Introduction to Digital Design in Verilog

Franco Fummi, Michele Lora
Stefano Spellini, Sebastiano Gaiardelli

UNIVERSITÀ
di VERONA
Department
of ENGINEERING FOR
INNOVATION MEDICINE

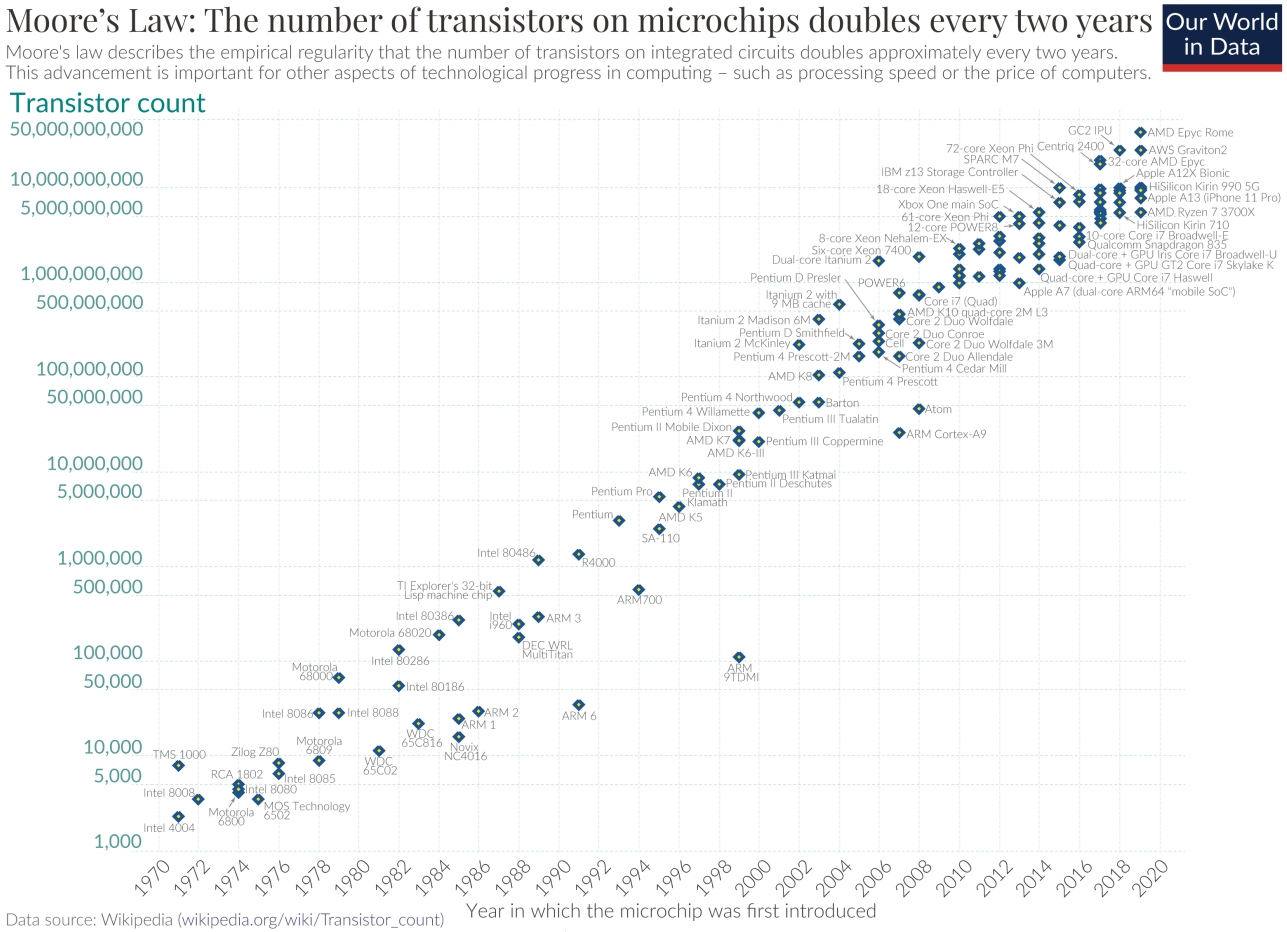
Cyber-Physical & IoT Systems Design

Introduction to EDA

A brief history of Electronic Design Automation

VLSI Circuits

- Very Large Scale Integrated Circuits
  - Many circuit components and wiring manufactured simultaneously into a compact, reliable and inexpensive chip



Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)
OurWorldinData.org – Research and data to make progress against the world's largest problems.

