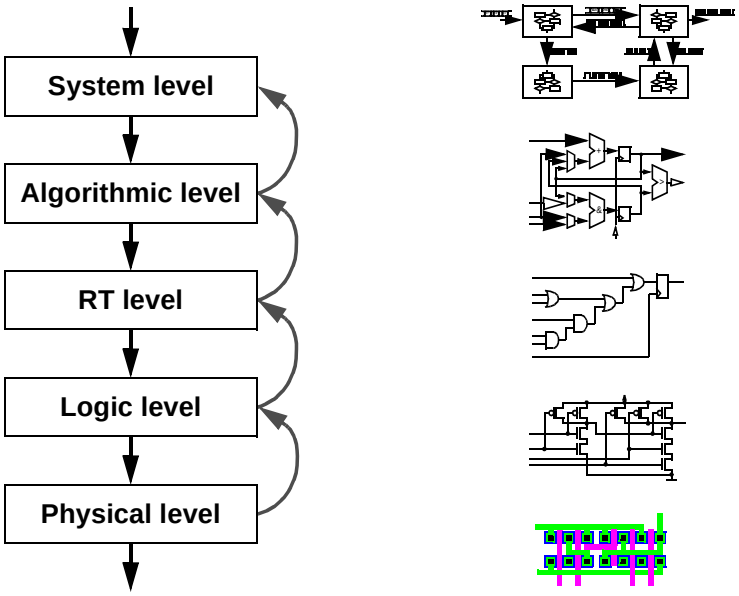
Design Methodology

Abstraction levels



Levels revealed

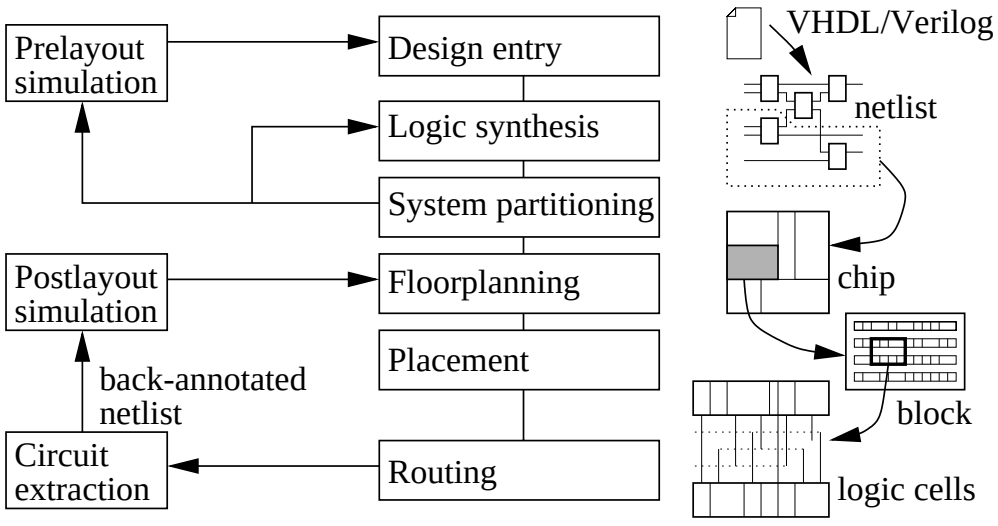
Hierarchy level	Abstraction	Supporting tools
System	space-time behavior as instruction, timing & pin assignment specifications	flow-charts, diagrams, high-level languages
Architecture	global organization of functional entities	HDLs, floor-planning block diagrams for clock cycle and area estimation
Register transfer	binding data flow functional modules and microinstructions	synthesis, simulation, verification, test analysis, resource use evaluation
Functional modules	primitive operations and control methods	libraries, module generators, schematic entry, test
Logic	Boolean function of gate circuits	Schematic entry, synthesis and simulation, verification, PLA tools
Switch	electrical properties of transistor circuits	RC extraction, timing verification, electrical analysis
Layout	geometric constraints	layout editor/compactor, netlist extractor, DRC, placement and routing

Synthesis levels and tasks

- System Level Synthesis
  - Clustering
  - Communication synthesis
- High-Level Synthesis
  - Resource or time constrained scheduling
  - Resource allocation
  - Binding
- Register-Transfer Level Synthesis
  - Data-path synthesis
  - Controller synthesis
- Logic Level Synthesis
  - Logic minimization
  - Optimization, overhead removal
- Physical Level Synthesis
  - Library mapping
  - Placement
  - Routing

