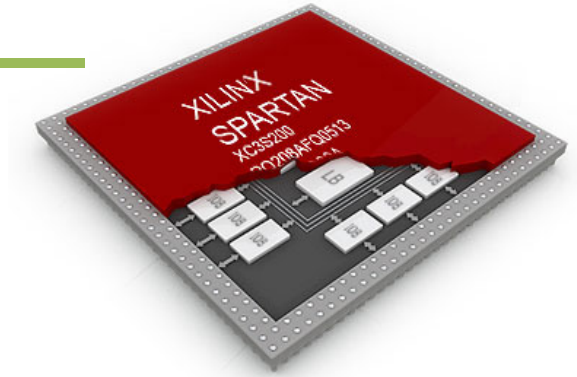
VHDL

Hardware description languages
Introduction to VHDL
Xilinx ISE Design Suite
Xilinx Vivado
Altera Quartus 2



Embedded Systems
Università degli Studi dell'Aquila

Outline

- Introduction
- Hardware description languages
- Introduction to VHDL
- Simulation and Synthesis
- Xilinx ISE Design Suite – Xilinx Vivado
- Altera Quartus 2

Component Instantiation

- Once a system is defined (and maybe included in a package) it can be used in a more complex system of higher level

- As seen, a component must be declared using the keyword *component* and instantiated via a *port map*. Multiple instances of the same component can coexist in a single design.

- VHDL 93 supports an alternative syntax that combines the declaration and the instantiation of a component

```
U0: entity dut_ent (dut_arch) port map(A, B);
```

Generic Components

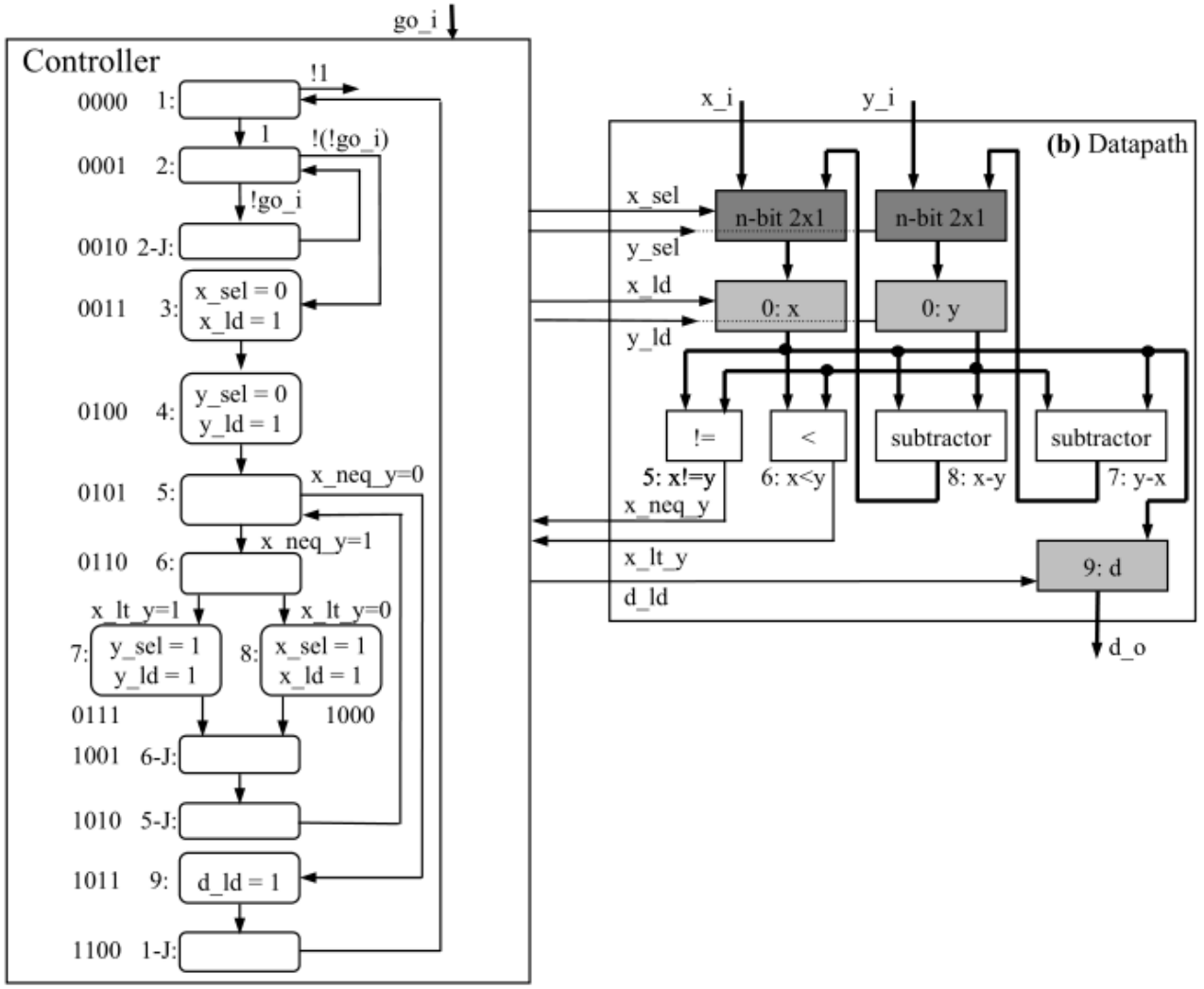
- VHDL supports the construction of parametric components through the *generic* construct

```
entity half_adder is

generic (N: integer);

port (A, B: in std_logic_vector (N-1 downto 0);
```

- In addition to the *port map*, the instantiation of a parametric component will include also a *generic map*

```
generic map (N => 10)
```

GCD Calculator (FSM + DP)



GCD Calculator (FSM + DP)

```
entity mux is
    port(         rst, sLine: in std_logic;
                  load, result: in std_logic_vector( 3 downto 0 );
                  output: out std_logic_vector( 3 downto 0 )
    );
end mux;

architecture mux_arc of mux is
begin
    process( rst, sLine, load, result )
    begin
        if( rst = '1' ) then
            output <= "0000";              -- do nothing
        elsif sLine = '0' then
            output <= load;                -- load inputs
        else
            output <= result;              -- load results
        end if;
    end process;
end mux_arc;
```

GCD Calculator (FSM + DP)

```
entity comparator is
    port(         rst: in std_logic;
                  x, y: in std_logic_vector( 3 downto 0 );
                  output: out std_logic_vector( 1 downto 0 )
    );
end comparator;

architecture comparator_arc of comparator is
begin
    process( x, y, rst )
    begin
          if( rst = '1' ) then
              output <= "00";          -- do nothing
          elsif( x > y ) then
              output <= "10";          -- if x greater
          elsif( x < y ) then
              output <= "01";          -- if y greater
          else
              output <= "11";          -- if equivalence.
          end if;
    end process;
end comparator_arc;
```

GCD Calculator (FSM + DP)

```
entity subtractor is
    port(       rst: in std_logic;
                cmd: in std_logic_vector( 1 downto 0 );
                x, y: in std_logic_vector( 3 downto 0 );
                xout, yout: out std_logic_vector( 3 downto 0 )
    );
end subtractor;
```

