Budapest University of Technology and Economics

Department of Electron Devices

Technology of IT Devices

Lecture 5

Digital system design

Digital system design

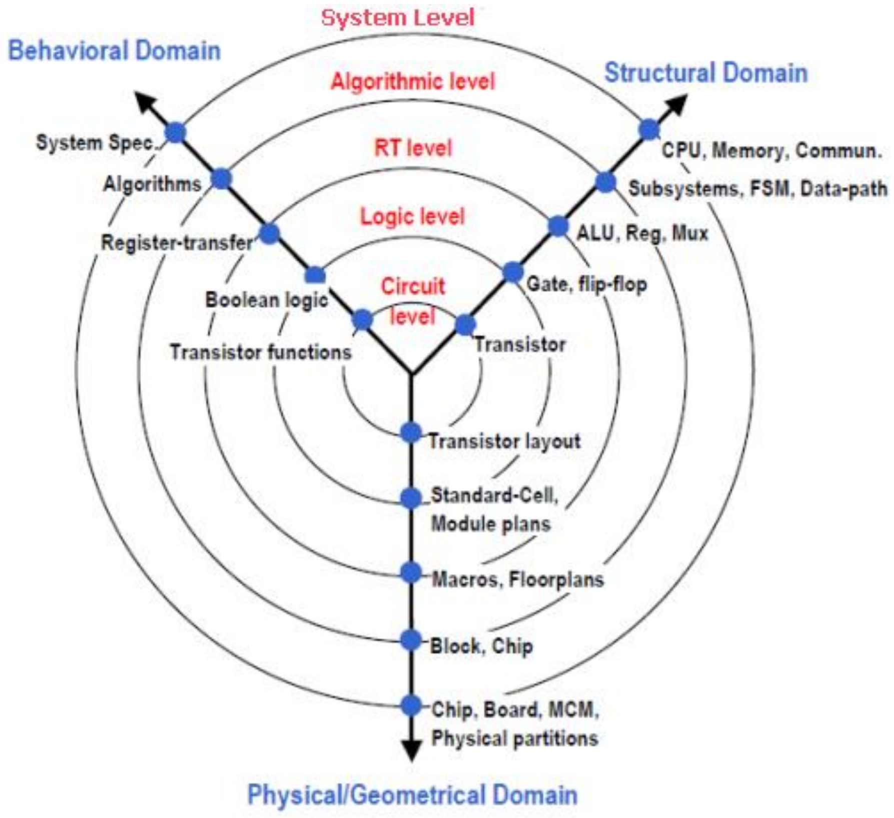
- Abstraction levels
- Design flow
- Tools of digital design flow
- Hardware Description Languages (HDLs)
  - Verilog
  - VHDL
  - Simple examples

Abstraction levels

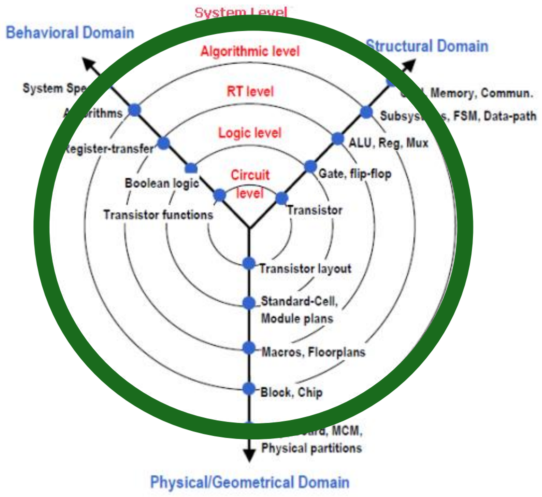
- The complexity of digital systems grows exponentially
  - Because of technological improvements, and user demands…
- Design methods have to cope with:
  - The design entries shifting towards higher levels of abstraction
  - Automated transformations into lower level models (with manual control and constraints)
  - Decreasing human interaction along the design flow
  - Automation cannot be avoided, even at the expense of efficiency
    - Who codes assembly in 20XX? No one implements a banking system using a technologically efficient language (such as C).

The Gajski-Kuhn Y diagram

- The concentric circles represent the abstraction levels
- The abstraction levels are examined from three viewpoints → arrows
  - How is the functionality described on that particular level?
  - What kind of structural elements are constituting the model?
  - What is the physical appearance of a design unit typical of that particular level?

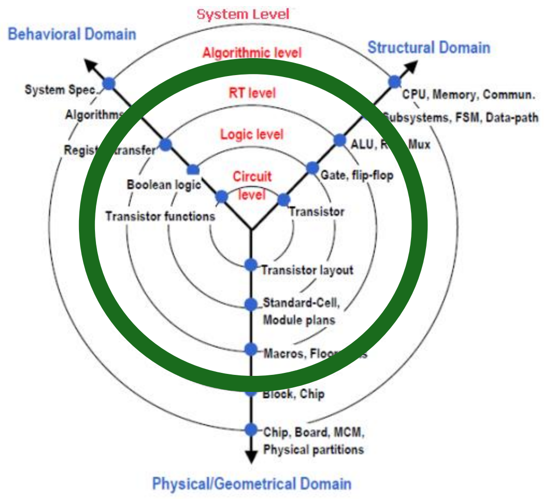


System level



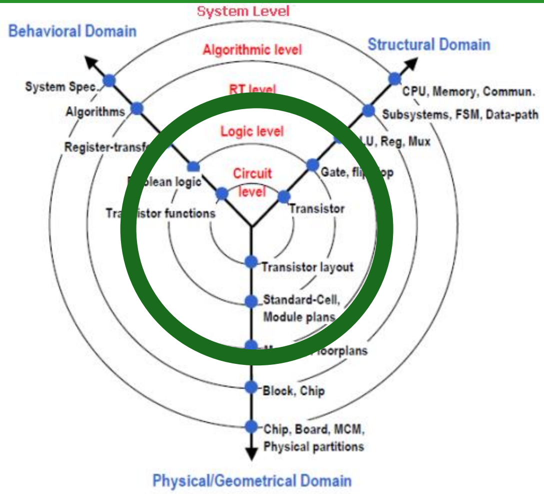
- The first aim is functional partitioning
  - Defining the subtasks of the components
  - Modeling the communication interfaces among design units and the external world
  - The implementation details are not important at this point (the models are independent of the technology)
  - HW/SW partitioning & co-design
- Tools
  - General purpose high level programming languages (mainly C, C++)

