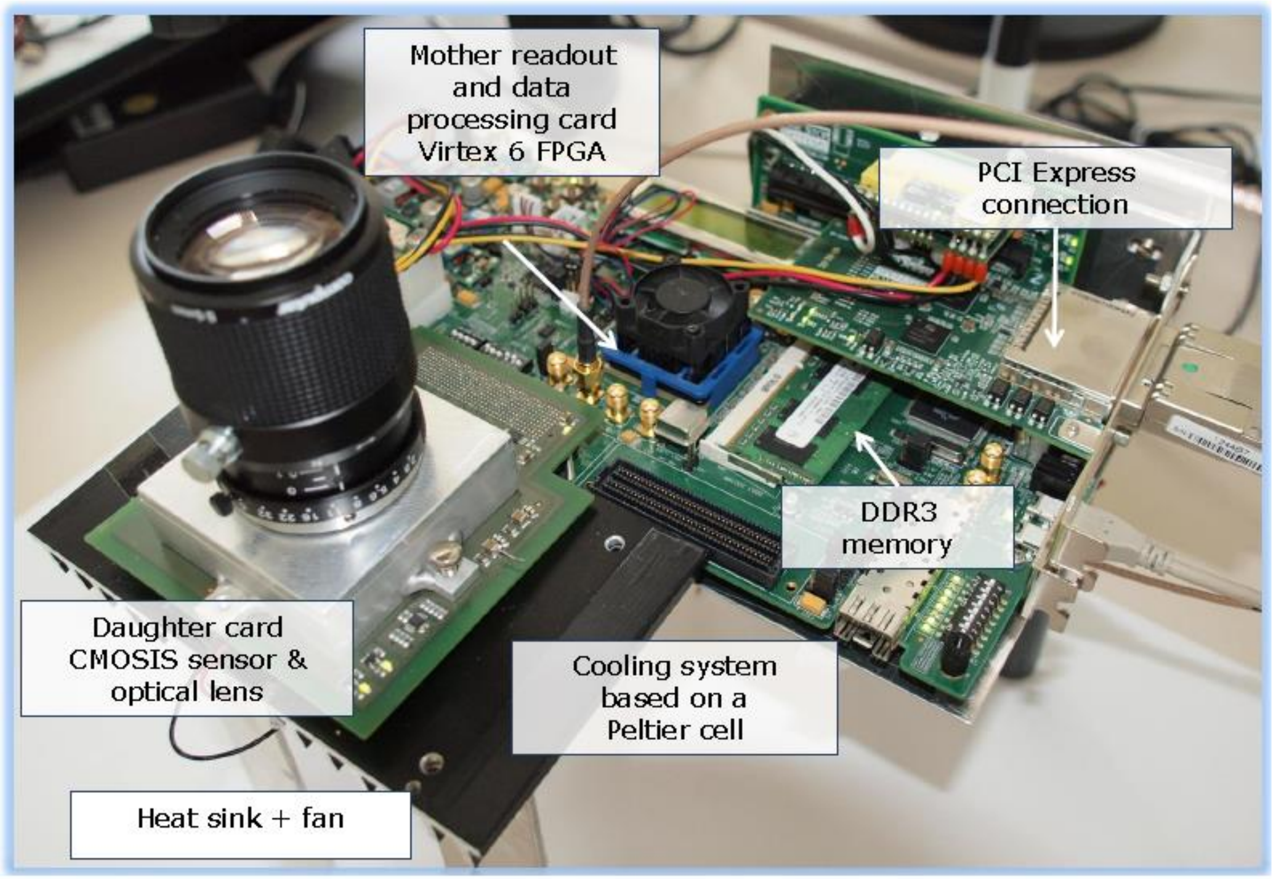
Banking applications

- 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

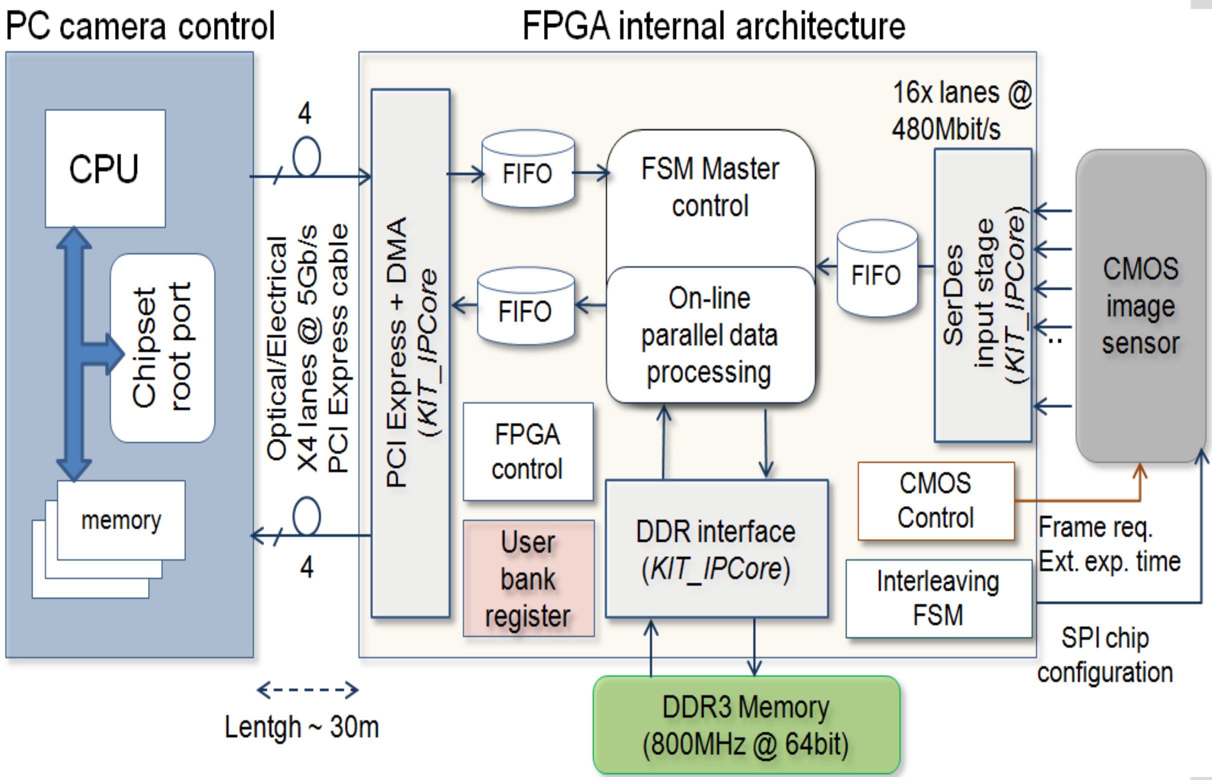
- Ultra Fast X-ray imaging of scientific processes with On-line assessment and data-driven process control

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


Mother readout and data processing card Virtex 6 FPGA
PCI Express connection
DDR3 memory
Daughter card CMOSIS sensor & optical lens
Cooling system based on a Peltier cell
Heat sink + fan

