



The Abdus Salam  
International Centre for Theoretical Physics



**310/1780-18**

**ICTP-INFN Advanced Training Course on  
FPGA and VHDL for Hardware Simulation and Synthesis  
27 November - 22 December 2006**

---

*Design Methodology Tools*

**Jorgen CHRISTIANSEN  
PH-ED  
CERN  
CH-1221 Geneva 23  
SWITZERLAND**

---

***These lecture notes are intended only for distribution to participants***

# IC design methodology and related tools

Jorgen Christiansen  
CERN

# IC design methodology and Tools

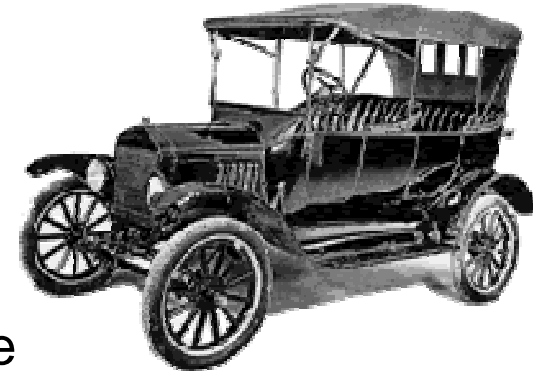
---

- Dealing with complexity – design methodology
- Low power
- Design styles
- Design tools:
  - Schematics
  - Layout,
  - Simulation,
  - HDL: VHDL and Verilog,
  - Place and route,
  - Synthesis
  - Design verification

# Evolution (revolution) of IC design

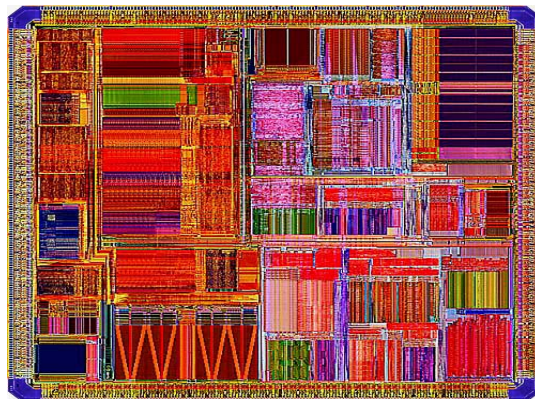
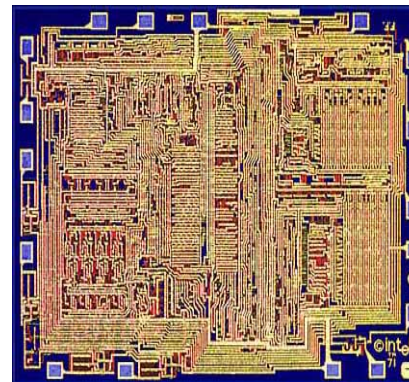
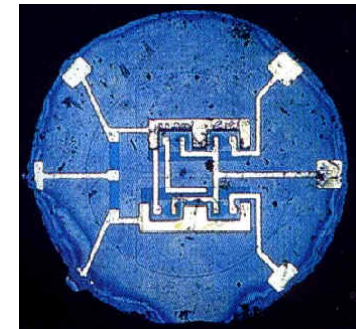
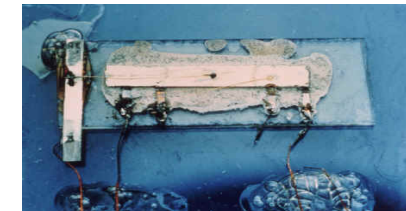
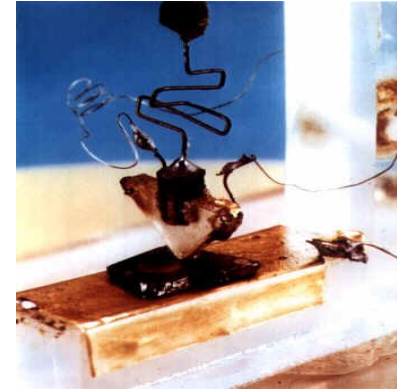
---

- If cars had the same rate of improvement as Integrated circuits a car today could:
  - Drive at the speed of light
  - Drive years on one single tank of petrol
  - Transport a whole city in one car
- The micro electronics industry only stays well alive (continuous growth) because of this rapid progress. (performance doubles every ~2 years)
  - This rate of progress **MUST** be maintained to keep IC industry in good shape.
  - The life time of a technology generation is ~5 years
- Production is cheap in large quantities because of lithographic processing (“like printing stamps”)
- Design is complicated and very expensive (design mistakes costs lot of time and money)



# How to put together millions of transistors and make it work ?

- Well chosen design methodologies
- Well chosen architectures
- Extensive use of power full CAE tools
- Strict design management
- Well chosen testing methodologies
- Design re-use
- One can NOT use same design methodologies and architectures when complexity increases orders of magnitude



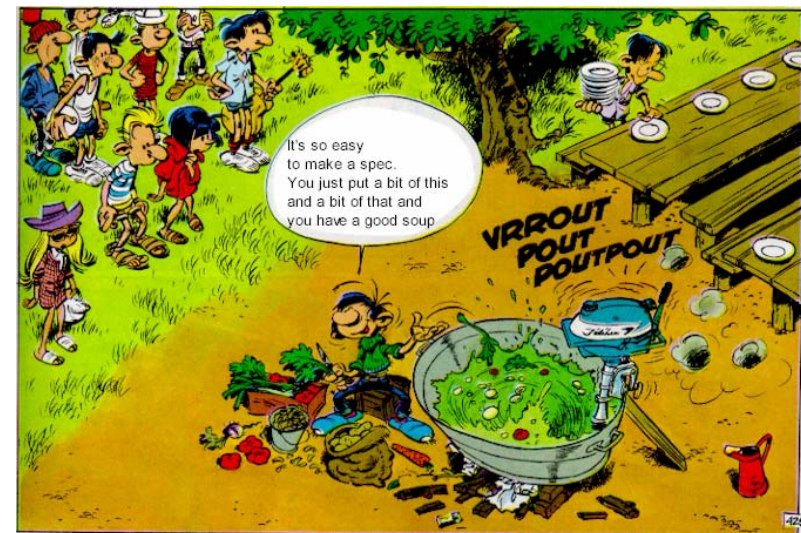
# Design Methodology

---

- Specification
- Trade-off's
- Design domains - abstraction level
- Top-down - Bottom up
- Schematic based
- Synthesis based
- Getting it right – Simulation and verification
- Lower power
- Design styles

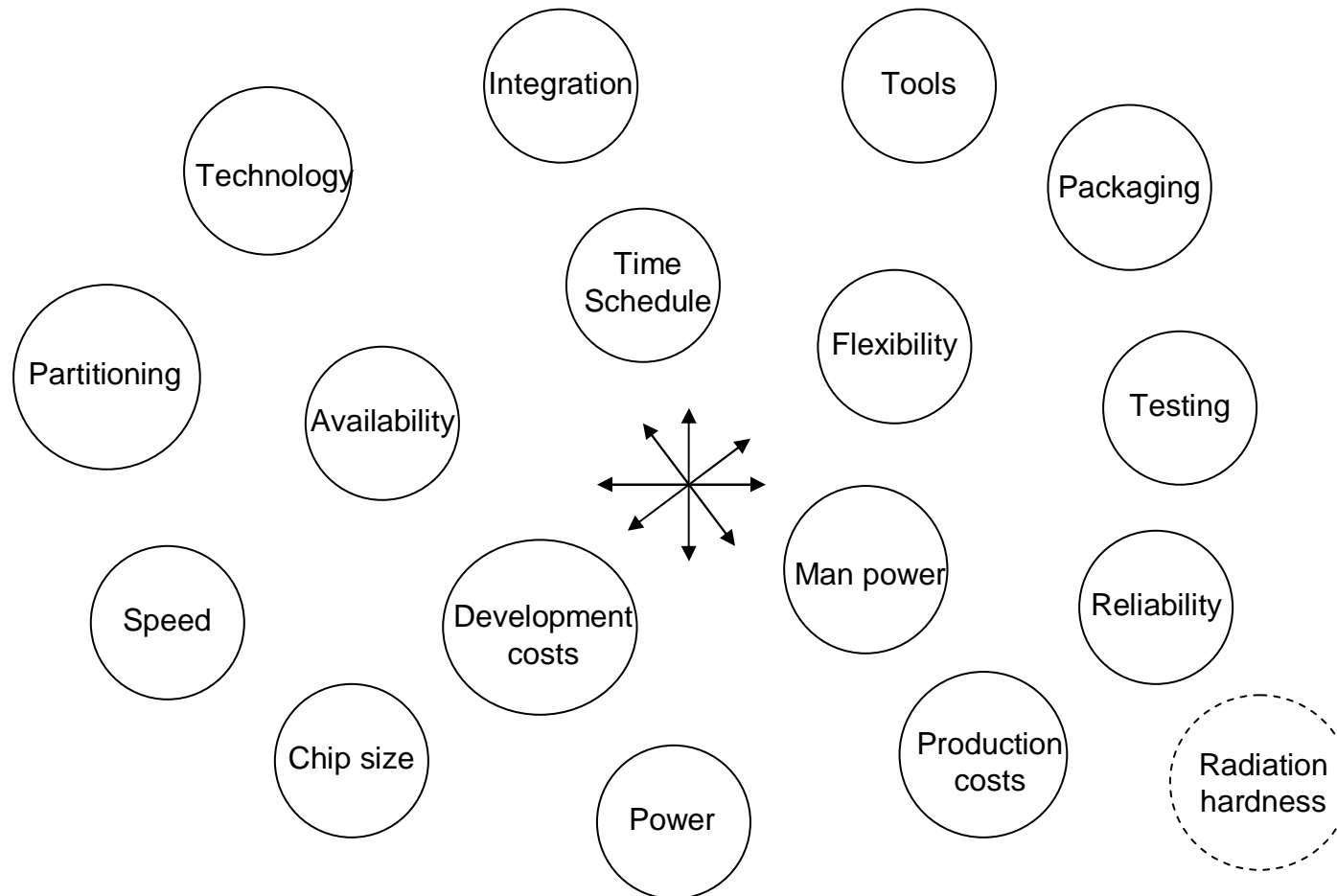
# Specification

- A specification of what to construct is the first major step.
- Compromise between what is wanted and what can be made  
Requires extensive experience to define best compromise
- A detailed specification must be agreed upon with the system people. Major changes during design will result in significant delays.
- Requirements must be considered at many levels  
System, sub-system, Board, Hybrid, IC
- Specifications can (must) be verified by system simulations.
- Specification is 1/4 - 1/3 of total IC project !.



