Chap. 8: Design Methodology and Tools

Jinsang Kim

Kyung Hee University

Outline

- Design Domains: Y chart
- Structured Design Strategies
- Design Methods
- Design Flow
- Design Economics
- Data Sheets and Documentation
- Closing the Gap between ASIC and Custom
- CMOS Physical Design Styles
- Interchange Formats

Why Design Methodology?

- **Design method gives impact on both effort and outcome of the design**
  - Borrows concept from multiple disciplines such as SW engineering
  - Broad principles: not changed
  - Details: evolved, need to monitor corresponding tools and literatures
- **Three domains**
  - Behavioral
    - **What to do:** ultra low-power radio for sensor network, wireless standard
  - Structural
    - **Interconnect of components:** connection between sensor, transceiver, up, memory,.., logic family, clocking strategy
  - Physical
    - **How to arrange component**: PCB layouts, chip or boards

Y Chart

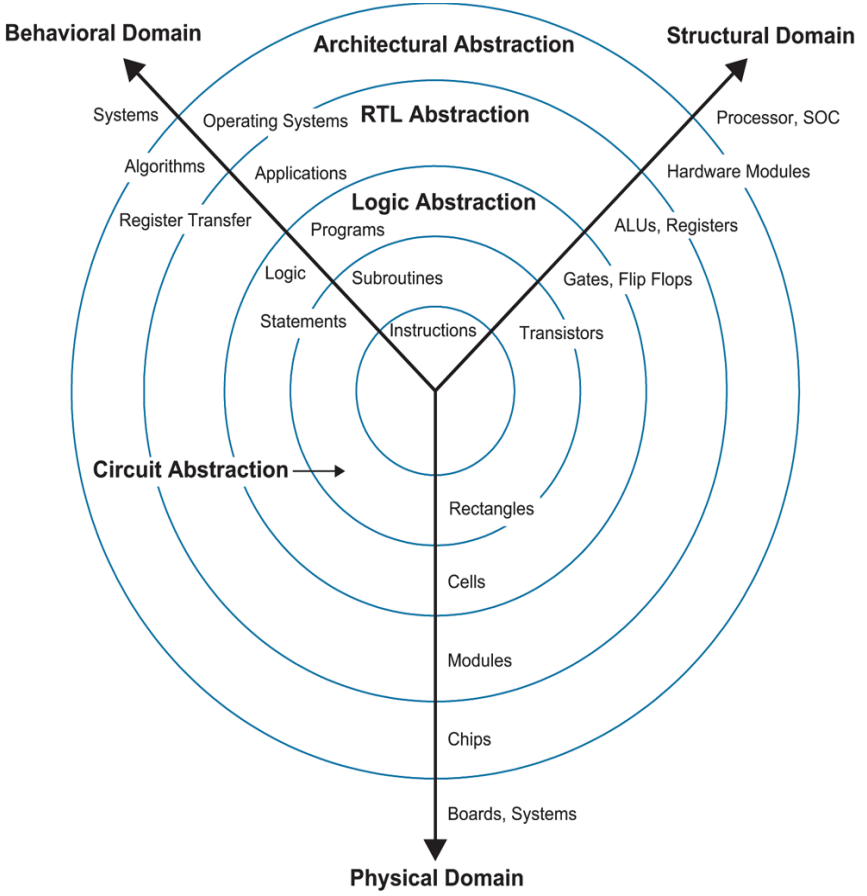


FIG 8.1 Gajski-Kuhn Y chart

- **Each domain can be hierarchically divided into different levels of abstraction, typically**
  - Architectural
  - RTL
  - Logic
  - Circuit
- **need verification of correctness after transformation between domains**

Structured Design

- **Design is a continuous tradeoff for the following parameters to measure the goodness of VLSI system on the three domains**
  - Performance: speed, power, function, flexibility
  - Size of die: cost
  - Time to design: cost
  - Ease of verification and test: cost
- **Why the structured design?**
  - For complex system design, the role of good VLSI design aids is to reduce this complexity and to increase productivity.
  - A good method of simplifying the approach is by the use of constraints and abstractions
    - Constraint: tool designer can automate the procedure
    - Abstraction: can collapse details and arrive at a simpler object to handle.

Software Radio

- **To explain the structured design, we take the abstract software radio as an example**
  - Information is modulated onto RF carrier

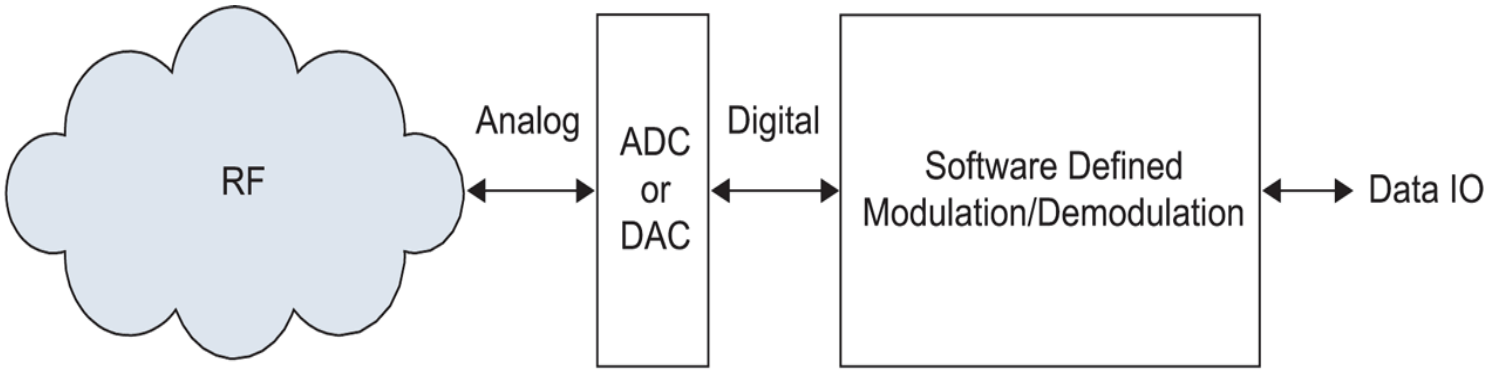


FIG 8.2 Software radio block diagram

Software Radio Transmit Path

- **Upconversion**
  - Intermediate frequency: 20Mhz
  - RF
    - 2.402 or 2.398GHz, analog frequency multiplier: mixer

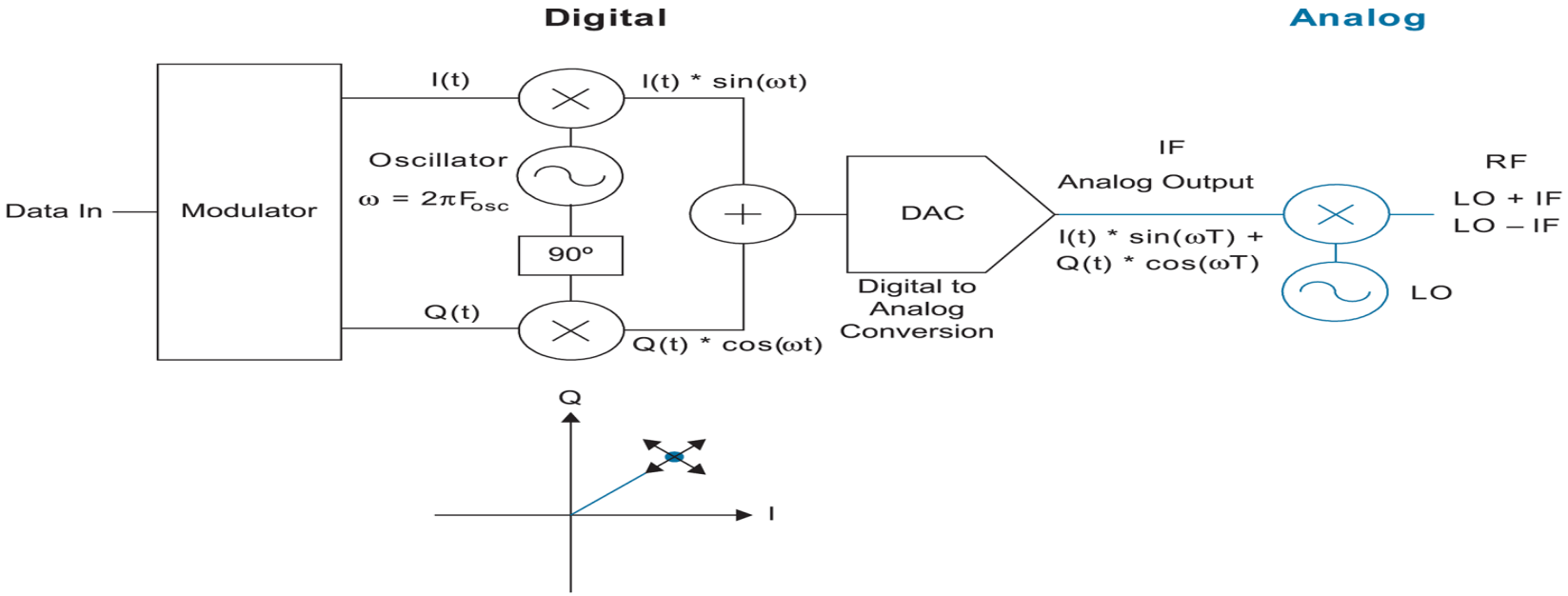


FIG 8.3 Software radio transmit path

Software Radio Receive Path

- **Downconversion**
  - 2.402 or 2.398GHz RF to 20Mhz IF for ADC

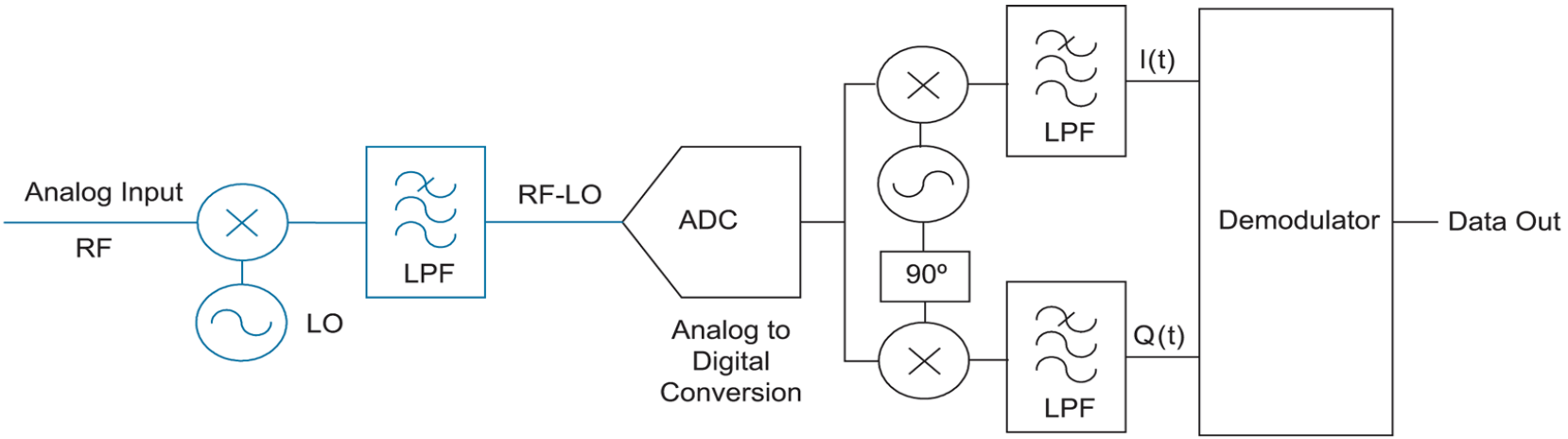


FIG 8.5 Software radio receive path

- **In summary, multiplication, sinewave generation, and filtering are important for SDR**

