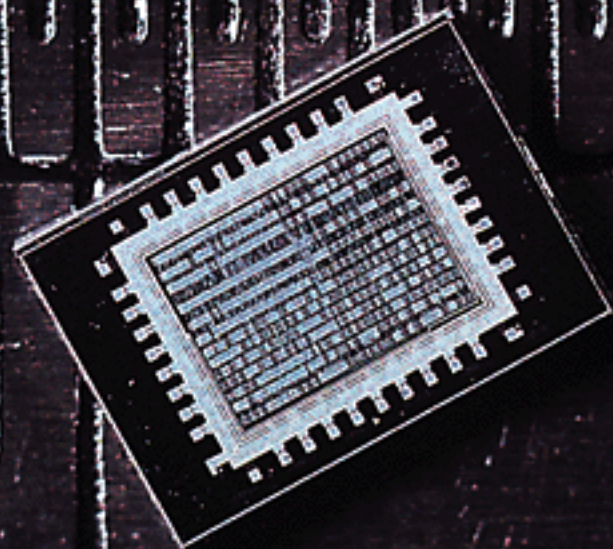


Application-Specific Integrated Circuits

Michael John Sebastian Smith



ASICs... the course

Michael John Sebastian Smith

This course is based on **ASICs... the book**

Application-Specific Integrated Circuits
Michael J. S. Smith
VLSI Design Series
1,040 pages
ISBN 0-201-50022-1
LOC TK7874.6.S63
Addison Wesley Longman, <http://www.awl.com>

Additional material (figures, resources, source code) is located at
ASICs... the website

<http://spectra.eng.hawaii.edu/~msmith/ASICs/HTML/ASICs.htm>

Some material in this work is reprinted from IEEE Std 1149.1-1990, "IEEE Standard Test Access Port and Boundary-Scan Architecture," Copyright © 1990; IEEE Std 1076/INT-1991 "IEEE Standards Interpretations: IEEE Std 1076-1987, IEEE Standard VHDL Language Reference Manual," Copyright © 1991; IEEE Std 1076-1993 "IEEE Standard VHDL Language Reference Manual," Copyright © 1993; IEEE Std 1164-1993 "IEEE Standard Multivalued Logic System for VHDL Model Interoperability (Std_logic_1164)," Copyright © 1993; IEEE Std 1149.1b-1994 "Supplement to IEEE Std 1149.1-1990, IEEE Standard Test Access Port and Boundary-Scan Architecture," Copyright © 1994; IEEE Std 1076.4-1995 "IEEE Standard for VITAL Application-Specific Integrated Circuit (ASIC) Modeling Specification," Copyright © 1995; IEEE 1364-1995 "IEEE Standard Description Language Based on the Verilog[®] Hardware Description Language," Copyright © 1995; and IEEE Std 1076.3-1997 "IEEE Standard for VHDL Synthesis Packages," Copyright © 1997; by the Institute of Electrical and Electronics Engineers, Inc. The IEEE disclaims any responsibility or liability resulting from the placement and use in the described manner. Information is reprinted with the permission of the IEEE. Figures describing Xilinx FPGAs are courtesy of Xilinx, Inc. ©Xilinx, Inc. 1996, 1997, 1998. All rights reserved. Figures describing Altera CPLDs are courtesy of Altera Corporation. Altera is a trademark and service mark of Altera Corporation in the United States and other countries. Altera products are the intellectual property of Altera Corporation and are protected by copyright laws and one or more U.S. and foreign patents and patent applications. Figures describing Actel FPGAs are courtesy of Actel Corporation.

The programs and applications presented in this work have been included for their instructional value. They have been tested with care but are not guaranteed for any particular purpose. The author does not offer any warranties, representations, or accept any liabilities with respect to the programs or applications.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this work, and the author was aware of a trademark claim, the designations have been printed in initial caps or all caps.

Figures copyright © 1997 by Addison Wesley Longman, Inc. Text copyright © 1997, 1998 by Michael John Sebastian Smith.

INTRODUCTION TO ASICs

1

Key concepts: The difference between full-custom and semicustom ASICs • The difference between standard-cell, gate-array, and programmable ASICs • ASIC design flow • Design economics • ASIC cell library

An **ASIC** (“a-sick”) is an **application-specific integrated circuit**

A **gate equivalent** is a NAND gate $F = \overline{A \cdot B}$ (IBM uses a NOR gate), or four transistors

History of integration: **small-scale integration (SSI)**, ~10 gates per chip, 60’s), **medium-scale integration (MSI)**, ~100–1000 gates per chip, 70’s), **large-scale integration (LSI)**, ~1000–10,000 gates per chip, 80’s), **very large-scale integration (VLSI)**, ~10,000–100,000 gates per chip, 90’s), **ultralarge scale integration (ULSI)**, ~1M–10M gates per chip)

History of technology: **bipolar technology** and **transistor–transistor logic (TTL)** preceded **metal-oxide-silicon (MOS)** technology because it was difficult to make metal-gate n-channel MOS (**nMOS** or **NMOS**); the introduction of **complementary MOS (CMOS)**, never cMOS) greatly reduced power

The **feature size** is the smallest shape you can make on a chip and is measured in μm or **lambda**

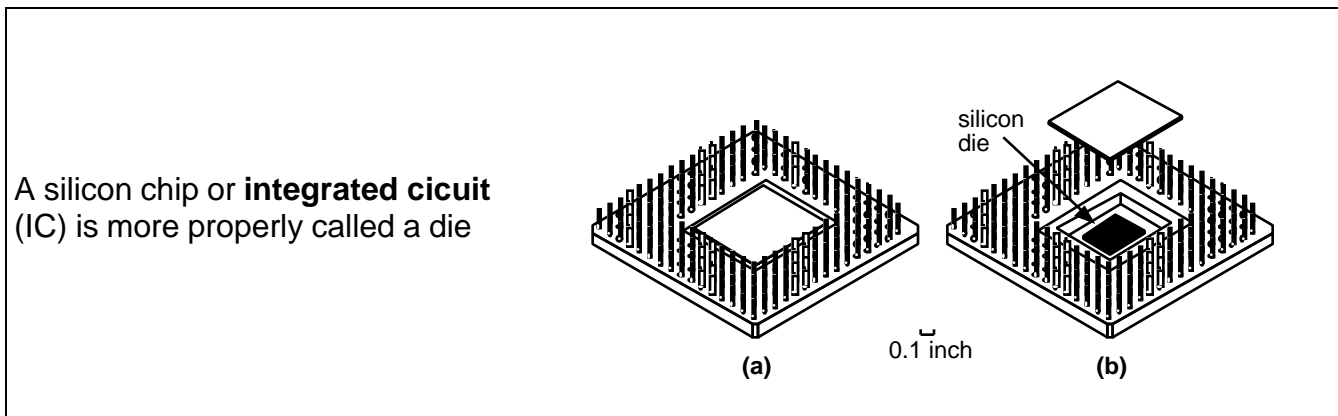
Origin of ASICs: the **standard parts**, initially used to design **microelectronic systems**, were gradually replaced with a combination of **glue logic**, **custom ICs**, **dynamic random-access memory (DRAM)** and **static RAM (SRAM)**

History of ASICs: The *IEEE Custom Integrated Circuits Conference (CICC)* and *IEEE International ASIC Conference* document the development of ASICs

Application-specific standard products (ASSPs) are a cross between standard parts and ASICs

1.1 Types of ASICs

ICs are made on a **wafer**. Circuits are built up with successive **mask layers**. The number of **masks** used to define the **interconnect** and other layers is different between **full-custom ICs** and **programmable ASICs**



1.1.1 Full-Custom ASICs

All mask layers are customized in a **full-custom ASIC**.

It only makes sense to design a full-custom IC if there are no libraries available.

Full-custom offers the highest performance and lowest part cost (smallest die size) with the disadvantages of increased design time, complexity, design expense, and highest risk.

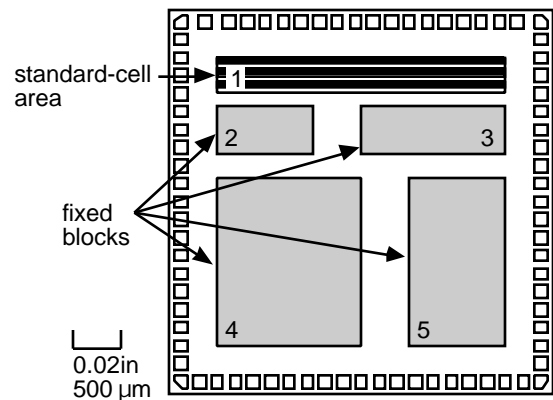
Microprocessors were exclusively full-custom, but designers are increasingly turning to semicustom ASIC techniques in this area too.

Other examples of full-custom ICs or ASICs are requirements for high-voltage (automobile), analog/digital (communications), or sensors and actuators.

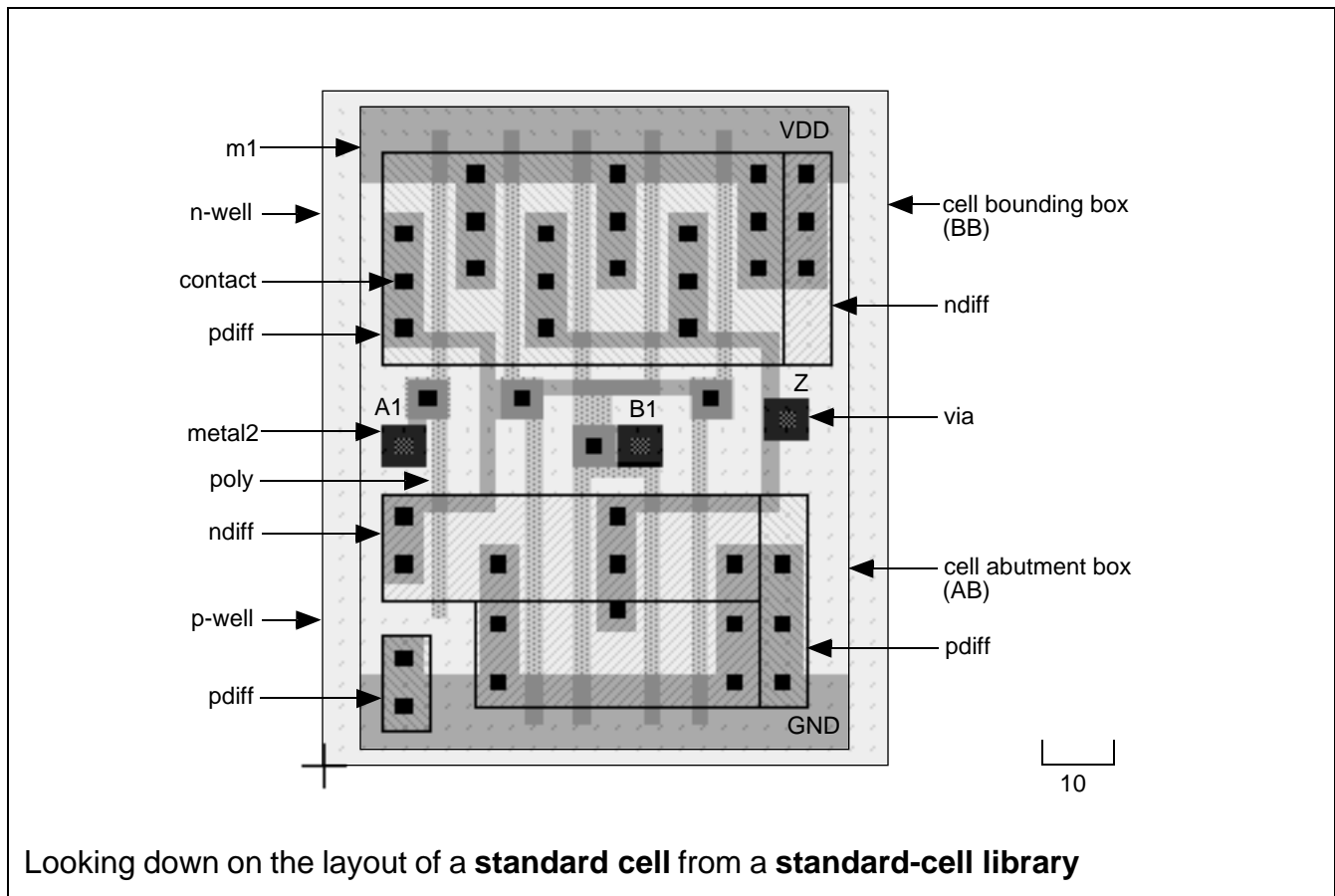
1.1.2 Standard-Cell-Based ASICs

A **cell-based ASIC (CBIC—“sea-bick”)**

- Standard cells
- Possibly **megacells, megafunctions, full-custom blocks, system-level macros (SLMs), fixed blocks, cores, or Functional Standard Blocks (FSBs)**
- All mask layers are customized—transistors and interconnect
- Custom blocks can be embedded
- Manufacturing lead time is about eight weeks.



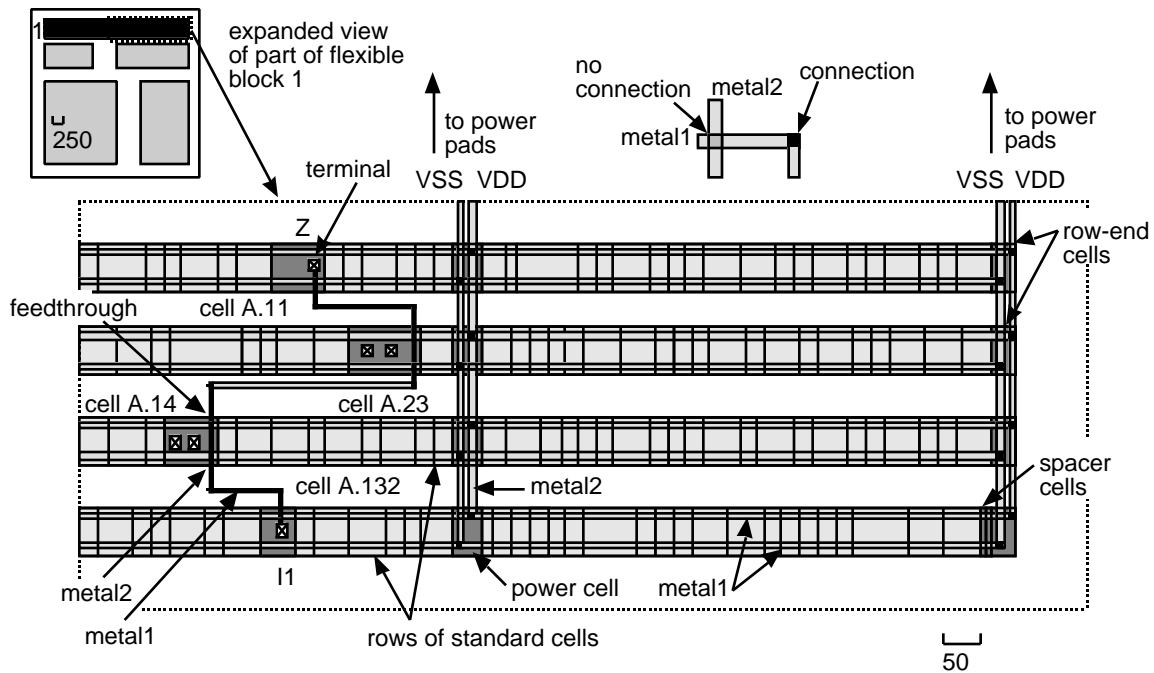
In **datapath (DP)** logic we may use a **datapath compiler** and a **datapath library**. Cells such as **arithmetic and logical units (ALUs)** are **pitch-matched** to each other to improve timing and density.