# Trade offs

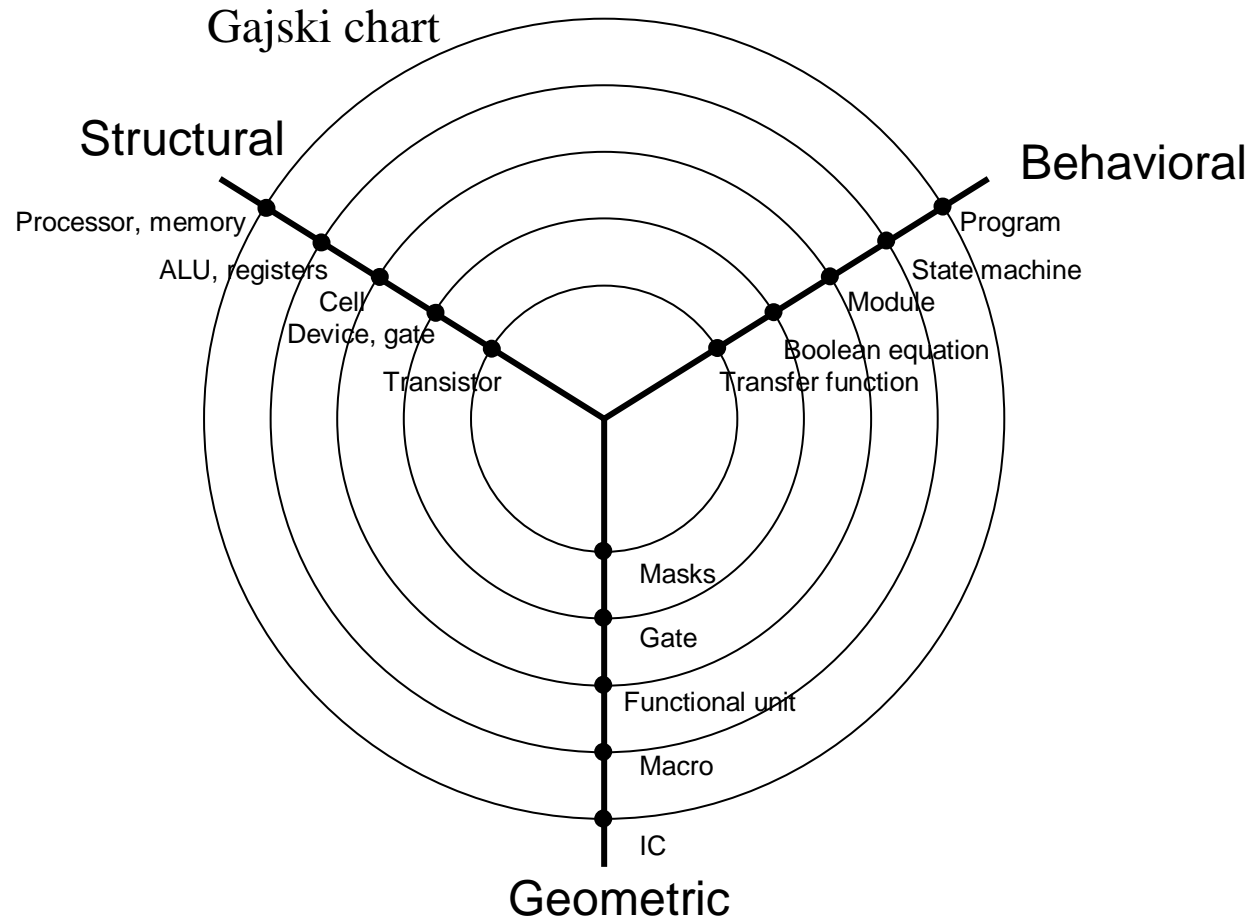
---



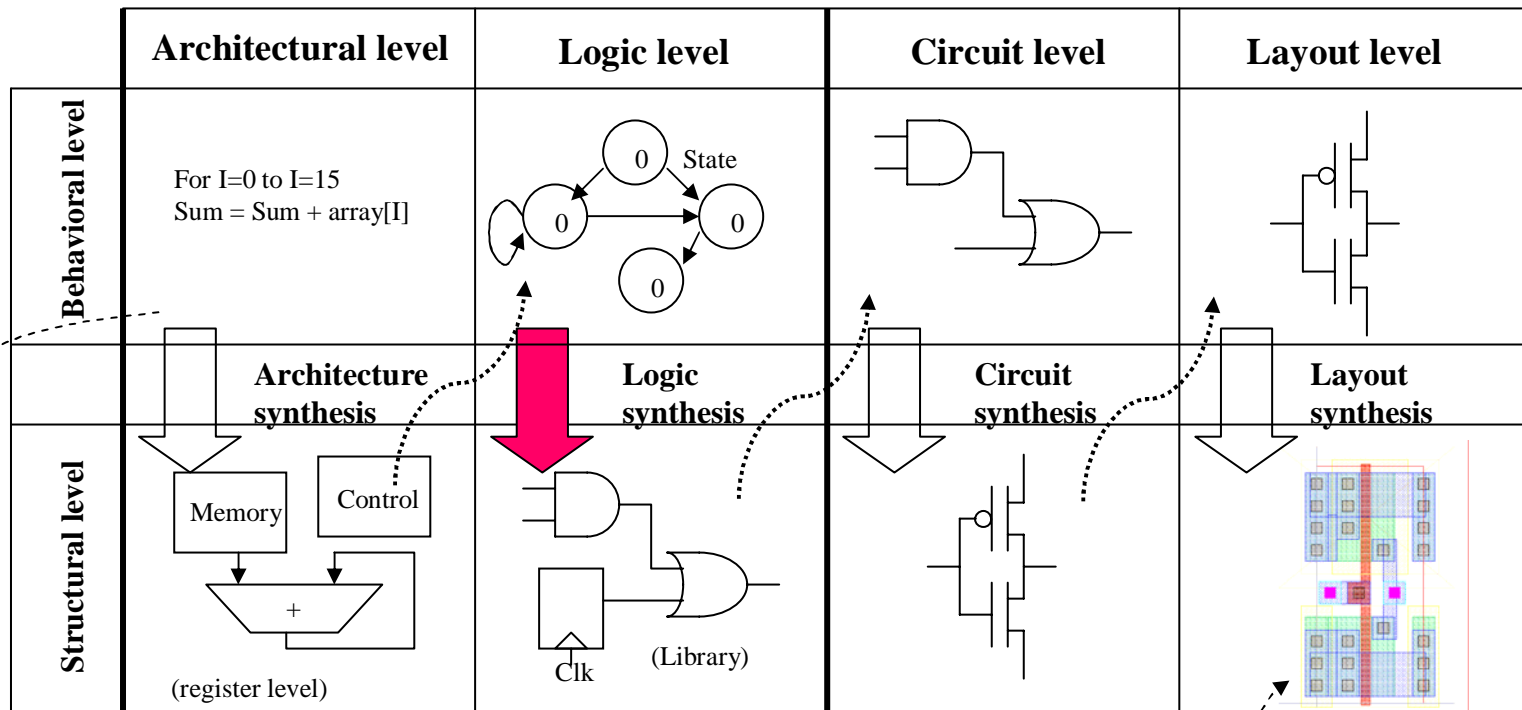


# Design domains

---



# Abstraction levels and synthesis

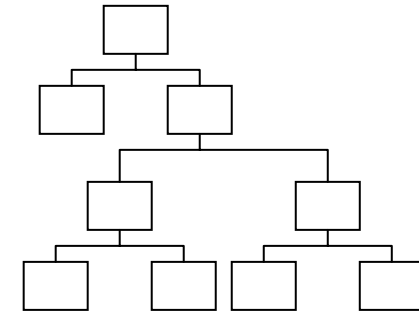


Silicon compilation (not a big success)

# Top - down design

---

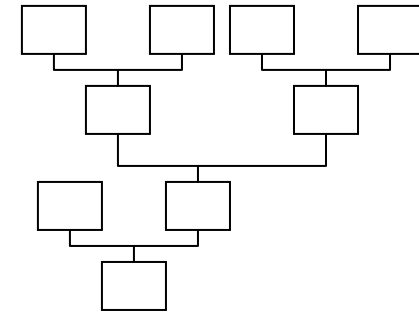
- Choice of algorithm (optimization)
- Choice of architecture (optimization)
- Definition of functional modules
- Definition of design **hierarchy**
- Split up in small boxes - split up in small boxes - split up in small boxes
- Define required units ( adders, state machine, etc.)
- Floor-planning
- Map into chosen technology  
(synthesis, schematic, layout)  
(change algorithms or architecture if speed or chip size problems)
- Behavioral simulation tools



# Bottom - up

---

- Build gates in given technology
- Build basic units using gates
- Build generic modules of use
- Put modules together
- Hope that you arrived at some reasonable architecture
- Gate level simulation tools



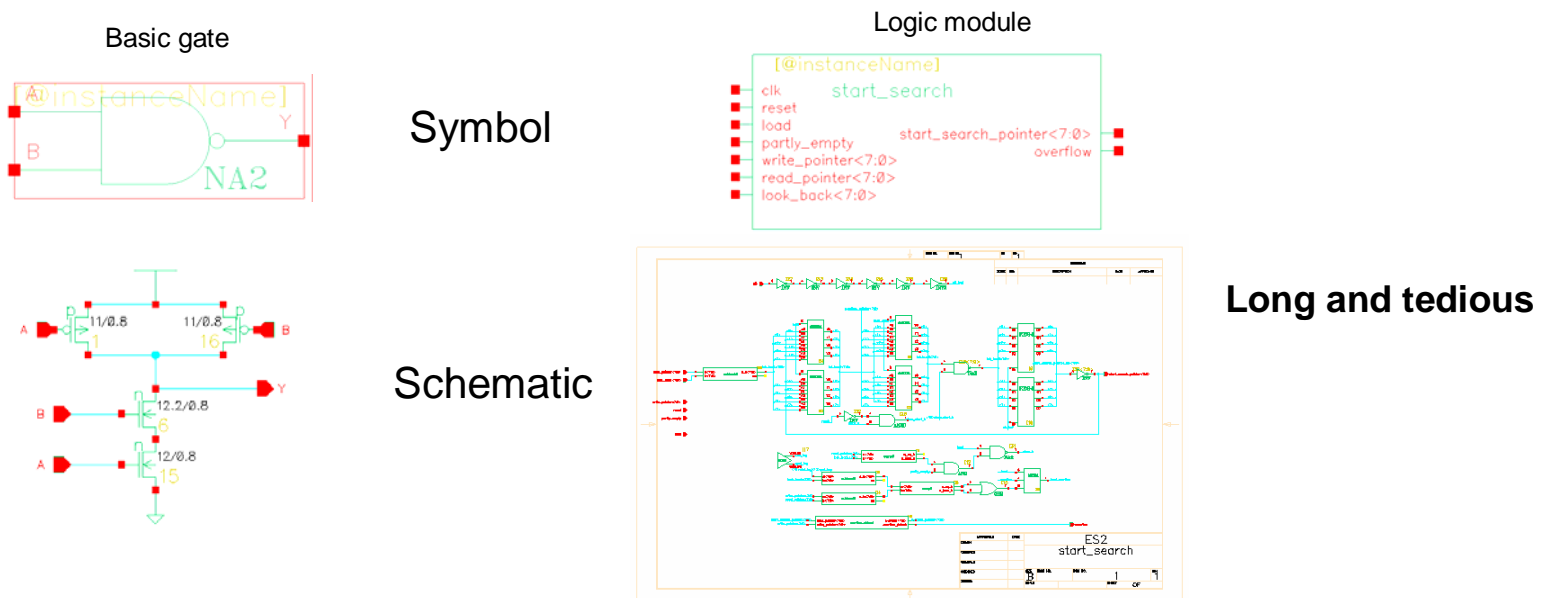
*Old fashioned design methodology a la discrete logic*