Algorithmic level

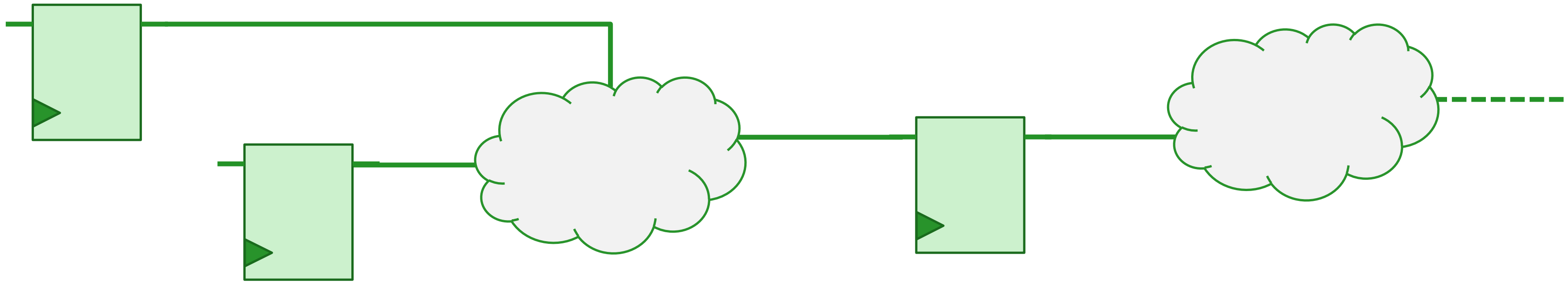


- Behavioral modeling of the functional blocks
- The behavioral models may be simulated
- Simulation time is the most important optimization goal → interpreted high level languages are rarely used, C & C++ are commonly applied
- There are several libraries that make hardware modeling more accurate and efficient (such as SystemC, Algorithmic C etc.)
  - Modeling of the concept of time
  - Modeling hardware-specific types (bit, logic vector etc.)
- E.g.: SystemC
  - Publicly available open standard http://accellera.org/downloads/standards/systemc

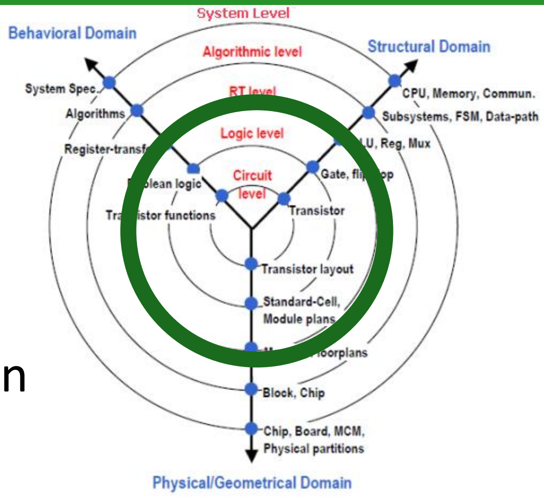
Register-Transfer Level (RTL)



- „The abstraction level defining the registers and the data transformation among them, including the interconnections and the timing (scheduling)”
- „The abstraction level defining what the circuit does between two rising edges of the clock.”
- „A set of continuous and scheduled assignments concurrently defining the behavior and the structure of a system.”



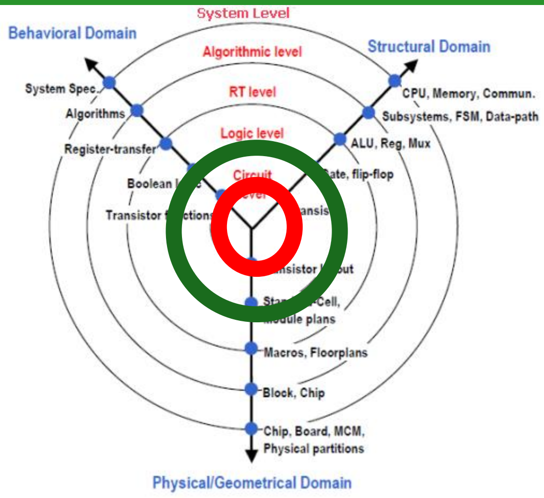
The RTL model



- Describing behavior
  - The RTL model expresses the behavior as consecutive steps of basic data transformations between registers (individually controlled storage elements)
- Structural elements
  - Datapath
    - registers storing the intermediate values of the variables
    - arithmetic/logic units performing data transformations
    - routing resources realizing the controllable interconnections of the resources (multiplexers, buses)
  - Control structures
    - Finite State Machines (FSMs) providing the datapath elements with appropriately scheduled control signals