Structured Design Strategies

- **Four guides**
  - Hierarchy
  - Regularity
  - Modularity
  - Locality

Hierarchy: Example 1

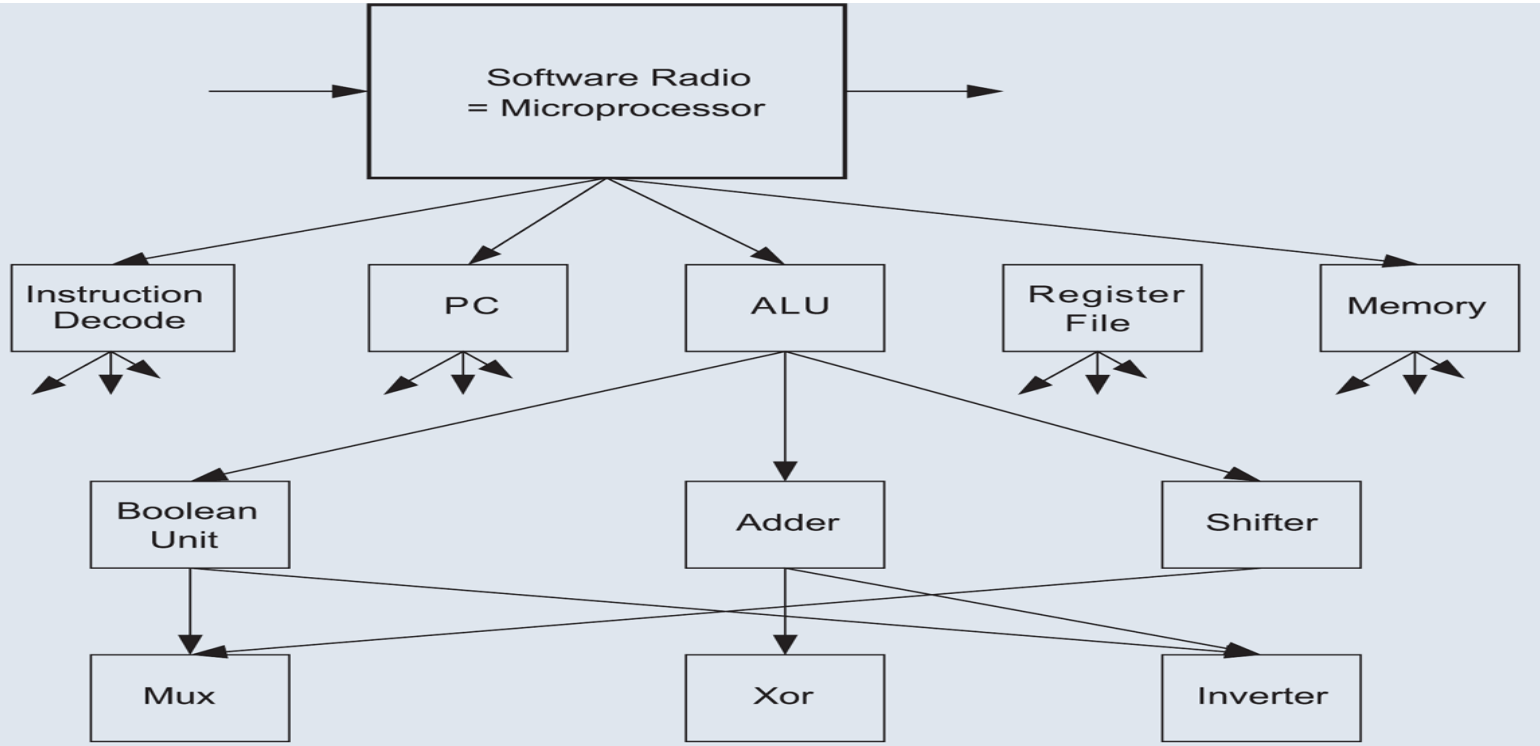


FIG 8.6 Possible hierarchy of software radio using a single microprocessor

- Software solution using a single processor

Hierarchy: Example 2

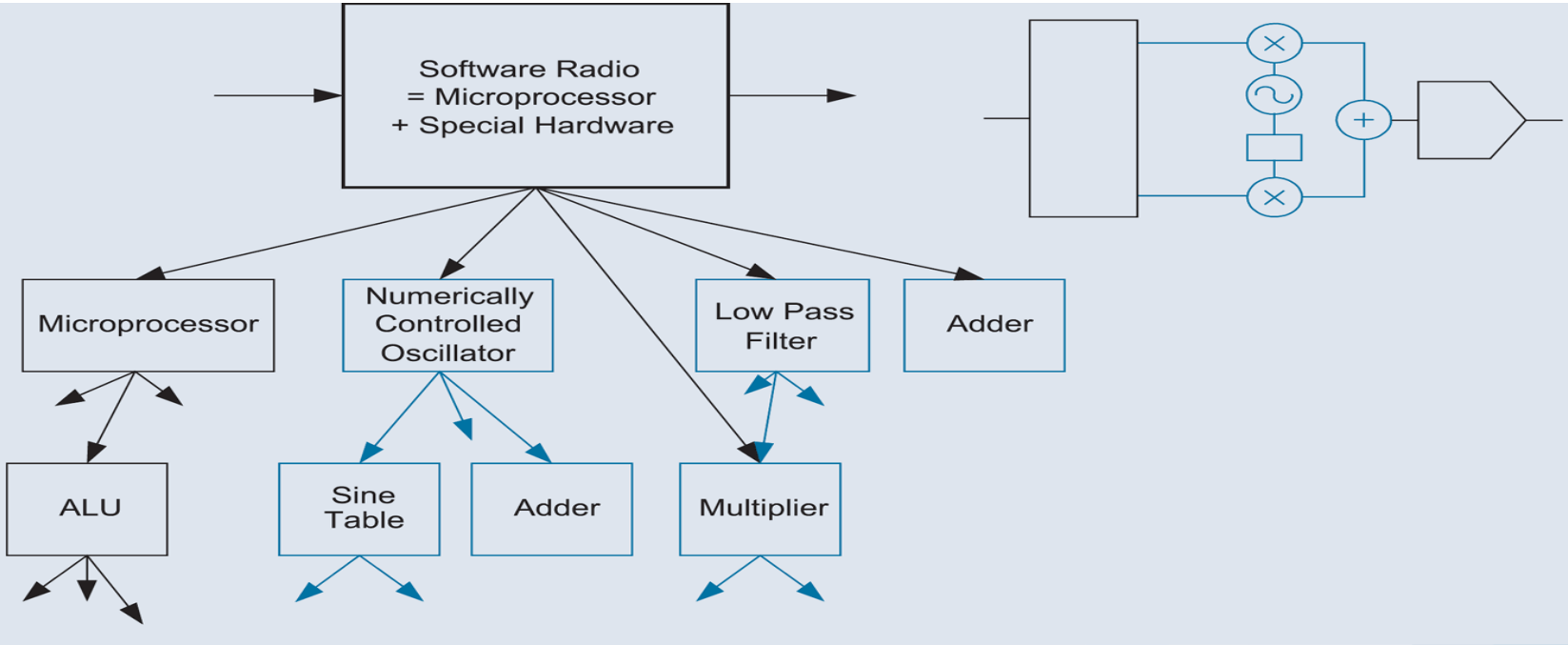


FIG 8.7 Transmit chain with dedicated IQ upconverter

- Better solution to use dedicated HW for the computationally intensive fixed-function blocks

Regularity

- **Hierarchy alone does not solve the complexity problem**
  - May end up with a large number of different submodules → still complex !!
- **With regularity as a guide, the designer attempts to divide the design hierarchy into a set of similar building block**
  - Circuit level: same sized TRs
  - Gate level: finite library of fixed height
  - Logic level: RAMs and ROMs in multiple places
- **Regularity**
  - Design reuse!!
  - Aids in verification efforts

Regularity: Example 1

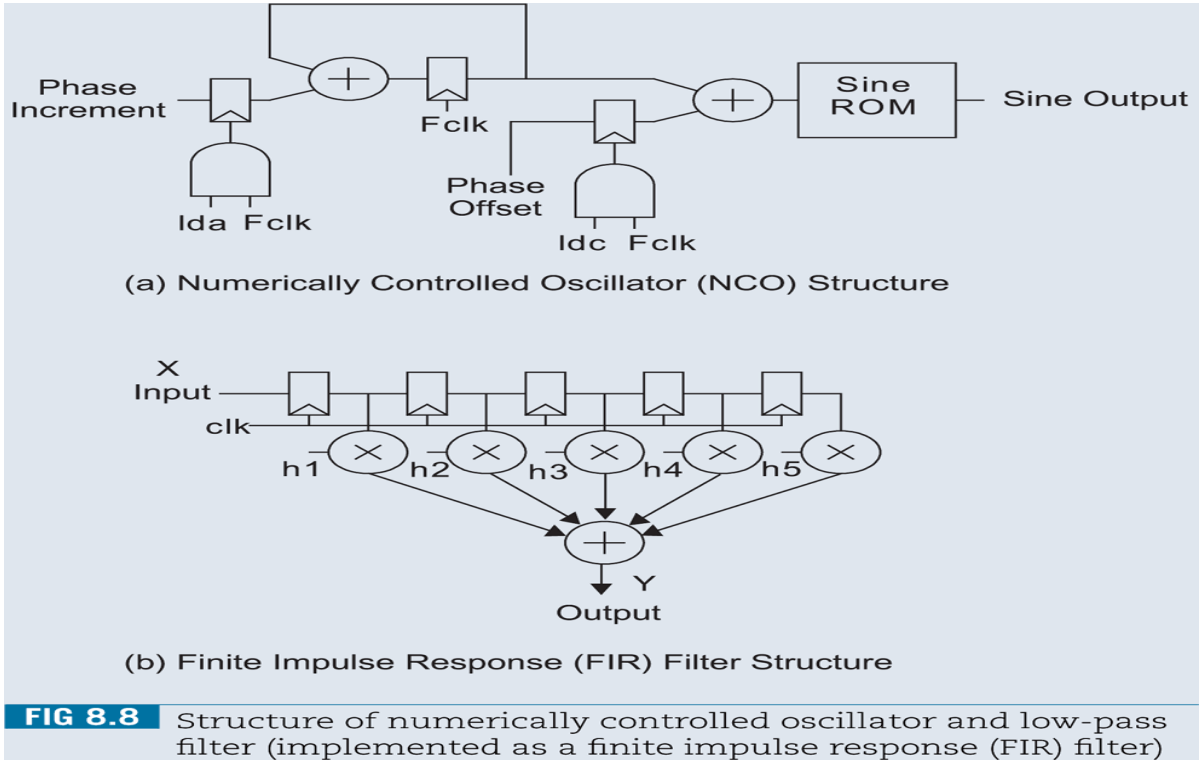


FIG 8.8 Structure of numerically controlled oscillator and low-pass filter (implemented as a finite impulse response (FIR) filter)

- **Common functions**
  - Register, adders and mutipliers

Regularity: Example 2

- IQ upconversion and downconversion: fixed hardware

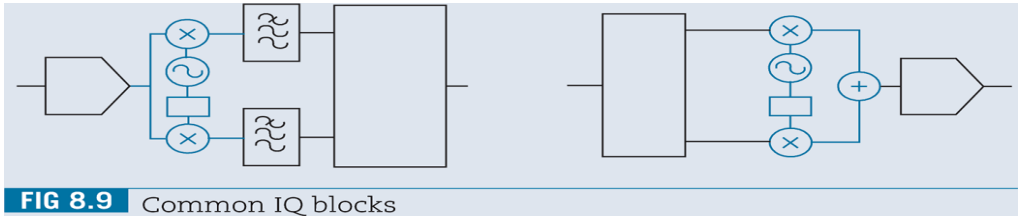


FIG 8.9 Common IQ blocks

- Key feature of the software defined radio
  - Programmability!!