Comment by one of the main designers of a Pentium processor

**The design was made in a typical top - down , bottom - up ,  
inside - out design methodology**

# Schematic based

- Symbol of module defines interface
- Schematic of module defines function
- Top - down: Make first symbol and then schematic
- Bottom - up: Make first Schematic and then symbol

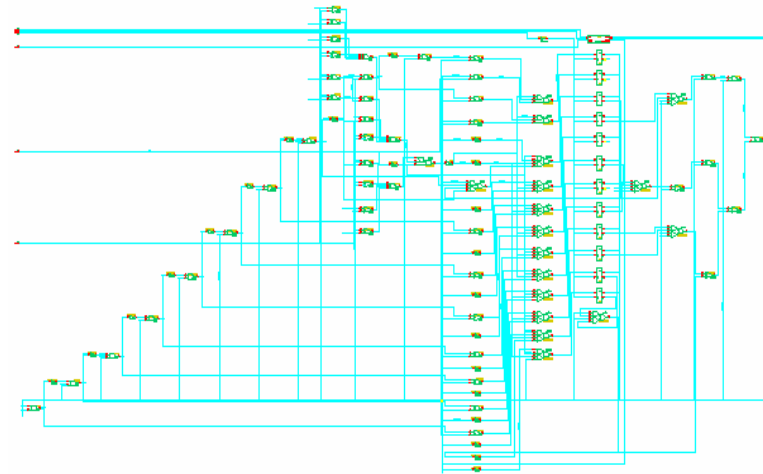


# Synthesis based

---

- Define modules and their behavior in a proper language (also used for simulation)
- Use synthesis tools to generate schematics (netlists)

```
always @(posedge clk)
  begin
    if (set) coarse <= #(test.ff_delay) offset;
    else if (coarse == count_roll_over)
      coarse <= #(test.ff_delay) 0;
    else coarse <= #(test.ff_delay) coarse + 1;
  end
```



**Only possible way to make designs with millions of gates**

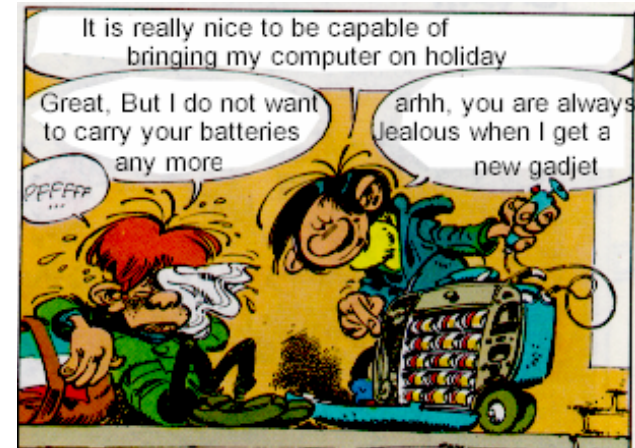
# Getting it right - Simulation

---

- Simulate the design at all levels (transistor, gate, system)
- Analog simulator (SPICE) for transistor level
- Digital gate level simulator for gate based design
- Mixed mode simulation of mixed analog-digital design
- Behavioral simulation at system/module level (Verilog, VHDL)
- All functions must be simulated and verified.
- Worst case data must be used to verify timing
- Worst - Typical - Best case conditions must be verified  
Process variations, Temperature range, Power supply voltage  
Factor two variation to both sides ( speed:  $\frac{1}{2}$  : 1 : 2)
- Use programming approach to verify large set of functions  
(not looking at waveform displays)

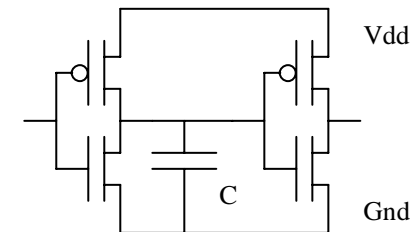
# Low power design

- Low power design gets increasingly important:  
Gate count increasing > increasing power.  
Clock frequency increasing > increasing power.  
Packaging problems for high power devices.  
Portable equipment working on battery.



- Where does power go:
  - 1: Charging and dis-charging of capacitance: Switching nodes
  - 2: Short circuit current: Both N and P MOS conducting during transition
  - 3: Leakage currents: MOS transistors (switch) does not turn completely off

- The power density of modern ICs are at the same level as the hot plate on your stove and is approaching the power density seen in a nuclear reactor !



$$P = N_{\text{switch}} * f * C * V_{\text{dd}}^2 + N_{\text{switch}} * f * E_{\text{short}} + N * I_{\text{leak}} * V_{\text{dd}}$$

$\swarrow$   
 $K * V_{\text{dd}}^2$



# Decrease power

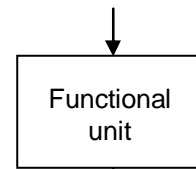
- Lower Vdd:

5v > 2.5v gives a factor 4 !

New technologies use lower Vdd because of risk of gate-oxide break-down and hot electron effect.

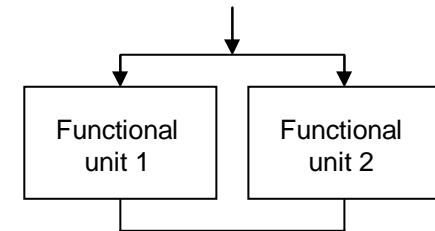
- Lower Vdd and duplicate hardware

One functional unit:  
frequency = 1  
Vdd = 1



$$P = 1 * 1^2 = 1$$

Two functional units:  
frequency = 1/2  
Vdd = 1/2 (optimistic)

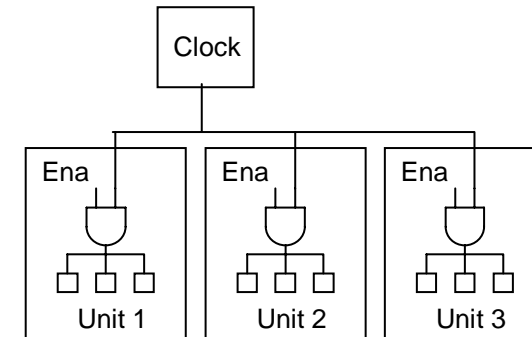


$$P = 2 * 1/2 * (1/2)^2 = 1/4$$

- Lower number of switching nodes

The clock signal often consumes 50% of total power

Clock gating



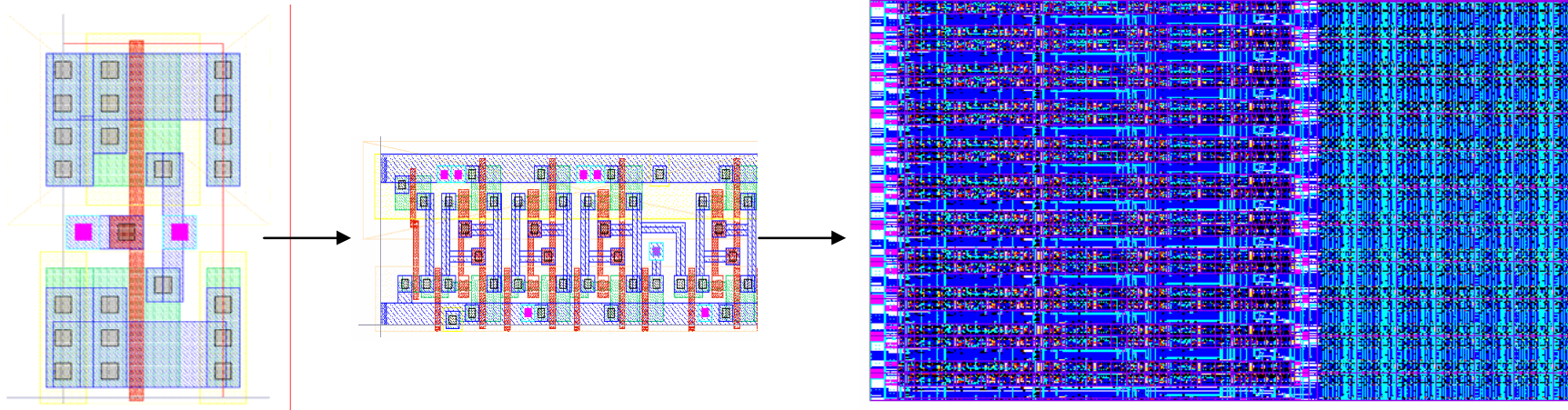
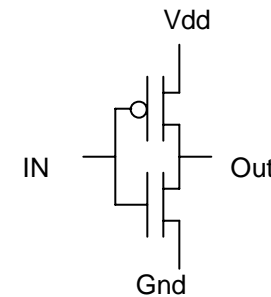
# Design styles

---

- Full custom
- Standard cell
- Gate-array
- IP-blocks - Macro-cell
- “FPGA”
- Combinations