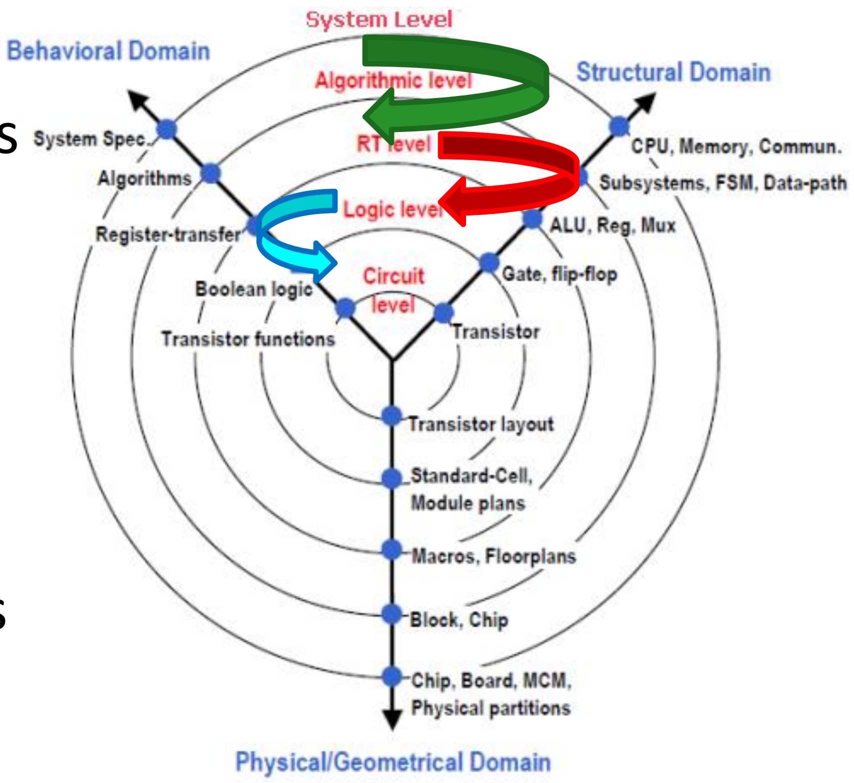
Gate level & circuit level

- Gate level
  - Logic gates and their interconnections
  - netlist
  - Independent from technology above the logic level
    - (more or less)
- Circuit level
  - Schematic
  - Physical design
    - Layout

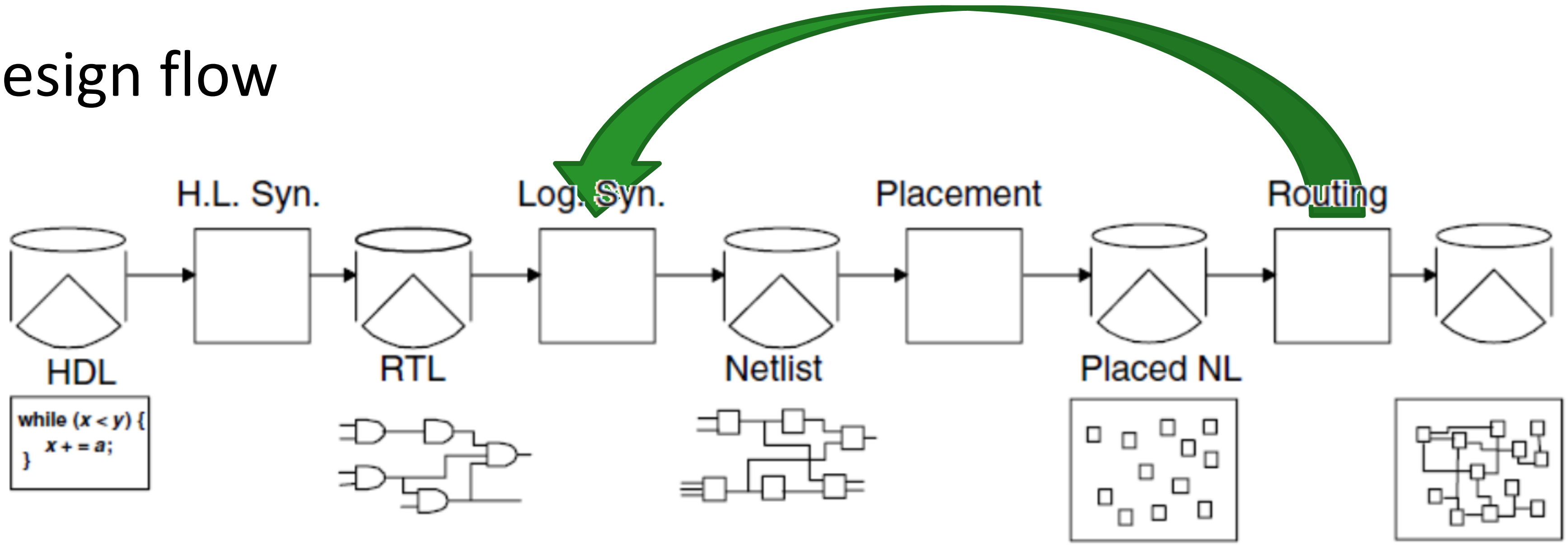


Synthesis

- The step between a higher and a lower abstraction level is synthesis
  - It can be done by a human or a PC
  - High-level synthesis (HLS)
  - Logic synthesis
  - Place and route
- On lower abstraction levels computer-based synthesis becomes more and more important.



Design flow



- Digital design is not that simple – it can be iterative
  - The exact propagation delay can only be calculated after the wiring/routing step (the length of the wires are needed).
  - It has an effect on logic synthesis, or maybe on higher abstraction levels.

High Level Synthesis (HLS)

- Algorithmic level → RTL
- HLS is used in conjunction with manual RTL design, however, modern HLS tools are more and more efficient
  - Cadence / C-to-Silicon compiler
  - Mentor Graphics / Catapult C
  - Synopsis / Synphony C compiler
- RTL models are generated automatically from the non-timed algorithmic models
- 5-10x productivity is promised by the HLS tool vendors

High Level Synthesis (HLS)