GCD Calculator (FSM + DP)

```
architecture subtractor_arc of subtractor is
begin
    process( rst, cmd, x, y )
    begin
        if( rst = '1' or cmd = "00" ) then          -- not active.
            xout <= "0000";
            yout <= "0000";
        elsif( cmd = "10" ) then            -- x is greater
            xout <= ( x - y );
            yout <= y;
        elsif( cmd = "01" ) then            -- y is greater
            xout <= x;
            yout <= ( y - x );
        else
            xout <= x;                      -- x and y are equal
            yout <= y;
        end if;
    end process;
end subtractor_arc;
```

GCD Calculator (FSM + DP)

```
entity gcd is
    port(     rst, clk, go_i: in std_logic;
              x_i, y_i: in std_logic_vector( 3 downto 0 );
              d_o: out std_logic_vector( 3 downto 0 )
    );
end gcd;

architecture gcd_arc of gcd is
begin



end gcd_arc;
```

GCD Calculator (FSM + DP)

```
component fsm is
    port(         rst, clk, proceed: in std_logic;
                  comparison: in std_logic_vector( 1 downto 0 );
                  enable, xsel, ysel, xld, yld: out std_logic
    );
end component;

component mux is
    port(         rst, sLine: in std_logic;
                  load, result: in std_logic_vector( 3 downto 0 );
                  output: out std_logic_vector( 3 downto 0 )
    );
end component;

component comparator is
    port(         rst: in std_logic;
                  x, y: in std_logic_vector( 3 downto 0 );
                  output: out std_logic_vector( 1 downto 0 )
    );
end component;
```

GCD Calculator (FSM + DP)

```
component subtractor is
    port(         rst: in std_logic;
                  cmd: in std_logic_vector( 1 downto 0 );
                  x, y: in std_logic_vector( 3 downto 0 );
                  xout, yout: out std_logic_vector( 3 downto 0 )
    );
end component;

component regis is
    port(         rst, clk, load: in std_logic;
                  input: in std_logic_vector( 3 downto 0 );
                  output: out std_logic_vector( 3 downto 0 )
    );
end component;

signal xld, yld, xsel, ysel, enable: std_logic;
signal comparison: std_logic_vector( 1 downto 0 );
signal result: std_logic_vector( 3 downto 0 );

signal xsub, ysub, xmux, ymux, xreg, yreg: std_logic_vector( 3 downto 0 );
```

GCD Calculator (FSM + DP)

```
-- doing structure modeling here

-- FSM controller
TOFSM: fsm port map(    rst, clk, go_i, comparison,
                            enable, xsel, ysel, xld, yld );
-- Datapath
X_MUX: mux port map( rst, xsel, x_i, xsub, xmux );
Y_MUX: mux port map( rst, ysel, y_i, ysub, ymux );
X_REG: regis port map( rst, clk, xld, xmux, xreg );
Y_REG: regis port map( rst, clk, yld, ymux, yreg );
U_COMP: comparator port map( rst, xreg, yreg, comparison );
X_SUB: subtractor port map( rst, comparison, xreg, yreg, xsub, ysub );
OUT_REG: regis port map( rst, clk, enable, xsub, result );

d_o <= result;
```

Simulation

- The simulation process allows the verification of the modeled circuit.

- The simulation can be:
  - A functional simulation, used to verify that the system effectively implements the desired functionality. This type of simulation requires the definition of an appropriate set of *stimuli*.

  - A timing simulation, used to verify the (temporal) performance of the circuit. This type of simulation requires the definition of the target platform and the *implementation* of the system.

Testbenches

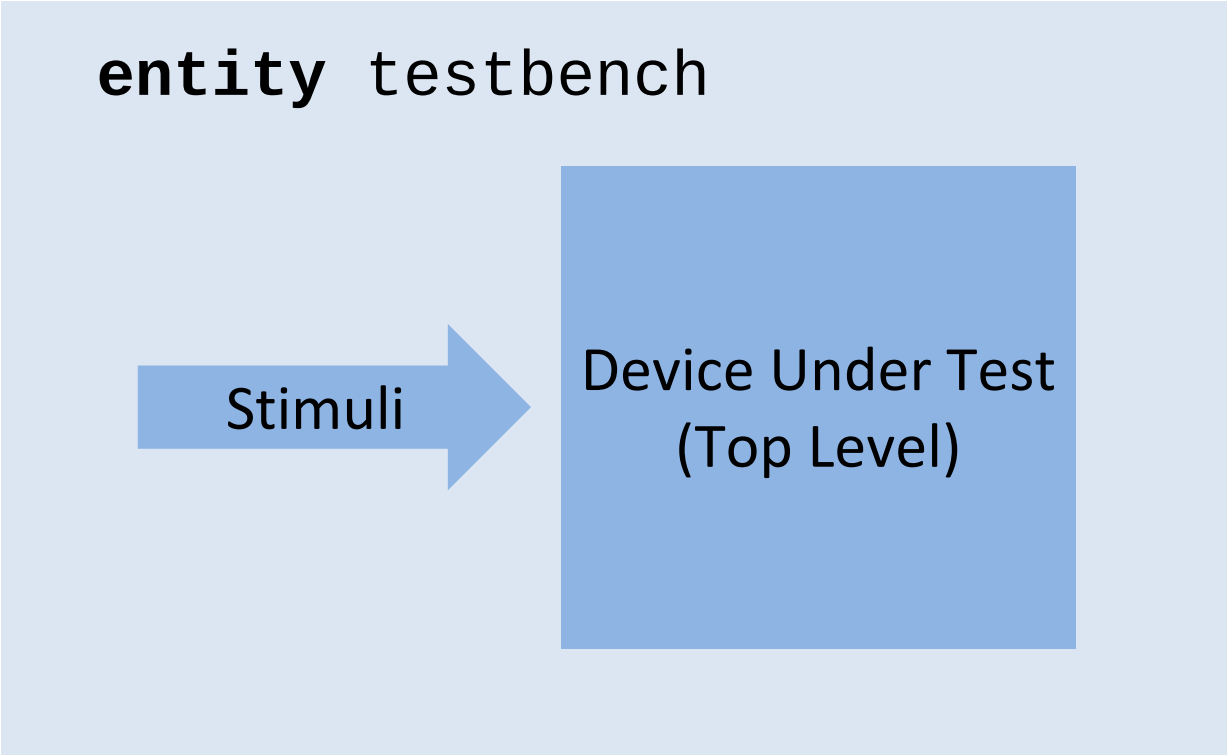
- The simulation of a circuit requires a model of the circuit and a description of the stimuli to be applied

- In VHDL this is accomplished using a testbench, i.e. a module:
  - With no inputs and no outputs
  - Including the system to be simulated
  - Whose only functionality is to apply a proper set of stimuli with the desired timing

- The testbenches are usually the only part of the VHDL code including temporal (non-synthesizable) statements

```
y <= "00" after 5s;
```

Working principle



Developing a testbench

- A testbench is usually the entity of higher level in a project
  - for this reason, this entity has no ports

```
entity testbench1 is
end testbench1;
```

- It will include the device to be tested as a components and as many signals as the inputs to be excited.
- Implementation of a testbench:
  - Connect declared signals to the inputs of the system under test using a *port map*
  - Specify the temporal behaviour of the signals to obtain the sequences of desired inputs