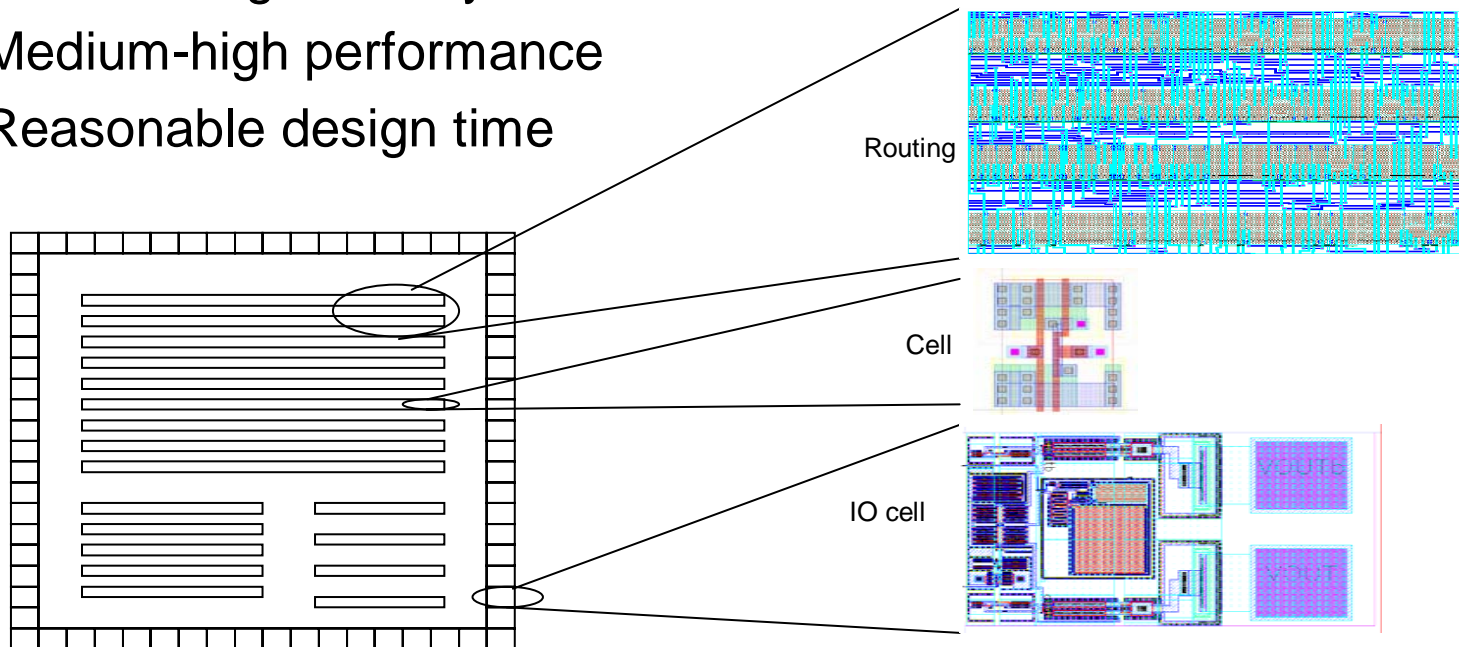
# Full custom

- Hand drawn geometry
- All layers customized
- Digital and analog
- Simulation at transistor level (analog)
- High density
- High performance
- Loong design time



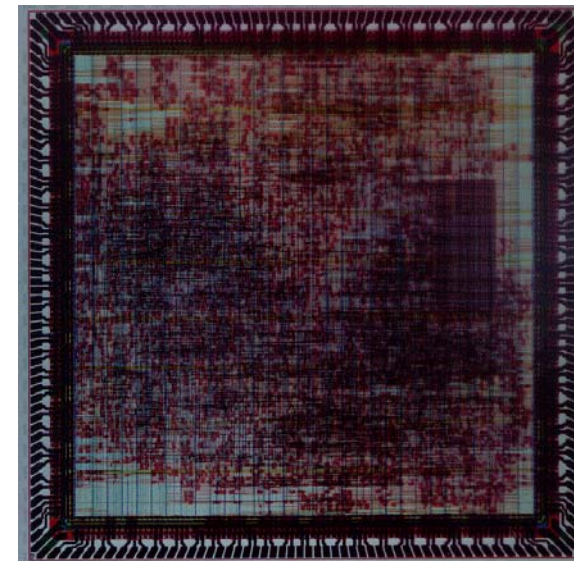
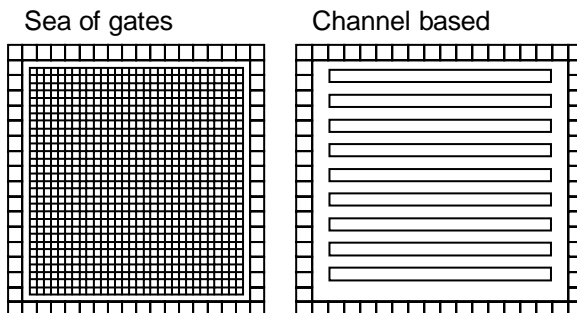
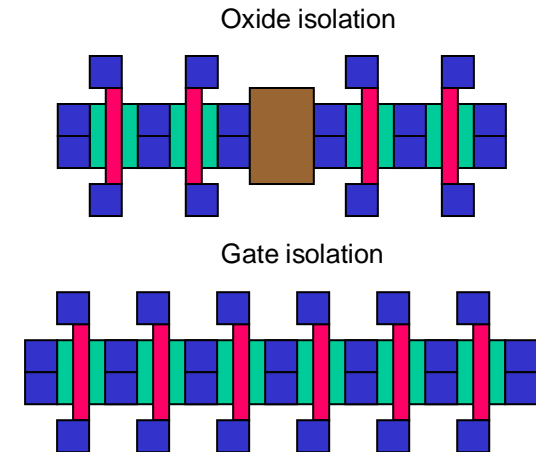
# Standard cells

- Standard cells organized in rows (and, or, flip-flops, etc.)
- Cells made as full custom by vendor (not user).
- All layers customized
- Digital with possibility of special analog cells.
- Simulation at gate level (digital)
- Medium- high density
- Medium-high performance
- Reasonable design time



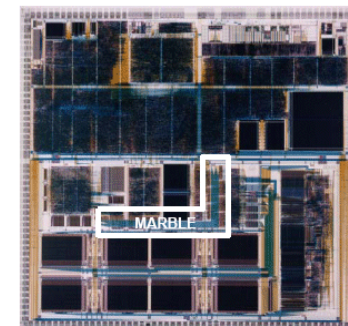
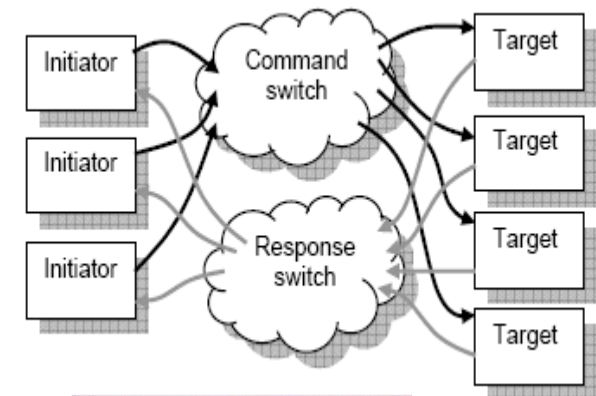
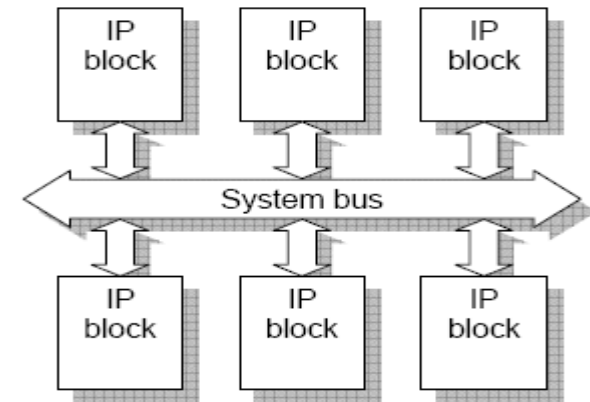
# Gate-array

- Predefined transistors connected via metal
- Two types: Channel based, Sea of gates
- Only metal layers customized
- Fixed array sizes (normally 5-10 different)
- Digital cells in library (and, or, flip-flops, etc.)
- Simulation at gate level (digital)
- Medium density
- Medium performance
- Reasonable design time



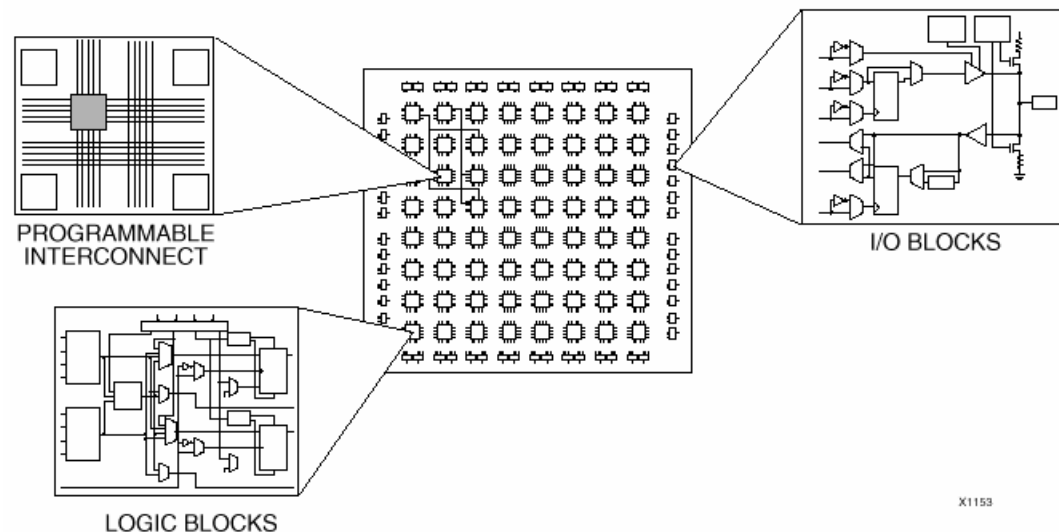
# IP blocks

- Functional blocks from specialized companies
  - Rely on external expertise to reduce design time
  - Quite a large selection now available
- Hard blocks
  - Full custom by vendor, **Technology dependent**
  - All layers customized: High density, High performance
  - Digital and analog (ADC)
  - Simulation at behavioral or gate level (digital)
  - Memories, ADC, DAC, PLL, CPU, etc.
- Soft blocks
  - Synthesizable HDL model, **Technology independent**
  - User to synthesize into given technology using available libraries and perform himself timing and design verification
  - Digital blocks: DSP, processor, MPEG, etc.
- Use standard on-chip busses (like on boards)
  - New trend: on-chip networks (like computer networks)
- “System On Chip”: **SOC**