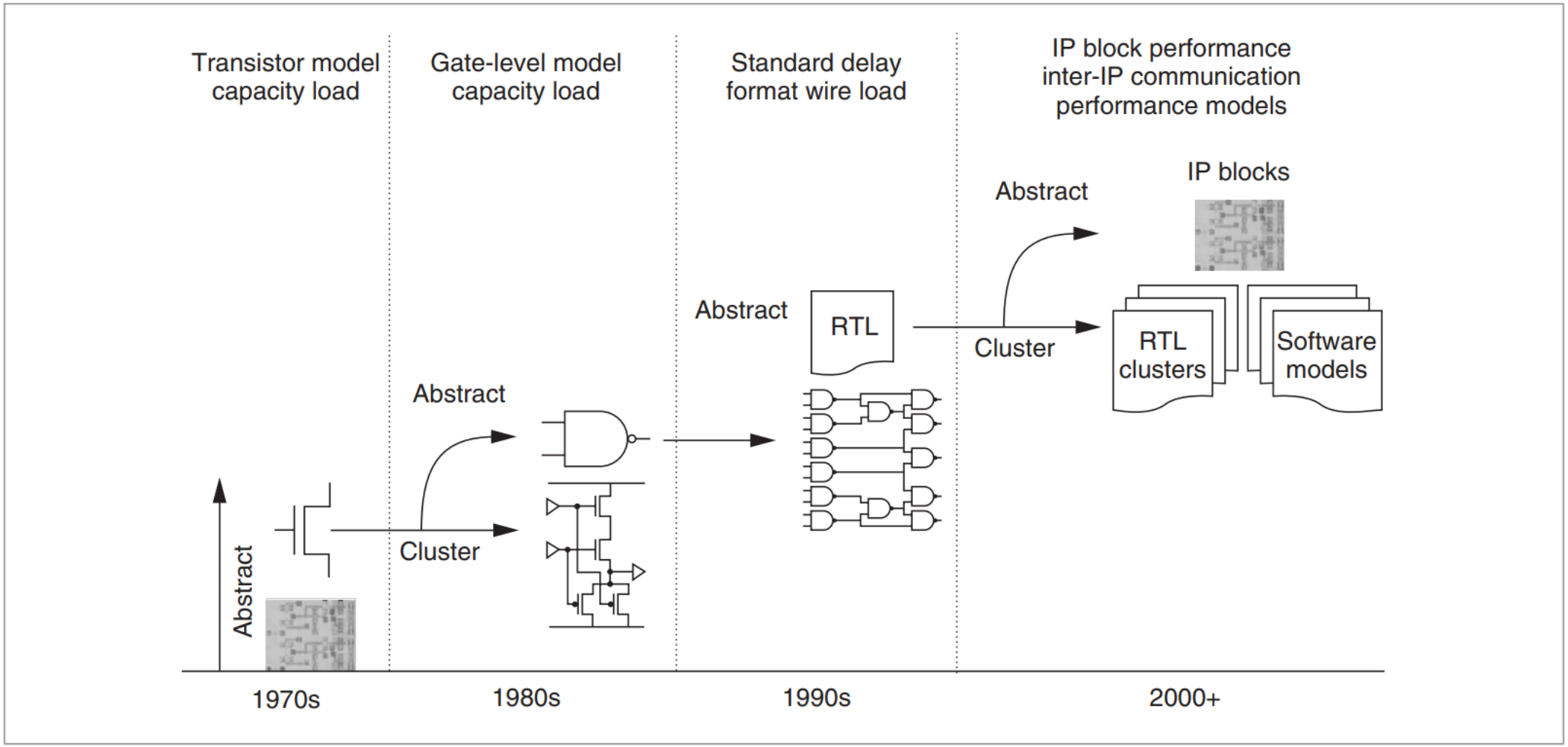
Electronic Design Automation

- Moore's Law pushes IC industry to grow exponentially
  - Exponential complexity in the design of Integrate Circuits