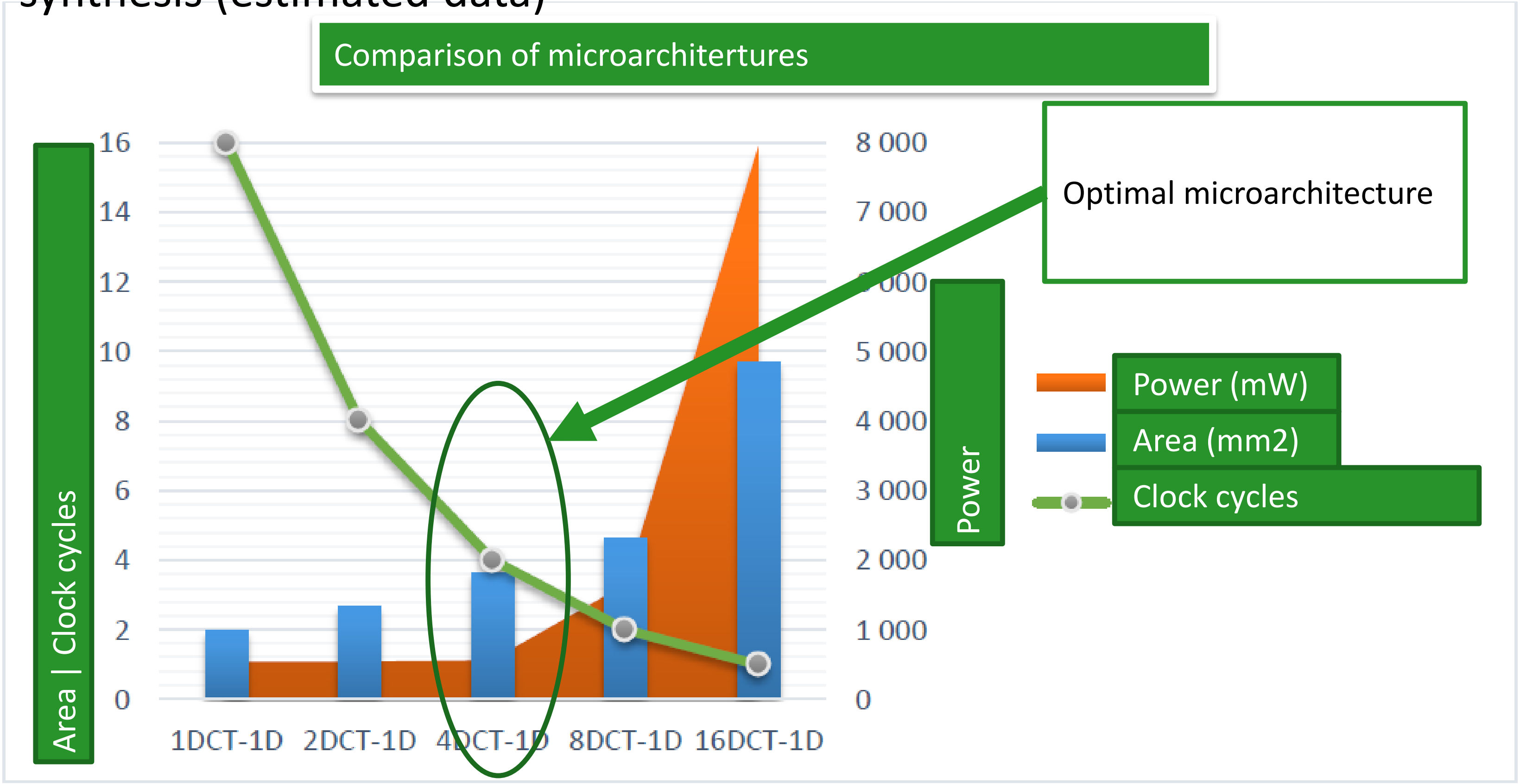
- During synthesis (automated or manual), the following problems have to be solved
- In case of control functions
  - Constructing FSMs
  - Managing the cooperation between FSMs and auxiliary circuits (counters, memory etc.)
- In case of data-processing functions
  - Resource-allocation
  - Scheduling
  - Binding

Advantages of HLS

- A 1M gate-equivalent design (which is actually not a very big one…)
  - „gate-equivalent”: area of the cells used / area of a single NAND2 cell
  - RTL model: ~300 KLOC (Kilo Lines Of Code)
  - Algorithmic model: ~30-40 KLOC
- Reusability issues
  - Although RTL is technology-independent, when switching from one technology to another, the RTL model is no longer optimal
- Design constraints are defined on a high level of abstraction
- Based on an algorithmic model, several RTL implementations may be generated, and optimized to a particular technology (e.g. FPGA vs. Std. cell ASIC).

HLS - example

- Discrete 2D cosine-transformation (8x8)
  - Technology: AMS 350nm, compiler: Cadence C-to-Silicon, without physical synthesis (estimated data)

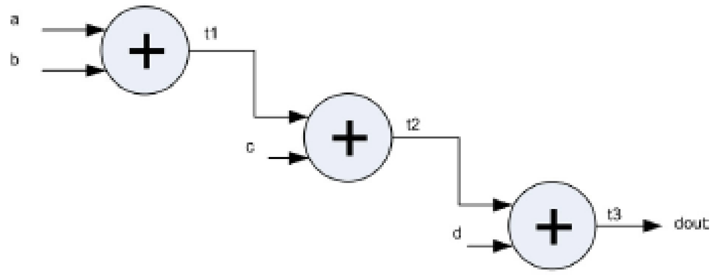


Example: microarchitecture selection

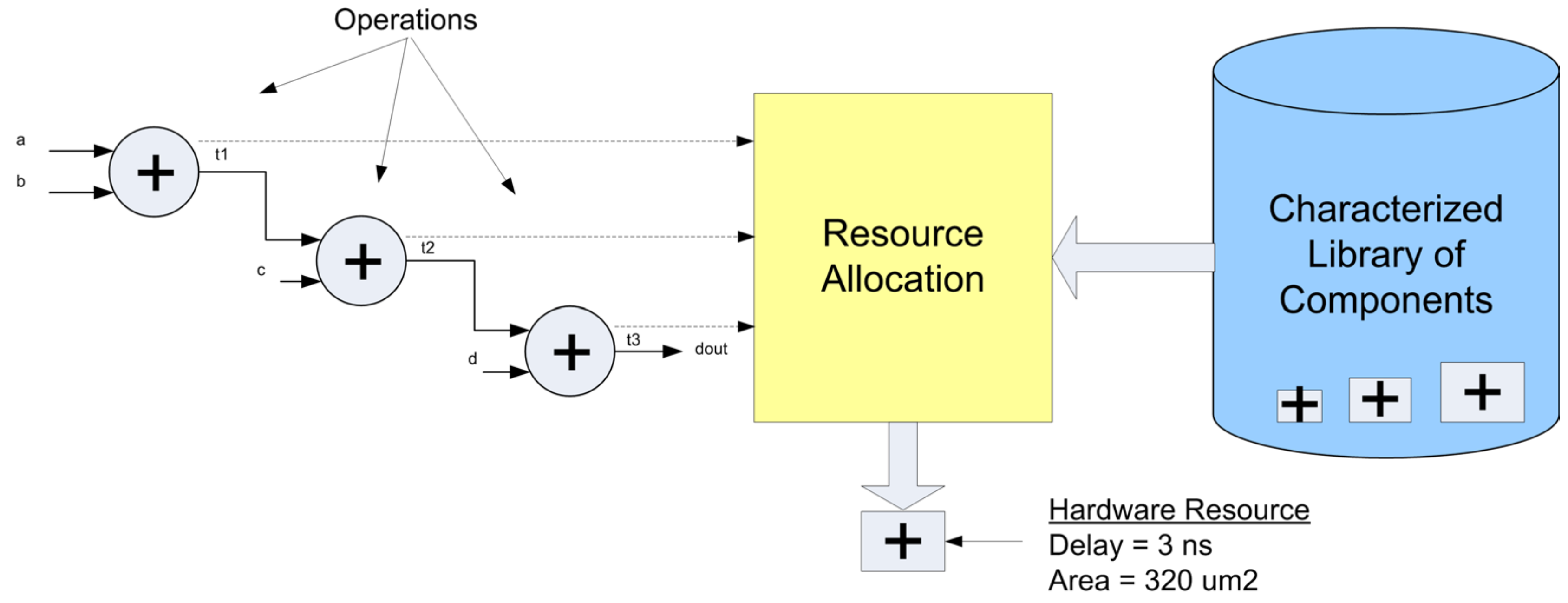
- Problem: add 4 32-bit unsigned integers
  - Source: Michael Fingeroff: High-Level Synthesis Blue Book

```
#include "accum.h"
void accumulate(int a, int b, int c, int d, int &dout){
  int t1,t2;

  t1 = a + b;
  t2 = t1 + c;
  dout = t2 + d;
}
```