Readout chain and FPGA architecture

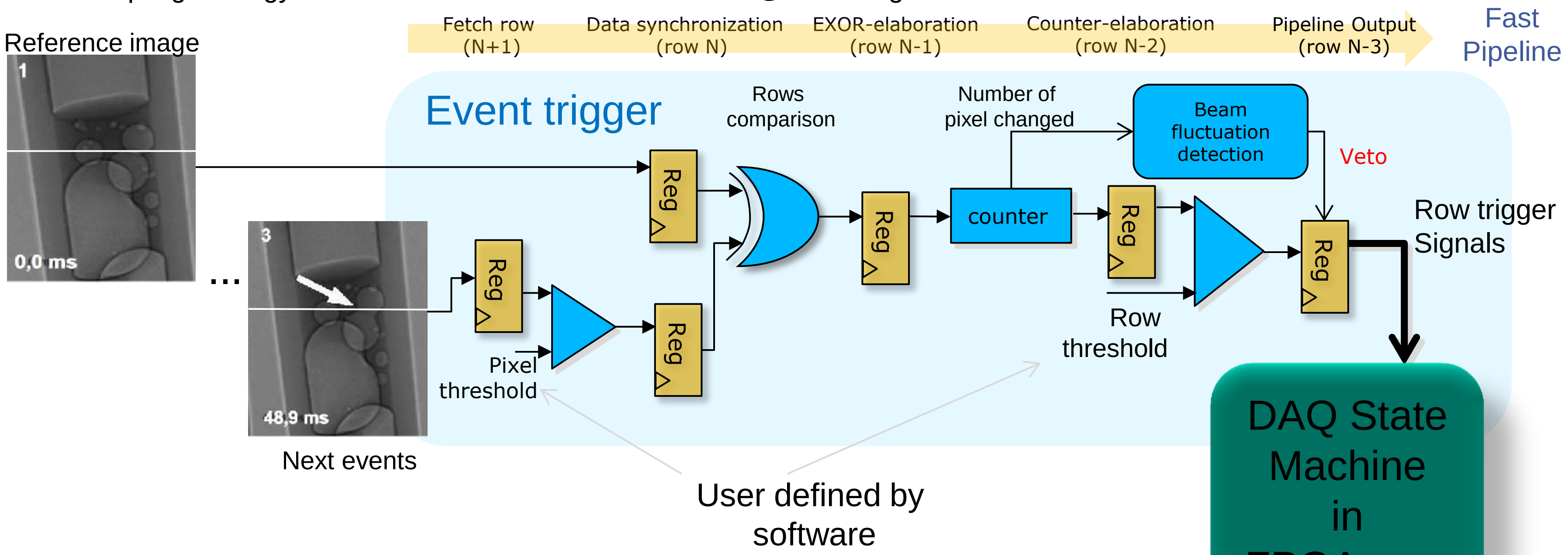
- 2.2 MP CMOS sensor
- 340 fps @ full view
- 14 Gb/s streaming with Bus Master DMA
- Virtex-6 Xilinx FPGA


PC camera control
FPGA internal architecture
CPU
Chipset root port
memory
4
4
Optical/Electrical X4 lanes @ 5Gb/s PCI Express cable
Lentgh ~ 30m
PCI Express + DMA (KIT_IPCore)
FIFO
FIFO
FSM Master control
On-line parallel data processing
FIFO
16x lanes @ 480Mbit/s
SerDes input stage (KIT_IPCore)
CMOS image sensor
FPGA control
User bank register
DDR interface (KIT_IPCore)
CMOS Control
Interleaving FSM
Frame req. Ext. exp. time
SPI chip configuration
DDR3 Memory (800MHz @ 64bit)