GCD Calculator Testbench (FSMD)

```
entity test_GCD is
end test_GCD;

architecture Bench of test_GCD is

component gcd
port(   clk:    in std_logic;
        rst:    in std_logic;
        go_i:   in std_logic;
        x_i:    in unsigned(3 downto 0);
        y_i:    in unsigned(3 downto 0);
        d_o:    out unsigned(3 downto 0)
);
end component;

signal T_clk,T_rst,T_go_i: std_logic;
signal T_x_i, T_y_i, T_d_o: unsigned(3 downto 0);
```

GCD Calculator Testbench (FSMD)

```
begin

    U1: GCD port map(T_clk,T_rst,T_go_i,T_x_i,T_y_i,T_d_o);

    Clk_sig: process
    begin
        T_clk<='1';
        wait for 5 ns;
        T_clk<='0';
        wait for 5 ns;
    end process;
```

GCD Calculator Testbench – alternative description

```
begin
    U1: GCD port map
        (clk => T_clk,
         rst => T_rst,
         go_i => T_go_i,
         x_i => T_x_i,
         y_i => T_y_i,
         d_o => T_d_o
         );

    Clk_sig: process
    begin
        T_clk<='1';
        wait for 5 ns;
        T_clk<='0';
        wait for 5 ns;
    end process;
```

Simulation tools

- FPGAs' Integrated Development Environments usually include a simulation tool or support the use of external ones.

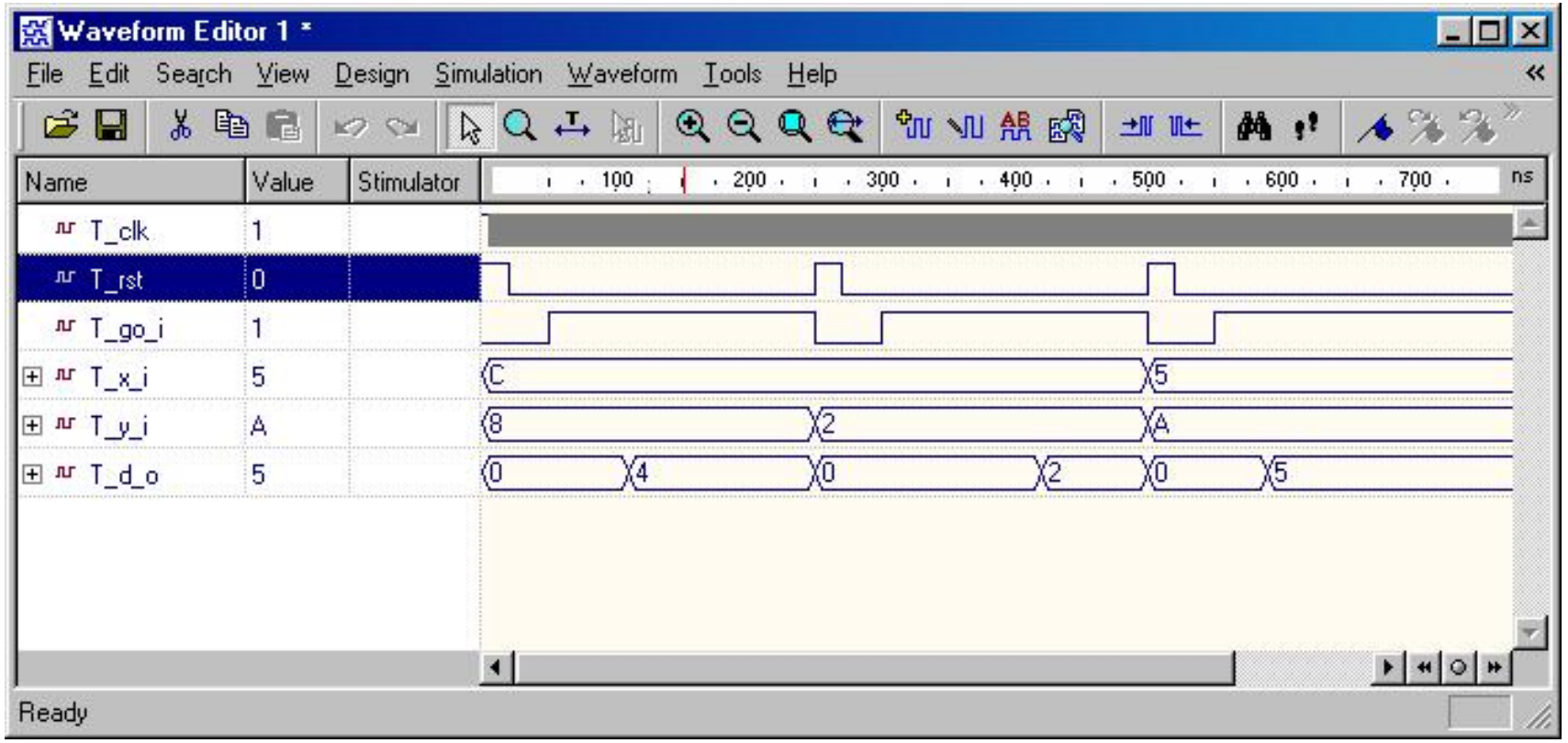
- One of the best known simulation tools is ModelSim from Mentor Graphics.
  - Manufacturers' ModelSim versions (specifically optimized for their FPGA) are provided by Mentor Graphics and often integrated within manufacturers' Design Suites.

- An open source tool to consider: GHDL (Gnu HDL)