  - The hardware is defined as a C++ function.
  - The inputs are passed by value, the outputs as reference
- The synthesizer creates a Data Flow Graph (DFG)
  - Every node represents a single instruction
  - The interconnections define the order of the instructions



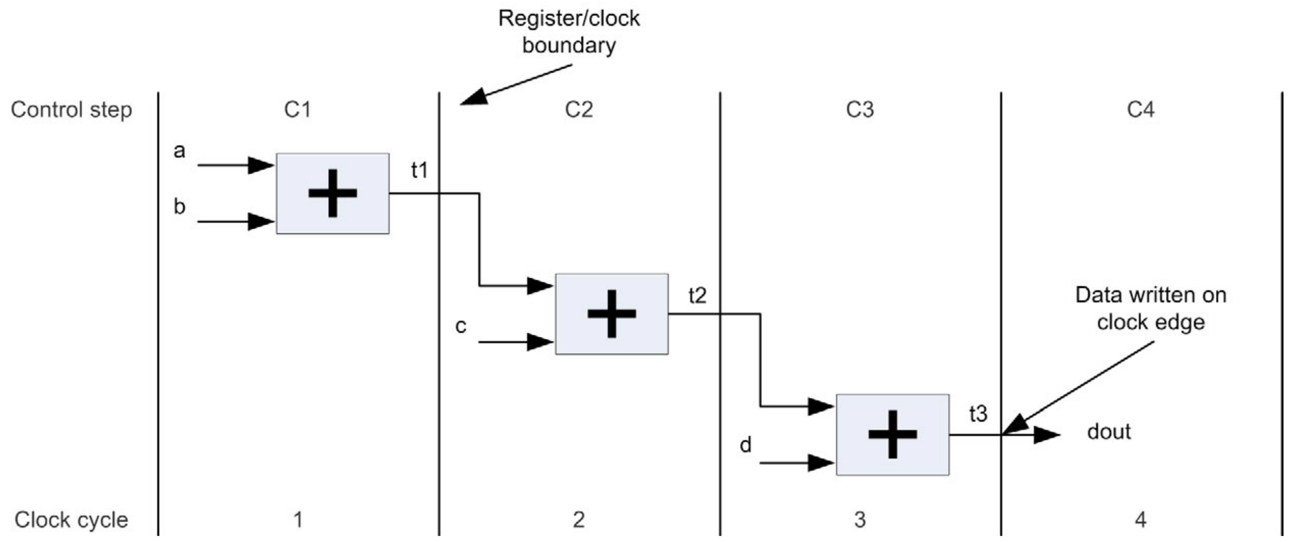
Example: resource-allocation



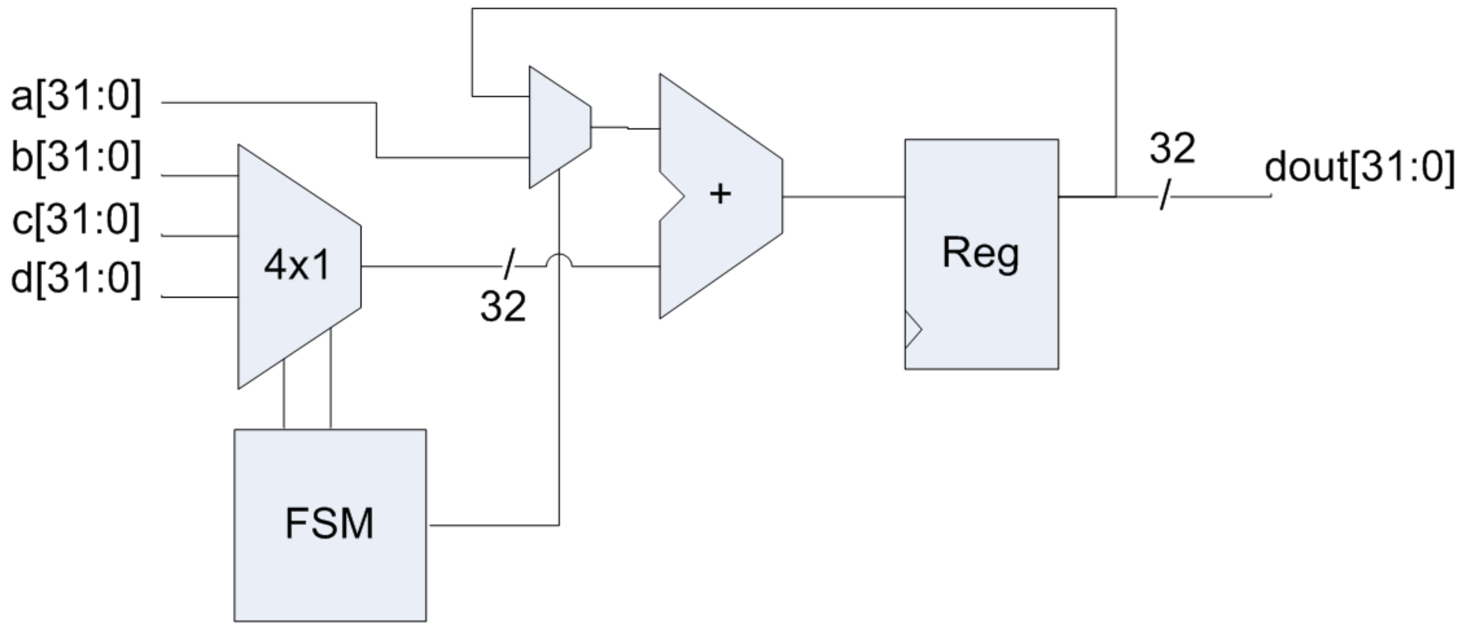
- The next step is called resource-allocation
  - Resources (storage and data manipulation) are selected from a library.
  - There are several components with the same functionality. They differ in area and/or timing. The HLS automatically selects the optimal one according to the design constraints.