- **Electronic Design Automation**: the tools to tame such complexity
  - Mathematics
  - Algorithms and Heuristics
  - Methodologies
  - Automatic Tools

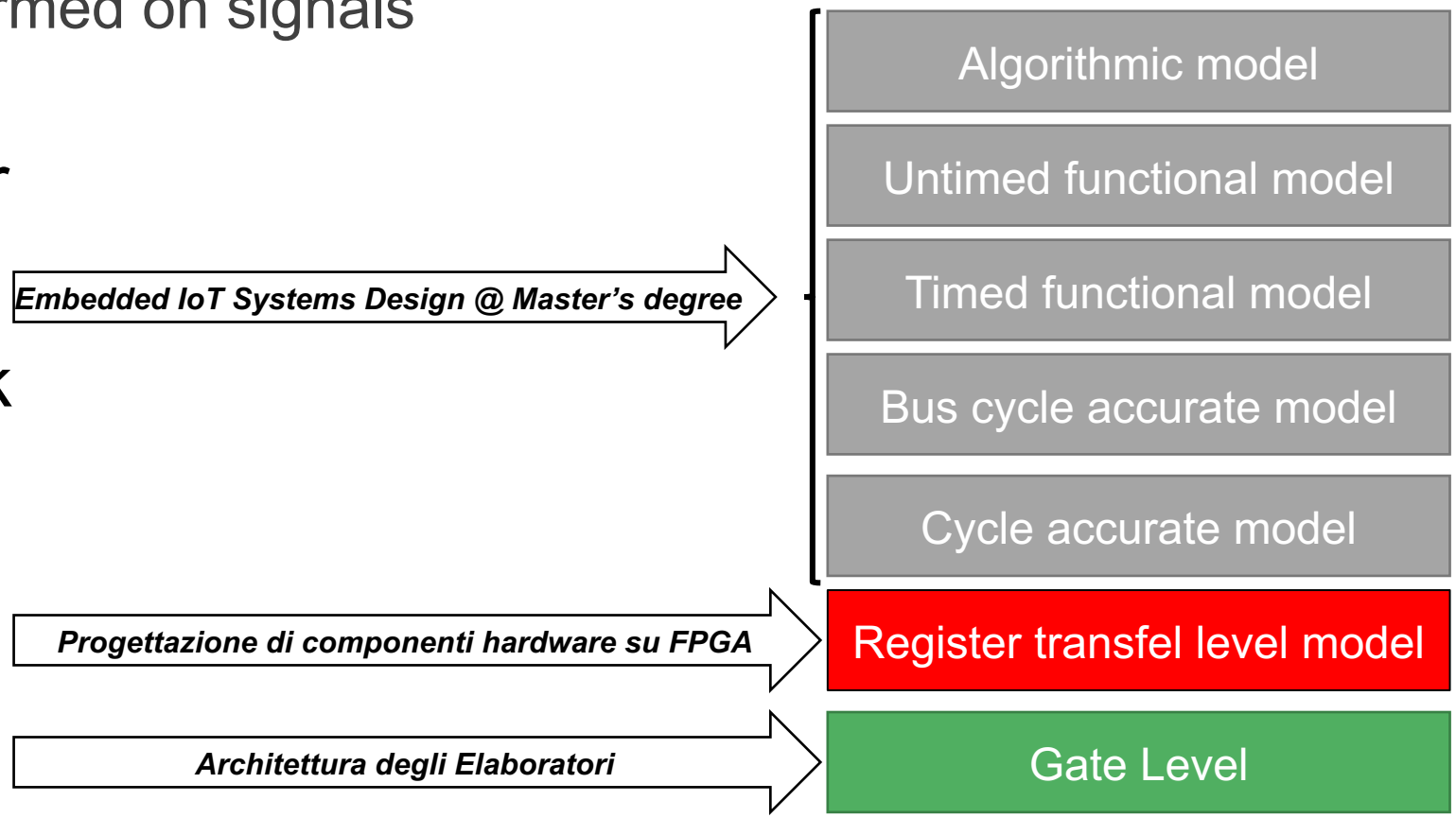
- **Automatic design and implementation of electronic systems**
  - Digital, Analog, and Mixed-Signal
  - Along all the steps of the design process
  - From the requirements specification to the system deployment