1.1.3 Gate-Array–Based ASICs

A **gate array**, **masked gate array**, **MGA**, or **prediffused array** uses **macros (books)** to reduce **turnaround time** and comprises a **base array** made from a **base cell** or **primitive cell**. There are three types:

- Channeled gate arrays
- Channelless gate arrays
- Structured gate arrays



Routing a CBIC (cell-based IC)

- A “wall” of standard cells forms a **flexible block**
- **metal2** may be used in a **feedthrough cell** to cross over cell rows that use **metal1** for wiring
- Other wiring cells: **spacer cells**, **row-end cells**, and **power cells**

A note on the use of hyphens and dashes in the spelling (orthography) of compound nouns: Be careful to distinguish between a “high-school girl” (a girl of high-school age) and a “high school girl” (is she on drugs or perhaps very tall?).

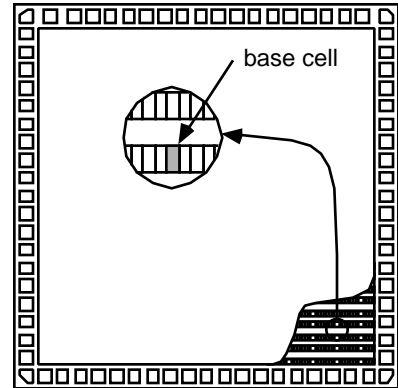
We write “channeled gate array,” but “channeled gate-array architecture” because the *gate array* is *channeled*; it is not “channeled-gate array architecture” (which is an array of channeled-gates) or “channeled gate array architecture” (which is ambiguous).

We write gate-array–based ASICs (with a en-dash between array and based) to mean (gate array)-based ASICs.

1.1.4 Channeled Gate Array

A **channeled gate array**

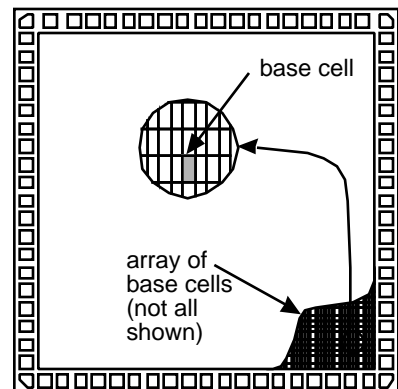
- Only the interconnect is customized
- The interconnect uses predefined spaces between rows of base cells
- Manufacturing lead time is between two days and two weeks



1.1.5 Channelless Gate Array

A **channelless gate array (channel-free gate array, sea-of-gates array, or SOG array)**

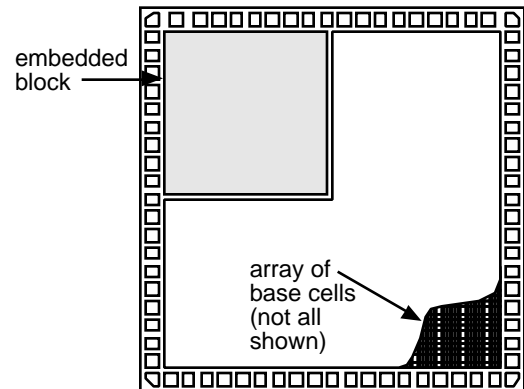
- Only some (the top few) mask layers are customized—the interconnect
- Manufacturing lead time is between two days and two weeks.



1.1.6 Structured Gate Array

An **embedded gate array** or **structured gate array (masterslice or masterimage)**

- Only the interconnect is customized
- Custom blocks (the same for each design) can be embedded
- Manufacturing lead time is between two days and two weeks.



1.1.7 Programmable Logic Devices

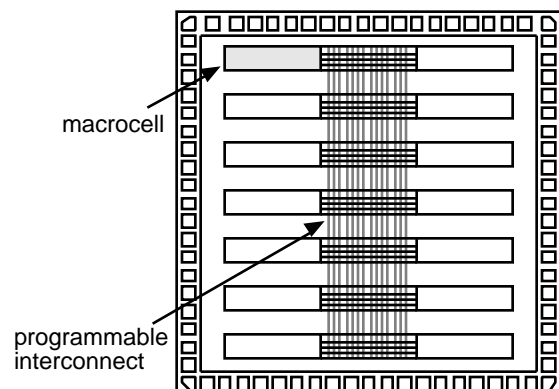
Examples and types of PLDs: **read-only memory (ROM)** • **programmable ROM** or **PROM** • **electrically programmable ROM**, or **EPROM** • An **erasable PLD (EPLD)** • **electrically erasable PROM**, or **EEPROM** • **UV-erasable PROM**, or **UVPROM** • **mask-programmable ROM**

• A **mask-programmed PLD** usually uses bipolar technology

Logic arrays may be either a **Programmable Array Logic (PAL[®])**, a registered trademark of AMD) or a **programmable logic array (PLA)**; both have an **AND plane** and an **OR plane**

A **programmable logic device (PLD)**

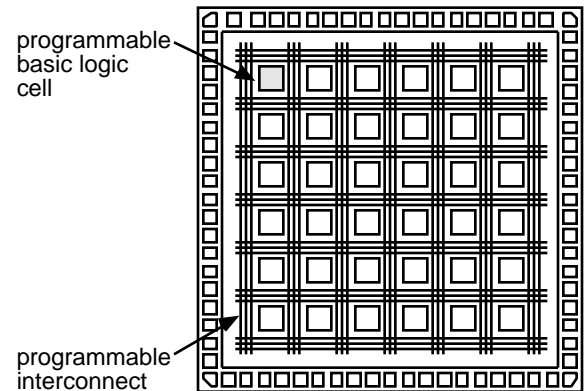
- No customized mask layers or logic cells
- Fast design turnaround
- A single large block of programmable interconnect
- A matrix of logic macrocells that usually consist of programmable array logic followed by a flip-flop or latch



1.1.8 Field-Programmable Gate Arrays

A **field-programmable gate array (FPGA)** or **complex PLD**

- None of the mask layers are customized
- A method for programming the basic logic cells and the interconnect
- The core is a regular array of programmable basic logic cells that can implement combinational as well as sequential logic (flip-flops)
- A matrix of programmable interconnect surrounds the basic logic cells
- Programmable I/O cells surround the core
- Design turnaround is a few hours



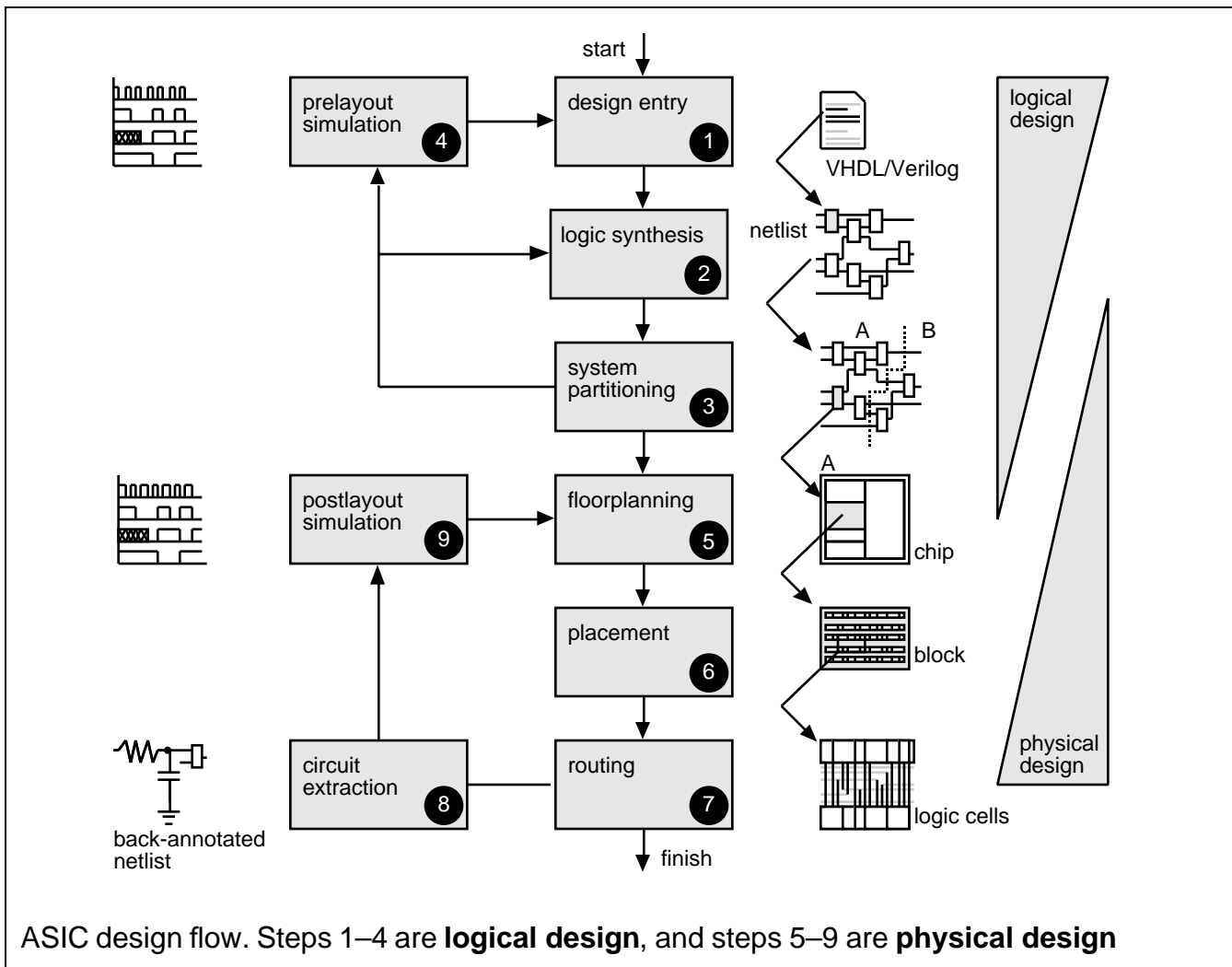
1.2 Design Flow

A **design flow** is a sequence of steps to design an ASIC

1. **Design entry.** Using a **hardware description language (HDL)** or schematic entry.
2. **Logic synthesis.** Produces a **netlist**—logic cells and their connections.
3. **System partitioning.** Divide a large system into ASIC-sized pieces.
4. **Prelayout simulation.** Check to see if the design functions correctly.
5. **Floorplanning.** Arrange the blocks of the netlist on the chip.
6. **Placement.** Decide the locations of cells in a block.
7. **Routing.** Make the connections between cells and blocks.
8. **Extraction.** Determine the resistance and capacitance of the interconnect.
9. **Postlayout simulation.** Check to see the design still works with the added loads of the interconnect.

1.3 Case Study

SPARCstation 1: Better performance at lower cost • Compact size, reduced power, and quiet operation • Reduced number of parts, easier assembly, and improved reliability



The ASICs in the Sun Microsystems SPARCstation 1

	SPARCstation 1 ASIC	Gates (k-gates)
1	SPARC integer unit (IU)	20
2	SPARC floating-point unit (FPU)	50
3	Cache controller	9
4	Memory-management unit (MMU)	5
5	Data buffer	3
6	Direct memory access (DMA) controller	9
7	Video controller/data buffer	4
8	RAM controller	1
9	Clock generator	1

The CAD tools used in the design of the Sun Microsystems SPARCstation 1		
Design level	Function	Tool
ASIC design	ASIC physical design	LSI Logic
	ASIC logic synthesis	Internal tools and UC Berkeley tools
	ASIC simulation	LSI Logic
Board design	Schematic capture	Valid Logic
	PCB layout	Valid Logic Allegro
	Timing verification	Quad Design Motive and internal tools
Mechanical design	Case and enclosure	Autocad
	Thermal analysis	Pacific Numerix
	Structural analysis	Cosmos
Management	Scheduling	Suntrac
	Documentation	Interleaf and FrameMaker

1.4 Economics of ASICs

We'll compare the most popular types of ASICs: an FPGA, an MGA, and a CBIC. The figures in the following sections are approximate and used to illustrate the different components of cost.

1.4.1 Comparison Between ASIC Technologies

Example of an ASIC **part cost**: A 0.5 μ m, 20k-gate array might cost 0.01–0.02 cents/gate (for more than 10,000 parts) or \$2–\$4 per part, but an equivalent FPGA might be \$20.

When does it make sense to use a more expensive part? This is what we shall examine next.

1.4.2 Product Cost

In a product cost there are **fixed costs** and **variable costs** (the number of products sold is the **sales volume**):

total product cost = fixed product cost + variable product cost × products sold

In a product made from parts the total cost for any part is

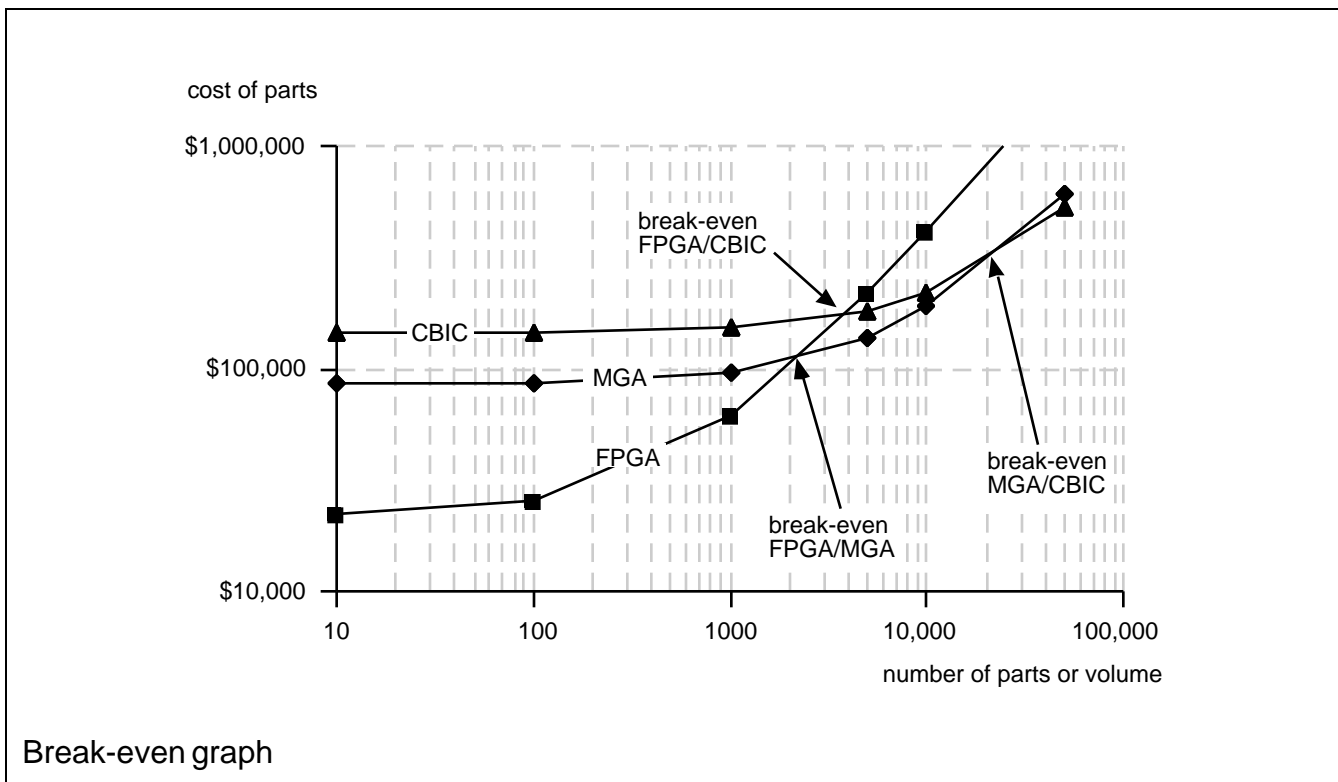
total part cost = fixed part cost + variable cost per part × volume of parts