# FPGA = Field Programmable Gate Array

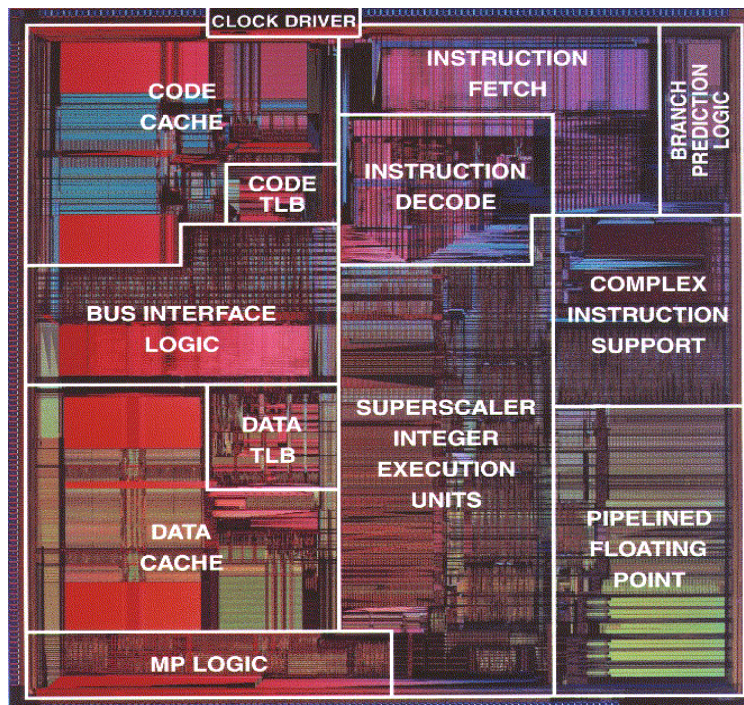
- Programmable logic blocks, Digital only
- Programmable connections between logic blocks (and memories)
- No layers customized (standard devices)
- Low - medium performance (up to a few hundred MHz)
- Low - medium density (~1M gates)
  - Be careful with how FPGA companies quote gate equivalents !
- Hardwired blocks can increase performance significantly
  - Memory, CPU, DSP, high speed serial, PLL, DLL, etc.
- Programmable: SRAM, EEROM, Flash, Anti-fuse, etc
- Easy and quick design changes
- Cheap design tools
- Low development cost
- High device cost
- **NOT a real ASIC**  
(Application Specific Integrated Circuit)
- Not part of my lecture



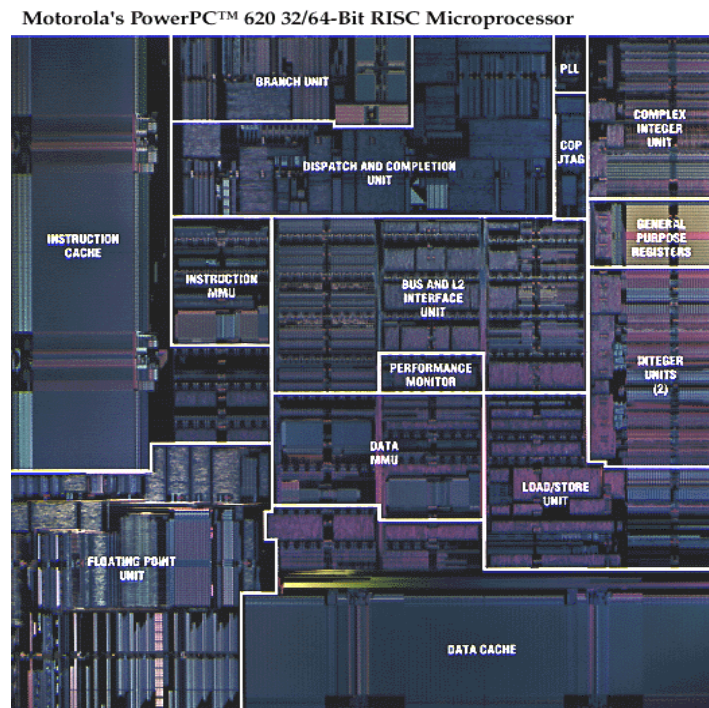
X1153

# High performance devices

- Mixture of full custom, standard cells and macro's
- Full custom for special blocks: Adder (data path), etc.
- Macro's for standard blocks: RAM, ROM, etc.
- Standard cells for non critical digital blocks



Pentium

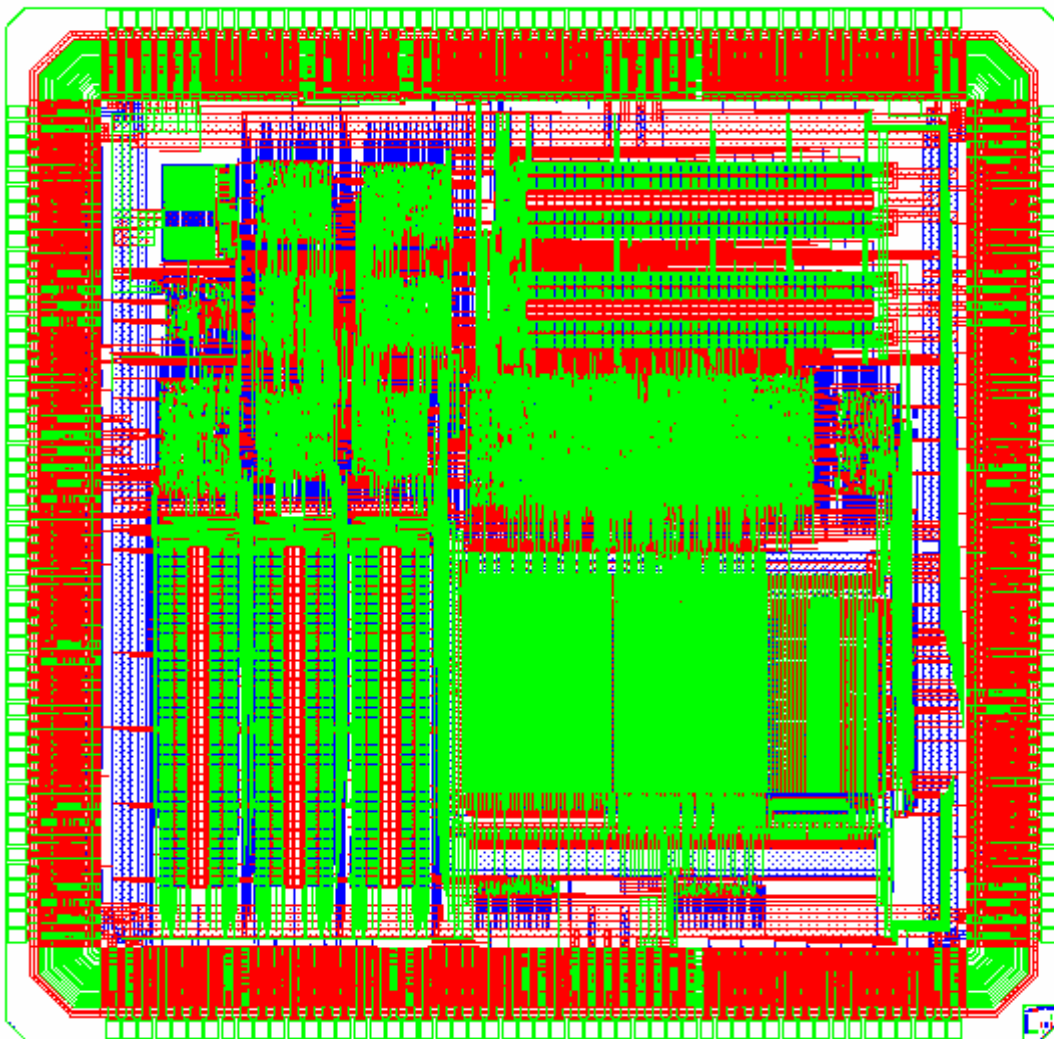


Power PC



# Example: High resolution TDC for HEP

---



High resolution TDC  
25 ps binning  
40 MHz external clock  
320 MHz internal TDC clock from PLL  
“Only” 1 million transistors  
0.25 um CMOS  
2 full custom macros  
5 memory macros  
50k Standard cells  
~5 man years design  
~2 man years test and design fix  
Total design price: ~1 million \$  
Production cost: 10\$/chip  
Production volume: ~50k chips  
Total production cost: 500k\$

# Tools for Cell development (Analog/digital)

---

- **Schematic entry** (transistor symbols)
- **Analog simulation** (SPICE models)
- **Layout** (layer definitions)
- **Design Rule Checking**, DRC ( design rules)
- **Extraction** (extraction rules and parameters)
- **Electrical Rule Checking**, ERC (ERC rules)
- **Layout Versus Schematic**, LVS ( LVS rules)
- Analog simulation.
- Characterization: delay, setup, hold, loading sensitivity,etc.
- Generation of digital simulation model with back annotation.
- Generation of synthesis model
- Generation of “black-box” for place & route

# Tools for Digital design

---

- **Behavioral simulation**
  - **Synthesis** (synthesis models)
  - Gate level simulation (gate models)
  - **Floor planning**
  - **Loading estimation** (loading estimation model)
  - Simulation/timing verification with estimated back-annotation
  - **Place and route** (place and route rules)
  - **Design Rule Check, DRC** (DRC rules)
  - Loading **extraction** (rules and parameters)
  - Simulation/**timing verification** with real back-annotation
  - Design export
  - Testing: **Test generation, Fault simulation, Vector translation**
- } Or direct **schematic entry**



---

- **Schematic**

- Drawing electrical circuit: Defines electrical hierarchy  
Defines electrical connections  
Defines circuit: transistors, resistors,,

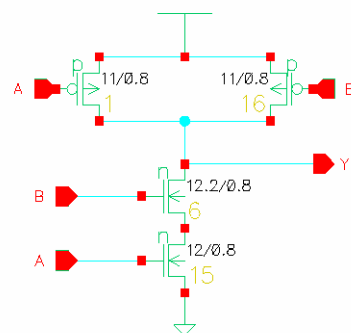
Requires good circuit design knowledge for analog design

Requires good logic design knowledge for digital design (boolean logic, state machines)

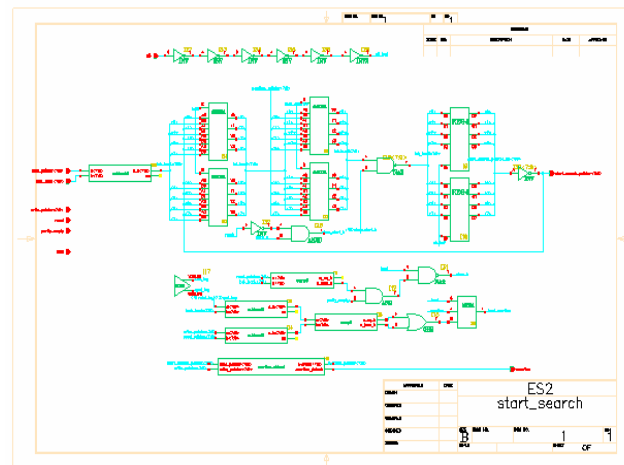
Gives good overview of design hierarchy

Significant amount of time used for manual optimization

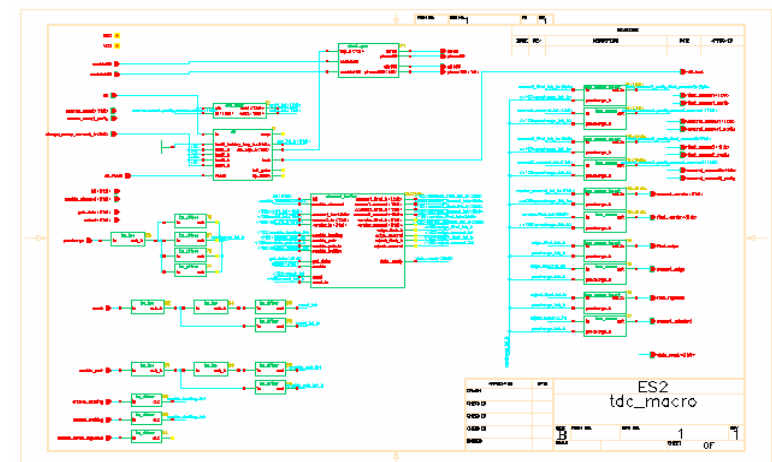
Transistor level



Gate level



Module level



---

- **Behavioral + Synthesis**

- Writing behavior (text):
  - Defines behavioral hierarchy
  - Defines algorithm
  - Defines architecture
- Synthesis tool required to map into gates
- Often integrated with graphical block diagram tool.

```
module add_and_mult( a,b,c, out)
input[31:0]  a,b,c;
output[31:0] out;
wire[31:0] internal_add;
```

```
adder32      add1(a,b, internal_add);
multiplier32 mult1( internal_add, c, out);
endmodule
```

```
assign #(test.logic_delay)
```

```
bsr_clk = ~(m_extest | m_sample | m_intest) | clk_dr,
```

```
bsr_shift = (m_extest | m_sample | m_intest) & shift_dr,;
```

```
always @(posedge clk)
```

```
begin
```

```
if (set) coarse <= #(test.ff_delay) offset;
```

```
else if (coarse == count_roll_over)
```