Simulation tools

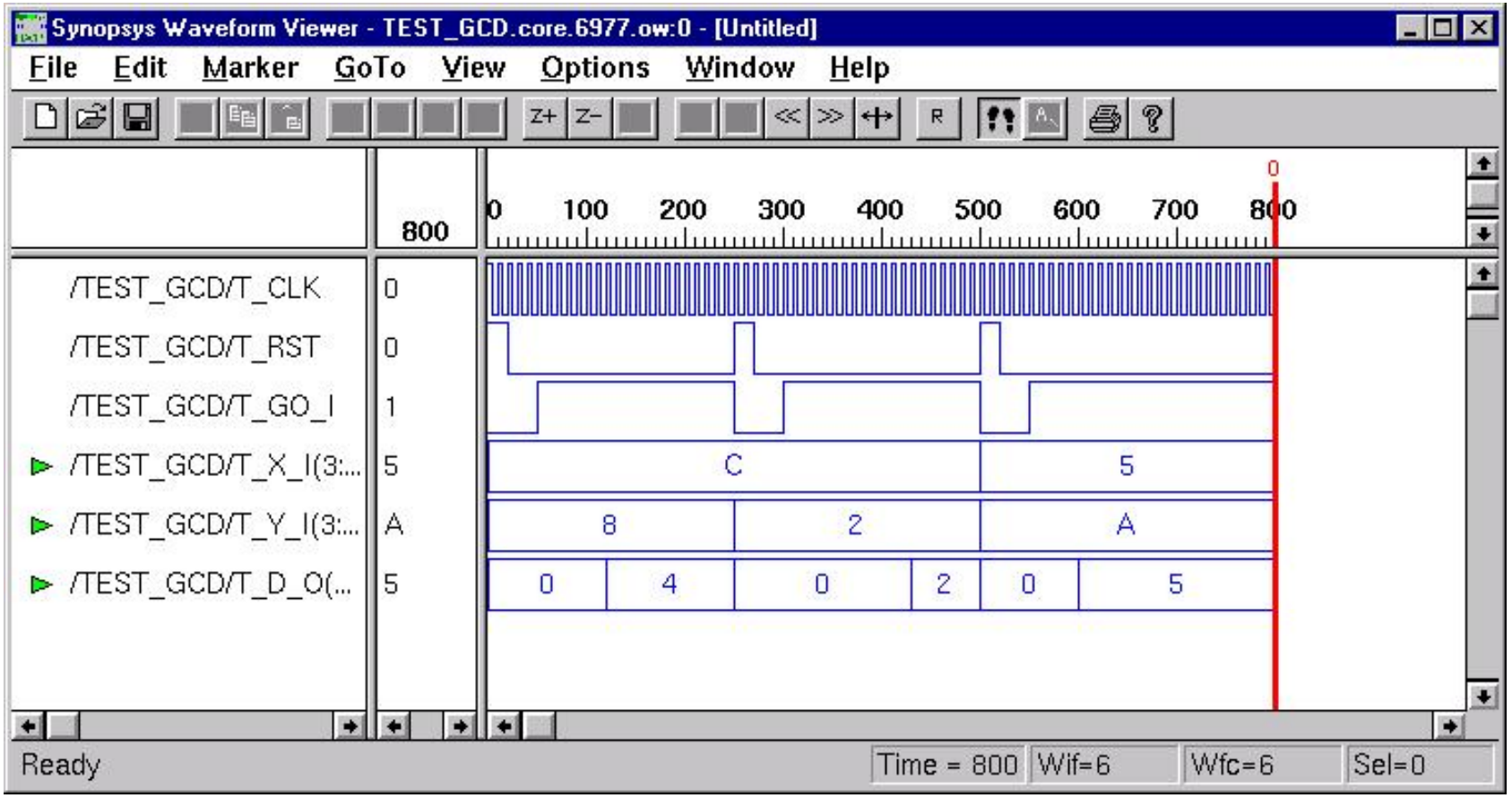
- A simulation tool allows the verification (functional or timing) of a system given the set of input signals.
- Besides testbenches, many simulators allow to manually specify the set of stimuli.
- Waveform Generators are usually graphical interface tools that allow users to define with relative ease the timing diagrams of the input signals.

Simulation of GCD (FSMD)



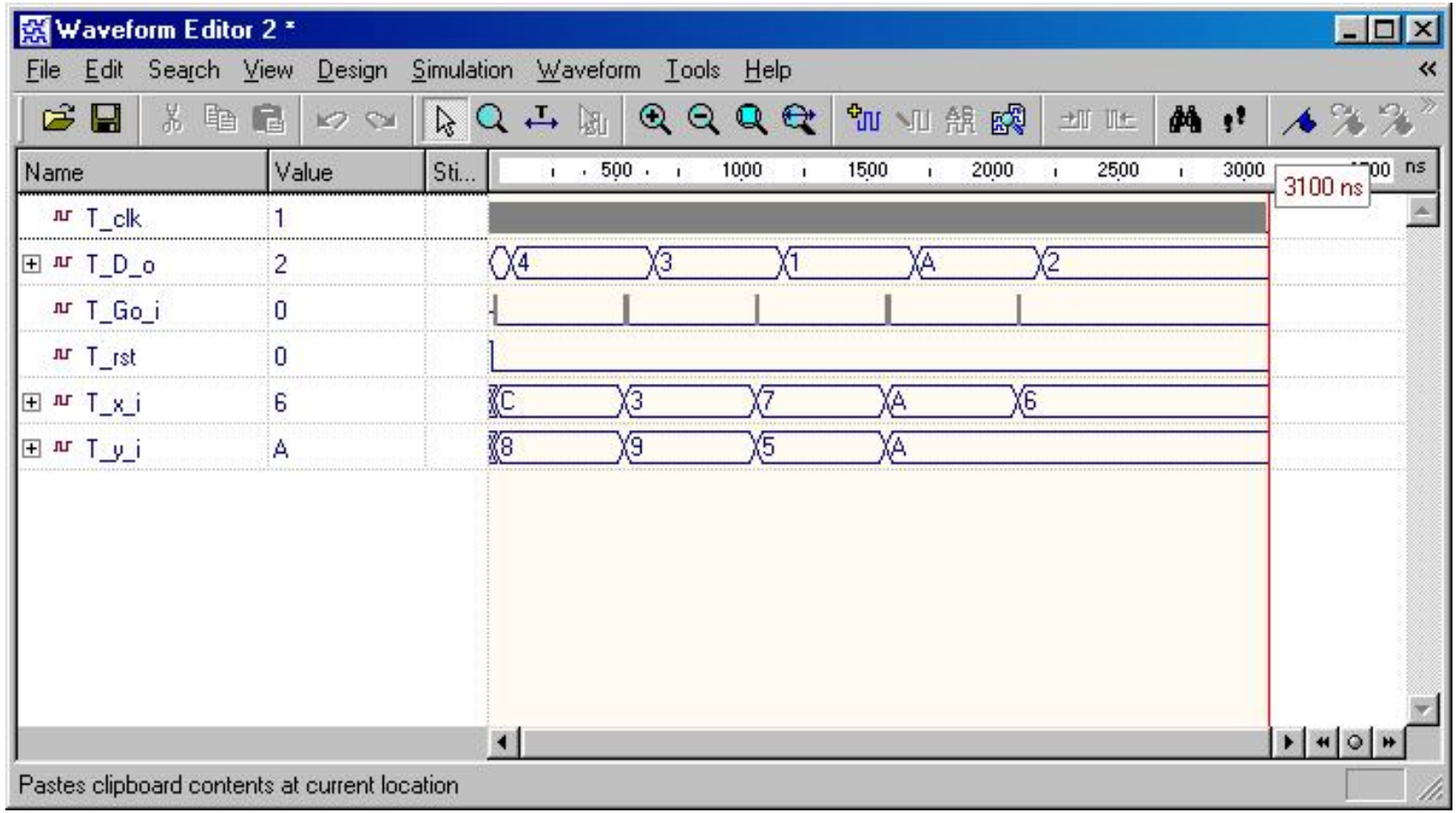
RTL Code Simulation

Simulation of GCD (FSMD)