For example, suppose we have the following (imaginary) costs:

- FPGA: \$21,800 (fixed) \$39 (variable)
- MGA: \$86,000 (fixed) \$10 (variable)
- CBIC \$146,000 (fixed) \$8 (variable)

Then we can calculate the following **break-even volumes**:

- FPGA/MGA 2000 parts
- FPGA/CBIC 4000 parts
- MGA/CBIC 20,000 parts

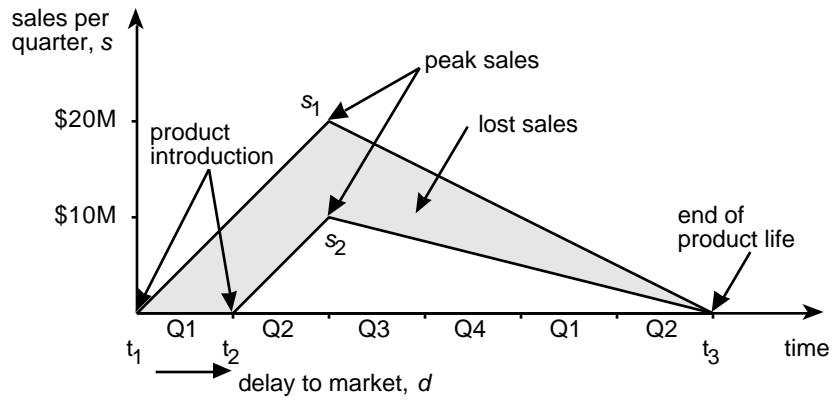


1.4.3 ASIC Fixed Costs

Examples of fixed costs: training cost for a new electronic design automation (EDA) system • hardware and software cost • productivity • production test and design for test • programming costs for an FPGA • nonrecurring-engineering (NRE) • test vectors and test-program development cost • pass (turn or spin) • profit model represents the profit flow during the product lifetime • product velocity • second source

	FPGA	MGA	CBIC
<u>Training:</u>	\$800	\$2,000	\$2,000
Days	2	5	5
Cost/day	\$400	\$400	\$400
<u>Hardware</u>	\$10,000	\$10,000	\$10,000
<u>Software</u>	\$1,000	\$20,000	\$40,000
<u>Design:</u>	\$8,000	\$20,000	\$20,000
Size (gates)	10,000	10,000	10,000
Gates/day	500	200	200
Days	20	50	50
Cost/day	\$400	\$400	\$400
<u>Design for test:</u>		\$2,000	\$2,000
Days		5	5
Cost/day		\$400	\$400
<u>NRE:</u>		\$30,000	\$70,000
Masks		\$10,000	\$50,000
Simulation		\$10,000	\$10,000
Test program		\$10,000	\$10,000
<u>Second source:</u>	\$2,000	\$2,000	\$2,000
Days	5	5	5
Cost/day	\$400	\$400	\$400
<u>Total fixed costs</u>	<u>\$21,800</u>	<u>\$86,000</u>	<u>\$146,000</u>

Spreadsheet, "Fixed Costs"



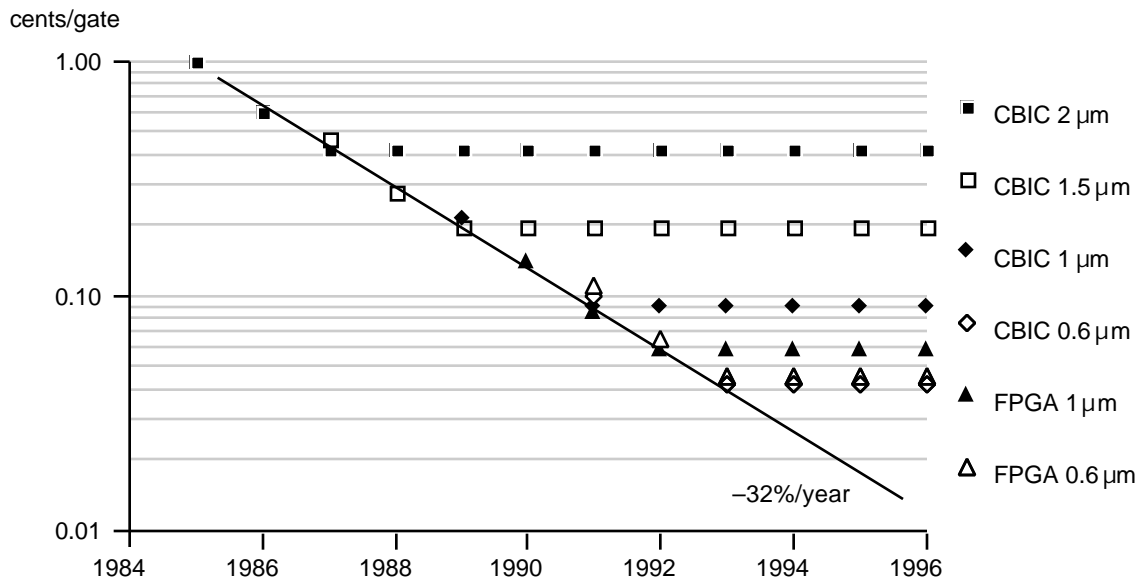
Profit model

1.4.4 ASIC Variable Costs

Factors affecting fixed costs: **wafer size • wafer cost • Moore’s Law** (Gordon Moore of Intel)
• gate density • gate utilization • die size • die per wafer • defect density • yield • die cost
• profit margin (depends on **fab** or **fabless**) • **price per gate • part cost**

	FPGA	MGA	CBIC	Units
Wafer size	6	6	6	inches
Wafer cost	1,400	1,300	1,500	\$
Design	10,000	10,000	10,000	gates
Density	10,000	20,000	25,000	gates/sq.cm
Utilization	60	85	100	%
Die size	1.67	0.59	0.40	sq.cm
Die/wafer	88	248	365	
Defect density	1.10	0.90	1.00	defects/sq.cm
Yield	65	72	80	%
Die cost	25	7	5	\$
Profit margin	60	45	50	%
Price/gate	0.39	0.10	0.08	cents
Part cost	\$39	\$10	\$8	

Spreadsheet, “Variable Costs”



Example price per gate figures

1.5 ASIC Cell Libraries

You can:

- (1) use a **design kit** from the **ASIC vendor**
- (2) buy an **ASIC-vendor library** from a **library vendor**
- (3) you can build your own cell library

(1) is usually a **phantom library**—the cells are empty boxes, or **phantoms**, you **hand off** your design to the ASIC vendor and they perform **phantom instantiation** (Synopsys CBA)

(2) involves a **buy-or-build decision**. You need a **qualified cell library** (qualified by the **ASIC foundry**) If you own the masks (the **tooling**) you have a **customer-owned tooling (COT**, pronounced “see-oh-tee”) solution (which is becoming very popular)

(3) involves a complex **library development** process: **cell layout • behavioral model • Verilog/VHDL model • timing model • test strategy • characterization • circuit extraction • process control monitors (PCMs) or drop-ins • cell schematic • cell icon • layout versus schematic (LVS) check • cell icon • logic synthesis • retargeting • wire-load model • routing model • phantom**

1.6 Summary

Key concepts:

- We could define an ASIC as a design style that uses a cell library
- The difference between full-custom and semicustom ASICs
- The difference between standard-cell, gate-array, and programmable ASICs
- The ASIC design flow
- Design economics including part cost, NRE, and breakeven volume
- The contents and use of an ASIC cell library

Types of ASIC

ASIC type	Family member	Custom mask layers	Custom logic cells
Full-custom	Analog/digital	All	Some
Semicustom	Cell-based (CBIC)	All	None
	Masked gate array (MGA)	Some	None
Programmable	Field-programmable gate array (FPGA)	None	None
	Programmable logic device (PLD)	None	None

1.7 Problems

Suggested homework: 1.4, 1.5, 1.9 (from *ASICs... the book*)

1.8 Bibliography

EE Times (ISSN 0192-1541, <http://techweb.cmp.com/ee>), *EDN* (ISSN 0012-7515, <http://www.ednmag.com>), EDAC (Electronic Design Automation Companies) (<http://www.edac.org>), The Electrical Engineering page on the World Wide Web (E2W3) (<http://www.e2w3.com>), SEMATECH (Semiconductor Manufacturing Technology) (<http://www.sematech.org>), The MIT Semiconductor Subway (<http://www-mtl.mit.edu>), EDA companies at <http://www.yahoo.comunder> Business_and_Economyin Companies/Computers/Software/Graphics/CAD/IC_Design The MOS Implementation Service (MOSIS) (<http://www.isi.edu>), The Microelectronic Systems Newsletter at <http://www-ece.engr.utk.edu/ece> NASA (<http://nppp.jpl.nasa.gov/dmg/jpl/loc/asi>)

1.9 References

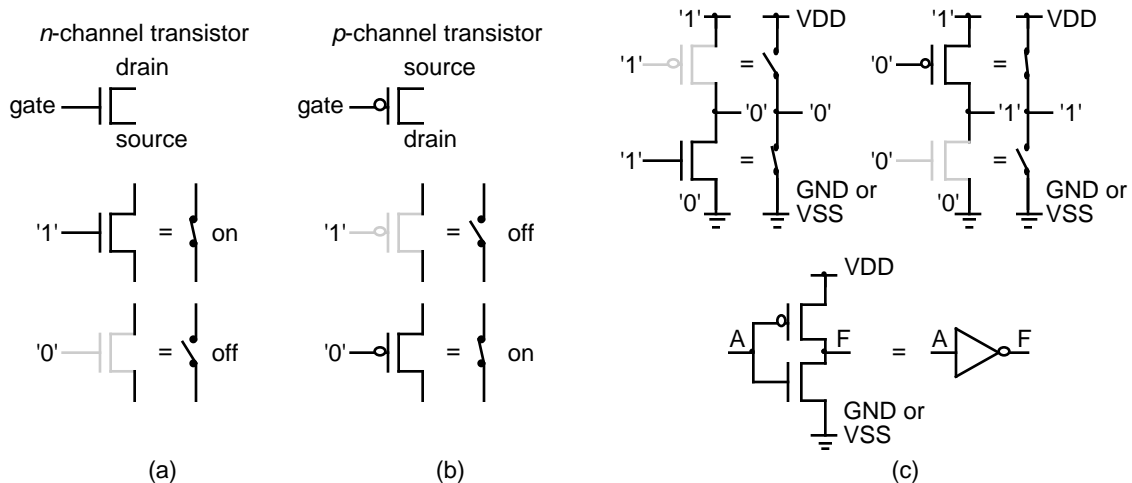
- Glasser, L. A., and D. W. Dobberpuhl. 1985. *The Design and Analysis of VLSI Circuits*. Reading, MA: Addison-Wesley, 473 p. ISBN 0-201-12580-3. TK7874.G573. Detailed analysis of circuits, but largely nMOS.
- Mead, C. A., and L. A. Conway. 1980. *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 396 p. ISBN 0-201-04358-0. TK7874.M37.
- Weste, N. H. E., and K. Eshraghian. 1993. *Principles of CMOS VLSI Design: A Systems Perspective*. 2nd ed. Reading, MA: Addison-Wesley, 713 p. ISBN 0-201-53376-6. TK7874.W46. Concentrates on full-custom design.

CMOS LOGIC

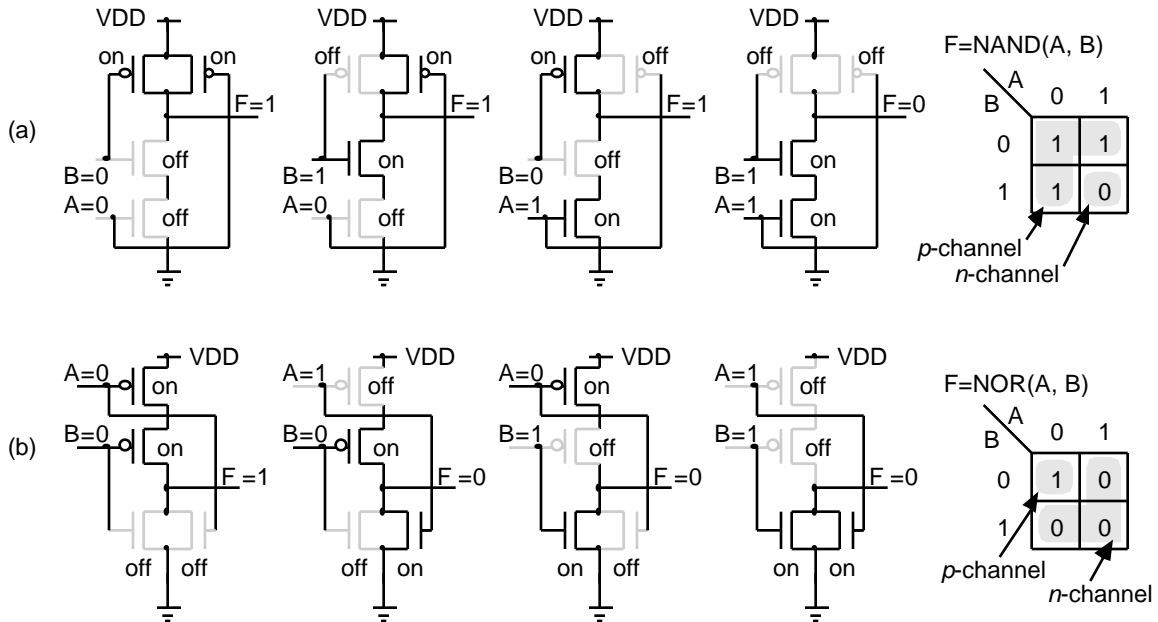
2

Key concepts: The use of transistors as switches • The difference between a flip-flop and a latch • Setup time and hold time • Pipelines and latency • The difference between datapath, standard-cell, and gate-array logic cells • Strong and weak logic levels • Pushing bubbles • Ratio of logic • Resistance per square of layers and their relative values in CMOS • Design rules and

- **CMOS transistor** (or device)
- A transistor has three terminals: **gate**, **source**, **drain** (and a fourth that we ignore for a moment)
- An MOS transistor looks like a switch (conducting/on, nonconducting/off, not open or closed)