Abstraction and Composition



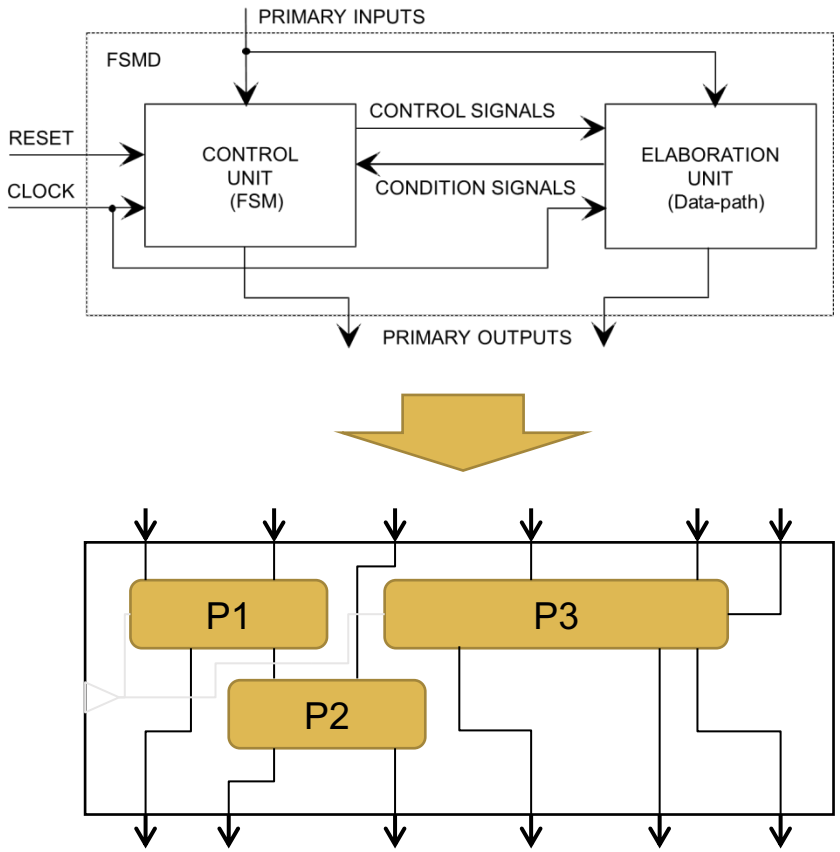
Register Transfer Level (RTL)

- Describe the operation of a synchronous digital circuit
  - Flow of signals, *i.e.*, transfer of data between registers
  - Logical operations performed on signals
- Managed by a controller
- Synchronized by a clock