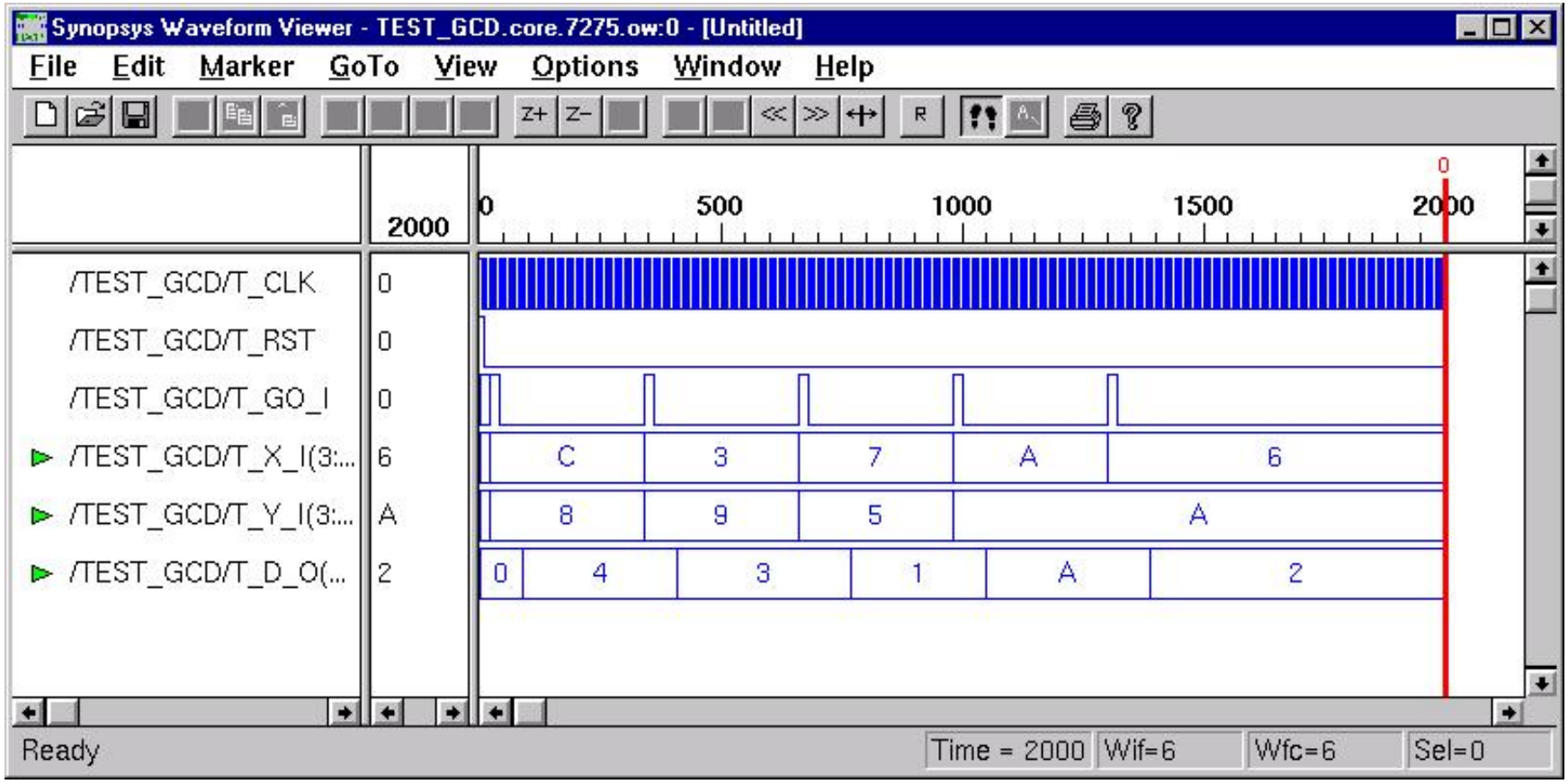
Gate level implementation Simulation

Simulation of GCD (FSM + DP)



RTL Code Simulation

Simulation of GCD (FSM + DP)



Gate level implementation Simulation

Synthesis

- *Synthesis* is process of translating a description of a hardware system at higher abstraction level into an (optimized) implementation on a lower abstraction level.
  - From HDL description to an implementation in terms of logic gates
- FPGA manufacturers usually provide appropriate synthesis tool optimized for the target devices.
- Not necessarily a simulated HDL model will be also synthesizable

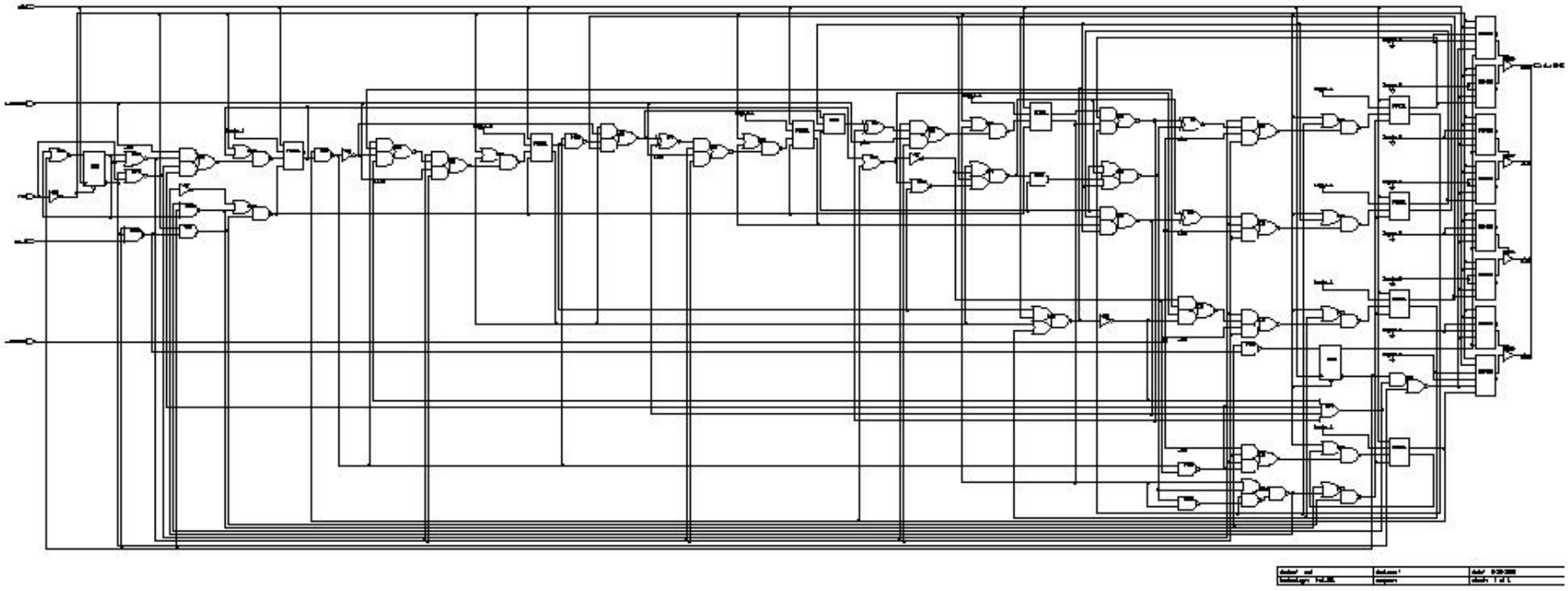
Synthesis tools

- The synthesis tool must interpret VHDL constructs and translate them into the corresponding circuit implementation
- The fundamental problem is to make sure that the translation process is correct (i.e. obtain the desired circuit)
- A frequent problem example: undesired latches