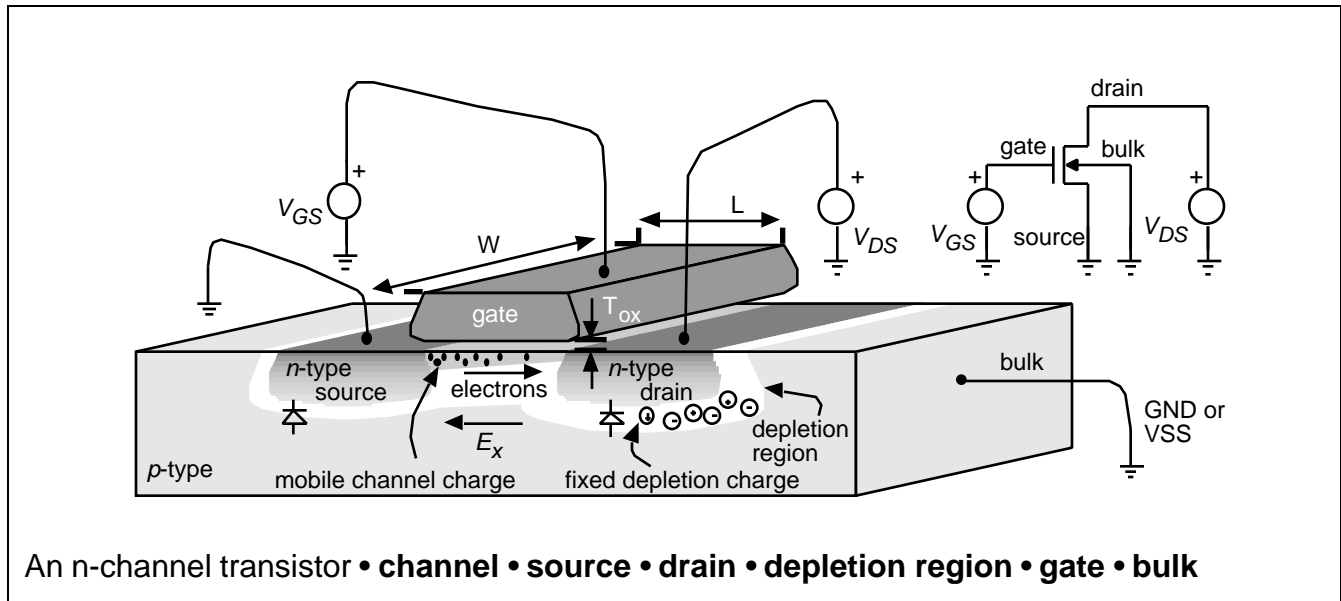


CMOS transistors viewed as switches • a CMOS inverter



CMOS logic • a two-input **NAND gate** • a two-input **NOR gate** • Good '1's • Good '0's

2.1 CMOS Transistors



current (amperes) = charge (coulombs) per unit time (second)

- Channel charge = Q (imagine taking a picture and counting the electrons)
- t_f is **time of flight** or **transit time**

The drain-to-source current $I_{DSn} = Q/t_f$

The (vector) velocity of the electrons $\mathbf{v} = -\mu_n \mathbf{E}$

- μ_n is the **electron mobility** (μ_p is the **hole mobility**)
- \mathbf{E} is the electric field (units Vm^{-1})

$$t_f = \frac{L}{v_x} = \frac{L^2}{\mu_n V_{DS}}$$

$$Q = C(V_{GC} - V_{tn}) = C[(V_{GS} - V_{tn}) - 0.5 V_{DS}] = WLC_{ox} [(V_{GS} - V_{tn}) - 0.5 V_{DS}]$$

$$I_{DSn} = Q/t_f$$

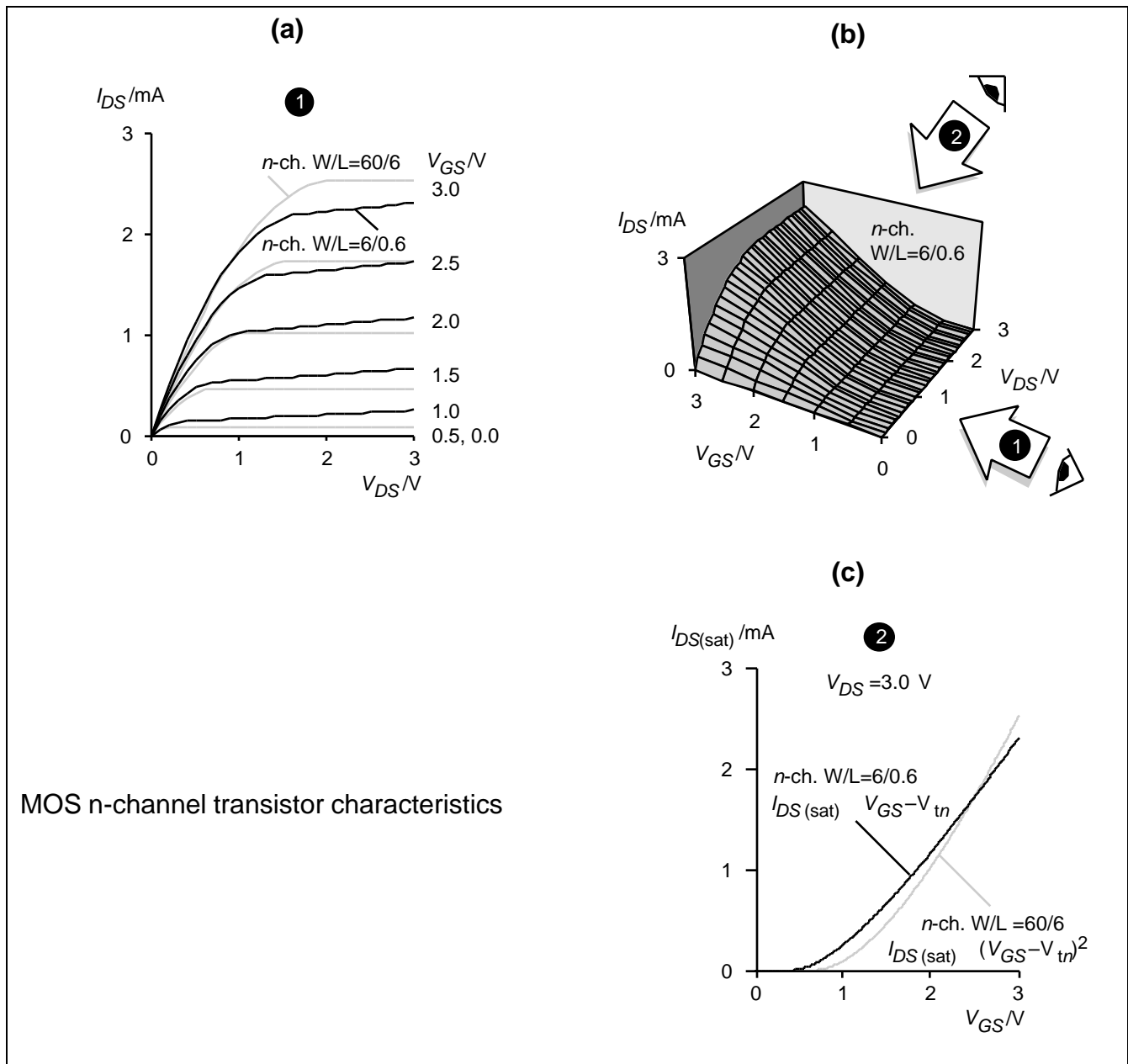
$$= (W/L)\mu_n C_{ox} [(V_{GS} - V_{tn}) - 0.5 V_{DS}] V_{DS} = (W/L)k'_n [(V_{GS} - V_{tn}) - 0.5 V_{DS}] V_{DS}$$

$k'_n = \mu_n C_{ox}$ is the process transconductance parameter (or **intrinsic transconductance**)

$k_n = k'_n(W/L)$ is the **transistor gain factor** (or just **gain factor**)

- The **linear region** (triode region) extends until $V_{DS} = V_{GS} - V_{tn}$
- $V_{DS} = V_{GS} - V_{tn} = V_{DS(sat)}$ (**saturation voltage**)
- $V_{DS} > V_{GS} - V_{tn}$ (the **saturation region**, or pentode region, of operation)
- **saturation current**, $I_{DSn(sat)}$

$$I_{DSn(sat)} = (k_n/2)(V_{GS} - V_{tn})^2; \quad V_{GS} > V_{tn}$$



MOS n-channel transistor characteristics

2.1.1 P-Channel Transistors

$$I_{DSp} = -k'_p(W/L)[(V_{GS} - V_{tp}) - 0.5 V_{DS}]V_{DS}; \quad V_{DS} > V_{GS} - V_{tp}$$

$$I_{DSp(sat)} = -\frac{\mu_p}{2} C_{ox} (W/L) (V_{GS} - V_{tp})^2; \quad V_{DS} < V_{GS} - V_{tp}$$

- V_{tp} is negative
- V_{DS} and V_{GS} are normally negative (and $-3V < -2V$)

2.1.2 Velocity Saturation

- $v_{\max n} = 10^5 \text{ ms}^{-1}$
- **velocity saturation**
- $t_f = L_{\text{eff}} / v_{\max n}$
- **mobility degradation**

$$I_{DSn(\text{sat})} = W v_{\max n} C_{\text{ox}} (V_{GS} - V_{tn}); \quad V_{DS} > V_{DS(\text{sat})} \text{ (velocity saturated).}$$

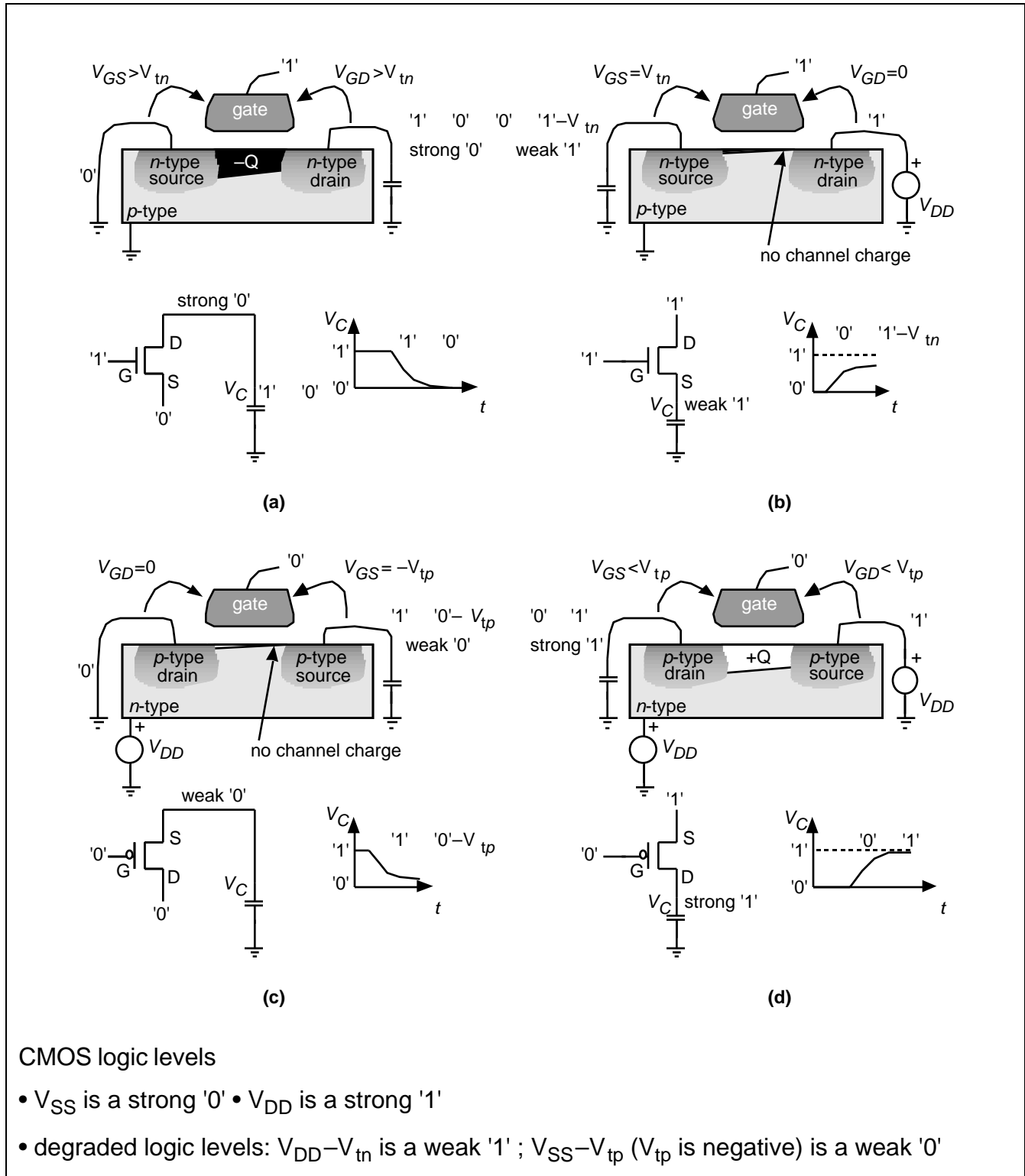
2.1.3 SPICE Models

- KP (in μAV^{-2}) = k'_n (k'_p)
- VTO and TOX = V_{tn} (V_{tp}) and T_{ox}
- U0 (in $\text{cm}^2\text{V}^{-1}\text{s}^{-1}$) = μ_n (and μ_p)

SPICE parameters

```
.MODEL CMOSN NMOS LEVEL=3 PHI=0.7 TOX=10E-09 XJ=0.2U TPG=1 VTO=0.65
DELTA=0.7
+ LD=5E-08 KP=2E-04 UO=550 THETA=0.27 RSH=2 GAMMA=0.6 NSUB=1.4E+17
NFS=6E+11
+ VMAX=2E+05 ETA=3.7E-02 KAPPA=2.9E-02 CGDO=3.0E-10 CGSO=3.0E-10
CGBO=4.0E-10
+ CJ=5.6E-04 MJ=0.56 CJSW=5E-11 MJSW=0.52 PB=1
.MODEL CMOSP PMOS LEVEL=3 PHI=0.7 TOX=10E-09 XJ=0.2U TPG=-1 VTO=-
0.92 DELTA=0.29
+ LD=3.5E-08 KP=4.9E-05 UO=135 THETA=0.18 RSH=2 GAMMA=0.47
NSUB=8.5E+16 NFS=6.5E+11
+ VMAX=2.5E+05 ETA=2.45E-02 KAPPA=7.96 CGDO=2.4E-10 CGSO=2.4E-10
CGBO=3.8E-10
+ CJ=9.3E-04 MJ=0.47 CJSW=2.9E-10 MJSW=0.505 PB=1
```

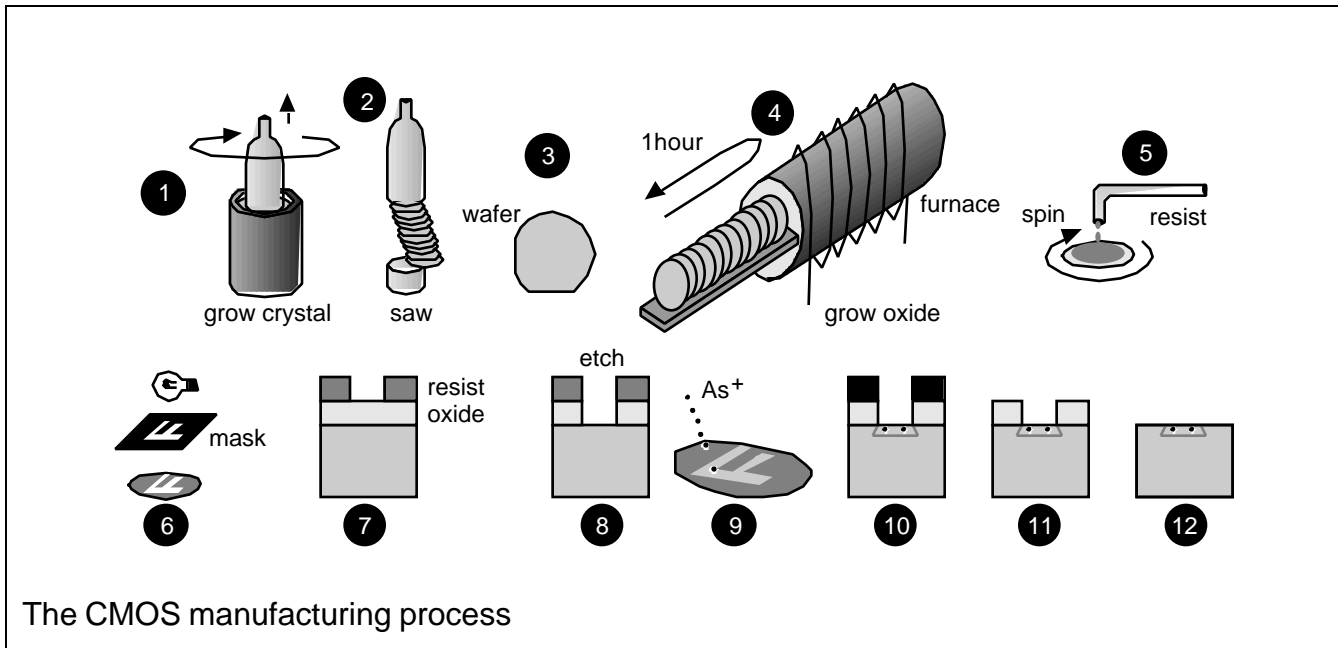
2.1.4 Logic Levels



CMOS logic levels

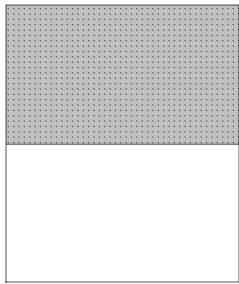
- V_{SS} is a strong '0' • V_{DD} is a strong '1'
- degraded logic levels: $V_{DD} - V_{tn}$ is a weak '1' ; $V_{SS} - V_{tp}$ (V_{tp} is negative) is a weak '0'