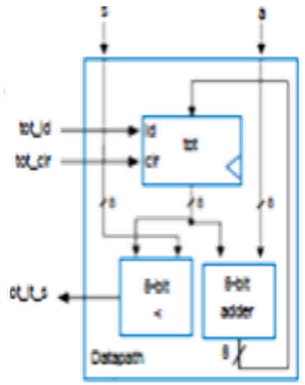
RTL: Controller and Datapath

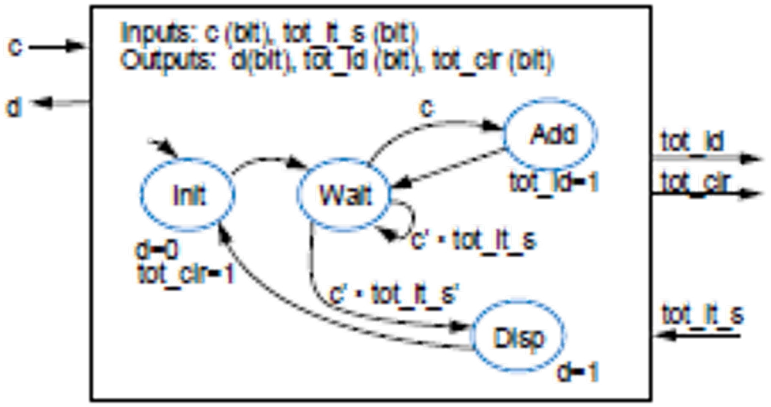
- RTL models the interconnection between
  - The **controller**: Finite State Machine (FSM) governing the behavior of the system
    - control/condition signals must be identified
  - The **data-path**: implementation of the component operations as combinatorial elements.
    - *E.g.*, registers, multiplexers, operators…
- **Synthesis**: automatic translation into a set of registers and library components

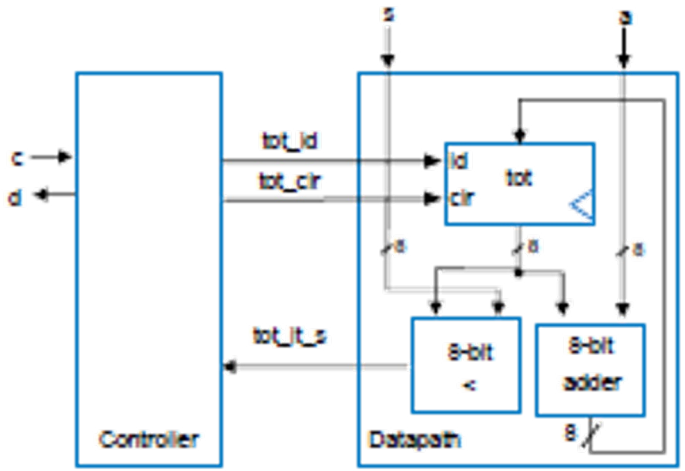


How to build an RTL module

1. Describe the behavior with a high-level FSM
2. Determine the datapath
   - What shall I do when the FSM is in a certain state?
3. Connections between datapath and FSM
4. Determine the controller FSM