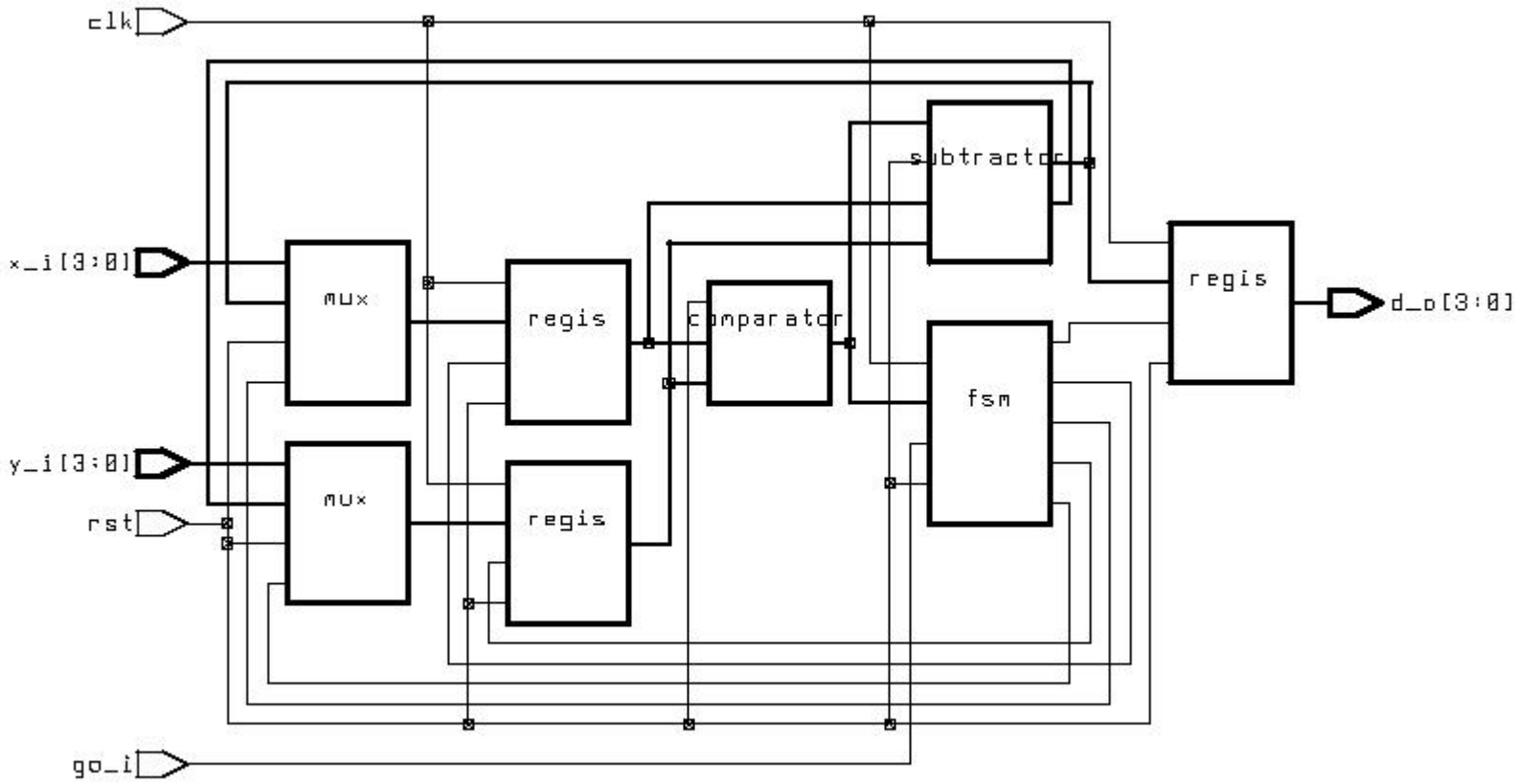
Constraints

- Besides the model, the synthesis process is based also on *constraints* fixed by the user
- Using the constraints, it is possible to try to obtain a certain behaviour of the synthesis tool, in order to optimize the final result
- Most of the development environments supports the definition of the constraints by using dedicated files
  - Example: Xilinx ISE User Constraints Files (.ucf)

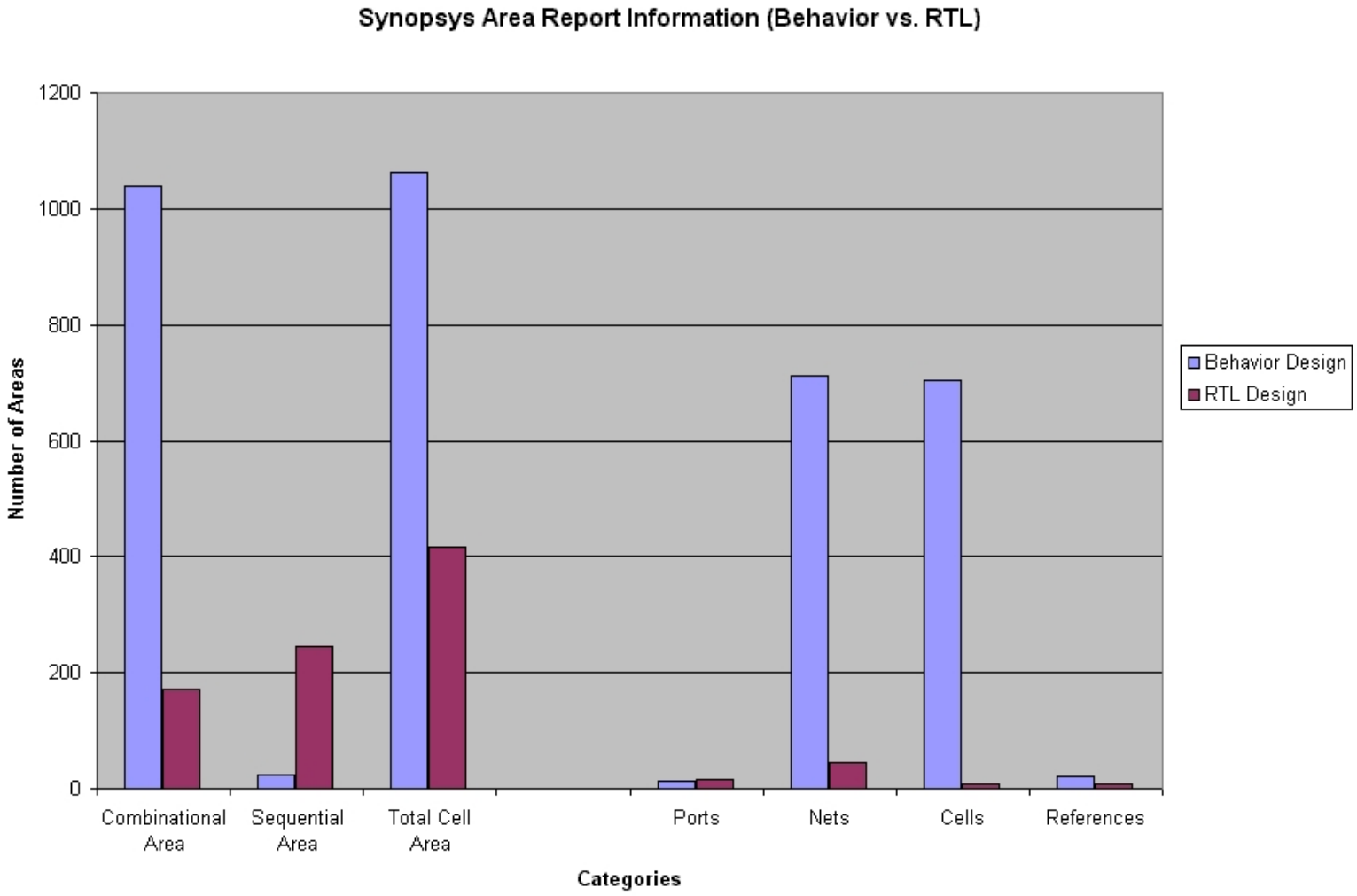
Synthesis of GCD (FSMD)