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



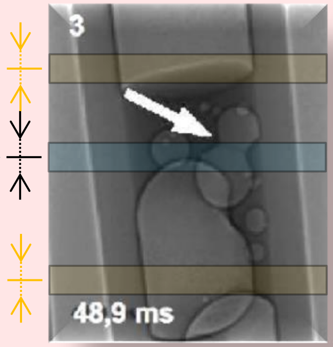
DAQ State Machine in FPGA core

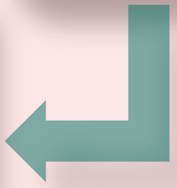
Event Readout

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

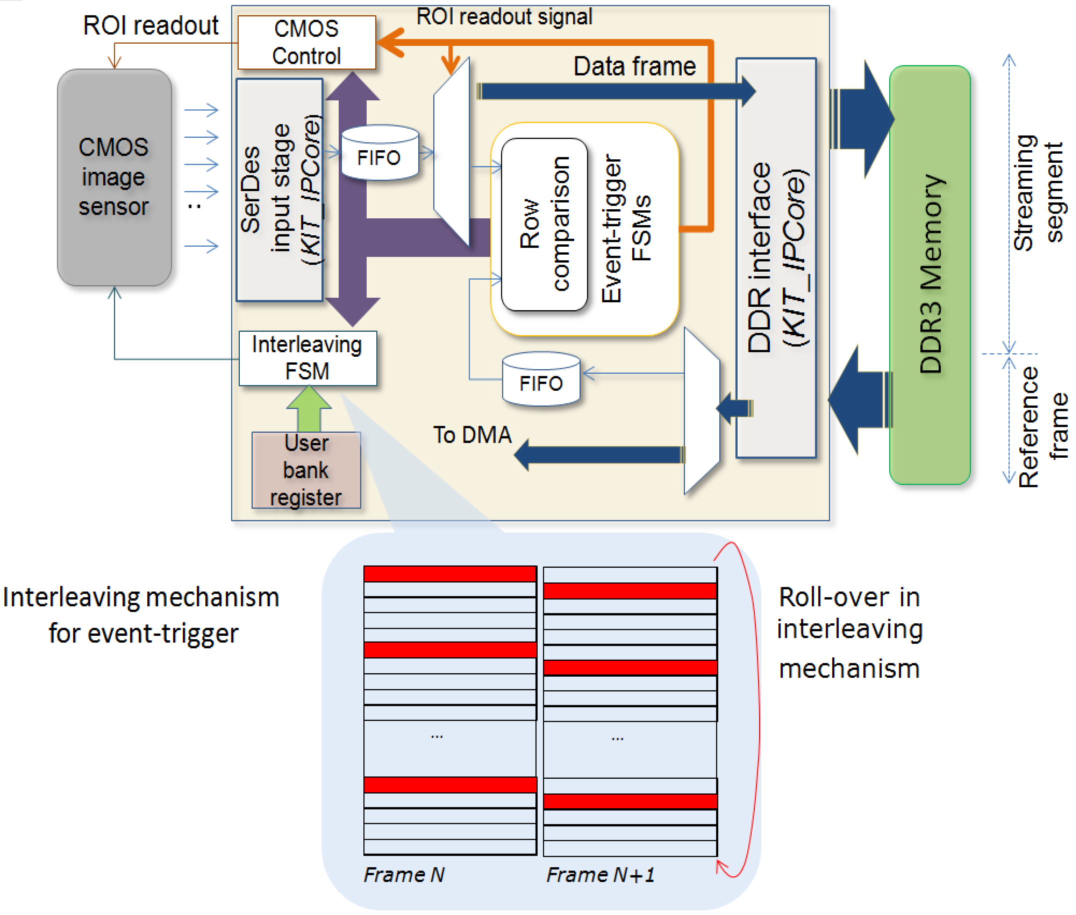
Optionally the multiple windows can be defined when a multiple rows trigger are presents



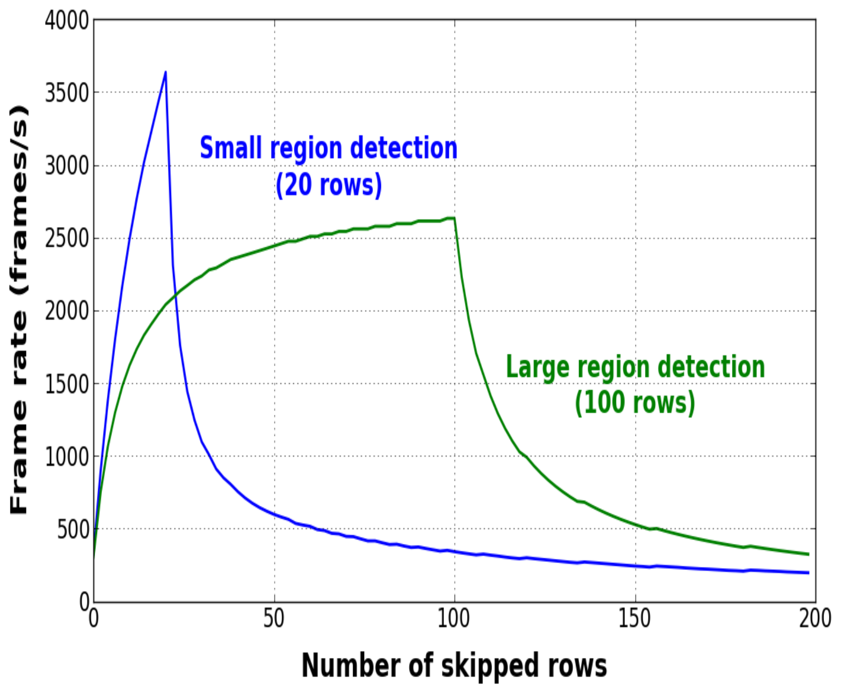


SPI chip configuration signals

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

Q & A