Example: scheduling

- A possible solution: all additions are performed in independent, consecutive clock cycles.
  - A FSM is created controlling the datapath element (multiplexing the inputs of the adders).
  - The sum is produced in 4 clock cycles (latency is 4), but the resource requirement is minimal (only a single adder is needed.)



Example: the datapath



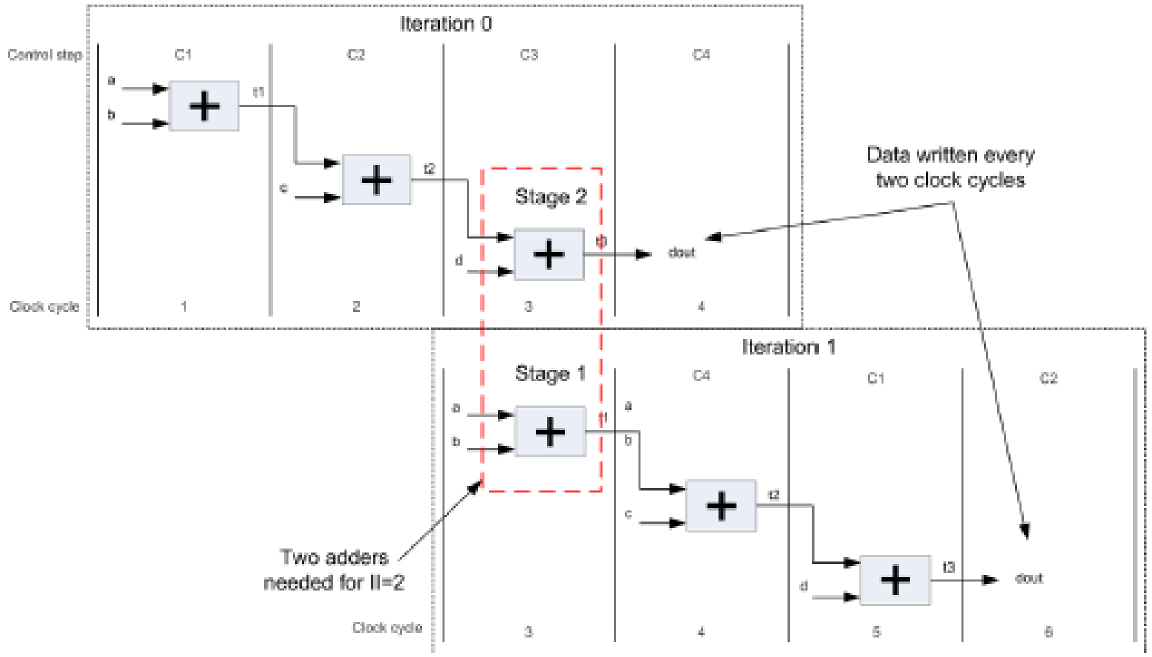
- Only a single adder is used, the operands are multiplexed from a clock cycle to another.

Example: pipeline

- To improve computation performance (decrease latency), the operations are overlapped.
- A pipeline is created
  - Initiation Interval (II): How frequently is the pipeline initiated (expressed in clock cycles)
  - Latency (L): The number of clock cycles a single input data set needs to get through the pipeline
  - Throughput (TP): How many clock cycles pass between two consecutive output data sets (e.g.: TP = 4 means that an output data set is provided by the pipeline every 4 clock cycles).
  - In the previous example
    - II: 4
    - L: 3
    - TP: 4

Example: pipe-line

Figure 4-9. Pipeline II=2, L=3, TP=2



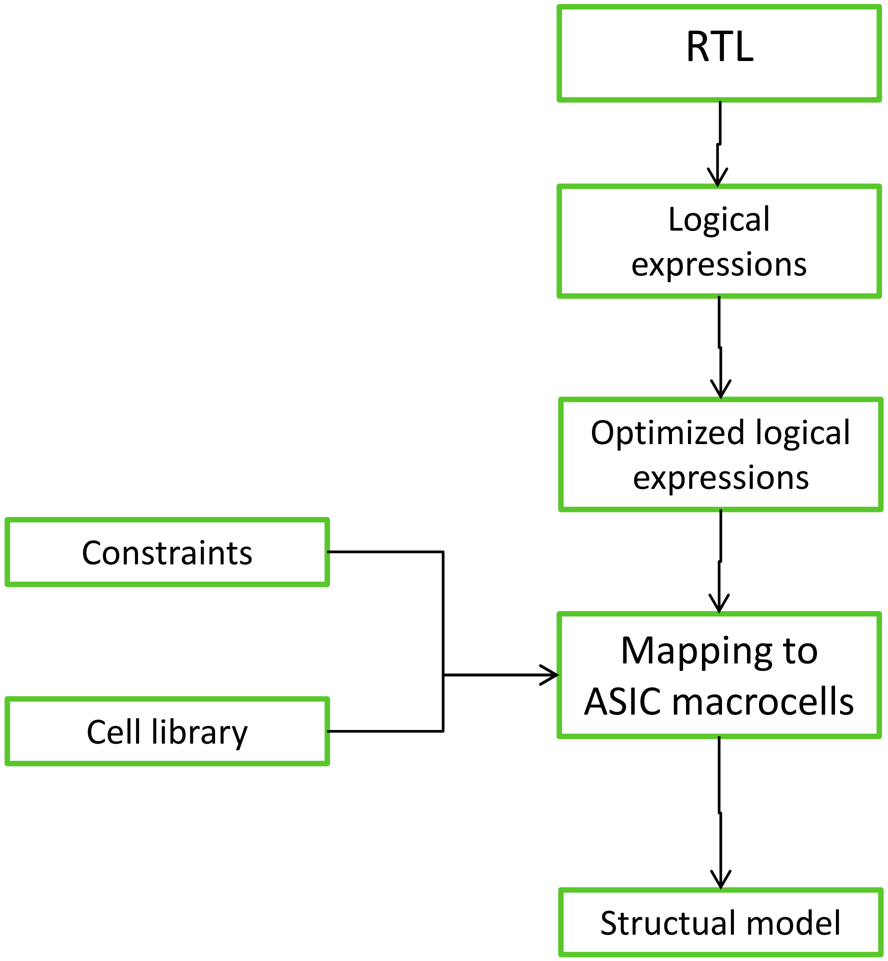
- By adding another adder, the computation performance may be doubled.
  - A new computation is initiated in every 2 clock cycles and a new result may be obtained in every 2 clock cycles