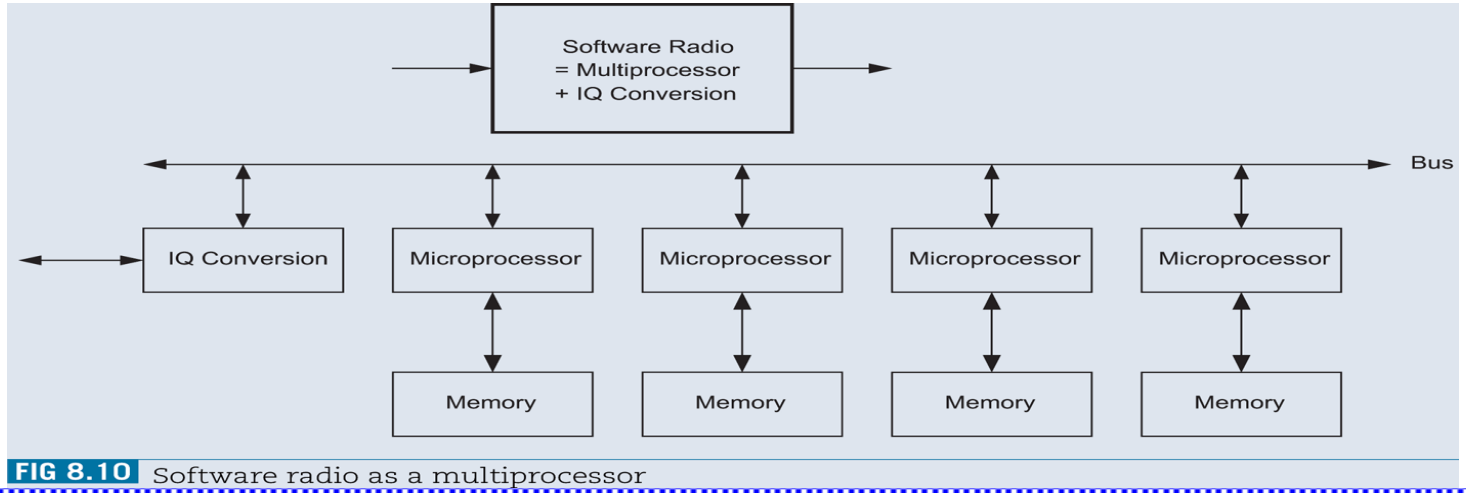


FIG 8.10 Software radio as a multiprocessor

Regularity: Example 2

- Enhanced up which embeds one-cycle mutiplication

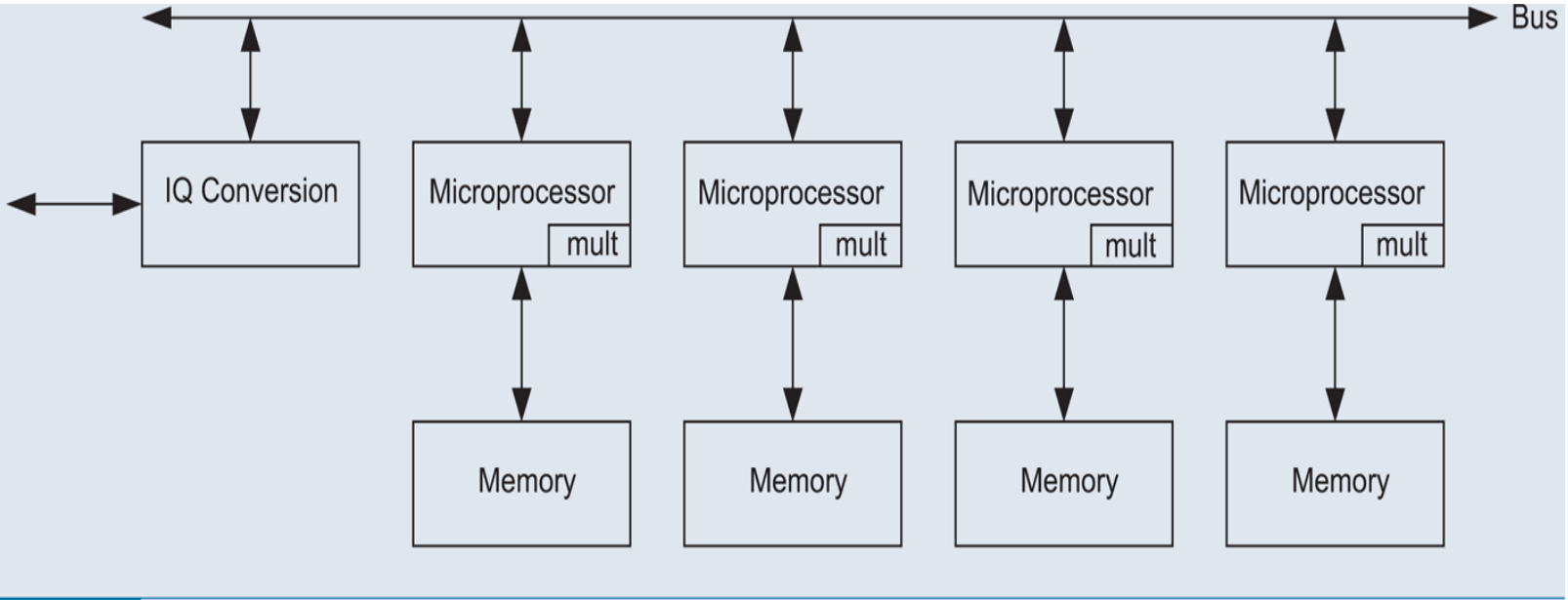


FIG 8.11 Enhanced multiprocessor for software radio

Modularity

- **Modularity states that "modules have** well-defined functions and interfaces"
- **"well-formed"?**
  - Clearly defined behavioral, structural, and physical interface that indicates functions
  - As well as name, signal type and electrical and timing constraints of the ports
- **Examples**
  - No too large fan-in or too small a drive capability
  - For noise immunity, inputs should only drive TR gates, not diffusion
  - Network on chip (NoC)

Modularity: Example 1

- Which FF?

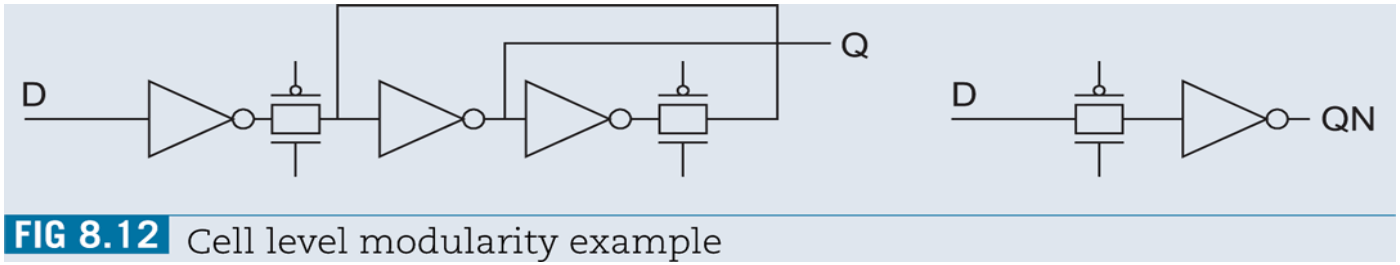


FIG 8.12 Cell level modularity example

- Another form of modularity: add testability for all register and register IO module for ease of input arrival time

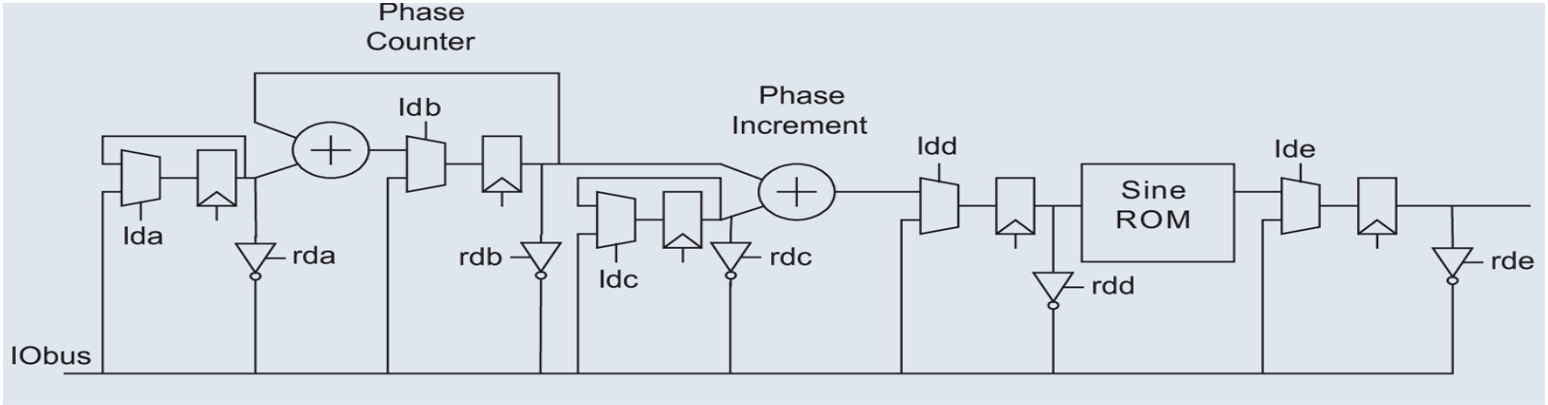


FIG 8.13 Modular test added to NCO

Modularity: Example 2

- **Compact sine ROM**
  - Store only the first quadrant of the sine wave
  - Good for small sine lookup table

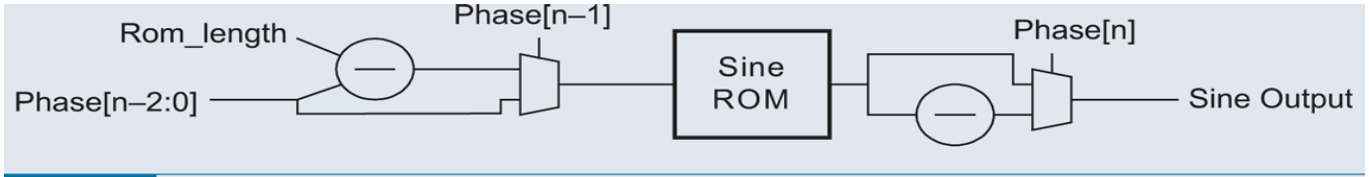


FIG 8.14 Sine ROM optimization

- Alternative structure: CORDIC

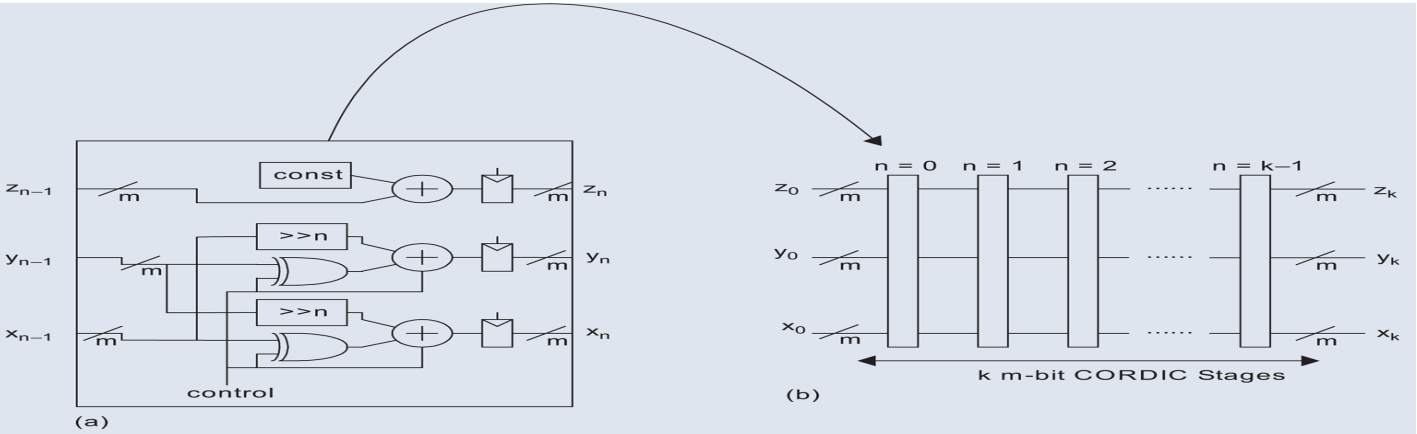


FIG 8.15 CORDIC processor as a sine generator

Locality

- **Internals of the modules are unimportant to other module**
  - "Information hiding" to reduce complexity
  - Ex: reduce global variables
- **Physical (spatial) locality:** above example
- **Temporal locality**
  - Clock protocol: the three clocking strategies in Chap. 7.