ASIC design flow

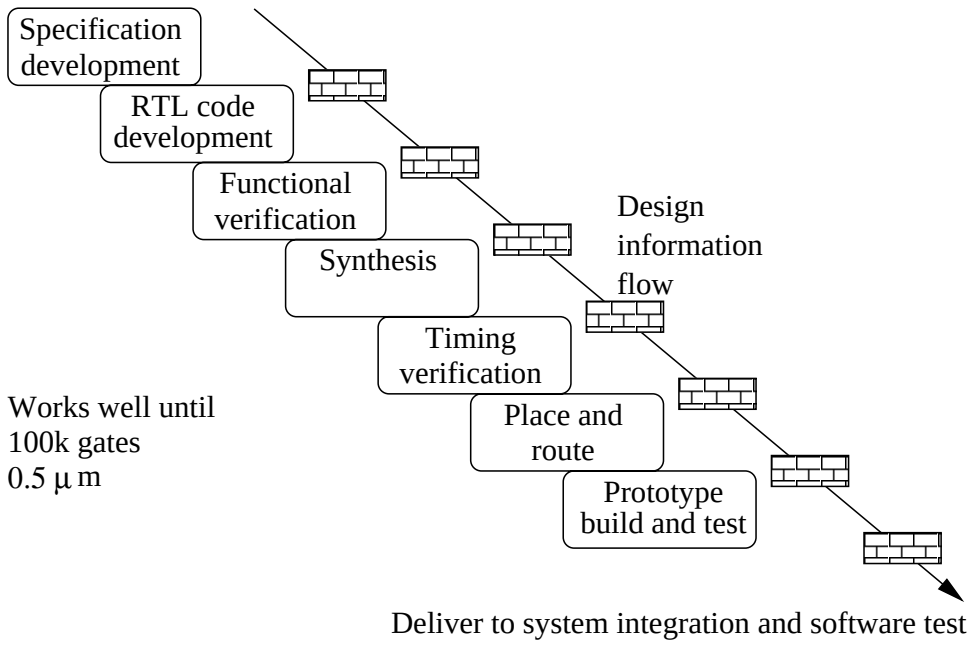
- [Smith97]



Waterfall vs. Spiral

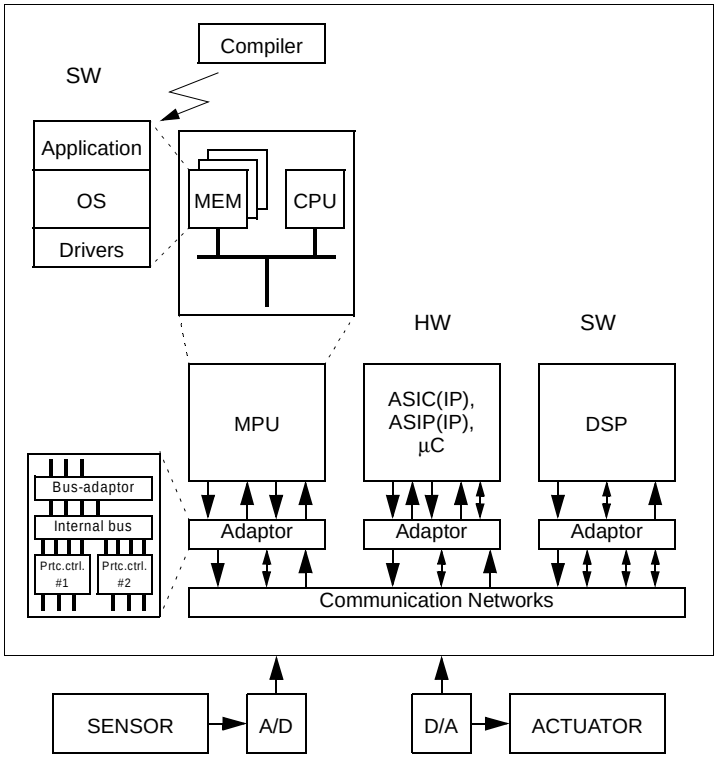
- **Traditional ASIC development follows so called *waterfall model*.**
- **In WF model, the project transitions from phase to phase in a step function, never returning to the activities of the previous phase.**
  (“Tossing” the project over the wall from one team to the next)
  But:
  - **Complexity increases**
  - **Geometry shrinks**
  - **Time-to-market pressure increases**
- **In the *spiral model*, the design teams work on multiple aspects of the design simultaneously, incrementally improving in each area as the design converges on completion.**