2.2 The CMOS Process

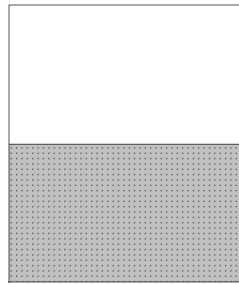


Key words: boule • wafer • boat • silicon dioxide • resist • mask • chemical etch • isotropic • plasma etch • anisotropic • ion implantation • implant energy and dose • **polysilicon** • chemical vapor deposition (CVD) • sputtering • photolithography • submicron and deep-submicron process • n-well process • p-well process • twin-tub (or twin-well) • triple-well • **substrate contacts** (well contacts or tub ties) • **active (CAA)** • **gate oxide** • **field** • field implant or channel-stop implant • **field oxide (FOX)** • bloat • dopant • **self-aligned process** • positive resist • negative resist • drain engineering • LDD process • lightly doped drain • LDD diffusion or LDD implant • stipple-pattern

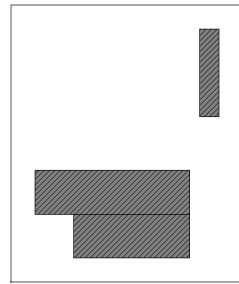
Mask/layer name	Derivation from drawn layers	Alternative names for mask/layer	Mask label
n-well	=nwell	bulk, substrate, tub, n-tub, moat	CWN
p-well	=pwell	bulk, substrate, tub, p-tub, moat	CWP
active	=pdiff+ndiff	thin oxide, thinox, island, gate oxide	CAA
polysilicon	=poly	poly, gate	CPG
n-diffusion implant	=grow(ndiff)	ndiff, n-select, nplus, n+	CSN
p-diffusion implant	=grow(pdifff)	pdiff, p-select, pplus, p+	CSP
contact	=contact	contact cut, poly contact, diffusion contact	CCP and CCA
metal1	=m1	first-level metal	CMF
metal2	=m2	second-level metal	CMS
via2	=via2	metal2/metal3 via, m2/m3 via	CVS
metal3	=m3	third-level metal	CMT
glass	=glass	passivation, overglass, pad	COG



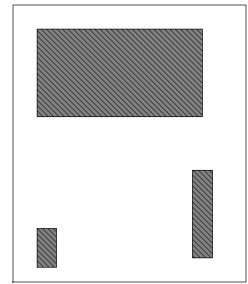
(a) nwell



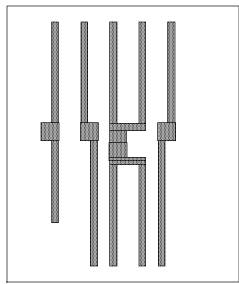
(b) pwell



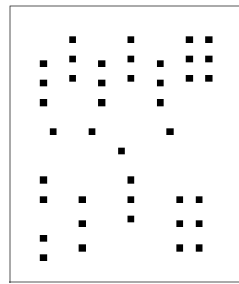
(c) ndiff



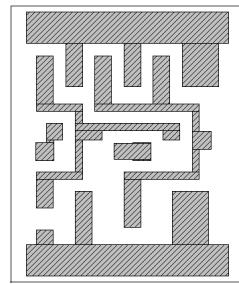
(d) pdiff



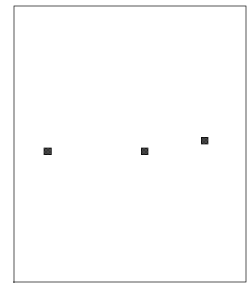
(e) poly



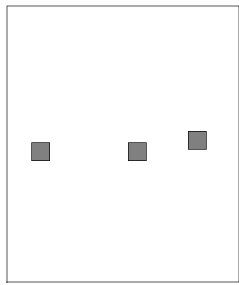
(f) contact



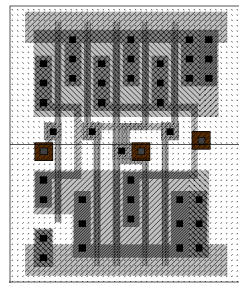
(g) m1



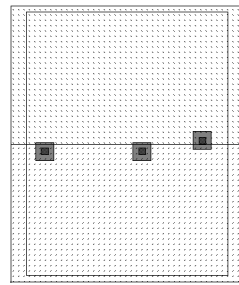
(h) via



(i) m2



(j) cell



(k) phantom

The mask layers of a standard cell

Active *mask*

CAA (mask) = ndiff (drawn) pdiff (drawn)

Implant select *masks*

CSN (mask) = grow (ndiff (drawn)) and

CSP (mask) = grow (pdiff (drawn))

Source and drain diffusion (on the *silicon*)

n-diffusion (silicon) = (CAA (mask) CSN (mask)) (¬ CPG (mask)) and

p-diffusion(silicon)=(CAA(mask) CSP(mask)) (¬ CPG(mask))

Source and drain diffusion (on the *silicon*) in terms of *drawn* layers

n-diffusion (silicon) = (ndiff (drawn)) (¬ poly (drawn)) and

p-diffusion (silicon) = (pdiff (drawn)) (¬ poly (drawn))

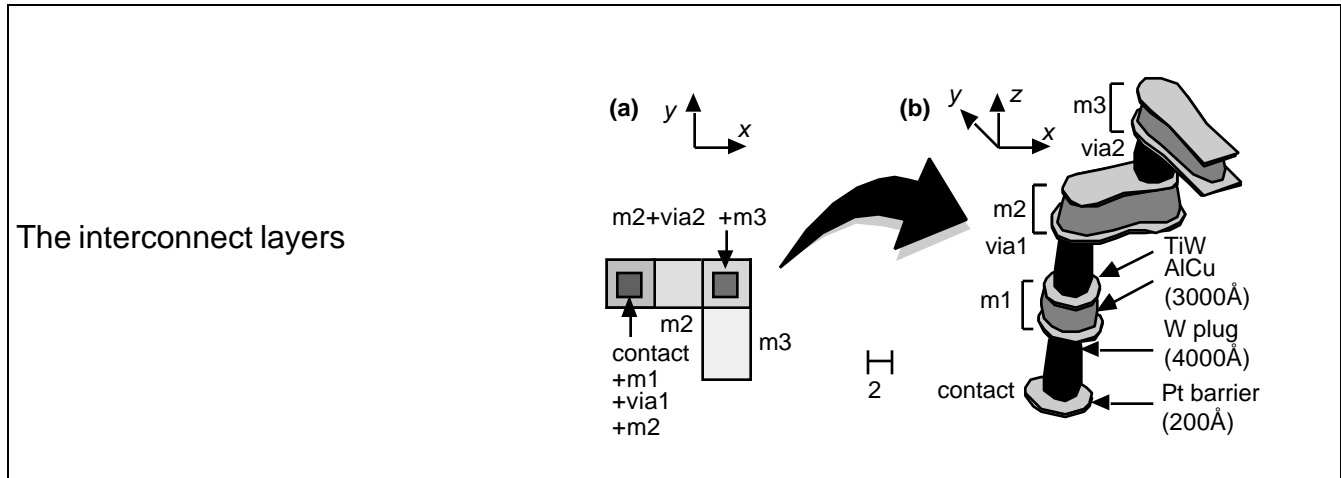
Drawn layers and stipple patterns

nwell	pwell	ndiff	pdiff	poly	contact
					(or solid)
m1	via1	m2	via2	m3	glass
	(or solid)		(or solid)		

The transistor layers

(a)

(b)

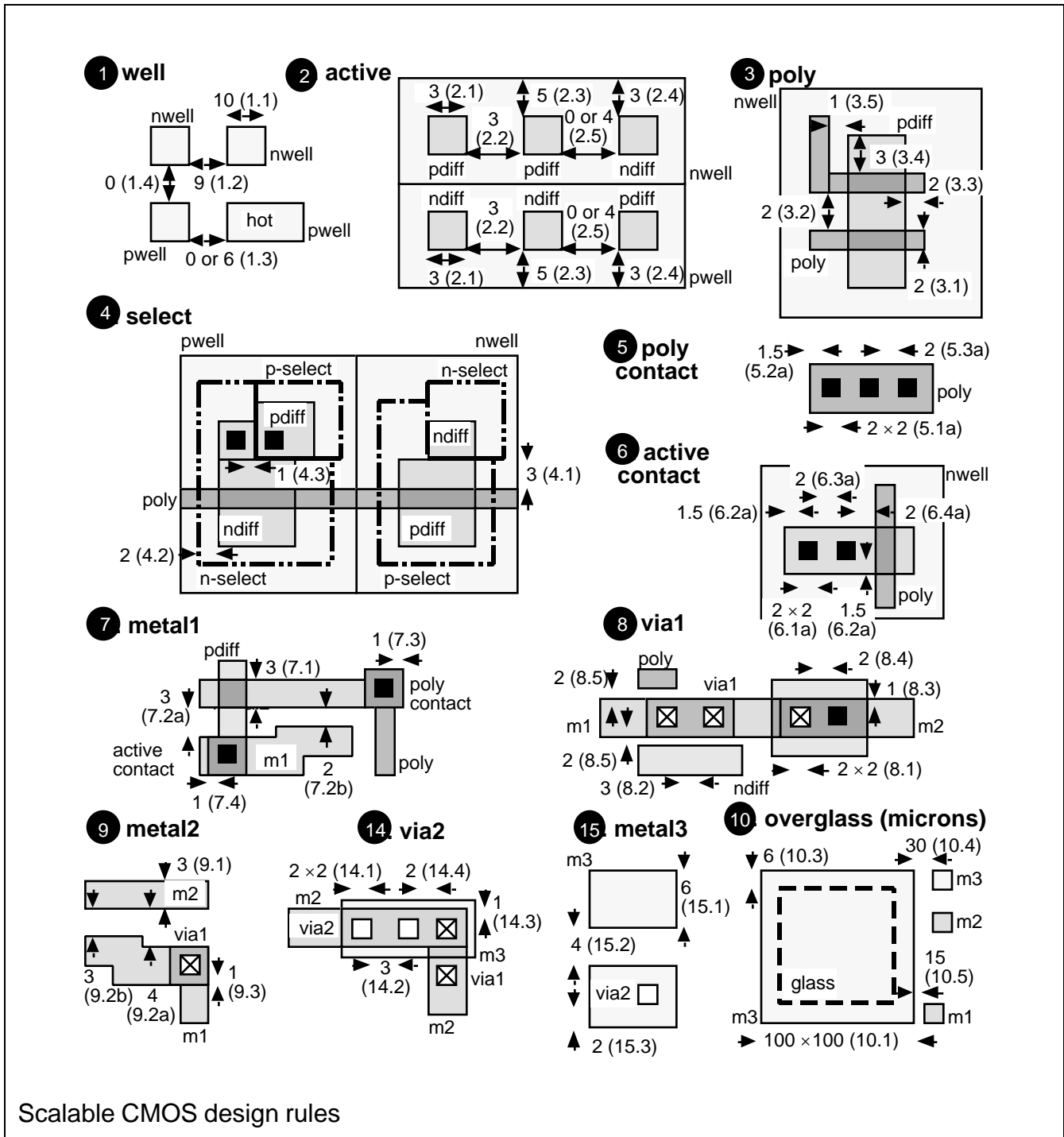


2.2.1 Sheet Resistance

Sheet resistance (1µm)			Sheet resistance (0.35µm)		
Layer	Sheet resistance	Units	Layer	Sheet resistance	Units
n-well	1.15± 0.25	k /square	n-well	1± 0.4	k /square
poly	3.5± 2.0	/square	poly	10± 4.0	/square
n-diffusion	75± 20	/square	n-diffusion	3.5± 2.0	/square
p-diffusion	140± 40	/square	p-diffusion	2.5± 1.5	/square
m1/2	70± 6	m /square	m1/2/3	60± 6	m /square
m3	30± 3	m /square	metal4	30± 3	m /square

Key words: diffusion • /square (ohms per square) • sheet resistance • silicide • self-aligned silicide (**salicide**) • LI, white metal, local interconnect, metal0, or m0 • m1 or metal1 • diffusion contacts • polysilicon contacts • barrier metal • contact plugs (via plugs) • chemical–mechanical polishing (CMP) • intermetal oxide (IMO) • interlevel dielectric (ILD) • metal vias, cuts, or vias • stacked vias and stacked contacts • two-level metal (2LM) • 3LM (m3 or metal3) • via1 • via2 • metal pitch • electromigration • contact resistance and via resistance

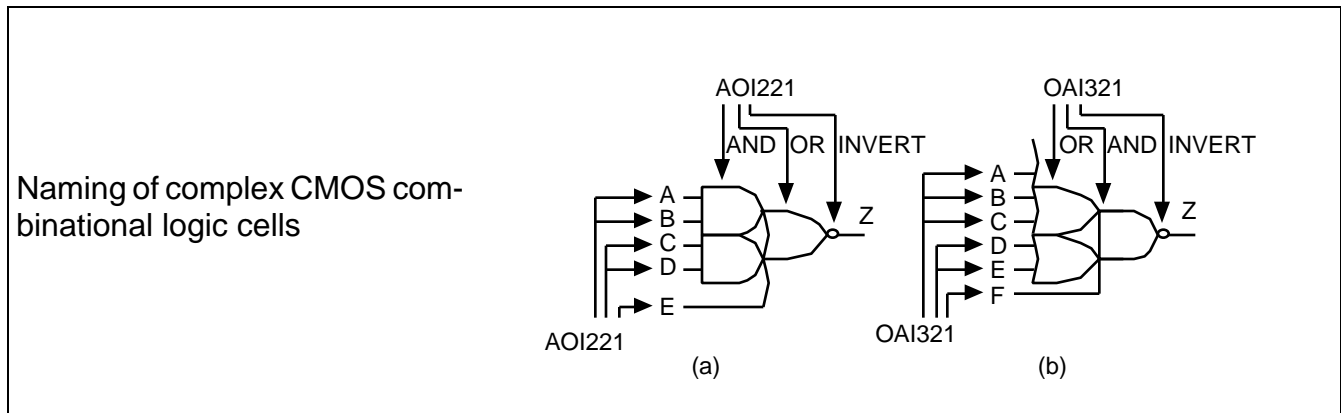
2.3 CMOS Design Rules



2.4 Combinational Logic Cells

The AOI family of cells with three index numbers or less		
Cell type ¹	Cells	Number of unique cells
Xa1	X21, X31	2
Xa11	X211, X311	2
Xab	X22, X33, X32	3
Xab1	X221, X331, X321	3
Xabc	X222, X333, X332, X322	4
Total		14