Locality: Examples

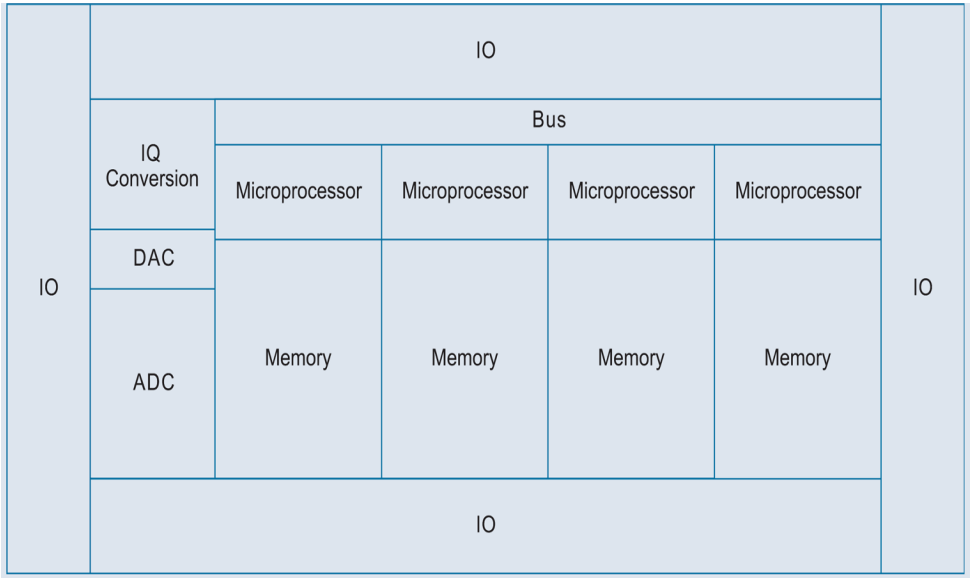


FIG 8.16 One possible floorplan for the software radio

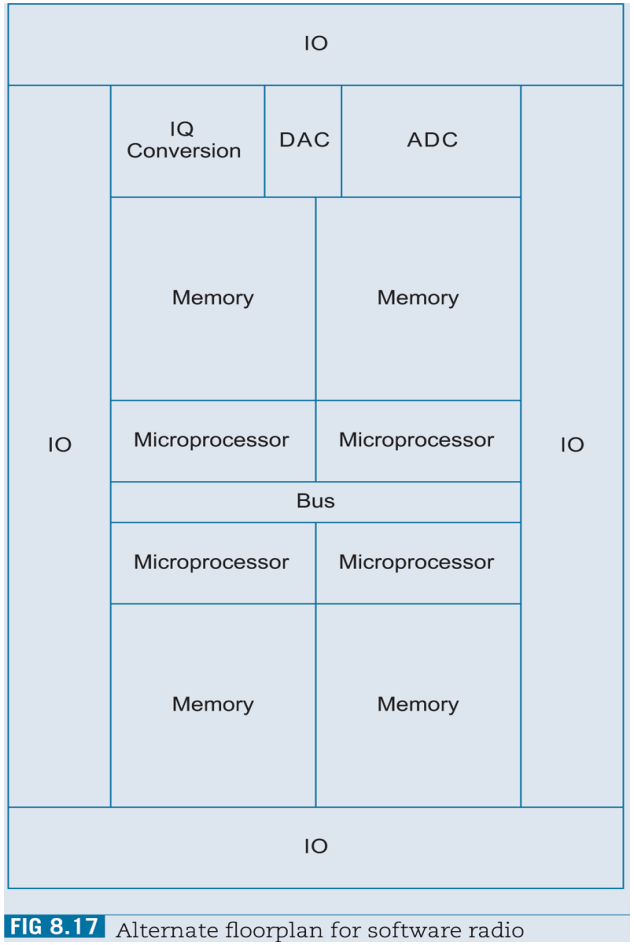


FIG 8.17 Alternate floorplan for software radio

- **Floorplanning: Analog components close to IO pads**
  - Large DC current → short Power busses
- **Which one has better locality?**

Structured Design: Summary

Design principle	Software	Hardware
Hierarchy	Subroutine, libraries	Modules
Regularity	Iteration, code sharing, object-oriented procedures	Datapaths, module reuse, regular arrays, gate arrays, standard cells
Modularity	Well-defined subroutine interfaces	Well-defined module interfaces, timing and loading data for modules, registered IOs
Locality	Local scoping, no global variables	Local connections through floorplaning

Design Methods

- **With what method to implement a CMOS system? → design method → implementation tech!**
- **Need to know the costs, capabilities, and limitations of a given implementation technology**
  - Microprocessor/DSP
  - Programmable logic: PLA, FPGA
  - Gate Array and Sea of Gates
  - Cell-based
  - Full custom
  - Platform-based

Microprocessor/DSP

- **Great flexibility: SW upgrade**
  - But SW development costs
- **Embedded commercial processor**
  - ARM
  - MIPS
  - IBM's PowerPC
  - Sun SPARC V8

Programmable logic: PLA

- **Cost, speed, or power dissipation of *uPs* may not meet system goal**
- **More efficient alternative than *uPs* and yet faster to develop than ASIC**
  - Chip with (re)programmable {logic arrays, interconnect}
- **Programmable logic array (PLA)**
  - Limited routing capability than FPGA
  - AND and OR planes

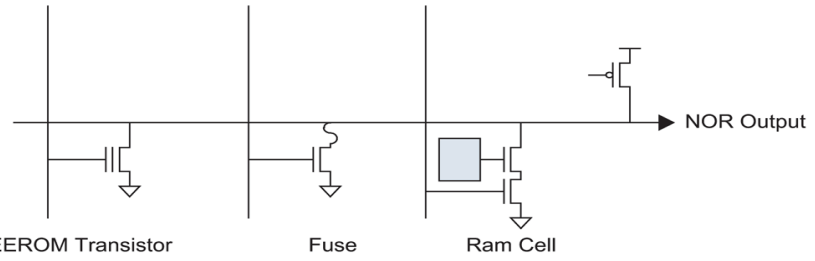


FIG 8.18 PLA NOR structure (one plane shown)

Programmable logic: FPGA

- **Field-programmable Gate Arrays**
  - Completely programmable even after shipped, or "in the field"
- **Programmability**
  - Actel: logic module (LM)
  - Xilinx: configurable logic block (CLB)
- **More advanced**
  - Add FFs, multipliers,…, etc.

Programmable Logic: FPGA

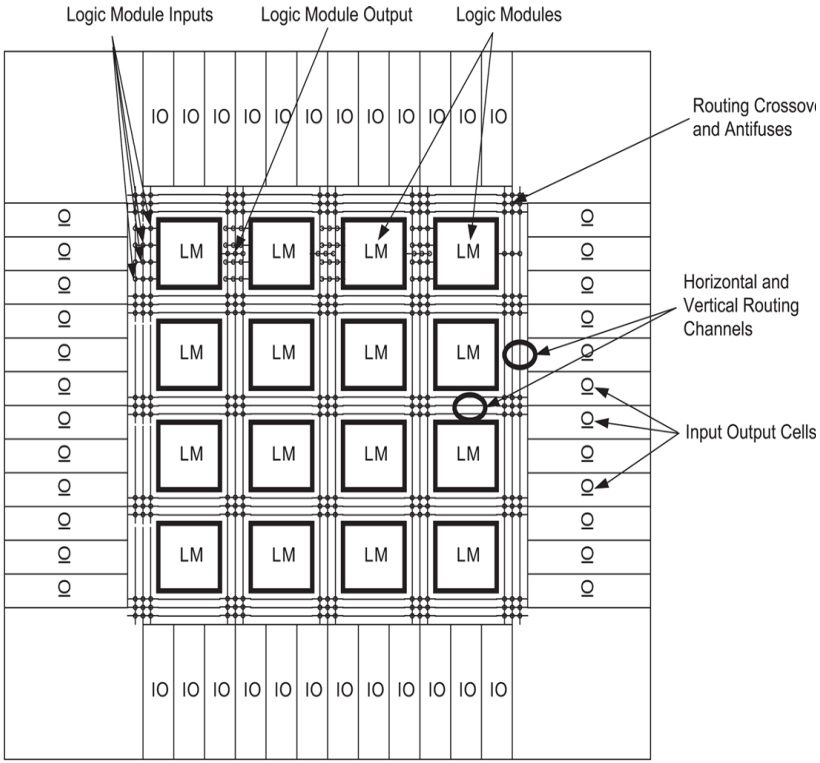


FIG 8.20 Representative Actel FPGA floorplan

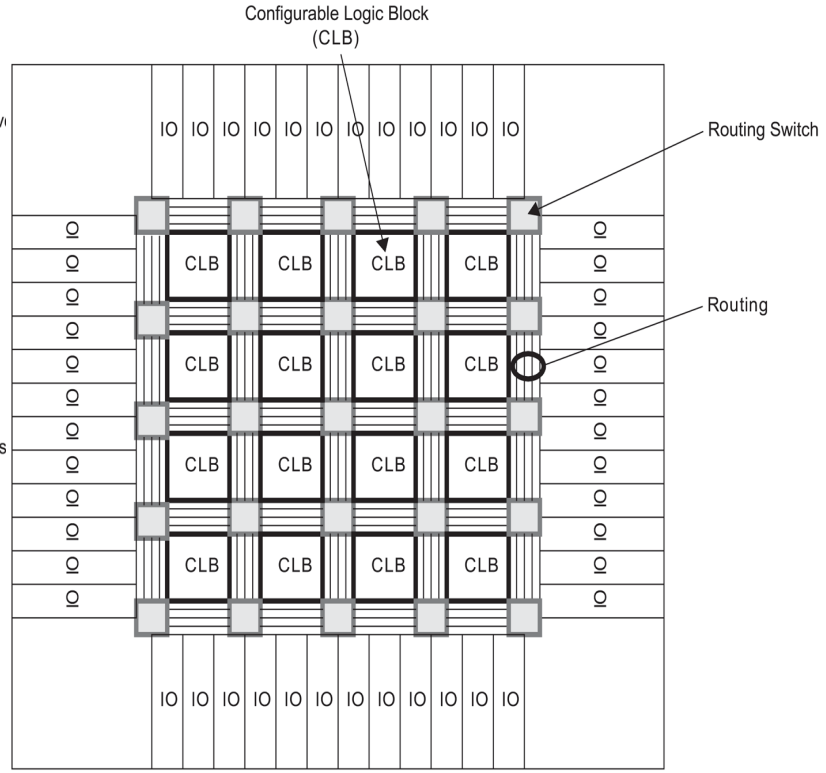


FIG 8.21 Simplified FPGA floorplan

See Fig. 8.22 for details of CLB!

Gate Array and Sea of Gates

- **To reduce nonrecurring engineering cost (NRE)**
  - Construct common base array of TRs (vendor) and personalize the chip by altering the metalization (per design basis)
  - Gate array (GA)
  - Sea-of-gate (SoG) (subclass of GA)
  - LSI RapidChip technology
    - RISC core + 10Mbit RAM + I/O up to 4.25Gbps → 300MHz