Waterfall model



Embedded Systems Components

System-on-Chip

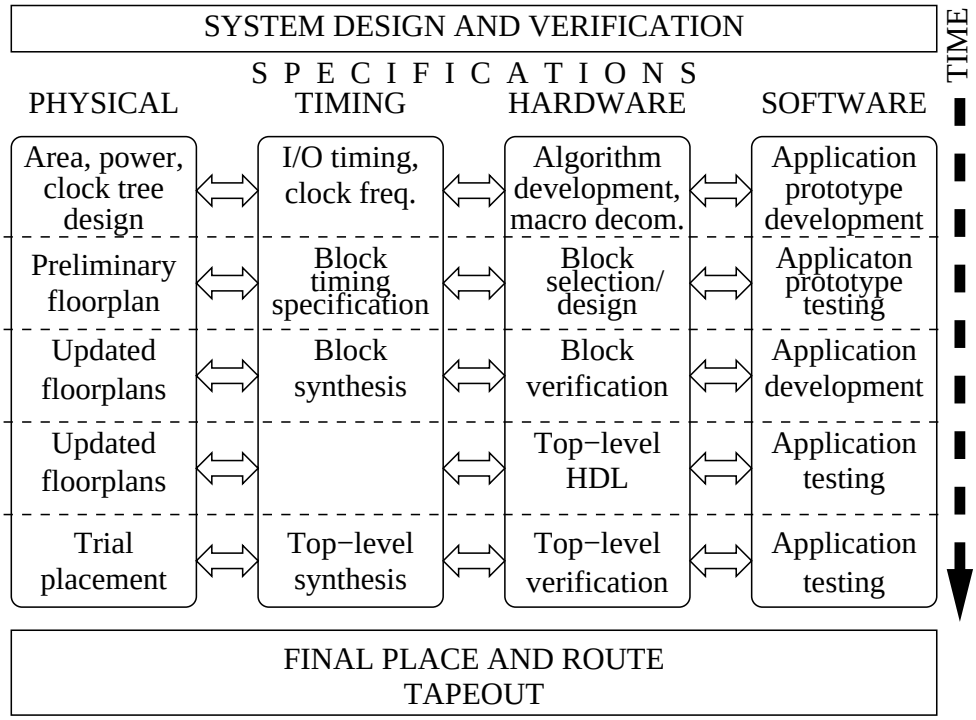


System-on-Chip design flow

- **Parallel, concurrent development of HW and SW**
- **Parallel verification and synthesis of modules**
- **Floor-planning and place-and-route included in the synthesis**
- **Modules developed only if a predesigned hard and soft macro is not available**
- **Planned iteration throughput**

- **All aspects of HW and SW design are addressed concurrently - functionality, timing, physical design, and verification**

System-on-Chip design flow



Top-down

- **Write complete specifications for the system or subsystem being designed**
- **Refine its architecture and algorithms, including SW design and HW/SW cosimulation if necessary**
- **Decompose the architecture into well-defined macros**
- **Design or select macros (recursion here!)**
- **Integrate macros into the top level; verify functionality and timing**
- **Deliver the subsystem/system to the next higher level of integration; at the top level this is tapeout**
- **Verify all aspects of the design (functionality, timing, etc.)**

... vs. Bottom-up

- **There exist synthesis and emulation tools, and libraries of reusable macros**
- **The design can be started at lowest level as well!**
- **At top level is difficult to estimate acceptable complexity of lowest level blocks ==> let's mix approaches to meet in the middle.**
  - **faster & fewer iterations**

System Design Process

- **System specification**
  - **Functions, performance, cost, development time**
- **Model refinement and test**
  - **Focus on the algorithm, not the implementation!**
- **HW/SW partitioning (decomposition)**
  - **... largely a manual process**
  - **Finally, define interfaces between SW and HW, and specify communication protocol**
- **Block specification**
  - **Elaborate *hardware specification* and *software specification***
- **System behavioral model and cosimulation**
  - **Cosimulate and refine (the cosimulation continues throughout the design process)**

Evolution of Silicon Process Technology

	1997	1998	1999	2002
Process technology	0.35 μ	0.25 μ	0.18 μ	0.13 μ
Cost of fab	$1.5 - 2.0 billion	$2.0 - 3.0 billion	$3.0 - 4.0 billion	$4.0 billion +
Design cycle	18 - 12 months	12 - 10 months	10 - 8 months	8 - 6 months
Derivative cycle	8 - 6 months	6 - 4 months	4 - 2 months	3 - 2 months
Silicon complexity	0.2 - 0.5 M gates	1 - 2 M gates	4 - 6 M gates	10 - 25 M gates
Applications	Cellular, PDAs, DVD	Set-top boxes, wireless PDA	Internet appliances, anything portable	Ubiquitous computing, intelligent, inter-connected controllers
Primary IP sources	Intragroup	Intergroup	Intercompany	Intercompany, interindustry

Evolution of Design Methodology

Historical Linchpin Technologies

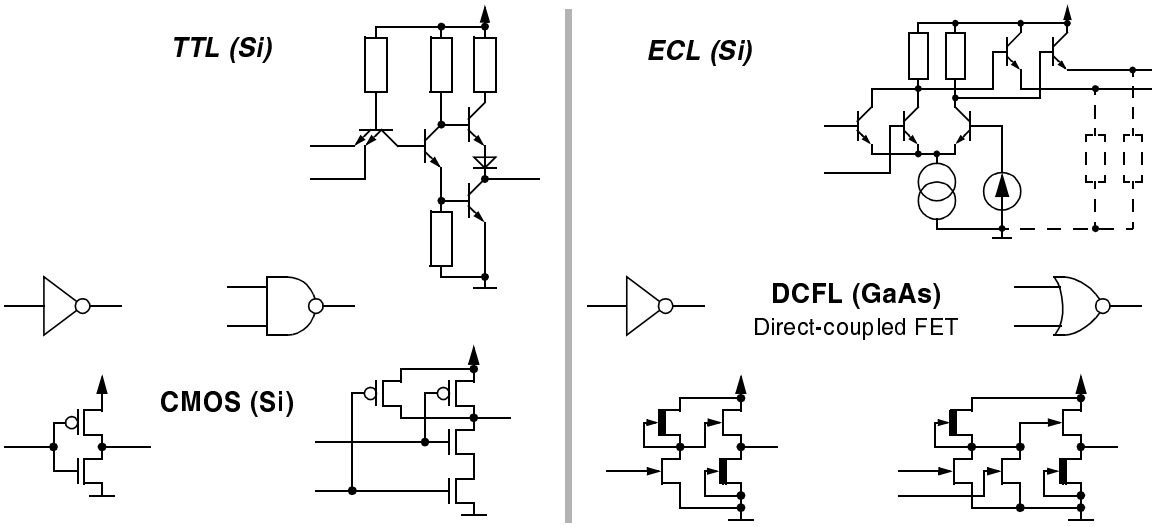
Time	1988 - 1990	1991 - 1994	1995 - 1996
Design	25-50 K gates 2.0 - 1.5 µ no design reuse	50-100 K gates 1.0 - 0.8 µ no design reuse	100-200 K gates 0.6 - 0.5 µ minimal reuse
Challenge	gate-level simulation	increase productivity	managing timing problems
Linchpin Technologies	gate-level simulation place & route	synthesis	design planning
		gate-level simulation place & route	synthesis gate-level simulation place & route