Logic synthesis

- Input
  - RTL code
  - HDL models of the cells
  - Timing and power-consumption characteristics of the cells (liberty)
- Output
  - Structural (gate-level) HDL model (netlist)
- Design constraints
  - Timing, area, power
  - The logic synthesis tool cooperates with the placement and routing algorithms

Logic synthesis

- RTL description → structural model (gate level)
- Constraints
  - Timing
  - Power
  - Area
- ASIC macrocell
  - Characteristic of the technology (std. Cell ASIC: gates), FPGA: Logic Element (LE)

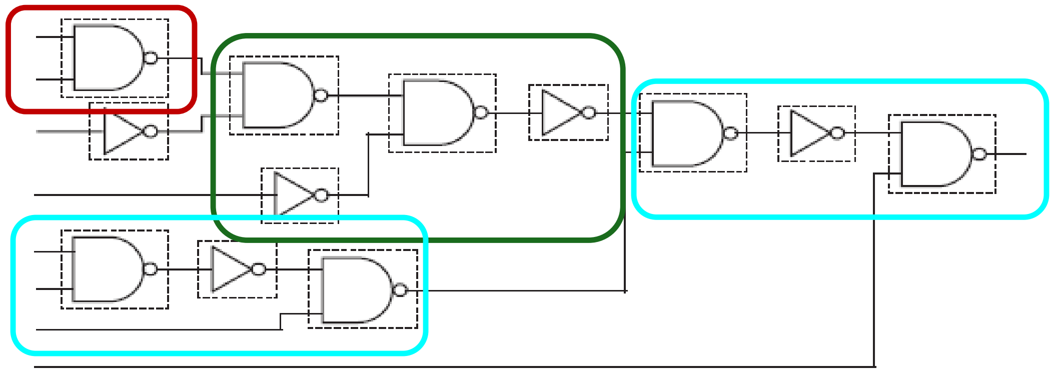


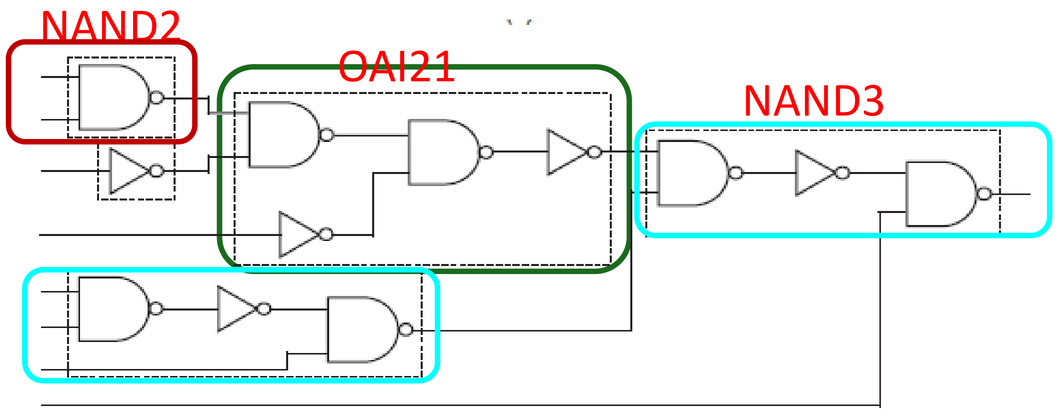
Synthesis steps

1. Read HDL (some optimization, such as dead code removal)
2. Flattening the design hierarchy
   - Global optimization may be performed (optimization between design units)
   - „spend CPU cycles rather than human cycles”
3. Optimizing logical expressions
   - Inferring widely used structures (adder, counter, memory, FSM etc.)
   - Inference of these resources is based on the requirements of the specific synthesis tool.
4. Mapping to macrocells
   - The macrocells implementing the logic functions in the design are selected from the cell library.

Mapping to macrocells

- A tree is constructed from the logic expressions
- The logic expressions of the design are implemented by multiple instances of the different macrocells provided by the cell library.





Hardware Description Languages (HDLs)

- VHDL (***V****HSIC* ***H****ardware* ***D****escription* ***L****anguage*, IEEE STD 1987)
  - VHSIC: ***V****ery* ***H****igh* ***S****peed* ***I****ntegrated* ***C****ircuit*
  - Development initiated by the US Department of Defense (DoD)
  - Original objective: documentation of already designed and manufactured ASICs.
- SystemVerilog (***VERI****fication and* ***LOG****ic*, IEEE STD 2005)
  - Gateway Design Automation → Cadence Design Systems
  - Original objective: logic simulation
  - The original language (Verilog) improved in 2005 → SystemVerilog

Terminology

- HDLs are not programming languages
  - The language construction is similar (syntax)
  - Partly or completely different meaning (semantics)
- The following widely used expressions are not accurate, and sometimes completely wrong
  - HDL program → HDL model
  - Compiling the HDL model → elaboration
  - Running the HDL program → HDL simulation
  - Programming the FPGA → configuration
  - etc…