Gate Array and Sea of Gates

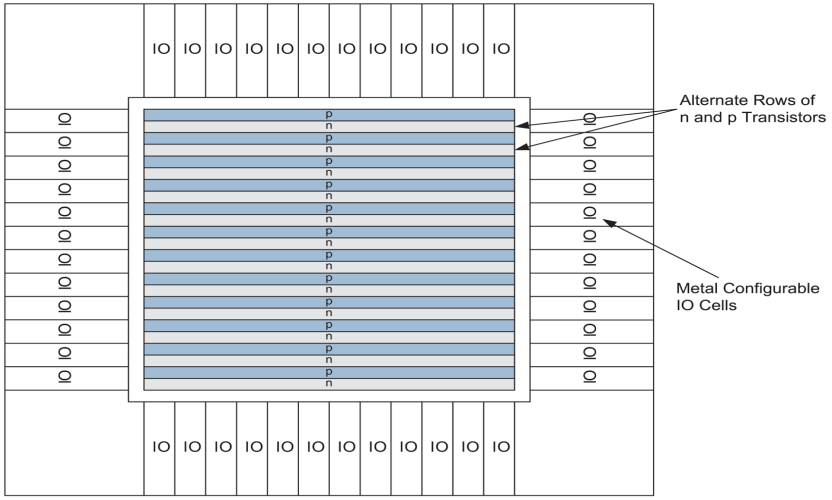


FIG 8.25 Sea of Gates floorplan and transistor layout

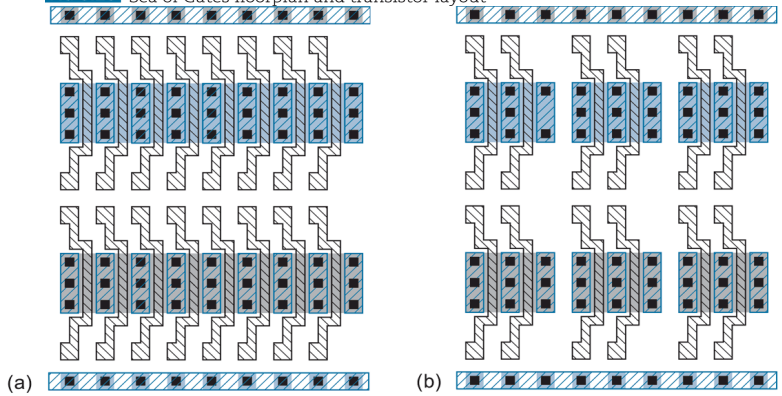


FIG 8.26 SOG and Gate Array cell layouts

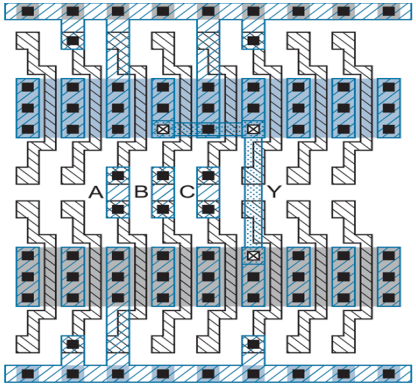


FIG 8.27 SOG programming example for 3-input NAND gate

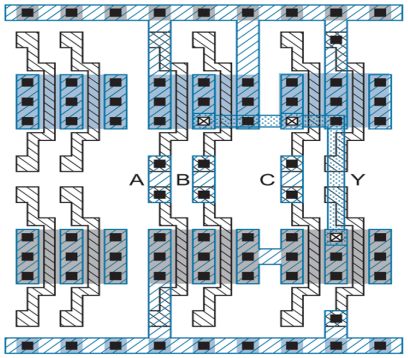


FIG 8.28 Gate Array programming example for 3-input NAND gate

Cell-based Design

- **Use a standard cell library as the basic building blocks of a chip**
  - Smaller, faster, and lower-power chips than GA or FPGA
  - High NRE costs for custom mask set → only economical for high volume parts
  - Much higher productivity than full-custom design
- **Foundries and library vendors supply cells with a wide range of functionality**
  - SSI (NAND, NOR, XOR, AOI, OAI, inverters, buffers, registers)
  - MSI (adders, multipliers, parity blocks,…)
  - Memories (RAM, ROM, CAM, register files)
  - System level module (*uPs*, interfaces, interfaces)
  - Possibility of mixed-signal and RF modules
  - See table 8.3, p 511

Full-Custom Design

- **Oldest and most traditional techniques: *custom mask layout!!***
  - Polygon pushing: laborious and prone to error
  - Symbolic layout
    - Use primitives: TRs, contacts, wires, ports
- **Analog and RF, cell libraries, memories and I/O cells still frequently use custom design**
- **Silicon complier: software generator to generate physical layout**

Platform-based

- **Increased system complexity → better to use predefined IP blocks**
- **Design frequently uses a number of common blocks**
  - RISC, memory, I/Os attached to common busses
- **Use *platform* by using common structures such as busses and common high level language to program the processor**

Design Methods: Summary

- See table 8.4 p. 519
- Design decision tradeoff

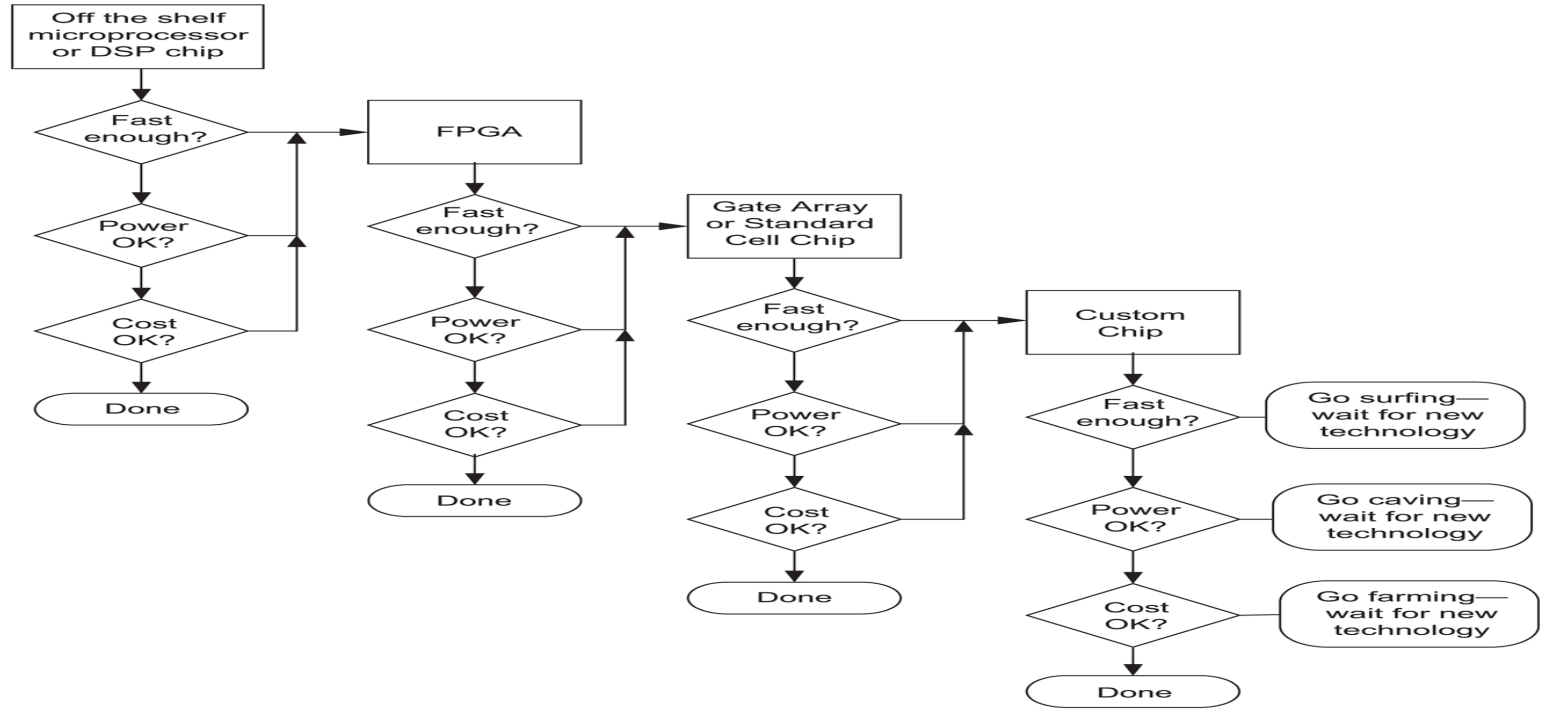


FIG 8.37 Design decision tradeoffs

Design Flows

- A set of procedure to progress from a spec. to final chip implementation in an error-free way
- General flow

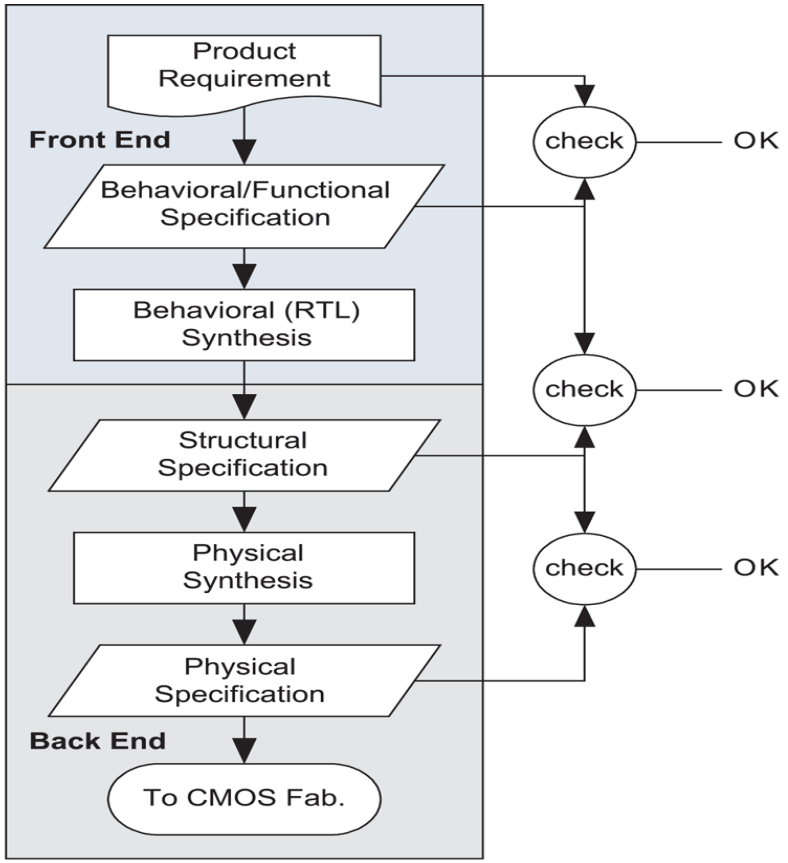


FIG 8.38 Generalized design flow

Behavioral Synthesis Flow

(ASIC Design Flow)

- From behavioral RTL description to a structural gate-level netlist

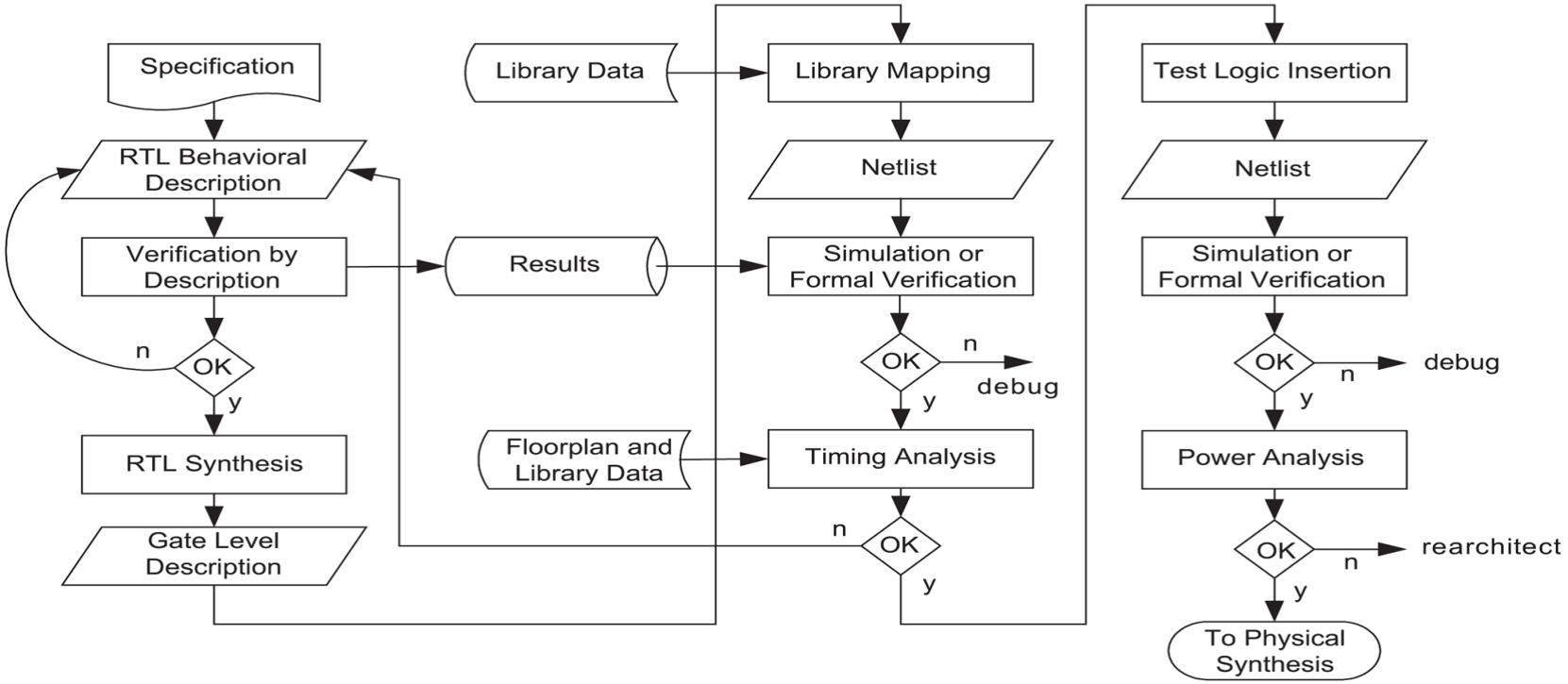


FIG 8.39 RTL synthesis flow

Logic Design and Verification

- **RTL description in an HDL and a set of test benches**
- **Tools**
  - Cadence: NC-verilog/SystemC/VHDL
  - Synopsys: VCS
  - Mentor Graphics: Modelsim
  - Aldec: ActiveHDL
- **Hierarchical functional verification**
  - usually from bottom up
- **see behavioral Verilog code p 523**