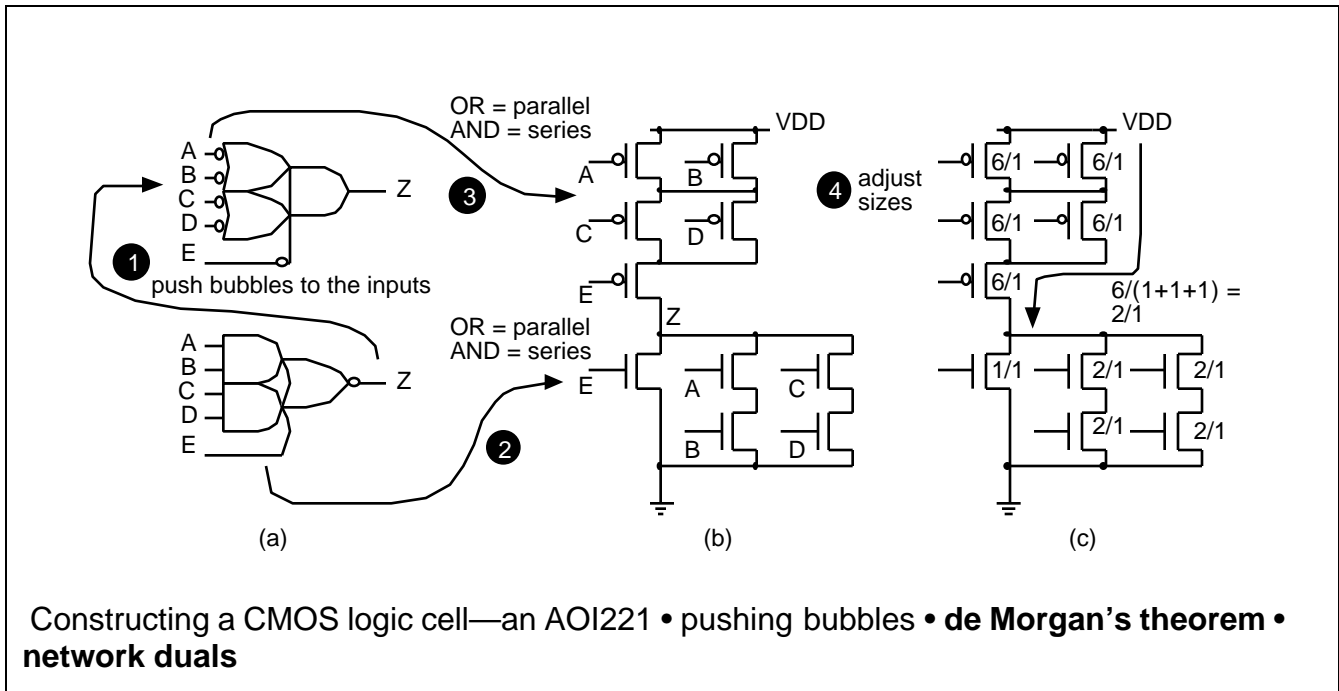
¹Xabc: X={AOI, AO, OAI, OA}; a, b, c = {2, 3}; {} means "choose one."



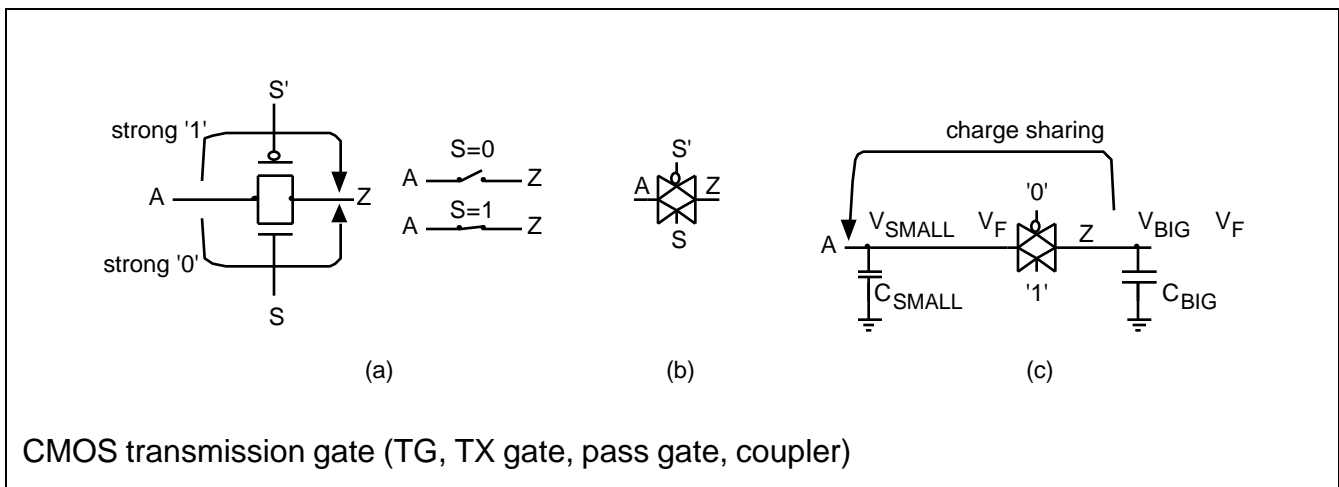
2.4.1 Pushing Bubbles

2.4.2 Drive Strength

We **ratio** a cell to adjust its **drive strength** and make $n = p$ to create equal rise and fall times



2.4.3 Transmission Gates



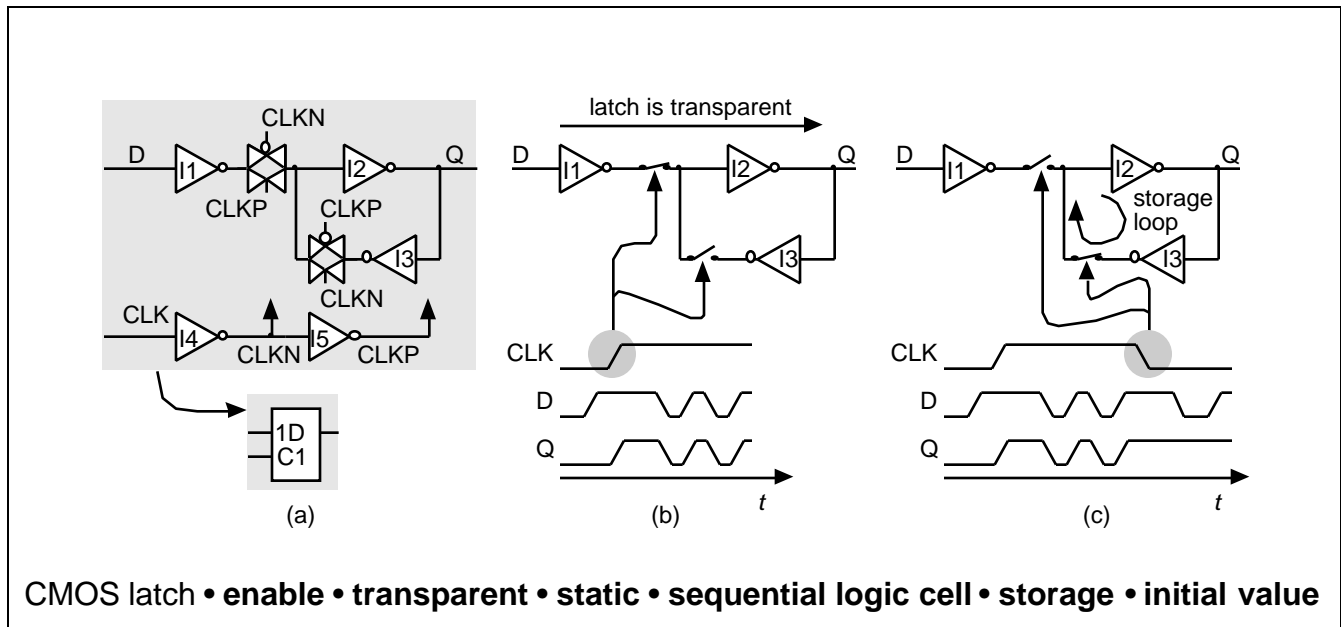
Charge sharing: suppose $C_{BIG}=0.2\text{pF}$ and $C_{SMALL}=0.02\text{pF}$, $V_{BIG}=0\text{V}$ and $V_{SMALL}=5\text{V}$; then

$$V_F = \frac{(0.2 \times 10^{-12})(0) + (0.02 \times 10^{-12})(5)}{(0.2 \times 10^{-12}) + (0.02 \times 10^{-12})} = 0.45 \text{ V}$$

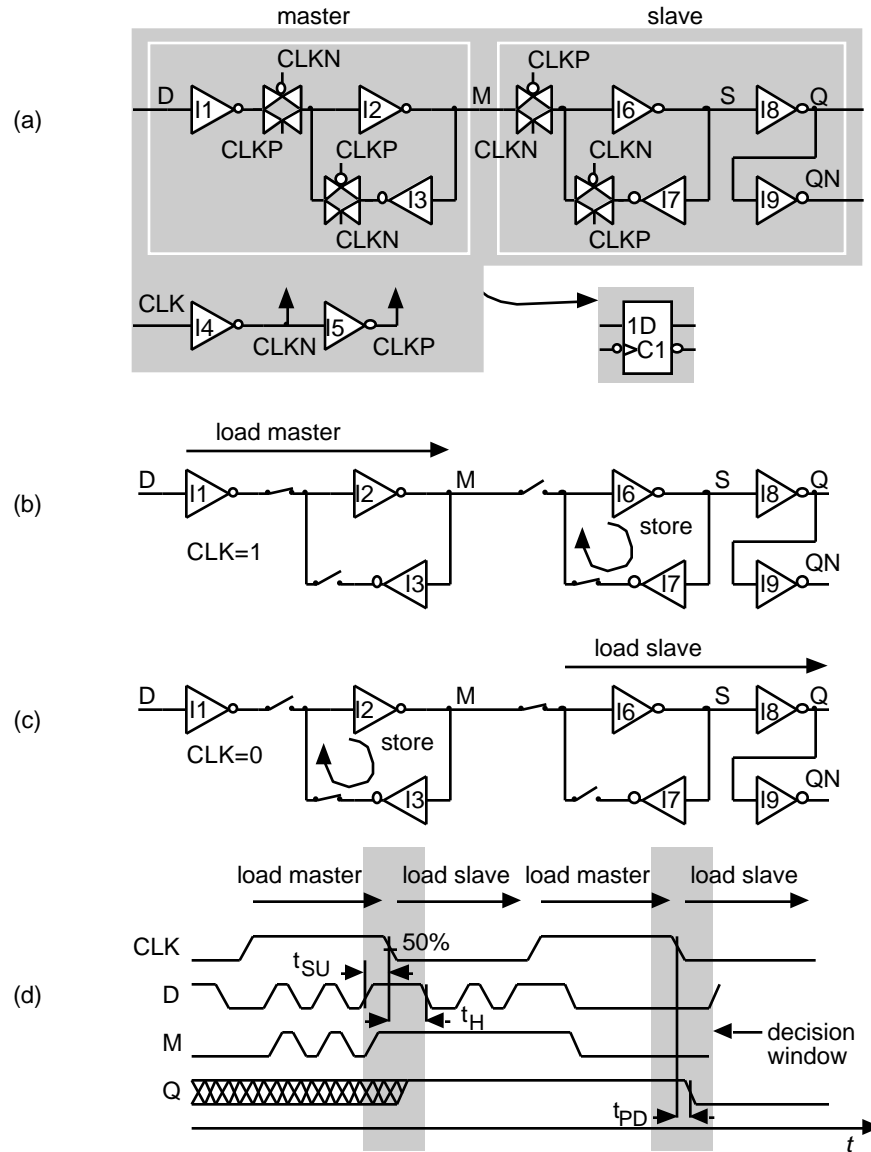
2.5 Sequential Logic Cells

Two choices for sequential logic: **multiphase clocks** or **synchronous design**. We choose the latter.

2.5.1 Latch



2.5.2 Flip-Flop



CMOS flip-flop

- master latch • slave latch
- active clock edge • negative-edge-triggered flip-flop
- setup time (t_{SU}) • hold time (t_H) • clock-to-Q propagation delay (t_{PD})
- decision window

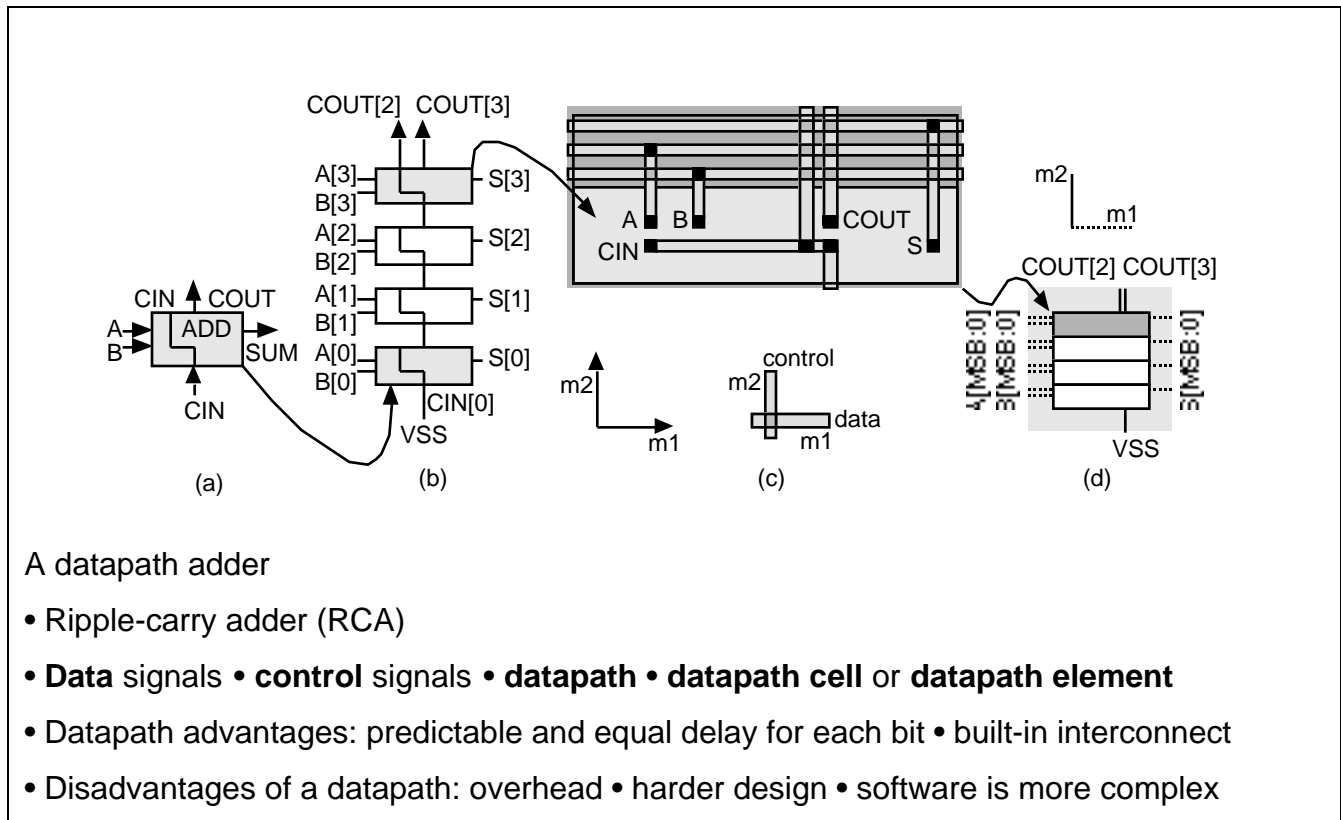
2.6 Datapath Logic Cells

full adder (FA): $SUM = A \oplus B \oplus CIN = PARITY(A, B, CIN)$,
 $COU = A \cdot B + A \cdot CIN + B \cdot CIN = MAJ(A, B, CIN)$.