History of VHDL

- Development initiated by the US Department of Defense (DoD)
- Original objective: documentation of the functional specification of already designed and manufactured ASICs.
- Later it was used for logic simulation, and then it was an input for circuit synthesis
- In 1987 IEEE standard (IEEE 1076-1987)
  - Update: IEEE STD 1164: 9-state logic signal (??? 01UXZWLH- ☺)
- Minor changes: 1993, 2000, 2002
- Latest version: VHDL 2008 (IEEE 1076-2008)
  - Modelling of reusable elements
  - Functional verification

VHDL

- Based on the programming language ADA
- Type-oriented, it uses keywords instead of symbols
  - It is easy to read
  - Case-insensitive
- Optimal for large designs
  - It supports hierarchical designs
  - It supports libraries and packages

VHDL example

```
library ieee;
use ieee.std_logic_1164.all;

entity d_flip_flop is
  port (clk:   in  std_logic;
        rst_n: in  std_logic;
        d:     in  std_logic;
        q:     out std_logic);
end entity d_flip_flop;

architecture rtl of d_flip_flop is
begin
  L_MAIN: process (clk, rst_n)
  begin
    if ( rst_n = ’0’ ) then
      q <= ’0’;
    elsif ( rising_edge(clk) ) then
      q <= d;
    end if;
  end process;
end architecture rtl;
```

Library and package declarations

Entity declaration

- generics
- portlist

Architecture

- Functional description of the module
- Instantiations of the submodules

History of SystemVerilog

- The Verilog language was developed by Gateway Design Automation.
- GDA was purchased by Cadence Design Systems, Verilog became an open standard.
- It became IEEE Standard 1364 in 1995.
- The standard was updated several times
  - In 2005 there was a major update (SystemVerilog), with many new features and capabilities to aid design verification and design modeling.
- The SystemVerilog contains the whole Verilog from 2009.

SystemVerilog example

```
module d_flip_flop (input  logic clk,
                    input  logic rst_n,
                    input  logic d,
                    output logic q);

  always_ff @ (posedge clk, negedge rst_n)
  begin
    if ( !rst_n ) q <= 0;
    else q <= d;
  end

endmodule
```

Module declaration

- generics
- portlist

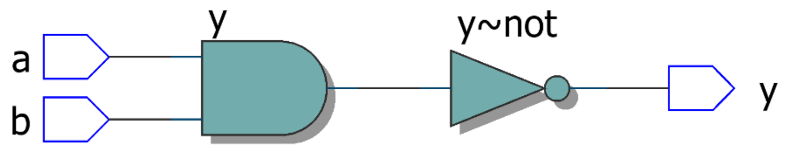
Module body

- Functional description of the module
- Instantiations of the submodules

NAND gate

```
library ieee;
use ieee.std_logic_1164.all;

entity nand_gate is
  port (a: in  std_logic;
        b: in  std_logic;
        y: out std_logic);
end entity nand_gate;

architecture rtl of nand_gate is
begin

  y <= a nand b;

end architecture rtl;
```

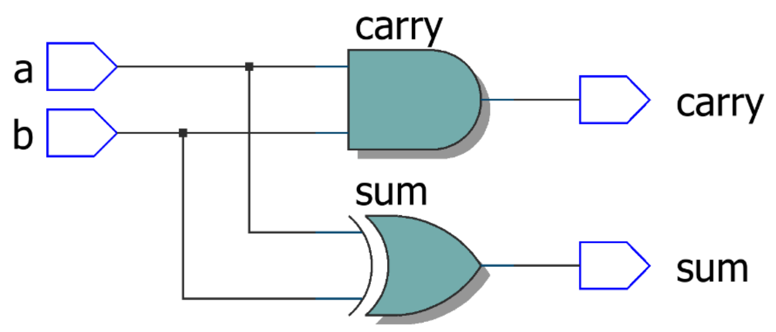
```
module nand_gate (input  logic a,
                  input  logic b,
                  output logic y);

  assign y = ~(a & b);

endmodule
```



Half-adder (RTL)

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity half_adder is
  port (a:     in  std_logic;
        b:     in  std_logic;
        sum:   out std_logic;
        carry: out std_logic);
end entity half_adder;

architecture rtl of half_adder is
begin

  sum <= a xor b;
  carry <= a and b;

end architecture rtl;
```

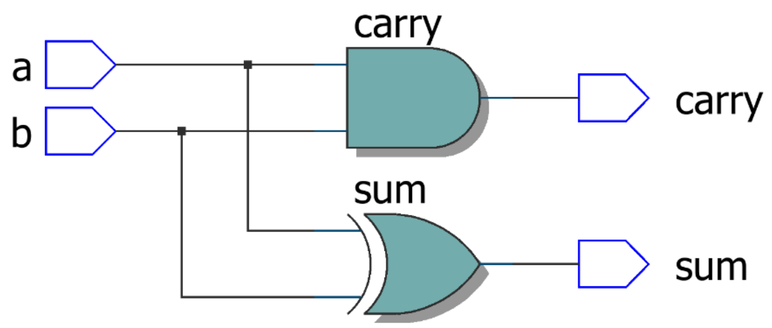
```systemverilog
module half_adder (input  logic a,
                   input  logic b,
                   output logic sum,
                   output logic carry);

  assign sum = a ^ b;
  assign carry = a & b;

endmodule
```



Half-adder – using components instantiation

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity half_adder is
  port (a:      in  std_logic;
        b:      in  std_logic;
        sum:    out std_logic;
        carry:  out std_logic);
end entity half_adder;

architecture rtl of half_adder is
begin

  L_XOR: entity work.xor_gate(rtl)
           port map (a => a, b => b,
                     y => sum);

  L_AND: entity work.and_gate(rtl)
           port map (a => a, b => b,
                     y => carry);

end architecture rtl;
```

```systemverilog
module half_adder (input  logic a,
                   input  logic b,
                   output logic sum,
                   output logic carry);

  xor_gate xor1(.a(a), .b(b),
                .y(sum));

  and_gate and1(.a(a), .b(b),
                .y(carry));

endmodule
```



Concept

- The synthesizable subsets of HDLs are very similar
- The statements are performed concurrently. The order of statements is irrelevant.
- Interface
  - The inputs and outputs are defined.
- Implementation of a combinational circuit
  - Definition of the logic functions between the inputs and the outputs
- Implementation of a sequential circuit
  - A sensitivity list is defined.
  - The body of the sequential statement (process / always block) describes what the circuit does at the rising edge of the clock