Logic Design and Verification

```
module nco_struct(input          fclock, reset,
                  input  [15:0]  initial_phase, phase_increment,
                  output [7:0] q);

   wire [6:0] ROM_data;
   wire [15:0] phase_0;
   wire [5:0] nbus_5;
   wire [7:0] wave_out;
   wire [5:0] nbus_2;
   wire [15:0] nbus_1;
   wire [15:0] phase;

   xor i_7(wave_out[7], phase[15], 1'b0);
   xor i_6(wave_out[6], phase[15], ROM_data[6]);
   .
   .
   .
   xor i_2(wave_out[2], phase[15], ROM_data[2]);
   .
   .
   not i_10(nbus_5[2], phase[10]);
   not i_9(nbus_5[1], phase[9]);
   not i_8(nbus_5[0], phase[8]);
   generic_flip_flop phase_reg_15(.q(phase[15]), .d(phase_0[15]),
      .clk(fclock));
   .
   .
   .
   .
   quarter_wave sine_table(.addr(the_address), .q(ROM_Table));
   .
endmodule
```

RTL Synthesis

- Convert RTL to generic gate and registers, then optimize the logic
- Also do state machine decomposition, datapath optimization, and power optimization
- **Commercial tools**
  - Synopsys: Design Compiler (Vision)
  - Cadence: BuildGates
  - Synplicity: Synplify Pro
- **see the synthesis code using the generic lib on p 524**

RTL Synthesis

```
module nco_struct(input          fclock, reset,
                  input  [15:0]  initial_phase, phase_increment,
                  output [7:0] q);

   wire [6:0] ROM_data;
   wire [15:0] phase_0;
   wire [5:0] nbus_5;
   wire [7:0] wave_out;
   wire [5:0] nbus_2;
   wire [15:0] nbus_1;
   wire [15:0] phase;

   xor i_7(wave_out[7], phase[15], 1'b0);
   xor i_6(wave_out[6], phase[15], ROM_data[6]);
   .
   .
   .
   xor i_2(wave_out[2], phase[15], ROM_data[2]);
   .
   .
   not i_10(nbus_5[2], phase[10]);
   not i_9(nbus_5[1], phase[9]);
   not i_8(nbus_5[0], phase[8]);
   generic_flip_flop phase_reg_15(.q(phase[15]), .d(phase_0[15]),
      .clk(fclock));
   .
   .
   .
   .
   quarter_wave sine_table(.addr(the_address), .q(ROM_Table));
   .
endmodule
```

Library Mapping

- **A generic HDL gate-level descript to a netlist using particular gates in the target library**
- **see the library mapping result on p 524**
  - Maps predefined blocks (memories) to their appropriate descriptions

Library Mapping

```
module nco_struct_mapped(input         fclock, reset,
                         input  [15:0] initial_phase, phase_increment,
                         output [7:0]  q);

   .
   .
   BUFX4 i_506(.A(n_355), .Y(q[7]));


  .
  MX2X1 i_00(.S0(reset), .B(initial_phase[15]), .A(nbus_1[15]),
     .Y(phase_0[15]));
  NAND2BX1 i_8(.AN(n_102), .B(n_101), .Y(n_104));
  XOR2X1 i_6(.A(phase[15]), .B(ROM_Table[6] ), .Y(n_103));
  .
  .
  DFFHQX1 phase_reg_0(.D(phase_0[15]), .CK(fclock), .Q(phase[15]));
  .
  .
  .
endmodule
```

Functional or Formal Verification

- **Check converted structural netlist = original behavioral HDL**
- **Verification strategies**
  1. Rerun the logic test bench and compare both results
  2. Formal verification
     - Compare logical equivalence mathematically
     - Tools are still maturing
     - Commercial tools
       - Synopsys: Formality
       - Cadence: Conformal
  3. Semantic and structural check on the HDL, eg.
     - Semantic: ensure all bus assignments match in bit width
     - Structural: make sure all outputs are connected

Static Timing Analysis

- **Check the *temporal* requirement**
- **Timing analyzer using actual gate timing is too slow**
- **Static timing analysis**
  - Use basic timing of library gates due to intrinsic gate delays and routing loads estimated from statistically or from floorplaning data
  - Runs quickly and exhaustively evaluates all timing paths
  - Check for both *max-delay* (for setup) and *min-delay* (for hold time)
  - Suffers from false path problems
    - Designer must manually flag such paths
  - Commercial tools
    - Cadence: Pearl
    - Synopysis: Pathmill and PrimeTime
- **see the static timing analysis result on p 527**
  - What's the slack time?

Static Timing Analysis

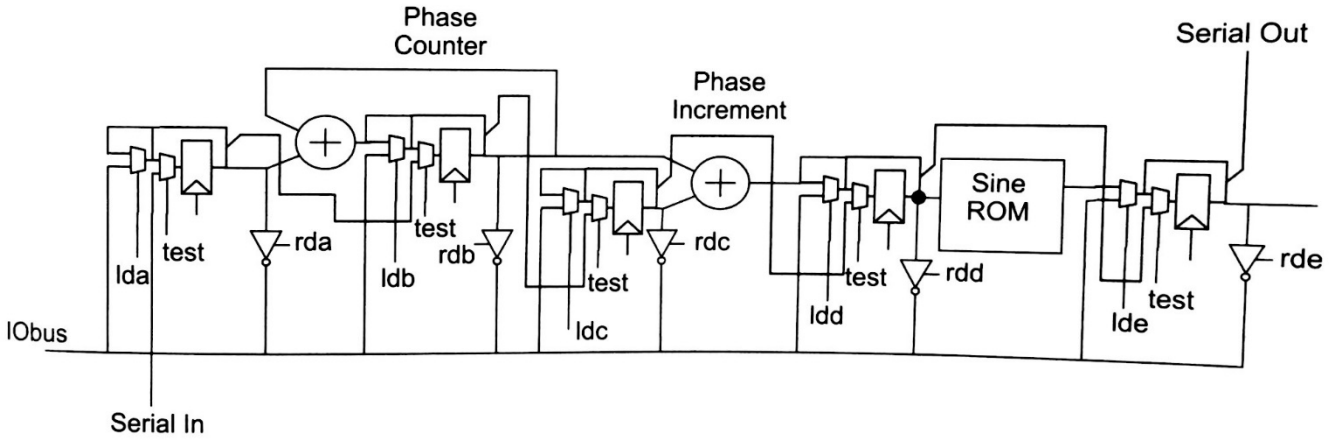
```
Beginpoint: phase_reg_14/Q (^) triggered by leading edge of 'clock'
Required Time 11.00
- Arrival Time 5.89
= Slack Time 5.11
  Clock Rise Edge 0.00
  + Source Insertion Delay 0.20
  = Beginpoint Arrival Time 0.20
```

Instance	Arc	Cell	Delay	Arrival Time	Required Time
	fclock ^			0.20	5.31
phase_reg_14	CK ^ -> Q ^	DFFHQX1	0.58	0.78	5.89
i_34	A ^ -> Y ^	XOR2X1	0.78	1.57	6.68
sine_table	addr[3] ^	quarter_wave		1.57	6.68
sine_table/i_371	A ^ -> Y v	INVX1	0.45	2.02	7.13
sine_table/i_42	B v -> Y ^	NOR2X1	0.19	2.21	7.32
sine_table/i_44	AN ^ -> Y ^	NAND2BX1	0.16	2.37	7.48
sine_table/i_145	A0 ^ -> Y v	AOI211X1	0.10	2.47	7.58
sine_table/i_6	C0 v -> Y ^	OAI211X1	0.13	2.61	7.72
sine_table/i_484	A ^ -> Y ^	BUFX4	0.96	3.57	8.68
sine_table	q[0] ^	quarter_wave		3.57	8.68
i_120	B ^ -> Y v	NOR2BX1	0.22	3.79	8.90
i_7	AN v -> Y v	NOR2BX1	0.20	3.99	9.10
i_9	AN v -> Y v	NOR2BX1	0.21	4.20	9.31
i_20	B v -> Y ^	NAND2BX1	0.12	4.32	9.43
i_3	AN ^ -> Y ^	NAND2BX1	0.17	4.49	9.60
i_8	AN ^ -> Y ^	NAND2BX1	0.19	4.67	9.78
i_31	A ^ -> Y ^	XNOR2X1	0.28	4.96	10.07
i_504	A ^ -> Y ^	BUFX4	0.93	5.89	11.00
	q[6] ^		0.00	5.89	11.00

Test Insertion

- **Insert scannable registers or optionally ATPG used to generate tests for a scannable design**
- **BIST to allow *in situ* testing**
- **Commercial tools**
  - Synopsys: DFT complier
  - Tetramax: ATPG
  - LogicVision: Logic BIST and Memory BIST
- **see the scan register insertion for testing on p 527**

Test Insertion



Power Analysis

- **Depends on the activity factors of the gates → depends on the inputs the chip receives**
- **Can be performed for a particular set of test vectors by running a simulator**
  - Evaluate the total capacitance switched at each clock transition at each node
  - If too high, return to architectural level
- **Commercial tools**
  - Synopsys: PrimePower and Powermill

Interchange Formats

- **Mostly text interchange formats**
- GDS2 stream
  - *De facto* standard binary format for describing mask geometry
- Caltech Intermediate Format (CIF)
  - Alternative text-based mask description language (academic use)
- Library Exchange Format (LEF)
  - Describe the physical attributes of library cells
  - Abstract the lower-level geometric details of a cell
  - See the example on p 559

Interchange Formats

```
LAYER metal1
   TYPE ROUTING ;
   WIDTH 0.2 ;
   SPACING 0.4 ;
   PITCH 1.0 ;
   DIRECTION HORIZONTAL ;
   CAPACITANCE CPERSQDIST 0.00003 ;
END metal 1
...
MACRO adc
   ORIGIN 0 0 ;
   SIZE 100 BY 200 ;
   PIN in
   DIRECTION INPUT ;
   PORT
   LAYER METAL4 ;
   RECT 10 10 10.5 10.5
   END in
   ...
   OBS
   LAYER METAL1 ;
   RECT 0 100 20 150 ;
   END
END adc
```

Interchange Formats

- Design Exchange Format (DEF)
  - Describe an actual design by listing the library elements and their placement and connectivity
  - See the example on p 560
- Standard Delay Format (SDF)
  - IEEE standard (P1497) to describe timing information
  - Specify pin-to-pin delays of modules, clock-to-data delays, and interconnect delays
  - See the example on p 560
  - INTERCONNECT clk_pad (0.1:0.2:0.3) (0.4:0.5:0.6)
    - Rising and falling delays, (slow:normal:fast) process corners
- Detailed Standard Parasitic Format (DSPF) and Standard Parasitic Exchange Format (SPEF)
  - Use to pass parasitic RC values between extraction tools and timing verification tools
  - See the example on p 561

Interchange Formats

✓ **DEF**

```
VERSION 5.2 ;
NAMESCASESENSITIVE ON ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN chip ;
UNITS DISTANCE MICRONS 1000 ;

COMPONENTS 1 ;
-adc_inst adc + FIXED ( -1000 -1000) N ;
END COMPONENTS

END DESIGN
```

✓ **SDEF**

```
(DELAYFILE
(SDFVERSION     "2.1")
(DESIGN         "chip")
(DATE           "March 24, 2003 11:57:3")
(VENDOR         "")
(PROGRAM        "PEARL")
(VERSION        "PEARL 5.1-s072 (64 bit)")
(DIVIDER        /)
(VOLTAGE        1.080:1.20:1.320)
(PROCESS        "slow=1.5:nom=1.0:fast=0.75")
(TEMPERATURE    80.000:25.000:-40.000)
(TIMESCALE      1ns)
(CELL
(CELLTYPE       "chip")
(INSTANCE)
(DELAY
   (ABSOLUTE
      (INTERCONNECT clk_mdi u_pad_clk_mdi/PAD (0.1:0.2:0.3) (0.4:0.5:0.6))
      (INTERCONNECT u_pad_clk_mdi/PAD clk_mdi (0.4:0.5:0.6) (0.7:0.8:0.9))
      )
      )
   (CELL
      (CELLTYPE "DFlipFlop")
      (INSTANCE foo/bar/reg)
      (DELAY
          (ABSOLUTE
             (IOPATH CK Q (.2:.25:.3) (.3:.35:.4))
          )
       )
    (TIMINGCHECK
       (WIDTH (posedge CK) (.06:.06:.06))
       (WIDTH (negedge CK) (.12:.12:.12))
       (SETUP (posedge D) (posedge CK) (0.1:0.1:0.1))
       )
    )
 )
```