- **parity function** ('1' for an odd numbers of '1's)
- **majority function** ('1' if the majority of the inputs are '1')

$S[i] = SUM(A[i], B[i], CIN)$

$COU = MAJ(A[i], B[i], CIN)$



2.6.1 Datapath Elements

Binary arithmetic				
Operation	Binary Number Representation			
	Unsigned	Signed magnitude	Ones' complement	Two's complement
	no change	if positive then MSB=0 else MSB=1	if negative then flip bits	if negative then {flip bits; add 1}
3=	0011	0011	0011	0011
-3=	NA	1011	1100	1101
zero=	0000	0000 or 1000	1111 or 0000	0000
max. positive=	1111=15	0111=7	0111=7	0111=7
max. negative=	0000=0	1111=-7	1000=-7	1000=-8
addition= S= A+B =addend+augend SG(A)=sign of A	S=A+B	if SG(A)=SG(B) then S=A+B else {if B<A then S=A-B else S=B-A}	S= A+B+COUT[MS B] COUT is carry out	S=A+B
addition result: OV=overflow, OR=out of range	OR=COUT[MSB] COUT is carry out	if SG(A)=SG(B) then OV=COUT[MSB] else OV=0 (impossible)	OV= XOR(COUT[MSB], COUT[MSB-1])	OV= XOR(COUT[MSB], COUT[MSB-1])
SG(S)=sign of S S= A+B	NA	if SG(A)=SG(B) then SG(S)=SG(A) else {if B<A then SG(S)=SG(A) else SG(S)=SG(B)}	NA	NA
subtraction= D= A-B =minuend -subtrahend	D=A-B	SG(B)=NOT(SG(B)); D=A+B	Z=-B (negate); D=A+Z	Z=-B (negate); D=A+Z

subtraction result:	OR=BOUT[M SB]	as in addition	as in addition	as in addition
OV=overflow, OR=out of range	BOUT is bor- row out			
negation: Z=-A (negate)	NA	Z=A; SG(Z)=NOT(SG(A))	Z=NOT(A)	Z=NOT(A)+1

2.6.2 Adders

Generate, $G[i]$ and **propagate**, $P[i]$

method 1

$$G[i] = A[i] \cdot B[i]$$

$$P[i] = A[i] \oplus B[i]$$

$$C[i] = G[i] + P[i] \cdot C[i-1]$$

$$S[i] = P[i] \oplus C[i-1]$$

method 2

$$G[i] = A[i] \cdot B[i]$$

$$P[i] = A[i] + B[i]$$

$$C[i] = G[i] + P[i] \cdot C[i-1]$$

$$S[i] = A[i] \oplus B[i] \oplus C[i-1]$$

Carry signal:

either $C[i] = A[i] \cdot B[i] + P[i] \cdot C[i-1]$

or $C[i] = (A[i] + B[i]) \cdot (P[i]' + C[i-1])$, where $P[i]' = \text{NOT}(P[i])$

Carry chain using two-input NAND gates, one per cell:

even stages

$$C1[i]' = P[i] \cdot C3[i-1] \cdot C4[i-1]$$

$$C2[i] = A[i] + B[i]$$

$$C[i] = C1[i] \cdot C2[i]$$

odd stages

$$C3[i]' = P[i] \cdot C1[i-1] \cdot C2[i-1]$$

$$C4[i]' = A[i] \cdot B[i]$$

$$C[i] = C3[i]' + C4[i]'$$

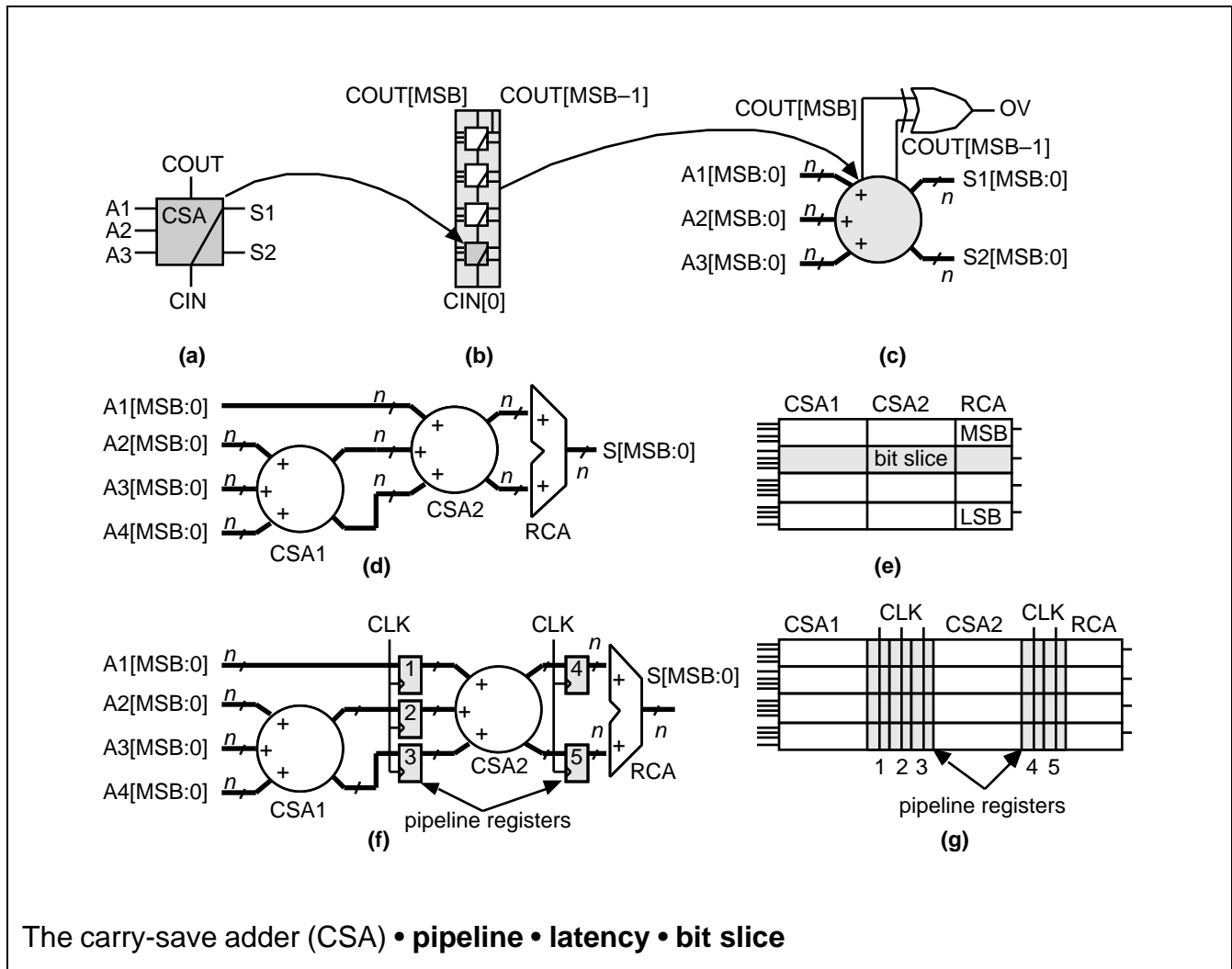
Carry-save adder (CSA) cell $\text{CSA}(A1[i], A2[i], A3[i], \text{CIN}, S1[i], S2[i], \text{COUT})$ has three outputs:

$$S1[i] = \text{CIN},$$

$$S2[i] = A1[i] \oplus A2[i] \oplus A3[i] = \text{PARITY}(A1[i], A2[i], A3[i])$$

$$\text{COUT} = A1[i] \cdot A2[i] + [(A1[i] + A2[i]) \cdot A3[i]] = \text{MAJ}(A1[i], A2[i], A3[i])$$

Carry-propagate adder (CPA)



carry-bypass adders (CBA):

$$C[7] = (G[7] + P[7] \cdot C[6]) \cdot \text{BYPASS}' + C[3] \cdot \text{BYPASS}$$

carry-skip adder:

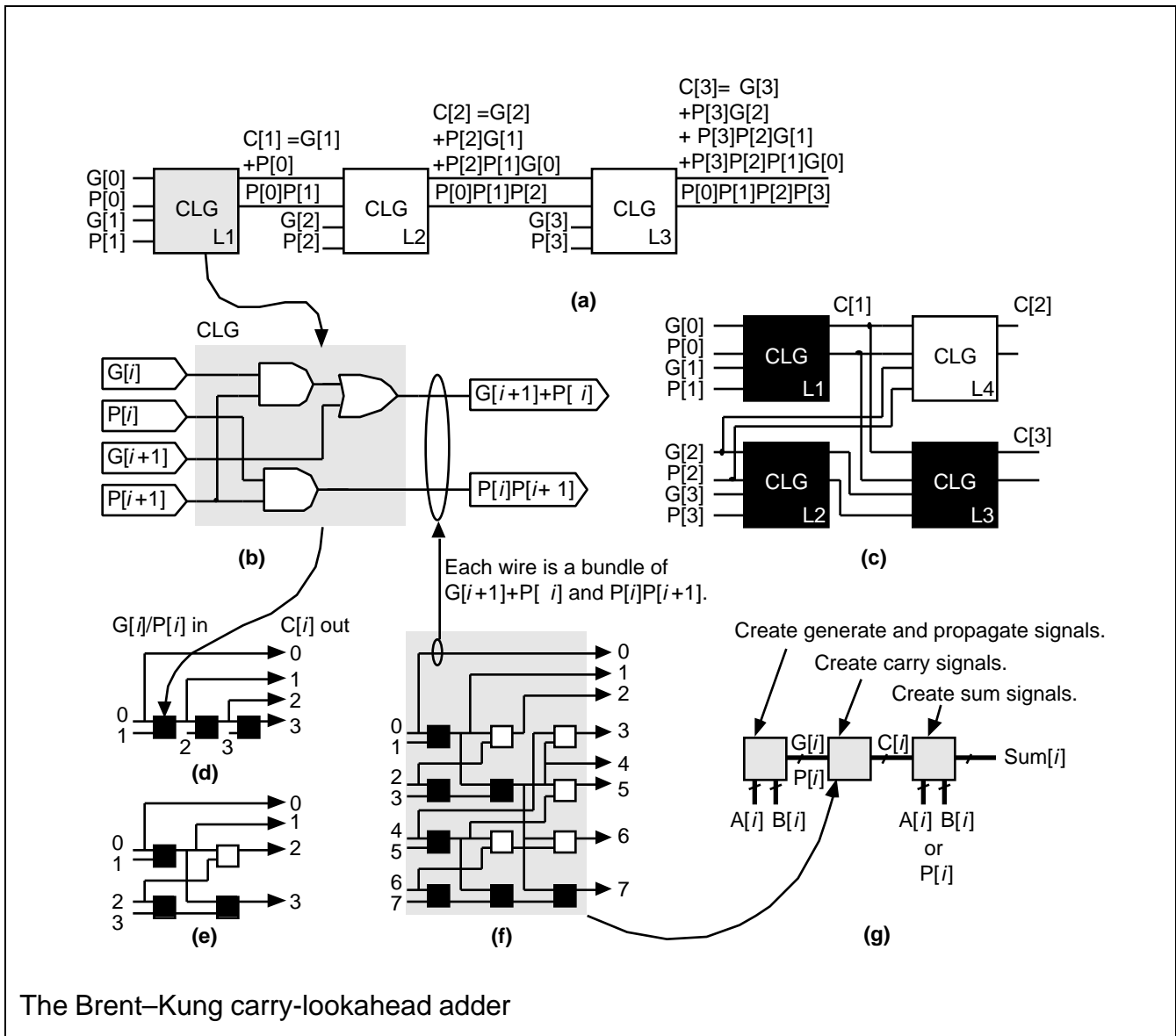
$$\text{CSKIP}[j] = (G[j] + P[j] \cdot C[j-1]) \cdot \text{SKIP}' + C[j-2] \cdot \text{SKIP}$$

Carry-lookahead adder (CLA, for example the Brent–Kung adder):

$$\begin{aligned}
 C[1] &= G[1] + P[1] \cdot C[0] \\
 &= G[1] + P[1] \cdot (G[0] + P[0] \cdot C[-1]) \\
 &= G[1] + P[1] \cdot G[0]
 \end{aligned}$$

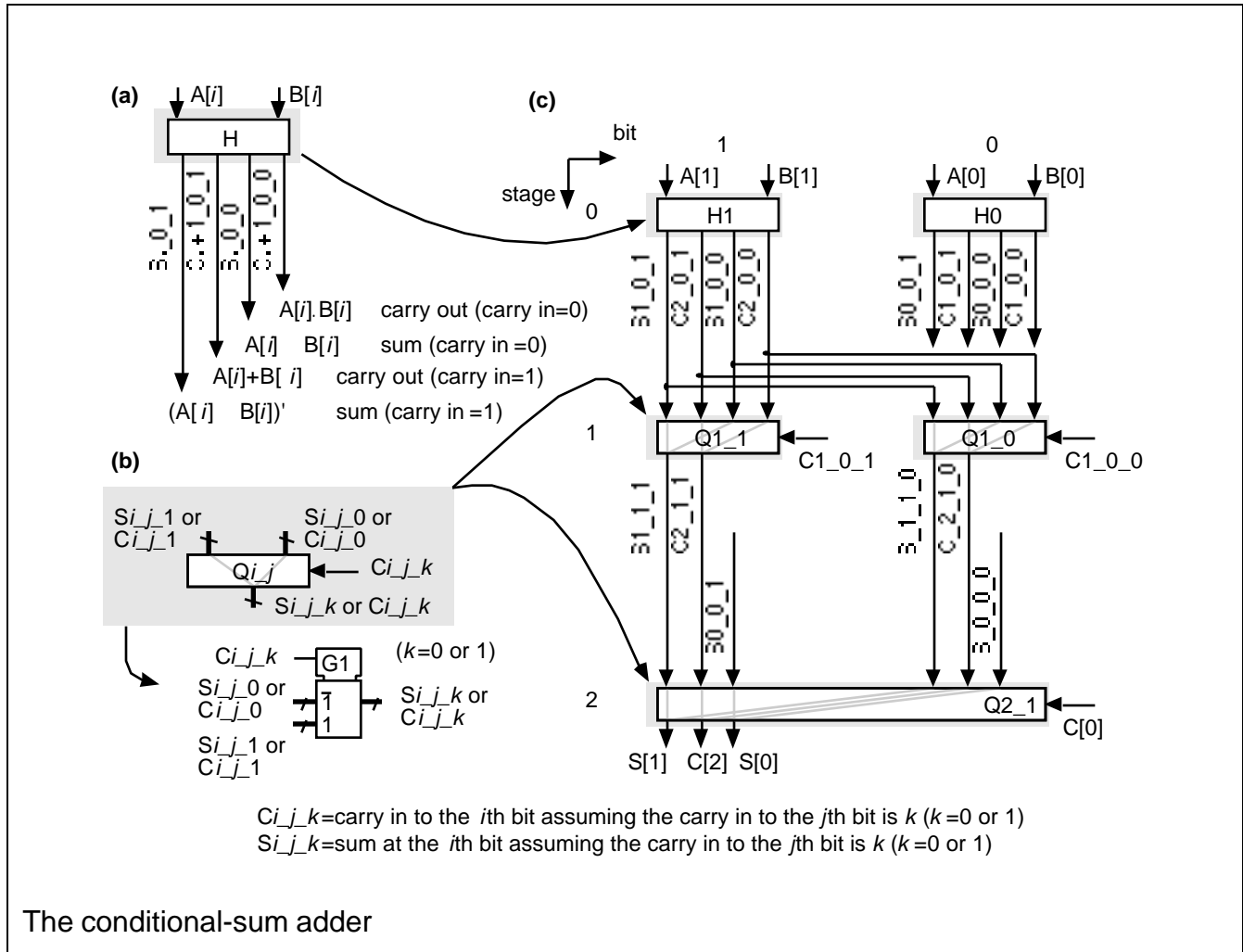
$$C[2] = G[2] + P[2] \cdot G[1] + P[2] \cdot P[1] \cdot G[0] ,$$

$$C[3] = G[3] + P[2] \cdot G[2] + P[2] \cdot P[1] \cdot G[1] + P[3] \cdot P[2] \cdot P[1] \cdot G[0]$$