Design Methodologies Today

Design characteristics	Timing-Driven Design	Block-Based Design	Platform-Based Design
Design complexity	5000 to 250 K gates	150 K to 1.5 M gates	300 K gates and greater
Design level	RTL	behavioural / RTL	architecture and VC evaluation
Design team	small, focused	multydisciplinary	multygroup, multydisciplinary
Primary design	custom logic	blocks in context, custom interfaces	interfacing to system and bus
Primary design granularity	gates and memory	functional clusters, cores	VCs
Bus architecture	none / custom	custom	standardized / multiple application specific
Mixed-signal	none	A/D, PLL	functions, interfaces
Hardware/software co-verification	none	HW/SW functionality and interfaces	HW/SW interfaces only
Partitioning focus	synthesis limitations	functions	f-ns / communications

VLSI Physical Design

Basic gates - NAND vs. NOR



- **Material properties**
  - **mobility (Si) -- μ_n = 1250 cm^2 / V sec & μ_p = 480 cm^2 / V sec & R ~ μ^{-1}**

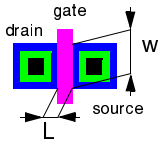
Material properties

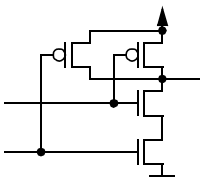
Properties	Unit	Si	GaAs	4H-SiC	Diamond
Electron mobility	[cm^2/ V·s]	1500	8500	1000	2200
Hole mobility	[cm^2/ V·s]	600	400	50	1600
Bandgap	[eV]	1.1	1·43	2	2.7
Dielectric constant		11.8	12.5	9.7	5.5
Thermal conductivity	[W / cm ·°K]	1.5	0.46	4.9	20
Saturation electron drift velocity	[x10^7 cm/s]	1	1	2	2.7
Melting point	[C]	1420	1238	2830	4000
Breakdown field	[x10^5 V/cm]	3	6	30	100

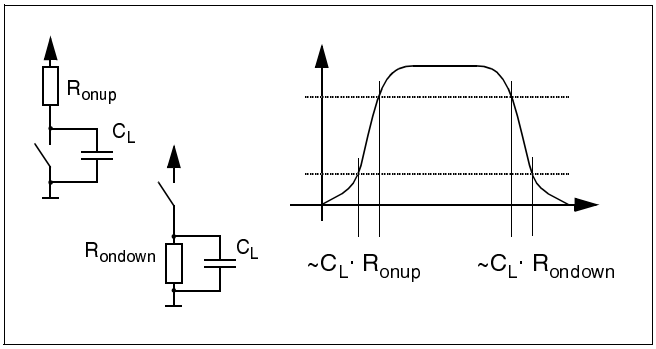
	CMOS	Bipolar	GaAs
Power dissipation (P_{DC})	Low	High	Medium
Input impedance	High	Low	High
Noise margin	High	Medium	Low
Speed	Medium	High	Very high
Packing density	High	Low	High
Delay sensitivity to load	High (o)	Low (o)	High (i/o)
Output drive	Low	High	Low
Bidirectional	Yes	No	Possible
Switching device	Ideal	Not ideal	Reasonable
f_t frequency	Medium	High (at low current)	Very high
Mask levels	12 to 16	12 to 20	6 to 10

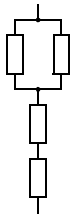
CMOS - why NAND?

- **Mobility --**
  μ_n = 1250 cm^2 / V sec & μ_p = 480 cm^2 / V sec
- **R ~ μ^{-1} & R ~ L w^{-1} (L-constant)**









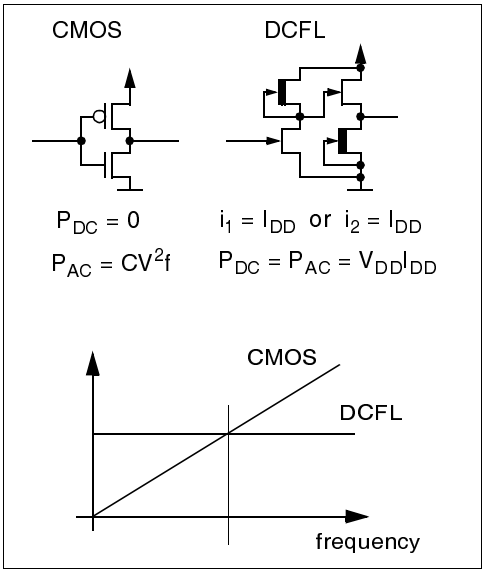
$R \sim \mu^{-1} w^{-1}$

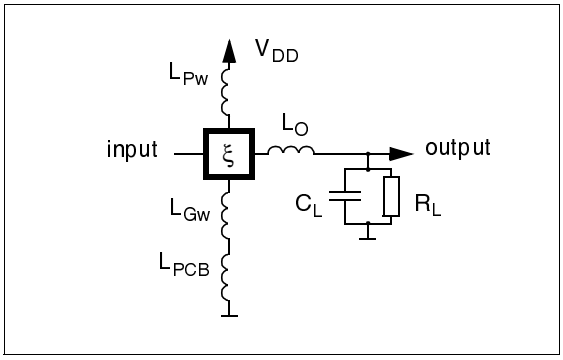
$w_p \sim 3 \cdot w_n \quad (R_p = R_n)$