Interchange Formats

- ✓ **SPEF**

```
*|DSPF 1.5
*|DESIGN "chip"
*|DATE "Mon Mar 24 07:26:34 2003"
*|VENDOR "Cadence Design Systems – HLD"
*|PROGRAM "hyperExtract 4.5.0"
*|VERSION "3.4E"
*|DIVIDER |
*|DELIMITER .
*|BUSBIT []

.SUBCKT chip in out
+ Vdd gnd

*Net Section
*
*|GROUND_NET gnd
*
*|NET name1 0.003PF
*I ???
*S ????
C1 node1 node2 1.2
C2 node3 node4 0.85
R1 node4 node5 0.5
*
.ENDS
```

Interchange Formats

- Advanced Library Format (ALF)
  - An alternate format used to describe primitive library elements
- WAVES Waveform and Vector Exchange Specification
  - WAVES: IEEE Std. 1029.1 and subset of the VHDL standard
  - Provides for the graphical definition of stimulus and response patterns
  - See the example on p 562
- Physical Design Exchange Format (PDEF)
  - Synopsys format to pass information between front-end and back-end tools
- **OpenAccess**
  - Provides API or set of software routines that communicate with a central database and extract data and insert data into that database
  - OpenAccess API for LEF and DEF descriptions

Interchange Formats

```
% ************************************************************
% * This file is automatically generated WAVES vector file,   *
% * and can be used in Test Bench generator.                  *
% * ACTIVE-HDL Testbench Generator ver. 3.5.                  *
% * Copyright (C) ALDEC Inc.                                  *
% *                                                           *
% * This file was generated on: 10:29:49 AM 12/9/2003         *
% ************************************************************
%
% Begin Comment
%
% reset : sig reg
% clk : sig reg
%
% 4.00 us : END_SIMULATION_TIME
% End Comment
%
% Begin of Vectors
%
%@ 0 0

 x 0 : 15 ns ; % 0 fs
 x 1 : 15 ns ; % 15 ns
 1 0 : 15 ns ; % 30 ns
 1 1 : 15 ns ; % 45 ns
 .
 1 0 : 15 ns ; % 330 ns
 1 1 : 15 ns ; % 345 ns
 1 0 : 15 ns ; % 360 ns
 1 1 : 15 ns ; % 375 ns
 0 0 : 15 ns ; % 390 ns
 0 1 : 15 ns ; % 405 ns
 0 0 : 15 ns ; % 420 ns
 0 1 : 15 ns ; % 435 ns
 0 0 : 15 ns ; % 450 ns
 0 1 : 15 ns ; % 465 ns
```

Automatic Layout Generation

- **Why we need automatic?**
  - For high productivity for most large digital chips with moderate performance (ASICs)
  - For high end ICs like Intel CPUs → need custom design
- **Physical synthesis when the structural netlist is converted to physical layout**
- **Two methods**
  - Software generator (see sec. 8.3.5)
  - Standard cell place and route (used on most ASICs) ← focused here

Standard cell P&R flow

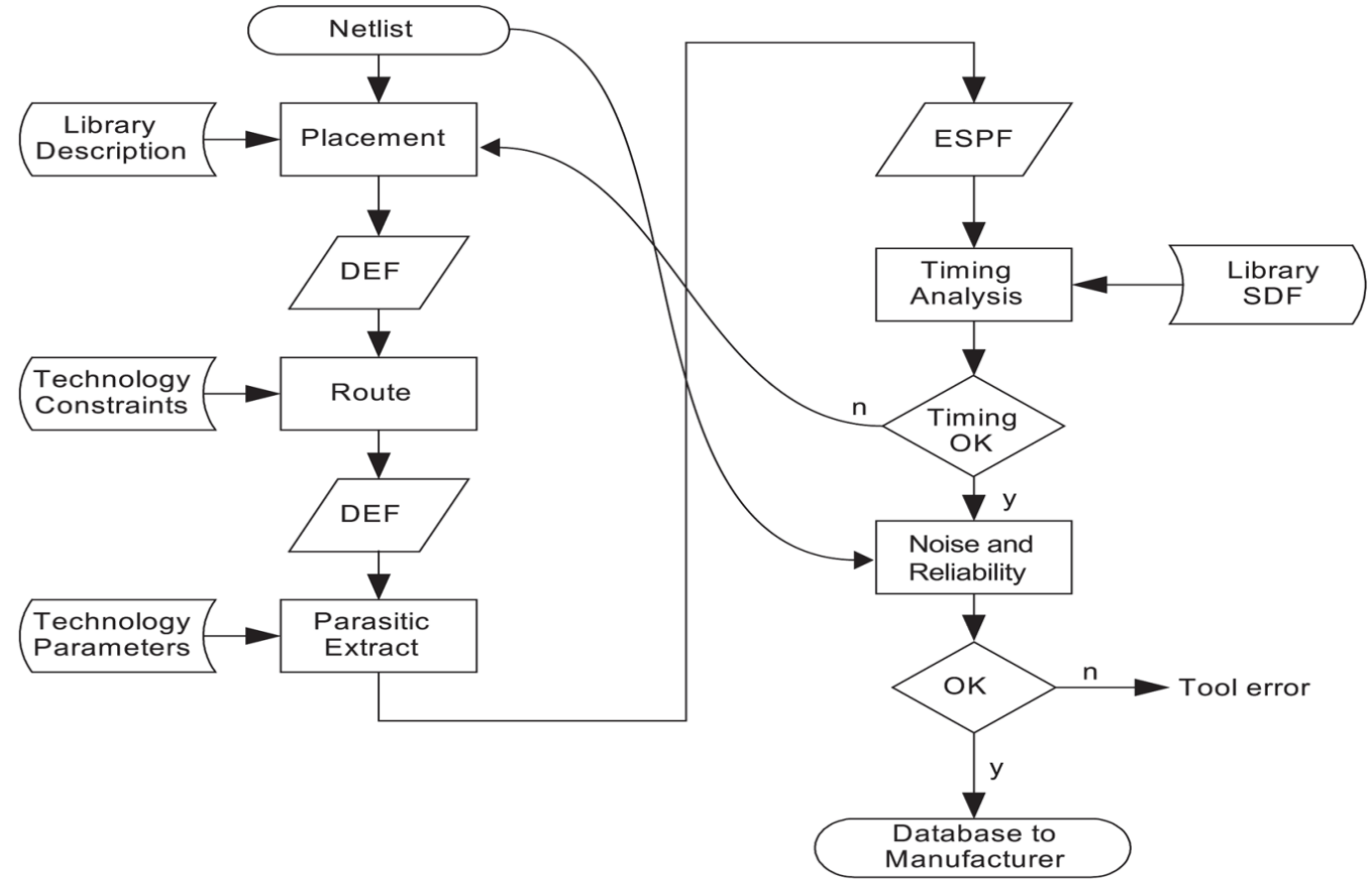


FIG 8.41 Standard cell place and route design flow

Standard cell P&R flow

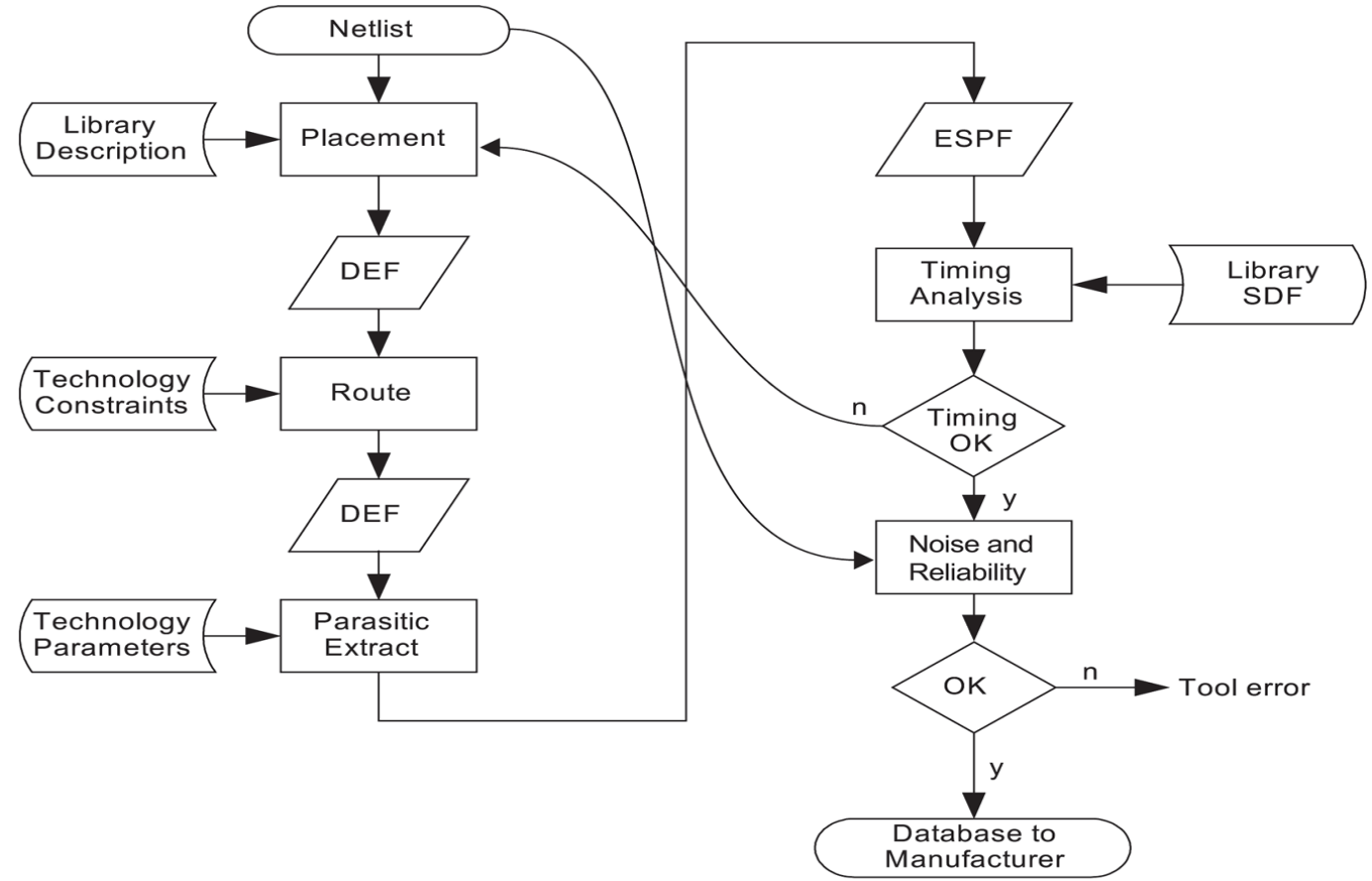


FIG 8.41 Standard cell place and route design flow

Placement

- **Constant height and variable-width cells**
- **Can add application specific custom block: memory, RF,..**
  - "flow" around
- **No separation between stand cell rows**
  - Rounting over the cells
- **Floorplanning**
  - Hierarchical manual floorplaning to increase locality before placement (optional)
  - See sec. 8.2.5