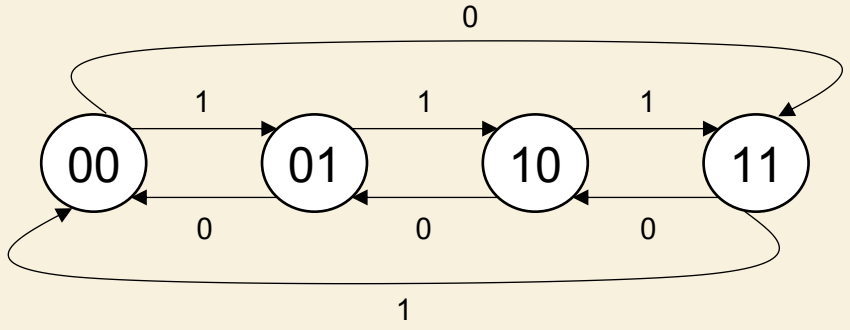


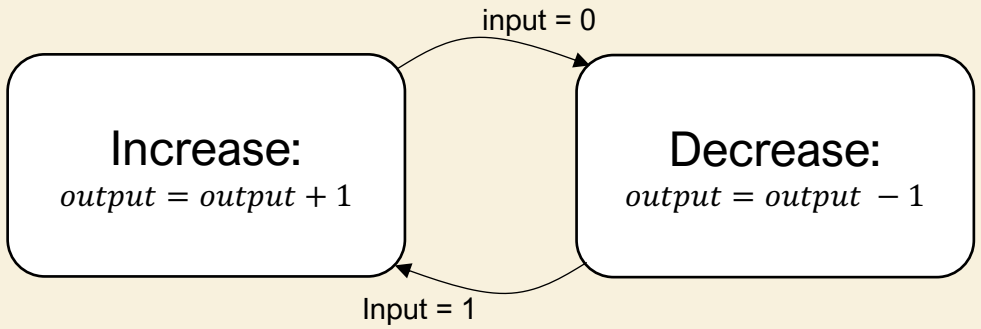
EFSM: Defining Controller and Datapath

- RTL models as FSM
  - the number of states may be impossible to represent

- Extended Finite State Machine (EFSM)
  - Generalization of FSMs
  - Allows to reduce the number of explicit states
    - A more compact, yet equivalent, representation for RTL models
    - **Datapath** operations expressed within **states**, **control** represented as **transitions**

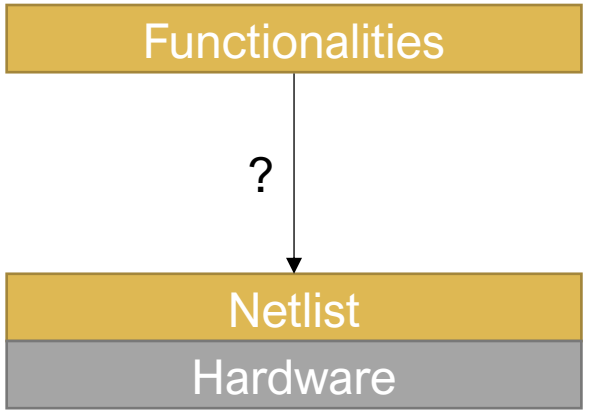
Example: two-bit counter with overflow





How to specify an RTL module

- SIS and Espresso:
  - Boolean functions expressed as Sum of Products
  - Net-list of gates
  - Truth table
  - State-transition table



- Necessity for a more abstract specification mechanism: ***Hardware Description Langages (HDLs)***!

A brief history of HDLs

- **‘60s-’70s**: Pioneer work on languages to specify hardware modules
  - Oriented to Programmable Logic Devices
- **Early ‘80s**: U.S. DoD starts to develop a language to simulate and document Application Specific Integrated Circuits (ASICs)
  - Shifting toward a VLSI perspective
- **1983**: The DoD introduces the VHSIC Hardware Description Language (**VHDL**)
- **1985**: Gateway Design System Corporation introduces **Verilog**
- **Late ’80s**: tools to automatically synthesis hardware from HDLs
- **1994:** Synopsys introduces behavioral synthesis from VHDL and Verilog
- **1998:** High-level synthesis: synthesis from SystemC and C code

Hardware Description Languages

- Characteristics:
  - Standard for all stages of hardware design
  - Different levels of abstraction
  - Mixture of description at different levels
  - Support the specification of both data and control paths

- Advantages:
  - Describes “what the hardware should do”
    - Executable functions as documentation
  - Behavior separated from the implementation

HDLs comparison

VDHL

- ADA-Like
- **Extensible** types and simulation engine
- Entities with **multiple architectures**
- Gate-level, dataflow, and behavioral modeling
- Synthesizable subset
- **Harder** to learn and use
- Extension (only) to Analog Mixed Signals

Verilog

- C-like syntax
- **Built-in** types and logic representations
- Modules with just **one implementation**
- Gate-level, dataflow, and behavioral modeling
- Synthesizable subset
- **Easy** to learn and use
- **Extension** to Analog Mixed Signals and System-level design
- **Faster** simulation