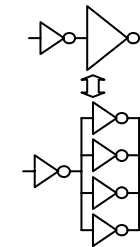
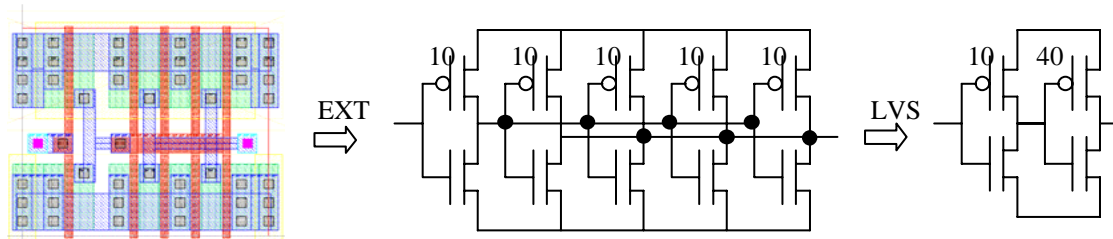
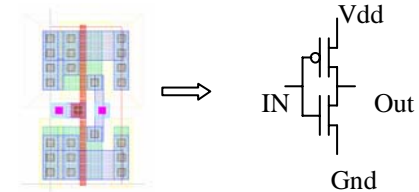
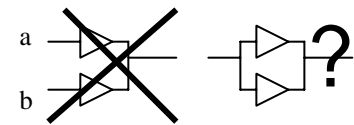
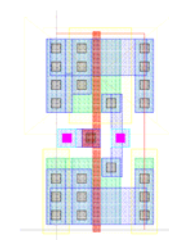
```
coarse <= #(test.ff_delay) 0;
```

```
else coarse <= #(test.ff_delay) coarse + 1;
```

```
end
```

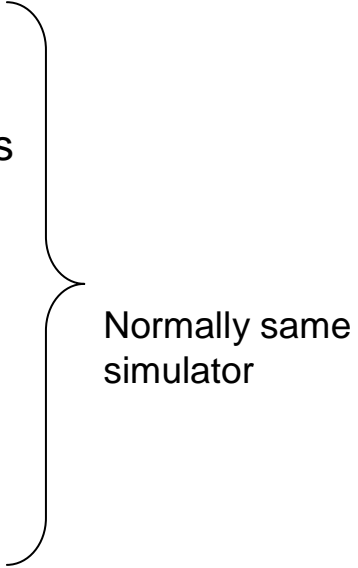
# Verification

- **Design Rule Check (DRC):**  
Checks geometrical shapes: width, length, spacing, overlap, etc.
- **Electrical rule check (ERC):**  
Checks electrical circuit: unconnected inputs  
shorted outputs  
correct power and ground connection
- **Extraction:**  
Extracts electrical circuit: transistors, connections, capacitance, resistance
- **Layout versus schematic (LVS):**  
Compares electrical circuits: transistors: parallel or serial  
(schematic and extracted layout)



# Simulation

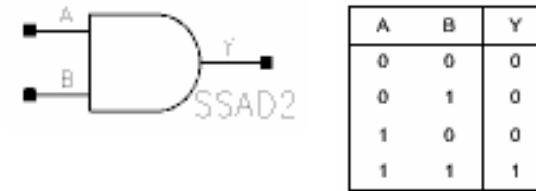
---

- **Simulates behavior of designed circuit**
    - Input: Models (transistor, gates, macro)  
Textual netlist (schematic, extracted layout, behavioral)  
User defined stimulus
    - Output: Circuit response (waveforms, patterns), Warnings
  - **Transistor level simulation using analog simulator (SPICE)**
    - Time domain
    - Frequency domain
    - Noise
  - **Gate level simulation using digital simulator**
    - Logic functionality
    - Timing: Operating frequency, delay, setup & hold violations  
Timing calculator needed to calculate delays from extracted parameters
  - **Behavioral simulation**
    - System and IC definition ( algorithm, architecture )
    - Partitioning
    - Complexity estimation
- 
- Normally same simulator



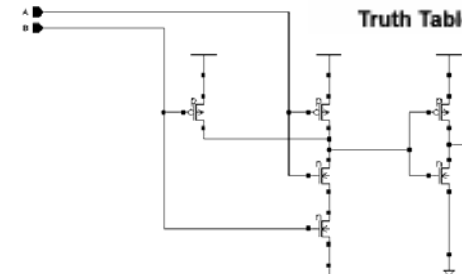
# Gate level models

- Border between transistor domain (analog) and digital domain
- Digital gate level models introduced to speed up digital simulation.
- Gate level model contains:
  - Logic behavior
  - Delays depending on: operating conditions, process, loading, signal slew rates
  - Setup and hold timing violation checks
- Gate level model parameters extracted from transistor level simulations and characterization of real gates.



A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Truth Table

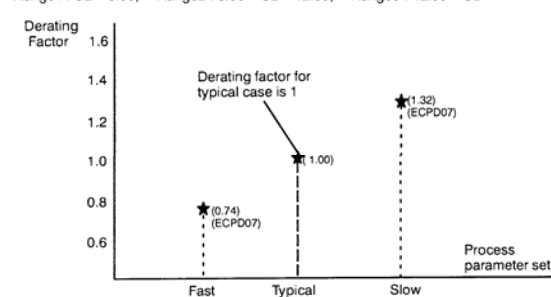
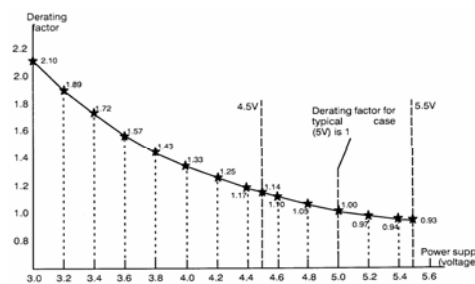
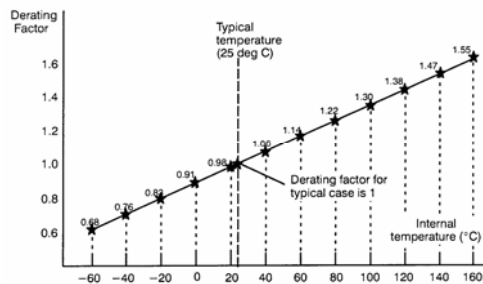


Schematic

SSAD2 Switching Characteristics  
 [Delays for typical process, 25.00°C, 2.50V, when  $t_{RQ}$  and  $t_F = 0.40ns$ ] [SL(Standard Load) = 0.0081 pF]

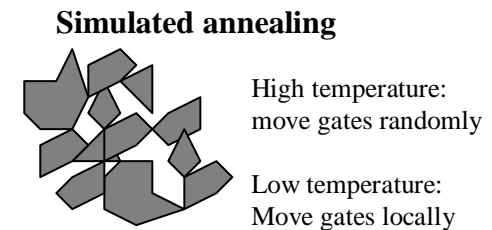
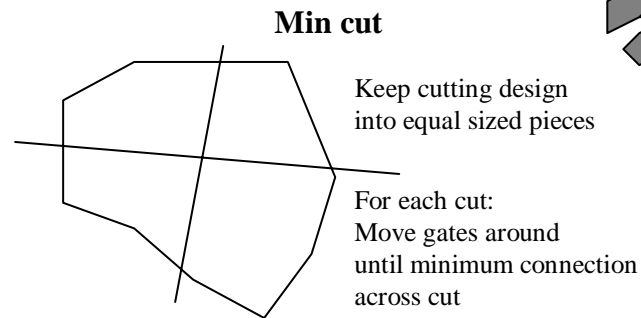
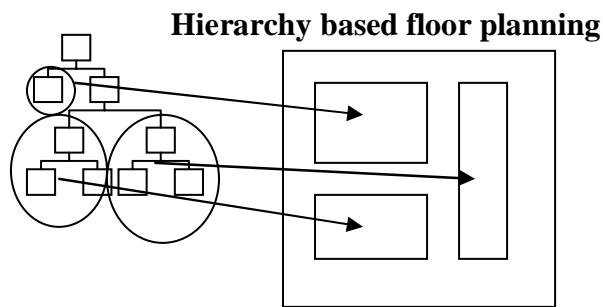
Path	Parameter	Delay [ns] SL = 2.00	Delay Equations [ns]		
			Range1*	Range2*	Range3*
A to Y	tPLH	0.13	$0.08 + 0.021 \cdot SL$	$0.09 + 0.019 \cdot SL$	$0.09 + 0.019 \cdot SL$
	tPHL	0.16	$0.14 + 0.012 \cdot SL$	$0.15 + 0.009 \cdot SL$	$0.15 + 0.008 \cdot SL$
	tR	0.13	$0.05 + 0.039 \cdot SL$	$0.05 + 0.041 \cdot SL$	$0.04 + 0.041 \cdot SL$
	tF	0.08	$0.05 + 0.014 \cdot SL$	$0.05 + 0.013 \cdot SL$	$0.04 + 0.015 \cdot SL$
B to Y	tPLH	0.12	$0.07 + 0.021 \cdot SL$	$0.08 + 0.019 \cdot SL$	$0.08 + 0.019 \cdot SL$
	tPHL	0.18	$0.16 + 0.012 \cdot SL$	$0.17 + 0.009 \cdot SL$	$0.17 + 0.008 \cdot SL$
	tR	0.13	$0.05 + 0.038 \cdot SL$	$0.05 + 0.041 \cdot SL$	$0.04 + 0.041 \cdot SL$
	tF	0.07	$0.04 + 0.015 \cdot SL$	$0.05 + 0.014 \cdot SL$	$0.04 + 0.015 \cdot SL$