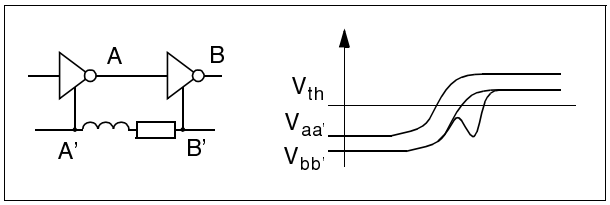
$R_p || R_p \sim R_n + R_n$

$R_p \sim 4 \cdot R_n \quad (w_p = w_n)$

Very high speed

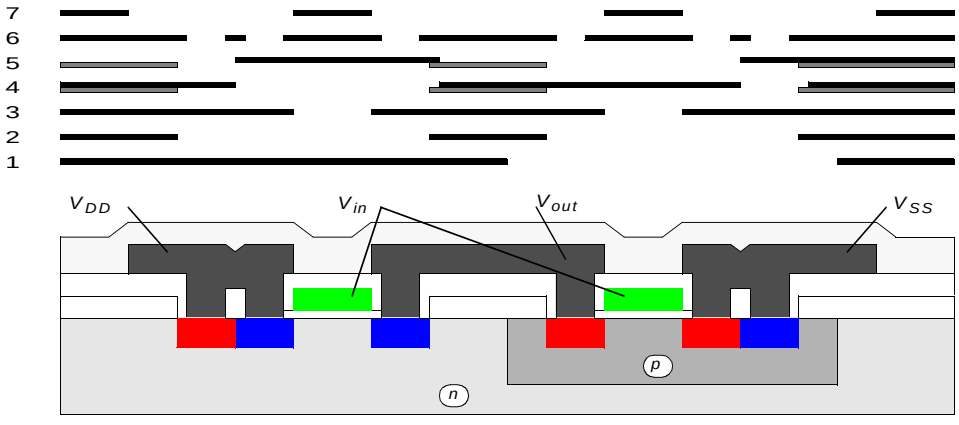






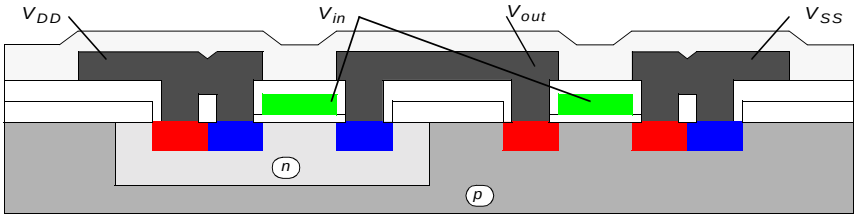
Process steps

- **Mask 1 - deep p-well diffusions** — **nMOS transistors**
- **Mask 2 - thinox regions**
- **Mask 3 - polysilicon** — **for "gate wires"**
- **Mask 4 - p-diffusion** — **pMOS transistors, p^+-mask & mask 2**
- **Mask 5 - n-diffusion** — **nMOS transistors, inverted p^+-mask & mask 2**
- **Mask 6 - contact cuts**
- **Mask 7 - metal layer**
- **Mask 8 - overglass layer** — **overall passivation, access to bonding pads**

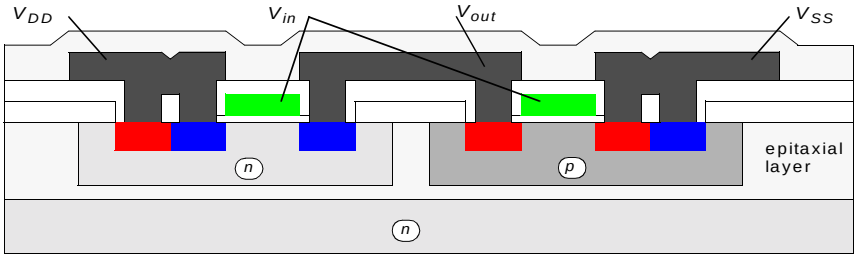


Process steps - variations

- Lithography masks - positive & negative
- Transistor wells - p-well, n-well, twin-tub & silicon-on-insulator