Hardware modeling in Verilog

Why Verilog

- **Advantages**:
  - In 2021 seems to be the HDL of choice of many companies
  - More people are familiar with C-like syntax
  - Simple module/port syntax
    - Familiar way to organize hierarchical building blocks and manage complexity
  - Well suited for both verification and synthesis
  - Easily extensible to other design domain
    - System-level design
    - Analog, and Analog-Mixed Signal systems design

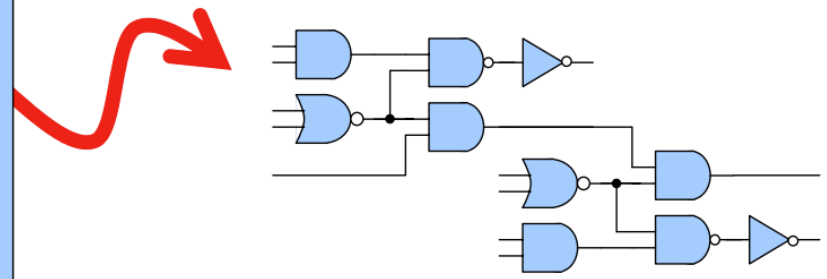
- Disadvantages
  - Easier syntax do not prevent errors and ugly code
  - C syntax may cause beginners to assume C semantics!

HDL modeling ≠ Software Programming

- Software Programming Language
  - Language which can be translated into machine instructions and then executed on a computer
- Hardware Description Language
  - Language with syntactic and semantic support for **modeling** the temporal behavior and spatial structure of hardware
    - **Concurrency is always implied by the physics of hardware components**!

```
module foo(clk,xi,yi,done);
  input [15:0] xi,yi;
  output done;

  always @(posedge clk)
    begin:
      if (!done) begin
        if (x == y) cd <= x;
        else (x > y) x <= x - y;
      end
    end
endmodule
```



Verilog Abstraction Layers

- Behavioral
  - Module's high-level algorithm is implemented with little concern for the actual hardware
- Dataflow
  - Module is implemented by specifying how data flows between registers
- Gate-Level
  - Module is implemented in terms of concrete logic gates (AND, OR, NOT) and their interconnections

Designers can create lower-level models from the higher-level models either manually or automatically.

The process of automatically generating a gate-level model from either a dataflow or a behavioral model is called:

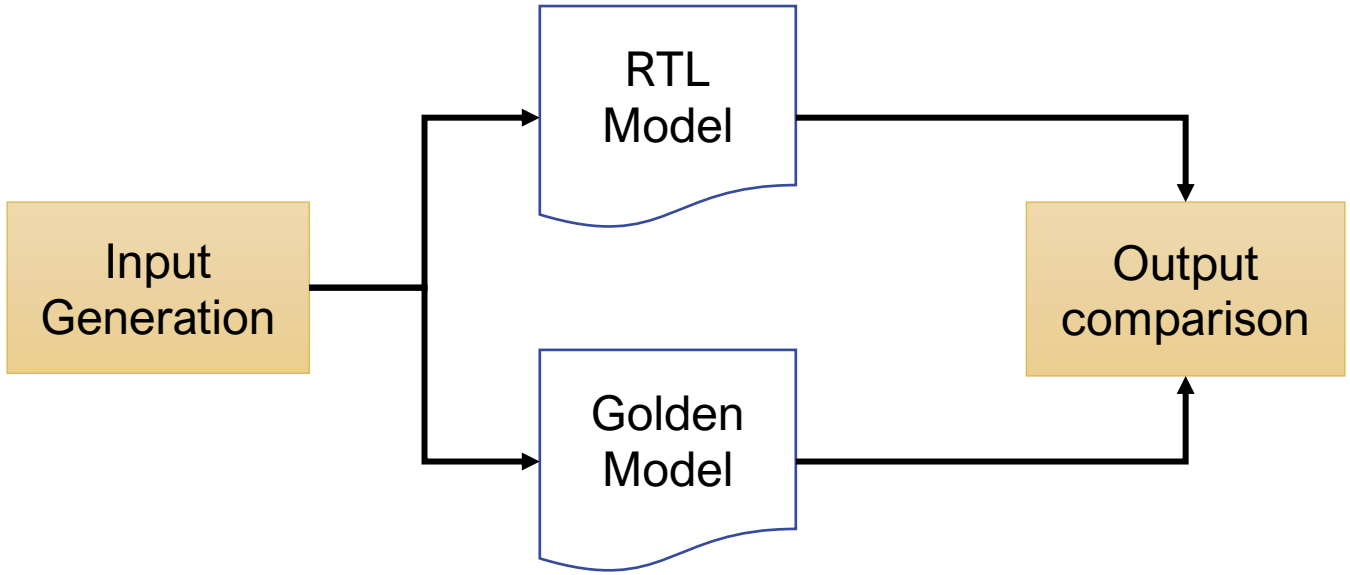
Logic Synthesis

System validation

Simulation of hardware models

Validation with Simulation

- Simulation is a very powerful (yet not complete) mean of validation
  - Pro: faster than (complete) formal methods
  - Cons: incomplete validation of correctness, false negative are highly probable
- Validation strategy: comparison between design and golden model
  - Golden model: a (usually more abstract) model that is (reasonably) correct



Discrete Event simulation

- HDLs simulation is event based
  - Events rule the evolution of the system
  - Blocks and processes are executed in response to the occurrence of events
  - Discrete Event Simulation is governed by a **dynamic scheduler**

- Scheduler
  - Exploits events to prepare the ***ready queue***
    - Processes that can be run at a given time
  - Chooses the process that must run at a given time
  - Whenever a process creates any events, sensitive processes and blocks are added to the ready queue
    - Executed at a later time

Verilog Simulation Semantic

```
Initialize the simulation
```

Initialization

```
while (there are events) {
     if (no active events) {
          if (there are inactive events) {
               activate all inactive events;
          }
          else if (there are nonblocking assign update events) {
               activate all nonblocking assign update events;
          }
          else if (there are monitor events) {
               activate all monitor events;
          }
          else
          {
               advance T to the next event time;
               activate all inactive events for time T;
          }
     }

     E = any active event;
     if (E is an update event) {
          update the modified object;
          add evaluation events for sensitive processes to event queue;
     }
     else
     {    /* shall be an evaluation event */
          evaluate the process;
          add update events to the event queue;
     }
}
```

Delta notification

Timed notification

Update

Simulation Loop

Testbench

- The testbench is the module in charge of governing the testing of the device being designed
  - It generates the input vectors
  - It analyzes the output values
  - It may be implemented in different ways:
    - Ad-hoc script languages
    - Verilog modules governing the timing evolution of the input and outputs
    - Automatic generation of test patterns
  - It should be as complete as possible
    - Maximize the coverage

Timing simulation

- Simulation depends on how time is defined
- The `` `timescale <time_unit>/<time_precision> `` compiler directive specifies:
  - **Time unit**: measurement of delays and simulation time
  - **Precision**: specifies how delay values are rounded before simulation

- # operator: Statement execution delayed by a fixed-time period

```
#num statement;      // Delay num time unit before executing
```

```
// Example
`timescale 10ns/1ns

reg a;
...
#1 a=0; //After 1 time_unit (10ns) assign 0 to a
#5 a=1; //After 5 time_unit (50ns) assign 1 to a
```

Testbench for the square root

```
`timescale 1ns / 1ps

module tb();

reg clk, rst,number_isready;
reg [31:0] number_port;

wire result_isready;
wire [31:0] result_port;

root dut(.clk(clk),
         .rst(rst),
         .number_isready(number_isready),
         .number_port(number_port),
         .result_isready(result_isready),
         .result_port(result_port)
         );

initial
begin
   clk <= 1'b0;
   rst <= #10 1'b0;
   rst <= #30 1'b1;
   number_isready <= #40 1'b1;
   number_isready <= #60 1'b0;
   number_port <= #40 32'd25;
end

always #10 clk <= !clk;

endmodule
```