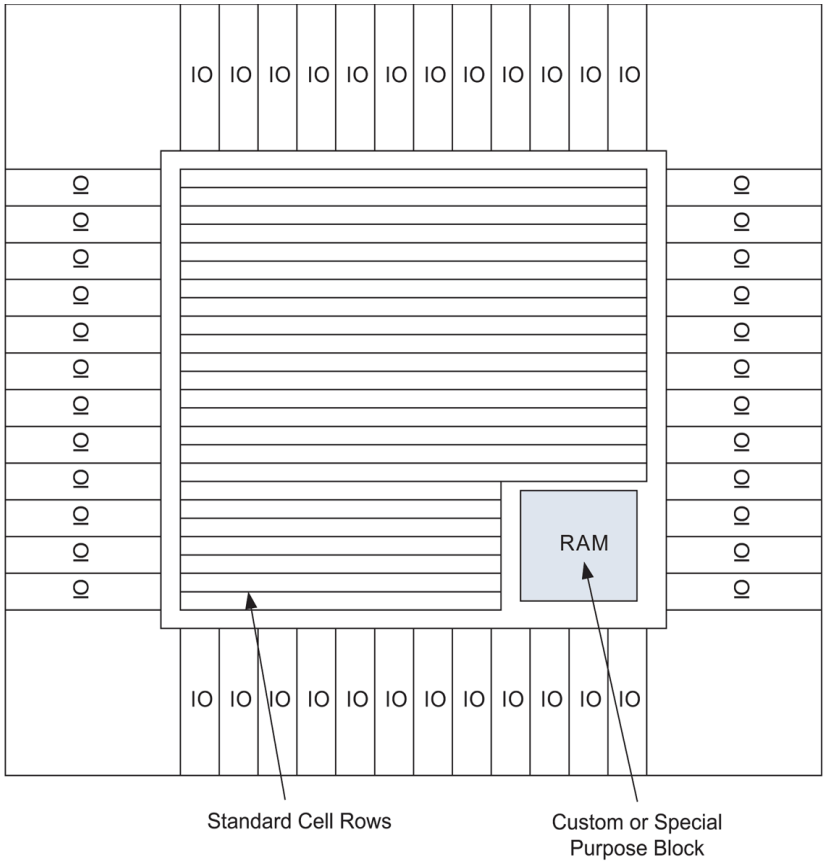


FIG 8.42 Standard cell chip layout

Routing

- **Two steps: global routing and detailed routing**
- **Global routing**
  - Abstract the routing problem to a notional set of abutting channels
- **Detailed routing**
  - Add wires to channels to complete signal connections

Parasitic Extraction

- Extract the parasitic RC associated with all nets in the layout after P&R
- 2D, 2.5D, 3D extractor

Timing Analysis

- **Rerun static timing analysis with actual routing loads placed on the gates**
  - Bottleneck!!
- **Usually multiple iteration of P&R to meet timing requirement**
- **If possible, a transistor-level timing simulation should be run**
  - Somewhat reduced transistor modeling accuracy
  - Almost SPICE accuracy
- **Commercial tools**
  - Synopsys: Pathmill and Nanosim
  - Cadence: UltraSim

Noise, VDD and Electromigration Analysis

- **Noise analysis to evaluate crosstalk due to interlayer routing capacitance**
- **Commercial tools**
  - Cadence: SignalStrorm, ElectronStorm, anc VoltageStorm

Timing-driven Placement

- **After the layout is completed, and extracting parasitic capacitance, if timing problem is found, we have to placement again → out of control for other complex parts in the chip!!**
- **The solution: timing-driven placement**
  - Takes into account the timing of the circuits as cells are placed
  - Cells on critical paths are given priority to minimize the wire delay

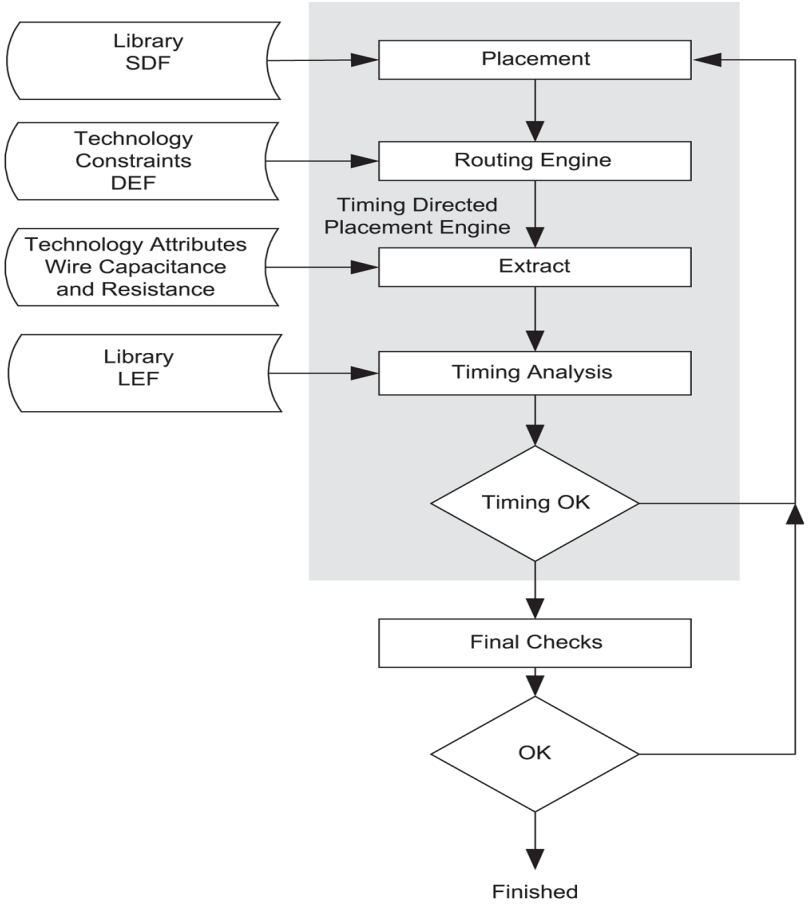


FIG 8.43 Timing directed placement design flow

Clock-tree Routing

- To minimize skew, it is often best to route the clock and its buffers before the main logic placement and routing are completed
- Many approaches to the clock distribution strategies

Power Analysis

- Now, real wire capacitances are available, and the power estimation can be repeated
- Similar techniques to those used during RTL synthesis are used

Mixed-signal or Custom-design Flow

- For small analog, RF, and high-speed digital IC designs, we use a custom-design flow

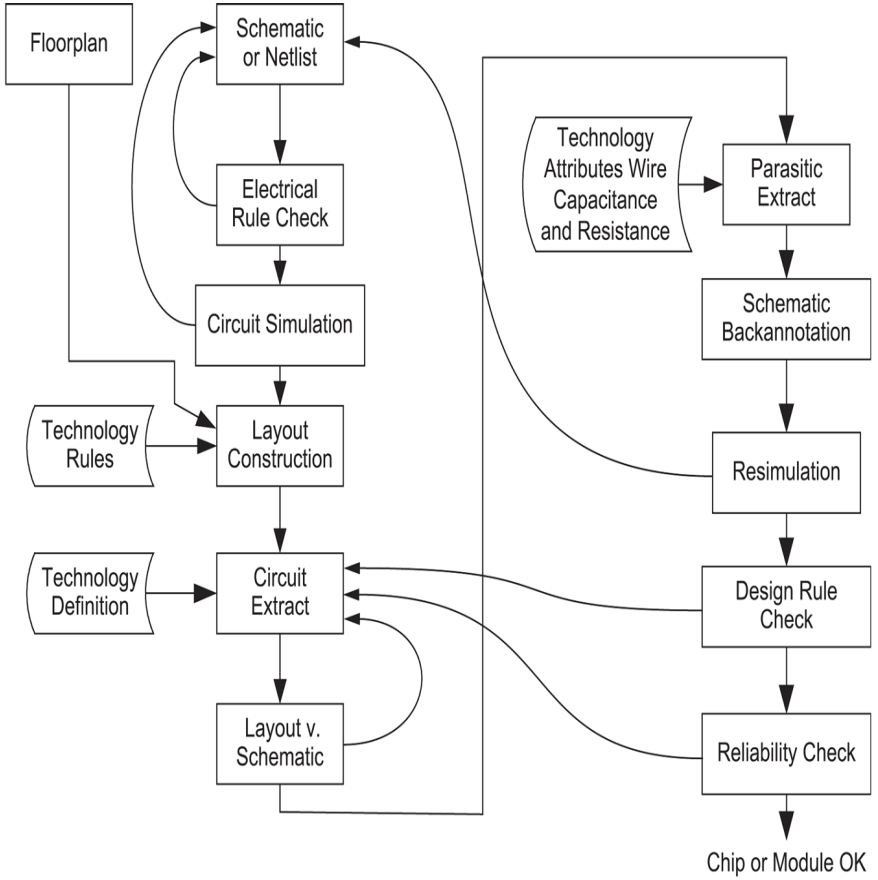


FIG 8.44 Mixed-signal or custom-design flow

Data Sheets and Documentation

- **Describe what it does and outlines the specs for making the IC work in a system**
- **Summary**
  - Designation and descriptive name of the chip
  - Concise description of what the chip does
  - A feature list
  - High-level block diagram of the chip function
- **Pinout**
  - Name of the pin, type of the pin, brief description of the pin function, and the package pin number
- **Description of Operation**

Data Sheets and Documentation

- **DC Specs**
  - Supply voltage, pin voltages, junction temperature,
  - V_{IL} and V_{IH}, V_{OL} and V_{IH}
  - Input loading for each input
  - Quiescent current, leakage current,…
- **AC specs**
  - Setup and hold times on all input
  - Clock to output delay times
  - Other critical timing such as minimum pulse widthds
- **Package diagram**
- **Principles of operational manual**
- **User manual**

Closing the Gap between ASIC and Custom

- **High speed *uPs* and DSP**
  - Custom design with which designers use a wide variety of circuit techniques and carefully tune all the critical parts of the chip
- **ASICs**
  - Built by automatic HDL synthesis and layout using a standard cell library
- **Performance gap of 180nm process**
  - Ups: 1-2 GHz
  - ASIC: 200-350 MHz
- **Overall, custom design = 3-8x frequency advantage over ASICs**
- **Delays**
  - ASICs: 50-100 FO4
  - Pentium II and III: 20-24 FO4
  - Pentium 4: 10 FO4
- **See table 8.5 p 548**

Closing the Gap between ASIC and Custom

Table 8.5 Relative performance of custom vs. ASIC design methodologies

Factor	vs. Poor ASIC	vs. Best Practice ASIC
Microarchitecture (e.g., pipelining)	1.8x	1.3x
Sequencing overhead: elements, skew, time borrowing	1.45x	1.1x
Circuit families (e.g., domino)	1.4x	1.2x
Logic design	1.3x	1.0x
Cell design, cell sizing, and wire sizing	1.45x	1.1x
Layout: floorplanning, placement, wire management	1.4x	1.0x
Exploiting process variation and accessibility	2x	1.2x

What are the factors?

- Microarchitecture
  - Pipelining or performing multiple operations simultaneously
  - ASICs: greater sequencing overhead
    - Most ASICs use very conservative FFs: 4-6 FO4 delay
    - Cannot benefit from pipelining
- Sequencing overhead
  - uPs: use more aggressive sequential circuits such as latches, 2-3 FO4 delay
    - Better for clock skew tolerance
- Circuit families
  - Most custom designs use skew-tolerant domino circuits on critical paths
    - 1.5-2x faster than CMOS

What are the factors?

- Logic design
  - Custom: optimized logic design, eg., lookahead adder not ripple carry adders and Booth-encoded multipliers
  - ASICs: synthesized from HDL → older tools: should specify manually Booth multipliers
- Cell and Wire design
  - ASICs: use a standard library. some library has a small selection of gates. → may be 25% slower than a rich selection!!
  - Custom: unlimited set of cell sizes
  - ASIC designer have little control over the wire selection
- Layout

What are the factors?

- Process variation
  - ASICs: designed for the SS process corner so that all of the chips will meet timing
    - long trail at the slow end
  - Custom: designed for the TT process
    - 17%-28% speedup at the expense of rejecting a few chips
    - Fastest 10% of the chips may be up to 30% faster than TT
    - Ups are commonly shorted by speed, and the fastest ones are sold at a premium ← *binning*

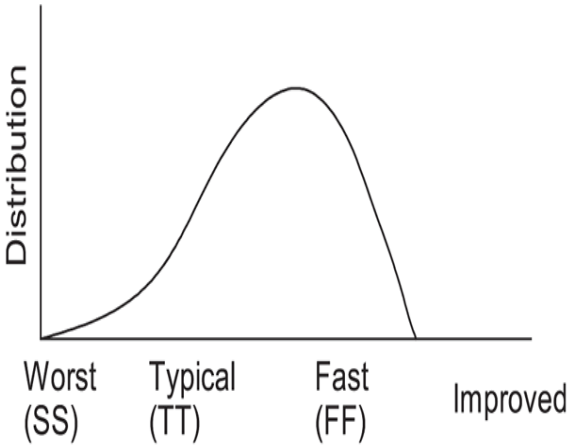


FIG 8.48 Distribution of chip behavior