n-well

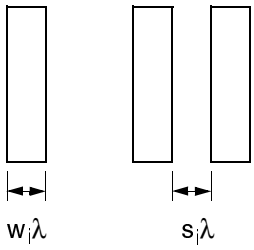


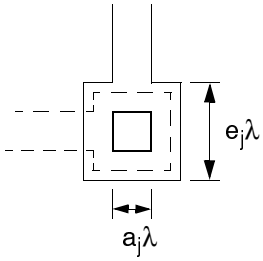
twin-tub

- "Educational Java Applets in Solid State Materials" - jas.eng.buffalo.edu

Layout rules

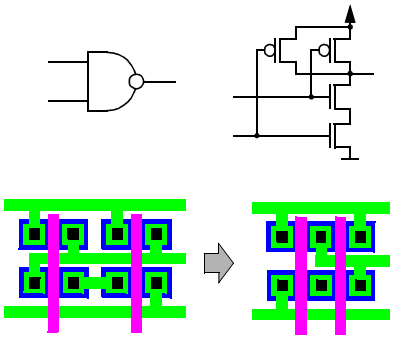
- Feature size - λ (2 μm, 0.8 μm, 0.13 μm, etc.)
- Layers - L_1, ... , L_v
- Wire width - w_i
  - $w_1>2$, $w_i>w_{i-1}$
- Wire separation - s_i
  - $s_1=3$, $s_i \geq s_{i-1}$
- Contact rule - layers L_i & L_j (i<j)
- Layout grid
  - $s_i\lambda=s_v\lambda=\lambda_s$, $e_i\lambda=e_v\lambda=\lambda_e$ --> $\lambda_s+\lambda_e$





Layout environments

- Cell generation
  - Programmable logic arrays (PLA)
  - Transistor chaining
  - Weinberger arrays & gate matrices
- Layout environments
  - Standard cells
  - Gate arrays
  - Sea-of-gates
  - Field-Programmable Gate Arrays (FPGA)

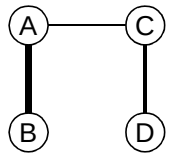


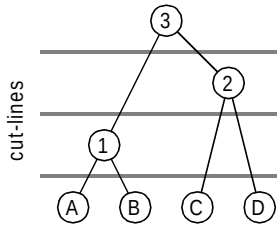
Transistor chaining

Layout methodologies

- Partitioning
- Floorplanning
  - initial placement
- Placement
  - fixed modules
- Global routing
- Detailed routing
- Layout optimization
- Layout verification

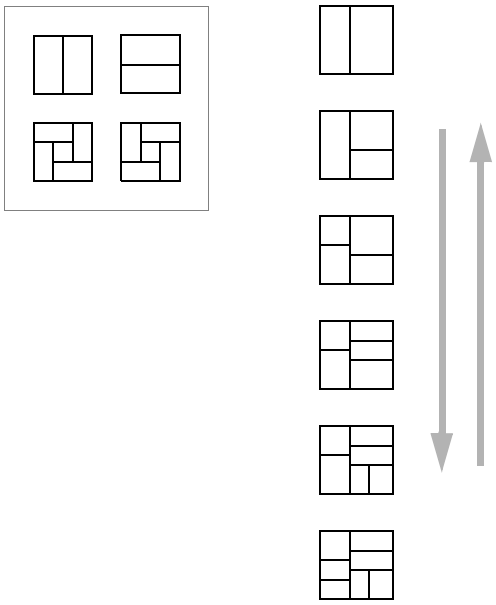
- Partitioning
  - Weighted compatibility graph partitioning
    - hypergraphs
  - Constructive approaches
    - hierarchical clustering
  - Iterative improvements
    - Kernighan-Lin heuristic
  - Weights
    - module size
    - number of connections
    - number of I/O-s





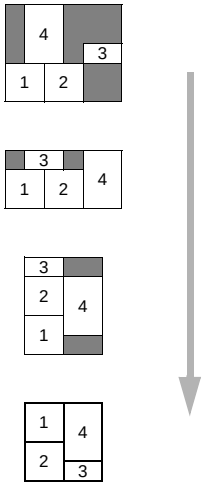
Floorplanning

- Sliceable floorplan
  - two slices
  - templates
- Rectangular dual graph approach
  - planar graph
  - $(M_i,M_j) \in E$ - modules are adjacent M_i, M_j
- Hierarchical approaches
  - bottom-up approach
  - top-down approach
- Soft-computational approaches
  - simulated annealing
  - genetic algorithms



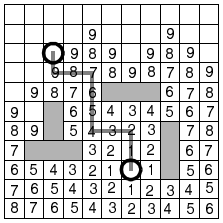
Placement

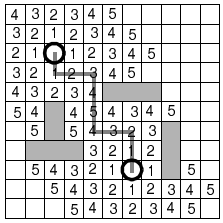
- Improvement of the initial floorplan
- Refined cost functions
  - known ports
- Constructive heuristics
- Iterative heuristics
- Soft-computing

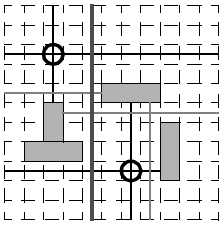


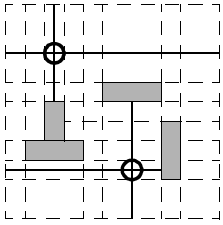
Routing

- Maze running
  - memory usage!
  - bidirectional search
  - minimum cost paths
  - multilayer routing
  - multiterminal routing
- Line searching
  - track graph
- Global routing
  - dividing routing task into smaller sub-tasks
  - routing channels
- Detailed routing
  - routing inside channels
- Layout optimization
  - channel compaction