\*Range1 : SL < 3.00, \*Range2 : 3.00 ≤ SL ≤ 12.00, \*Range3 : 12.00 < SL



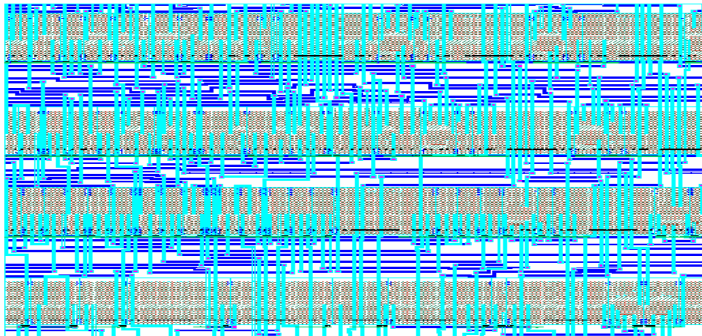
# Place and Route

- Generates final chip from gate level netlist
  - Goals: Minimum chip size  
Maximum chip speed.
- Placement:
  - Placing all gates to minimize distance between connected gates
    - Floor planning tool using design hierarchy
    - Specialized algorithms ( min cut, simulated annealing, etc.)
    - Timing driven
    - Manual intervention
  - Very compute intensive

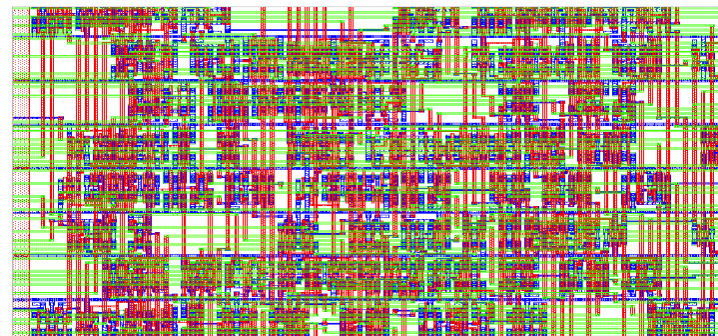


- 
- Routing:
    - Channel based: Routing only in channels between gates (few metal layers: 2)
    - Channel less: Routing over gates (many metal layers: 3 - 6)
    - Often split in two steps:
      - Global route: Find a coarse route depending on local routing density
      - Detailed route: Generate routing layout

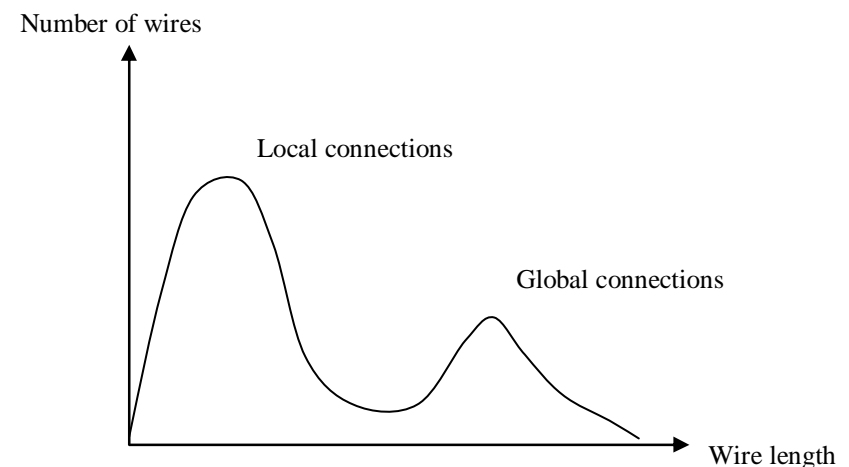
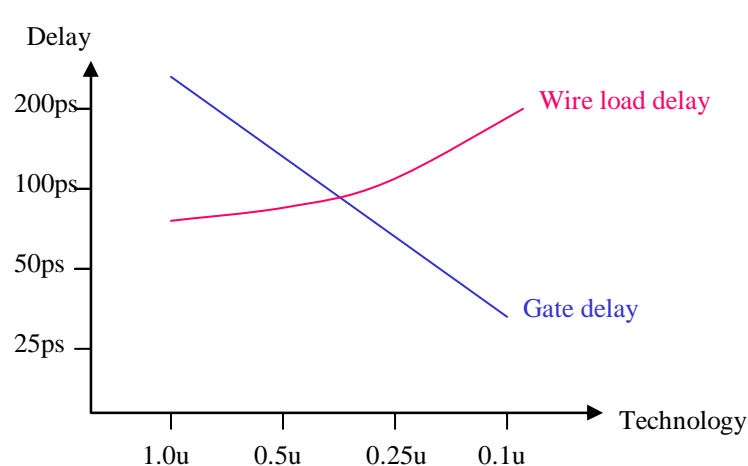
Channel based



Channel less



- Performance of sub-micron CMOS IC's are to a large extent determined by place & route.
  - Loading delays bigger than intrinsic gate delays
  - Wire R-C delays becomes important in sub-micron
  - Clock distribution over complete chip gets critical at operating frequencies above 100Mhz.



# Design tool framework

---

- Design tools from one vendor normally integrated into a framework which enables tools to exchange data.
  - Common data base
  - Automatic translation from one type to another
  - (Allows third part tools to be integrated into framework)
- Few standards to allow transport of designs between tools from different vendors.
  - VHDL and Verilog behavioral models and netlists
  - EDIF netlist, SPICE netlist for analog simulation
  - GDSII layout
  - Standard Delay Format (SDF) for gate delays.
  - Small vendors must be compatible with large vendors.

**Transporting designs between tools from different vendors often cause problems**

# Source of CAE tools

---

- Cadence, Mentor
  - Complete set of tools integrated into framework
- Synopsis + Avant
  - Power full synthesis tools
  - VHDL simulator
  - Power full place and route tools
  - Hspice simulator with automatic characterization tools
- Div commercial:
  - View-logic, Summit, Tanner, etc.
- Complete set of commercial high performance CAE tools cost ~1 M\$ per seat ! (official list price).

- 
- Free shareware:
    - Spice, Magic, Berkley IC design tools, Aliance
    - Diverse from the web.
  - University programs: tools ~10K\$, MPW runs
    - Europe: Europractice: Tools and MPW
    - US: Mosis: Now private MPW run  
Each tool supplier have separate university programs

# Hardware describing languages (HDL)

---

- Describe behavior not implementation
- Make model independent of technology
- Model complete systems
- Specification of sub-module functions
- Speed up simulation of large systems
- Standardized text format
- CAE tool independent



---

- VHDL

- Very High speed integrated circuit Description Language
- Initiated by American department of defense as a specification language.
- Standardized by IEEE

- Verilog

- First real commercial HDL language from gateway automation (now Cadence)
- Default standard among chip designers for many years
- Started as proprietary language
- Now also a IEEE standard because of severe competition from VHDL. Result: multiple vendors

---

- **Compiled/Interpreted**

- **Compiled:**

- Description compiled into C and then into binary or directly into binary
    - Fast execution
    - Slow compilation

- **Interpreted:**

- Description interpreted at run time
    - Slow execution
    - Fast “compilation”
    - Many interactive features

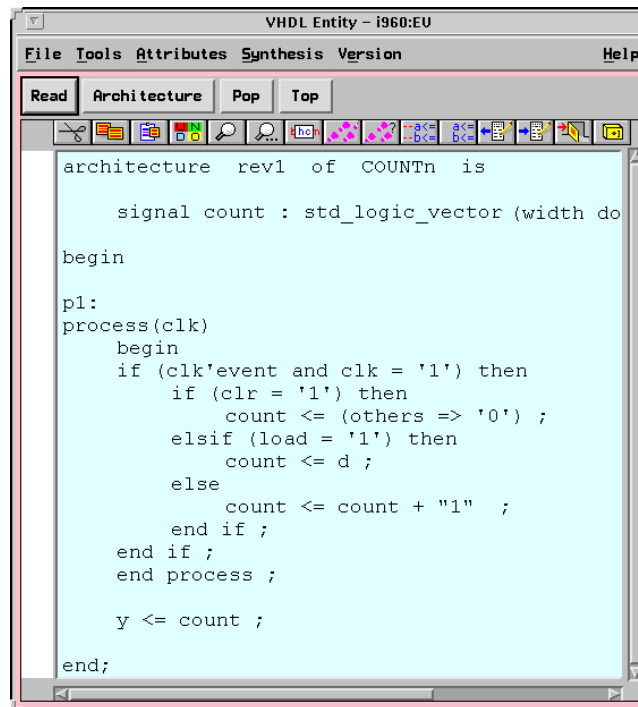
- **VHDL normally compiled**

- **Verilog exists in both interpreted and compiled versions**

# HDL design entry

---

- Text:
  - Tool independent
  - Good for describing algorithms
  - Bad for getting an overview of a large design



The image shows a screenshot of a VHDL editor window titled "VHDL Entity - i960:EU". The window has a menu bar with "File", "Tools", "Attributes", "Synthesis", "Version", and "Help". Below the menu bar is a toolbar with various icons for editing and navigation. The main text area contains the following VHDL code:

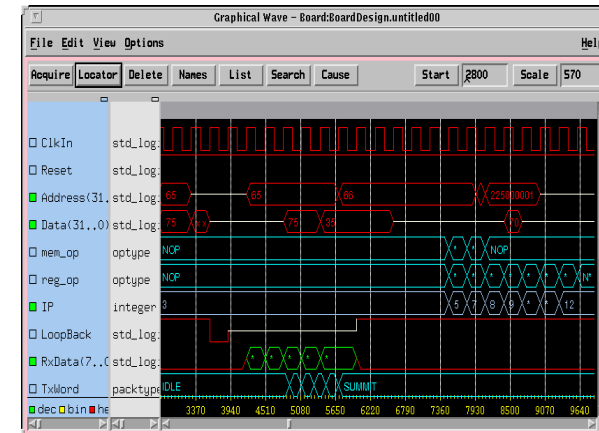
```
architecture rev1 of COUNTn is
    signal count : std_logic_vector (width do
begin
p1:
process(clk)
begin
    if (clk'event and clk = '1') then
        if (clr = '1') then
            count <= (others => '0') ;
        elsif (load = '1') then
            count <= d ;
        else
            count <= count + "1" ;
        end if ;
    end if ;
end process ;

y <= count ;

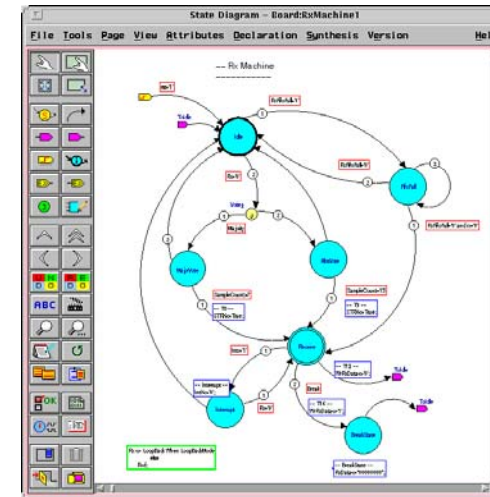
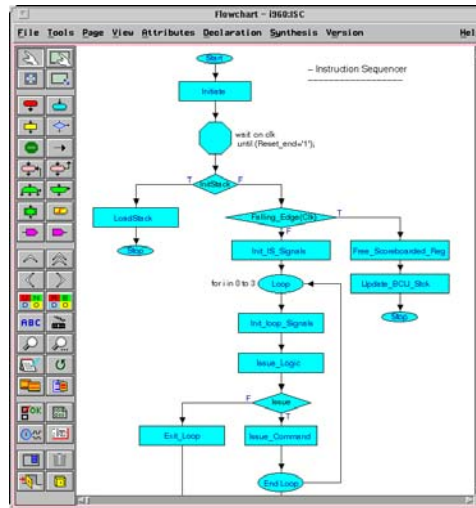
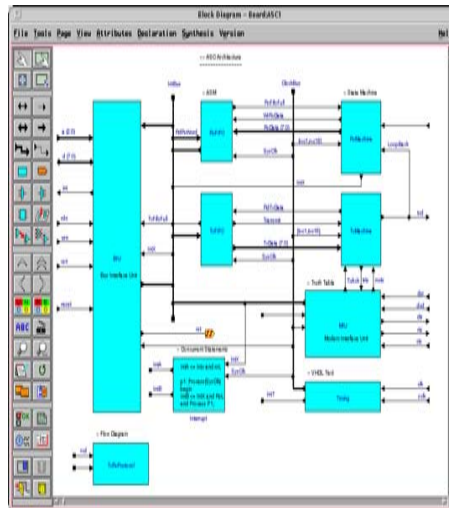
end;
```

- Add-on tools

- Block diagrams to get overview of hierarchy
- Graphical description of final state machines (FSM)
  - Generates synthesizable HDL code
- Flowcharts
- Language sensitive editors
- Waveform display tools



From Visual HDL, Summit design



# Synthesis and Technology dependence

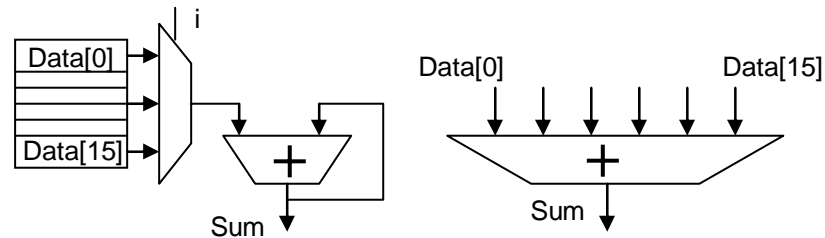
## Algorithm

0% technology dependent

```
For i = 0 ; i = 15  
sum = sum + data[i]
```

## Architecture

10% technology dependent

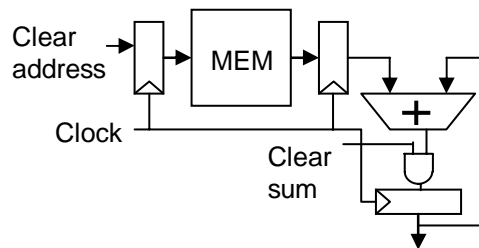


Behavioral synthesis

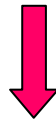


## Register level

20% technology dependent

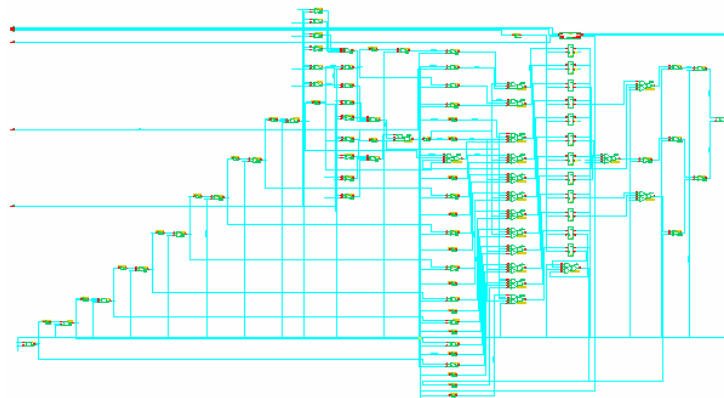


Logic synthesis



## Gate level

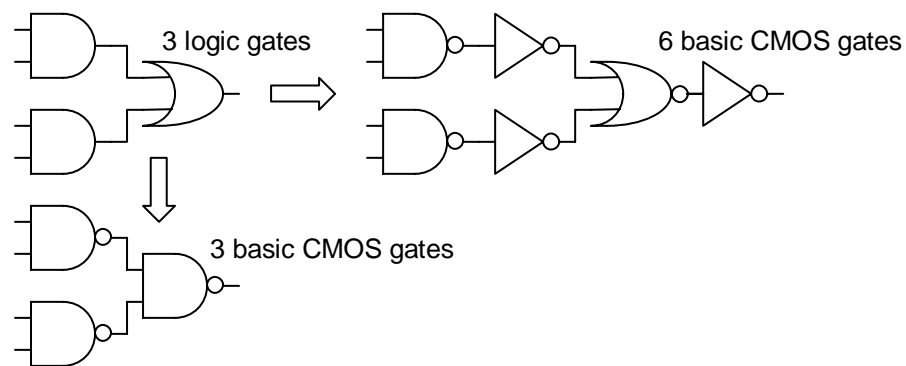
100% technology dependent



# Logic synthesis

---

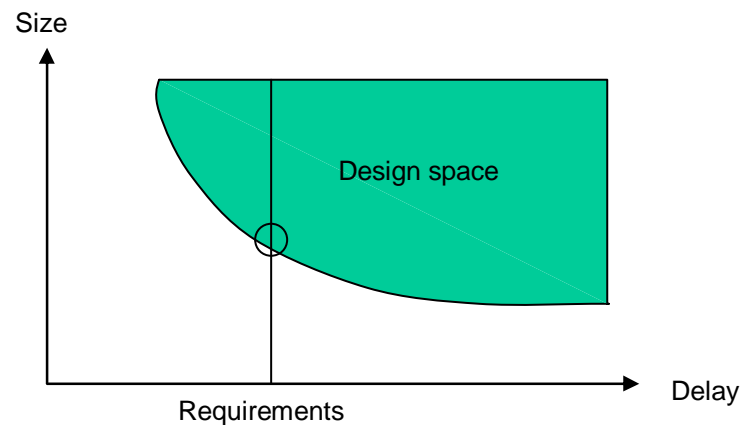
- HDL compilation (from VHDL or Verilog)
  - Registers: Where storage is required
  - Logic: Boolean equations, if-then-else, case, etc.
- Logic optimization
  - Logic minimization (similar to Karnaugh maps)
  - Finds logic sharing between equations
  - Maps into gates available in given technology
  - Uses local optimization rules



# Synthesis goals

---

- Combined timing - size optimization
  - Smallest circuit complying to all timing constraints



- Best solution found as a combination of special optimization algorithms and evaluation of many alternative solutions (Similar to simulated annealing)

---

- Problems in synthesis

- Dealing with “single late signal”
- Mapping into complex library elements
- Regular data path structures:

- Adders: ripple carry, carry look ahead, carry select, etc.
- Multipliers, etc.

Use special guidance to select special adders, multipliers, etc..

Performance of sub-micron technologies are dominated by wiring delays (wire capacitance + R-C delays)

- Synthesis in many cases does a better job than a manually optimized logic design.

(in much shorter time)



# Timing estimation in synthesis

- **Wire loading**

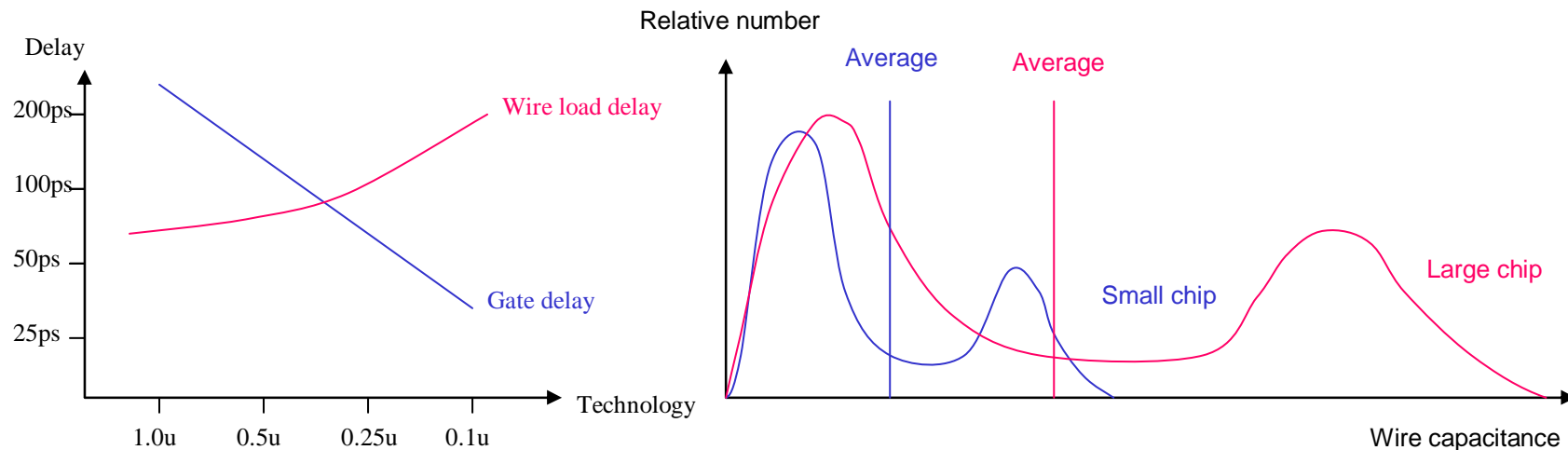
Timing optimization is based on a wire loading model.

Loading of gate = input capacitance of following gates + wire capacitance

Gate loading known by synthesizer

Wire loading must be estimated

R-C delay calculation very complicated





# Trends in synthesis

---

- Integration of synthesis and P&R
- Synthesizable standard modules (Processor, PCI interface, Digital filters, etc.), IP blocks
- Automatic insertion of scan path for production testing.
- Synthesis for low power
- Synthesis of self-timed circuits (asynchronous)
- Behavioral synthesis
- Formal verification
- Hardware and software co-design
  - What to put in hardware and what in software ?
    - System C tools