Synthesis of GCD (FSM + DP)



Behaviour vs. RTL Synthesis



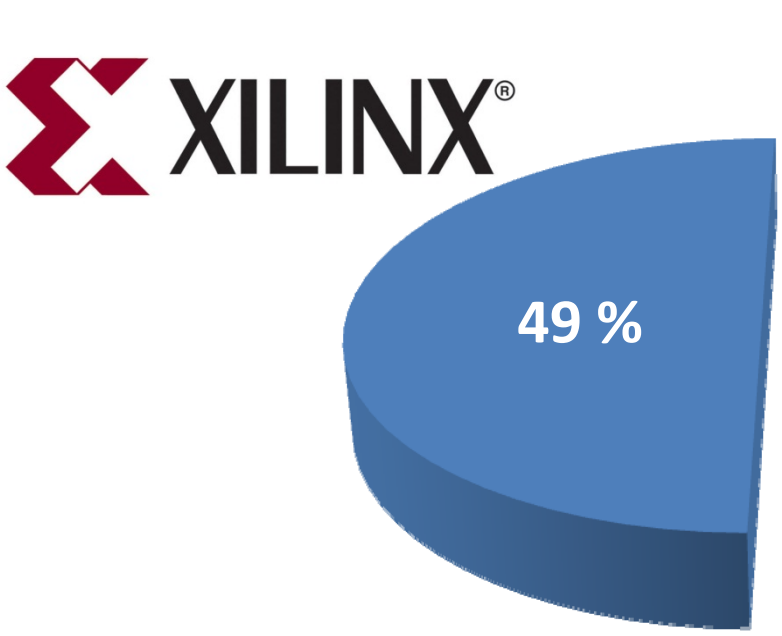
Implementation

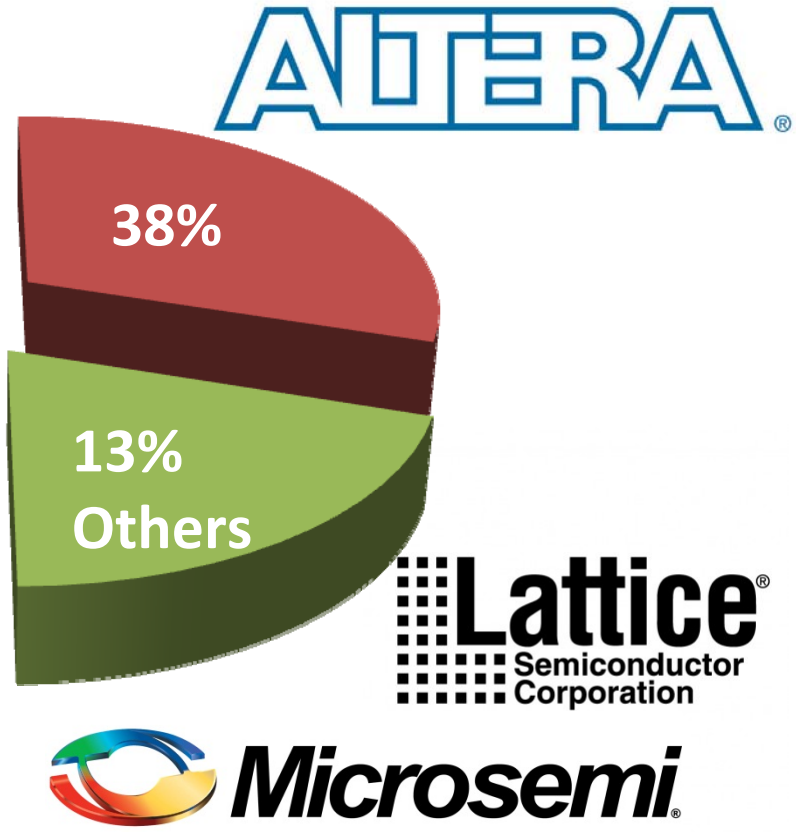
- Implementation is the process of generating an FPGA configuration file (bitstream) starting from a synthesized circuit

- It includes:
  - Map: fitting the design into the resources of the target platform
  - Place and route: placing and interconnecting logic elements satisfying timing specifications
  - Bitstream generation: creating the device configuration file

- Implementation constraints are used to direct the implementation tools about mapping, placement, timing etc.

Manufacturers and market shares





* Xilinx Investor Overview, 2015

Xilinx ISE Design Suite

- ISE Design Suite is a design environment from Xilinx for the development of CPLD and FPGA based systems.

- ISE is an integrated environment for simulation and synthesis:
  - Simulation: by ISIM (ISE Simulator), with support for external tools (ModelSim, NC-Sim, VCS);
  - Synthesis: by XST (Xilinx Synthesis Technology), with support for external tools (Precision RTL, Synplify)