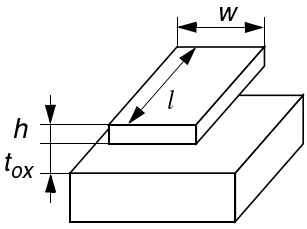


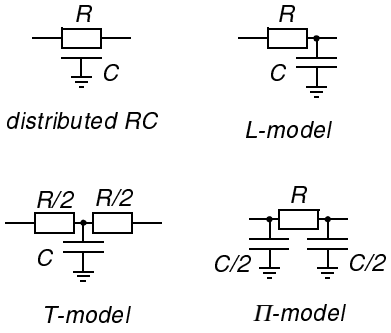




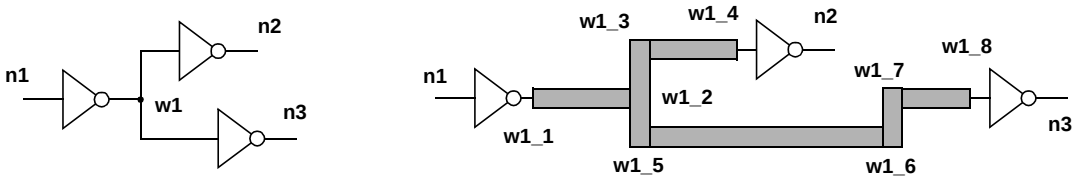
Delay modelling

- Delays in wires can not be neglected anymore!
- $C_w = f_r\, \varepsilon_{ox}\, w\, l / t_{ox}$ $\quad R_w = \rho\, l / w\, h$
- Standard unit of capacitance □C_g - gate-to-channel capacitance having W=L=λ
- 1.2 μm technology □C_g - 2.3 fF & τ - 46 ps
- Elmore delay - $T_{ELM} = ½R_wC_w + R_wC_L$





Delay modelling



- **Analog (Spice) models**
  - transmission line based models ~ distributed RC
  - approximate models - L-model, Elmore delay
- **Digital (VHDL) models**
  - wire segment == assignment with delay
  - back-annotation -- layout -> VHDL model

```
w1_1 <= not n1 after gate_delay ps;
w1_2 <= w1_1 after delay_w1_1 ps;
. . .
w1_8 <= w1_7 after delay_w1_8 ps;
n3   <= not w1_8 after gate_delay ps;
```


Logic Synthesis

- **Transforming logic functions (Boolean functions) into a set of logic gates**
  - transformations at logic level from behavioral to structural domain
- **Optimizations / Transformations**
  - area
  - delay
  - power consumption
- **Implementation of Finite State Machines (FSM)**
  - state encoding
  - generating next state and output functions
  - optimization of next state and output functions

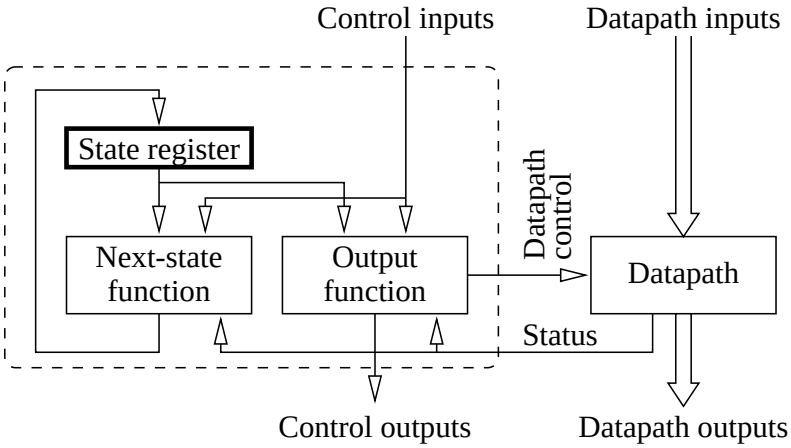
RT Level Synthesis

***Definition*: Register-Transfer level synthesis means transformation from RT-level structural description (in terms of registers, multiplexers and operations) to Logic level description (in terms of combinational logic blocks and storage elements)**

- **Data path synthesis**
  - **Maximizing the clock frequency**
  - **Retiming**
  - **Operator selection**
- **Controller synthesis**
  - **Architecture selection**
  - **FSM optimizations for area and performance**
  - **State assignment / coding**
  - **Decomposition**

FSM with Data-Path Model

- **FSMD models are used to describe digital systems on the register-transfer level.**

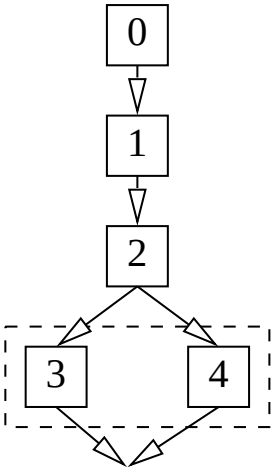


- **Data-path consists of storage units (registers, register files, memories) and combinational units (ALUs, multipliers, shifters, comparators etc.), connected by buses.**

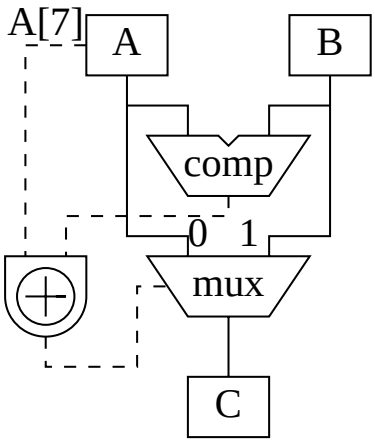
Data-Path Optimization

```
read_port(A);
read_port(B);
if A[7] = 0 then
  begin
    if A > B then C := A;
    else C := B;
  end
else
  begin
    if A > B then C := B;
    else C:= A;
  end
```

Control flow



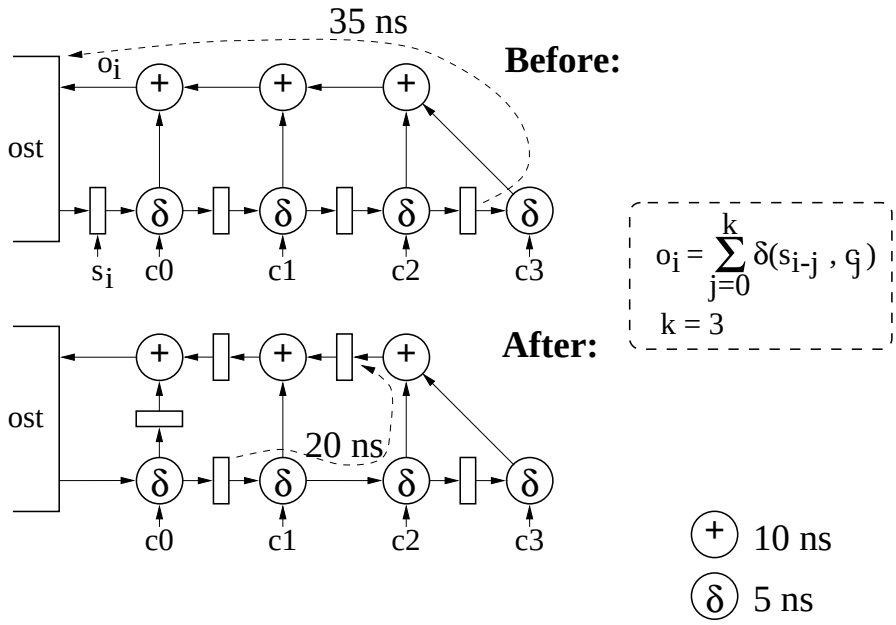
Data flow



Retiming

- ***Retiming*** **is a transformation of sequential circuits at the RT-level, whereby registers are moved across combinational blocks such a way as to minimize the clock cycle, or to minimize the number of registers**
- **The functionality of the circuit is not changed by register relocation process**
  - **Optimization:**
    - **control-step #1:** $r_1 \leftarrow c_1(v_i,...)$,
    - **control-step #2:** $r_2 \leftarrow c_2(r_1,v_j,...)$
    - **where r_1,r_2 are registers; c_1,c_2 combinational blocks; v_i,v_j variables**
    - $f_{max} = 1 / \max(delay(c_1),delay(c_2))$,
    - $delay(c_1) > delay(c_2)$: then $c_1^{new} = g(f_1(v_i,...), f_2(v_i,...))$
  - **After resynthesis, when** $delay(g)+delay(c_2) < delay(c_1)$ **:**
    - **control-step #1:** $r_1 \leftarrow f_1(v_i,...)$; $r_2 \leftarrow f_2(v_i,...)$
    - **control-step #2:** $r_3 \leftarrow c_2(g(r_1,r_2),v_j,...)$

Retiming: Digital Correlator



Resource Allocation and Assignment

- **The task of *operator selection* is the selection of an appropriate operator implementation from a library**
- **Selecting the architecture of a complex operation**
  - **parallel versus sequential execution**
  - **bit-parallel versus bit-serial**
- **Example - addition**
  - **bit-serial adder**
    - **1 full-adder, 1-bit register, n clock cycles**
    - **Manchester adder - m bits in parallel (m<n)**
  - **bit-parallel adders**
    - **ripple-carry, carry-look-ahead, carry-skip and carry-select adders**

`http://www.ecs.umass.edu/ece/koren/arith/simulator`

Arithmetic Unit Architecture Selection

- Parallel versus sequential execution
- Adder/subtracter architectures
  - ripple-carry - sequence of full-adders, small but slow
  - carry-look-ahead - separate calculation of carry generation and/or propagation
  - carry-select adders - duplicated hardware plus selectors
    - speculative calculation one case with carry and another without, the answer will be selected when the actual carry has arrived
- Multiplier architectures
  - sequential algorithms -- register + adder, 1/2/... bit(s) at a time
  - “parallel” algorithms - array multipliers -- AND gates + full-adders
- Multiplication/division with constant
  - shift+add -- 5*n = 4*n + n = (n<<2) + n

Controller synthesis

- Controller synthesis is also task at algorithmic level. Controller is the implementation of the scheduling task in hardware, specified by states and state transitions. It is called Finite State Machine (FSM).
- *The canonical implementation* of a sequential system is based directly on its state description. It consists of *state register*, and a *combinational network* to implement the transition and output functions.
- Sub-tasks:
  - Generation of the state graph;
  - Selecting the proper controller architecture, and
  - Finite state machine optimizations for area and performance.

FSM encoding

- ***Task*: Encoding inputs, outputs, and states.**
  - **Studied intensively in the sixties and seventies.**
- **Let *z* be the number of states, then minimum code length is t=ceil($\log_2 z$).**
- **Current methods follow one of the two objectives:**
  - **Improve the testability**
  - **Minimize the area of control logic**

    Known tools are KISS (‘85) and NOVA (‘89)
- **Two encoding approaches:**
  - **Minimal code length encoding**
  - **“One-hot” encoding**

FSM architectures