- Quartus II development environment is the Altera equivalent of ISE

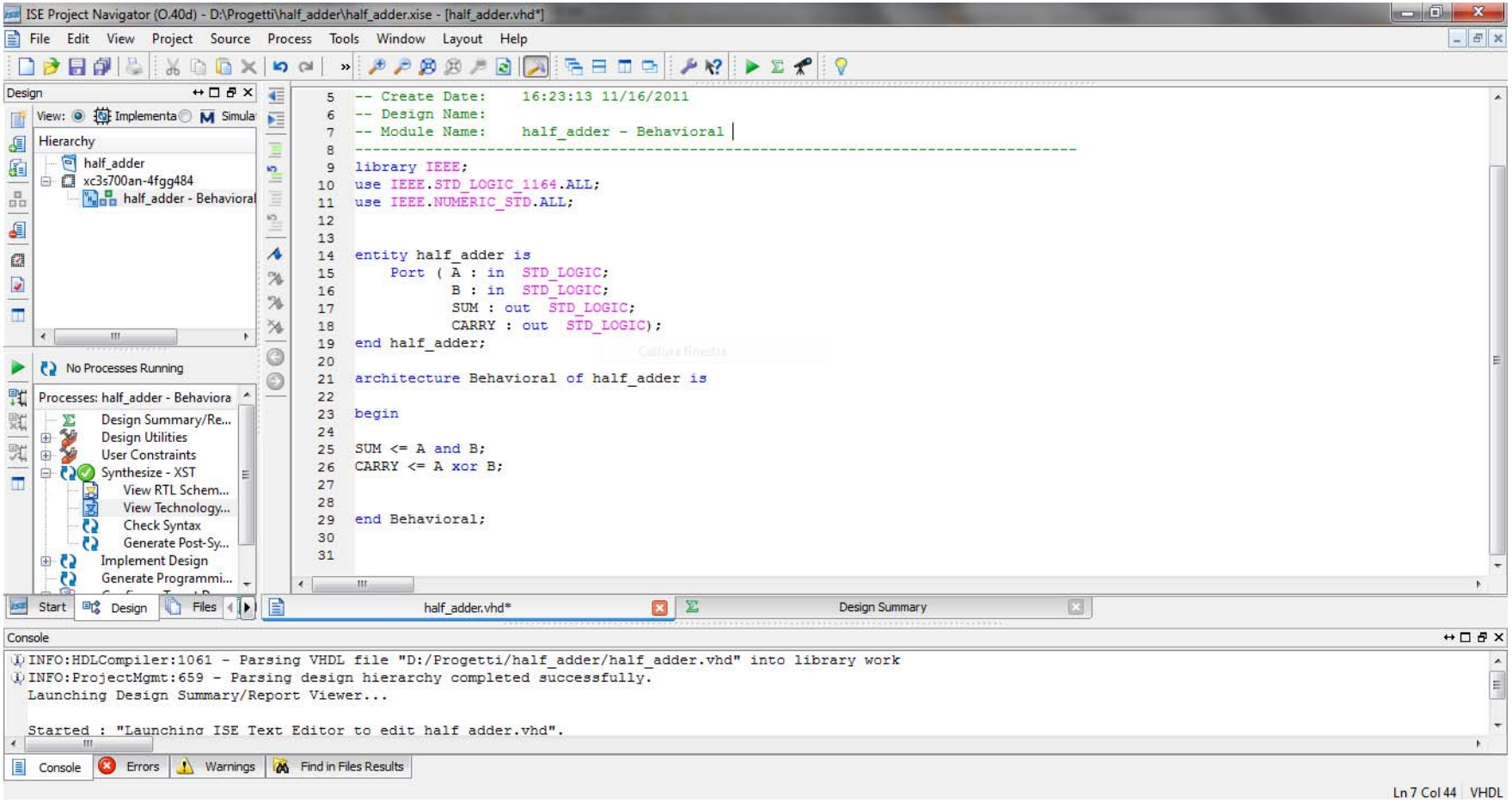
ISE Design Suite - Tools

- Basic tools: editor (graphical environment as an alternative), simulation tool, synthesis tool
- Embedded system development: rapid development of System-on-Chip using the Embedded Design Kit
- Management Tools: Effective management of the design flow through PlanAhead
- Other utilities and tools: DSP design using System Generator, analysis tools for performance or occupied area, etc..

Project Navigator

- The Project Navigator is the central environment of ISE Design Suite
- It is the entry point for coding, simulation and synthesis
- It also allows a consistent and integrated management of the other software tools

Project Navigator



Embedded Development Kit (EDK)

- EDK is a system-on-chip design environment

- It includes:
  - XPS: hardware design environment (microprocessor systems' design, also supporting the development of multicore architectures)
  - SDK: software development environment

- Altera Alternative : Nios II Embedded Design Suite (EDS);

System Generator

- System Generator is a development environment for FPGA-based digital signal processing architectures
  - Simulink-based tool, supports the use of all Simulink's features
  - Large number of DSP blocks immediately available
  - Possibility of exploiting Matlab to generate vectors of stimuli

- System Generator also supports hardware co-simulation

- Altera also provides a similar tool (DSP Builder)

Vivado Design Suite

- Since 2012 Xilinx discontinued ISE Design Suite in favor of Vivado Design Suite.

- Vivado is “a ground-up rewrite and re-thinking of the entire Xilinx design flow”
  - IP Integrator
  - Simulator
  - High-Level Synthesis

- We will return on this…

IP Cores

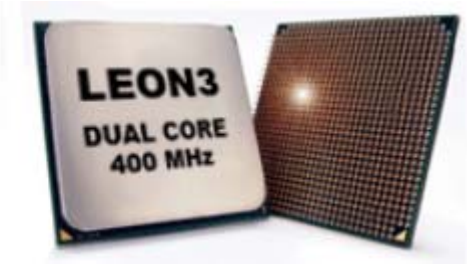
- An advantage of the use of manufacture's IDEs is the availability of IP Cores
  - Free (as in beer)
  - Ready to use
  - Optimized

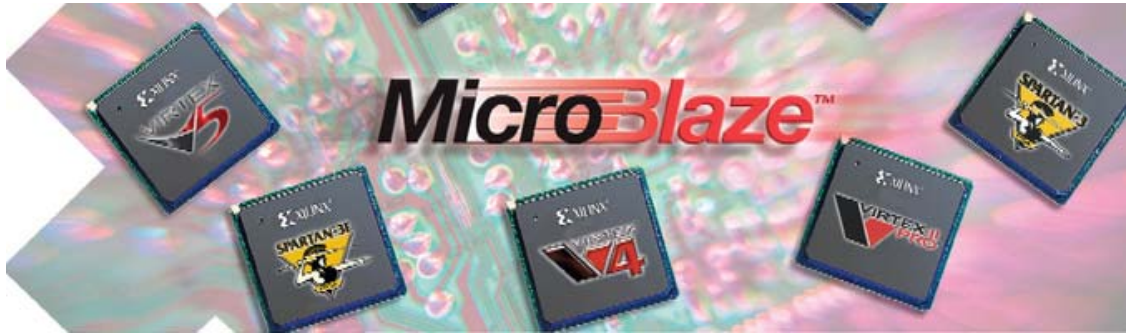
- Xilinx provides various IP cores inside of EDK and a dedicated signal processing core library (Core Generator)

- IP Cores are provided also by third parties
  - OpenCores.org provides a large number of IP Cores for many application areas (free of charge, registration needed)

Embedded (Soft)-Processors

LEON3
Synthesizable
processor





Embedded Processors

- Xilinx provides two types of processors for the development of embedded systems:
  - Hard-processor: PowerPC (available in various models of Xilinx FPGAs) and ARM based processors, available on the Zynq-7000 devices
  - Soft-processors: MicroBlaze, PicoBlaze

- PowerPC and MicroBlaze are directly supported within the Embedded Development Kit

PicoBlaze

- PicoBlaze is actually more a low-end, programmable state machine than a programmable microprocessor
  - KCPSM, Constant (K) Coded Programmable State Machine
- It is often used within the System Generator as datapath control element
- 8-bit RISC architecture
- Programmable assembly (max 1024 instructions programs)

References

- ANSI/IEEE Std 1076
- Frank Vahid and Tony Givargis, *Embedded System Design: A Unified Hardware/Software Introduction*. John Wiley & Sons, 2002
- Peter J. Ashenden, *The Designer's Guide to VHDL*, Morgan Kaufmann Publishers, 2008
- Volnei A. Pedroni, *Circuit Design and Simulation with VHDL - 2nd Edition*, MIT Press, 2010
- Pong P. Chu, *RTL Hardware Design using VHDL*, Wiley, 2006
- Pong P. Chu, *FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version*, Wiley, 2008

Free & open resources

- vhdl.org
- OpenCores: open source hardware resource
- Vendors websites: Xilinx (forum recommended), Altera
- GHDL: GPL VHDL Simulator
- ActiveHDL: simulator and various tutorials
- Cobham Gaisler: Leon 3 processor

Thank you!