The Brent–Kung carry-lookahead adder

Carry-select adder duplicates two small adders for the cases $CIN='0'$ and $CIN='1'$ and then uses a MUX to select the case that we need



2.6.3 A Simple Example

An 8-bit conditional-sum adder

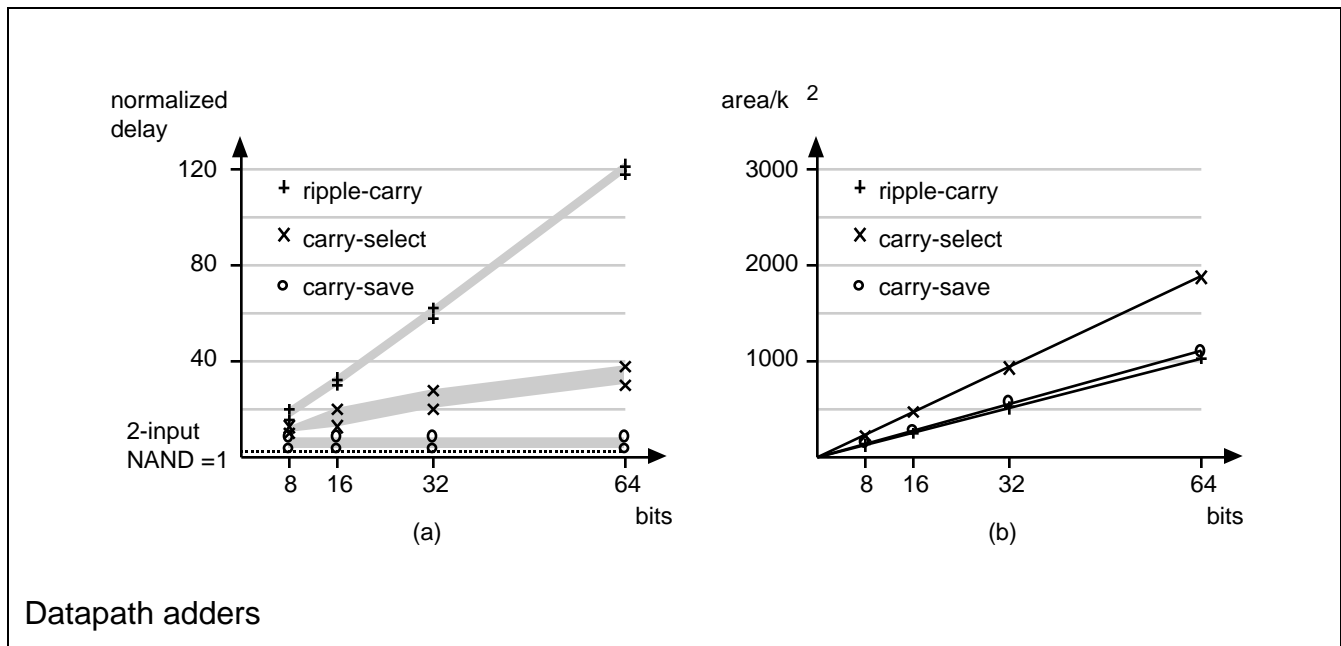
```

module m8bitCSum (C0, a, b, s, C8); // Verilog conditional-sum adder
for an FPGA //1
input [7:0] C0, a, b; output [7:0] s; output C8; //2
wire
A7,A6,A5,A4,A3,A2,A1,A0,B7,B6,B5,B4,B3,B2,B1,B0,S8,S7,S6,S5,S4,S3,S2
,S1,S0; //3
wire C0, C2, C4_2_0, C4_2_1, S5_4_0, S5_4_1, C6, C6_4_0, C6_4_1,
C8; //4
assign {A7,A6,A5,A4,A3,A2,A1,A0} = a;assign
{B7,B6,B5,B4,B3,B2,B1,B0} = b; //5
assign s = { S7,S6,S5,S4,S3,S2,S1,S0 }; //6
assign S0 = A0^B0^C0 ; // start of level 1: & = AND, ^ = XOR, | =
OR, ! = NOT //7
assign S1 = A1^B1^(A0&B0|(A0|B0)&C0) ; //8
assign C2 = A1&B1|(A1|B1)&(A0&B0|(A0|B0)&C0) ; //9
assign C4_2_0 = A3&B3|(A3|B3)&(A2&B2) ;assign C4_2_1 =
A3&B3|(A3|B3)&(A2|B2) ; //10
assign S5_4_0 = A5^B5^(A4&B4) ;assign S5_4_1 = A5^B5^(A4|B4) ; //11
assign C6_4_0 = A5&B5|(A5|B5)&(A4&B4) ;assign C6_4_1 =
A5&B5|(A5|B5)&(A4|B4) ; //12
assign S2 = A2^B2^C2 ; // start of level 2 //13
assign S3 = A3^B3^(A2&B2|(A2|B2)&C2) ; //14
assign S4 = A4^B4^(C4_2_0|C4_2_1&C2) ; //15
assign S5 = S5_4_0&
!(C4_2_0|C4_2_1&C2)|S5_4_1&(C4_2_0|C4_2_1&C2) ; //16
assign C6 = C6_4_0|C6_4_1&(C4_2_0|C4_2_1&C2) ; //17
assign S6 = A6^B6^C6 ; // start of level 3 //18
assign S7 = A7^B7^(A6&B6|(A6|B6)&C6) ; //19
assign C8 = A7&B7|(A7|B7s)&(A6&B6|(A6|B6)&C6) ; //20
endmodule //21

```

2.6.4 Multipliers

- Mental arithmetic: 15 (**multiplicand**) \times 19 (**multiplier**) = $15 \times (20-1) = 15 \times 2\bar{1}$
- Suppose we want to multiply by $B=00010111$ (decimal $16+4+2+1=23$)
- Use the canonical signed-digit vector (**CSD vector**) $D=0010\bar{1}001$ (decimal $32-8+1=23$)
- B has a **weight** of 4, but D has a weight of 3 — and saves hardware



To **recode** (or encode) any binary number, B, as a CSD vector, D: $D_i = B_i + C_i - 2C_{i+1}$, where C_{i+1} is the carry from the sum of $B_{i+1} + B_i + C_i$ (we start with $C_0=0$).

If $B=011$ ($B_2=0, B_1=1, B_0=1$; decimal 3), then:

$$D_0 = B_0 + C_0 - 2C_1 = 1 + 0 - 2 = \bar{1},$$

$$D_1 = B_1 + C_1 - 2C_2 = 1 + 1 - 2 = 0,$$

$$D_2 = B_2 + C_2 - 2C_3 = 0 + 1 - 0 = 1,$$

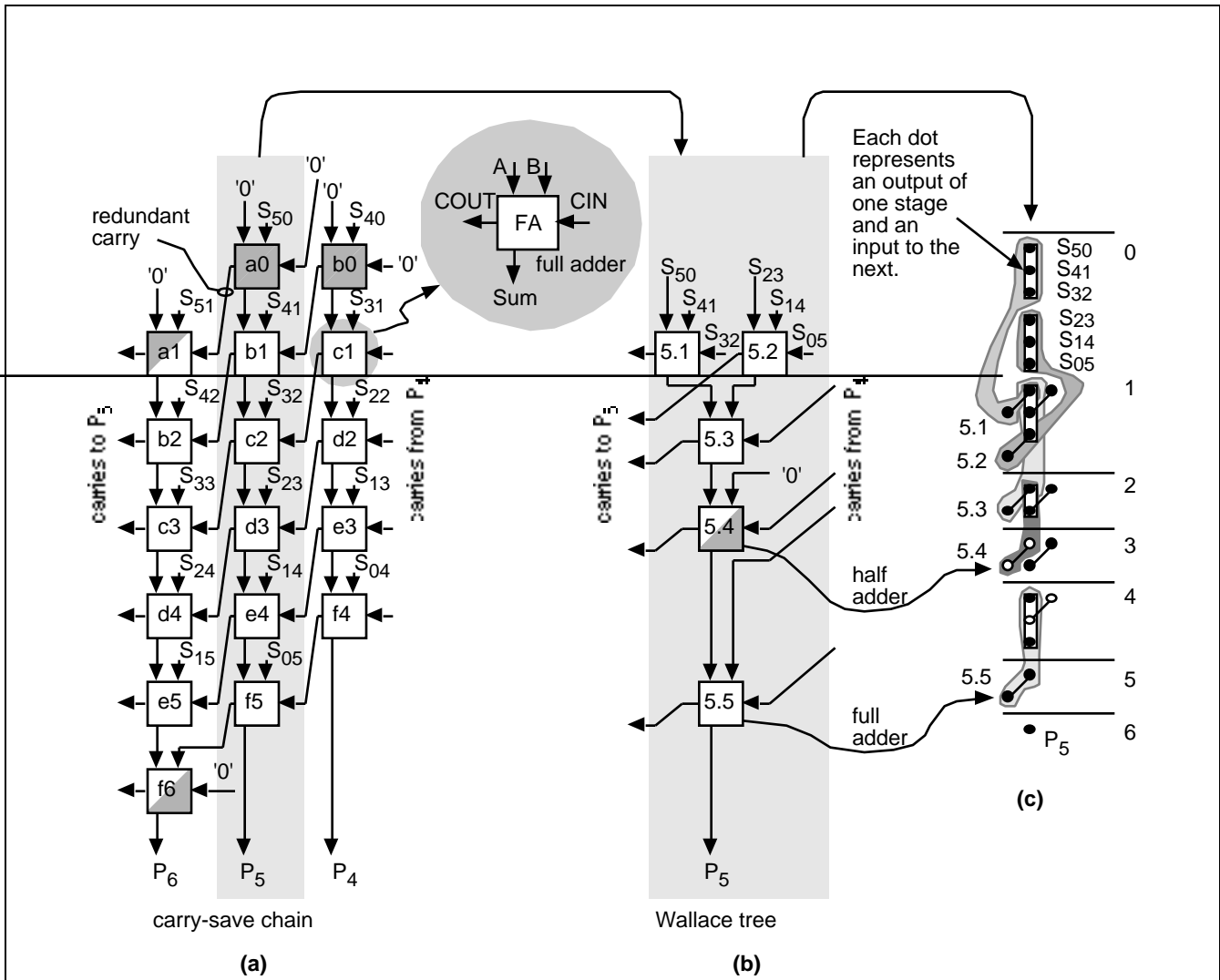
so that $D = 10\bar{1}$ (decimal $4-1=3$).

We can use a **radix** other than 2, for example **Booth encoding** (radix-4):

$B=101001$ (decimal $9-32=-23$) $E=\bar{1}\bar{2}1$ (decimal $-16-8+1=-23$)

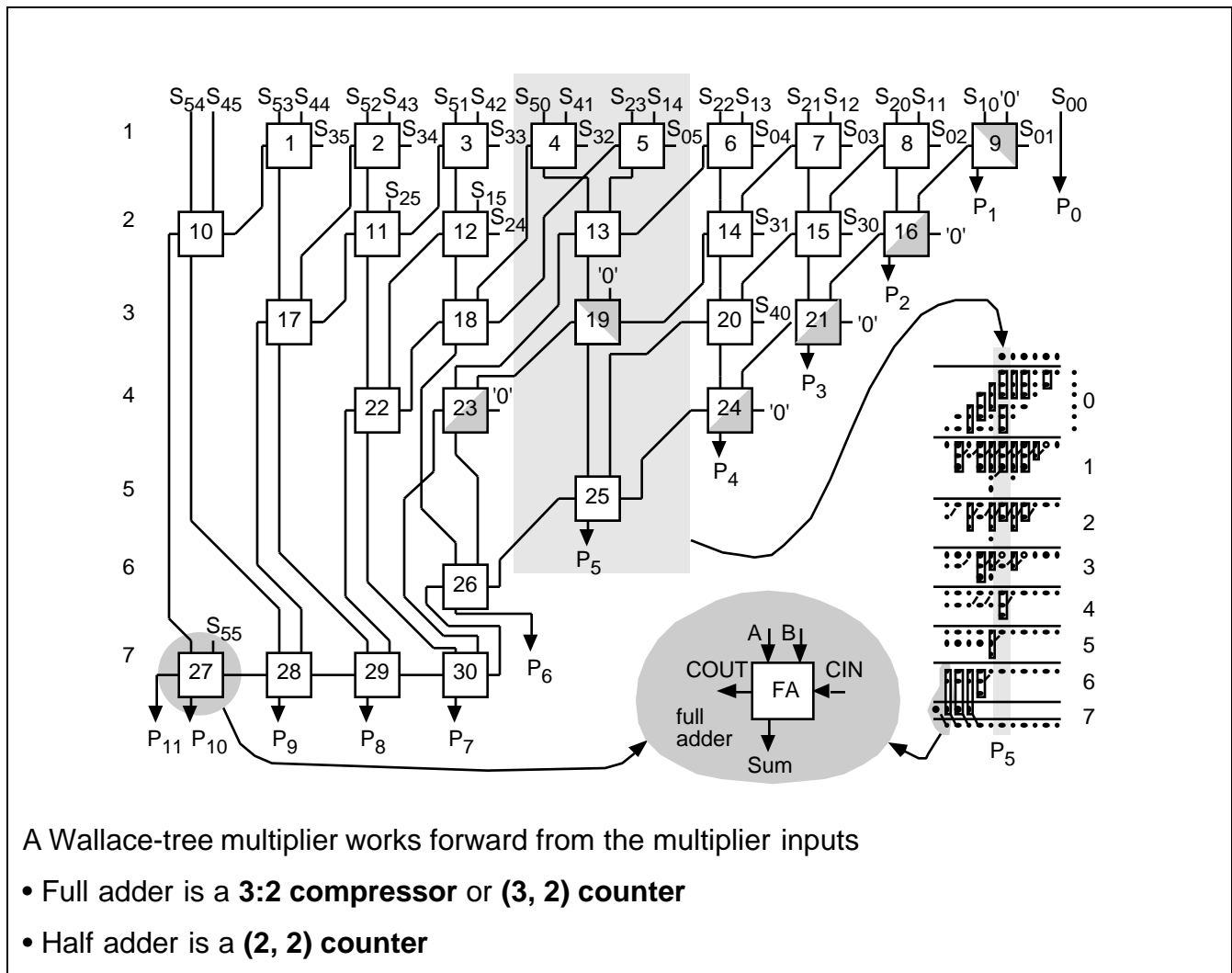
$B=01011$ (eleven) $E=1\bar{1}\bar{1}$ ($16-4-1$)

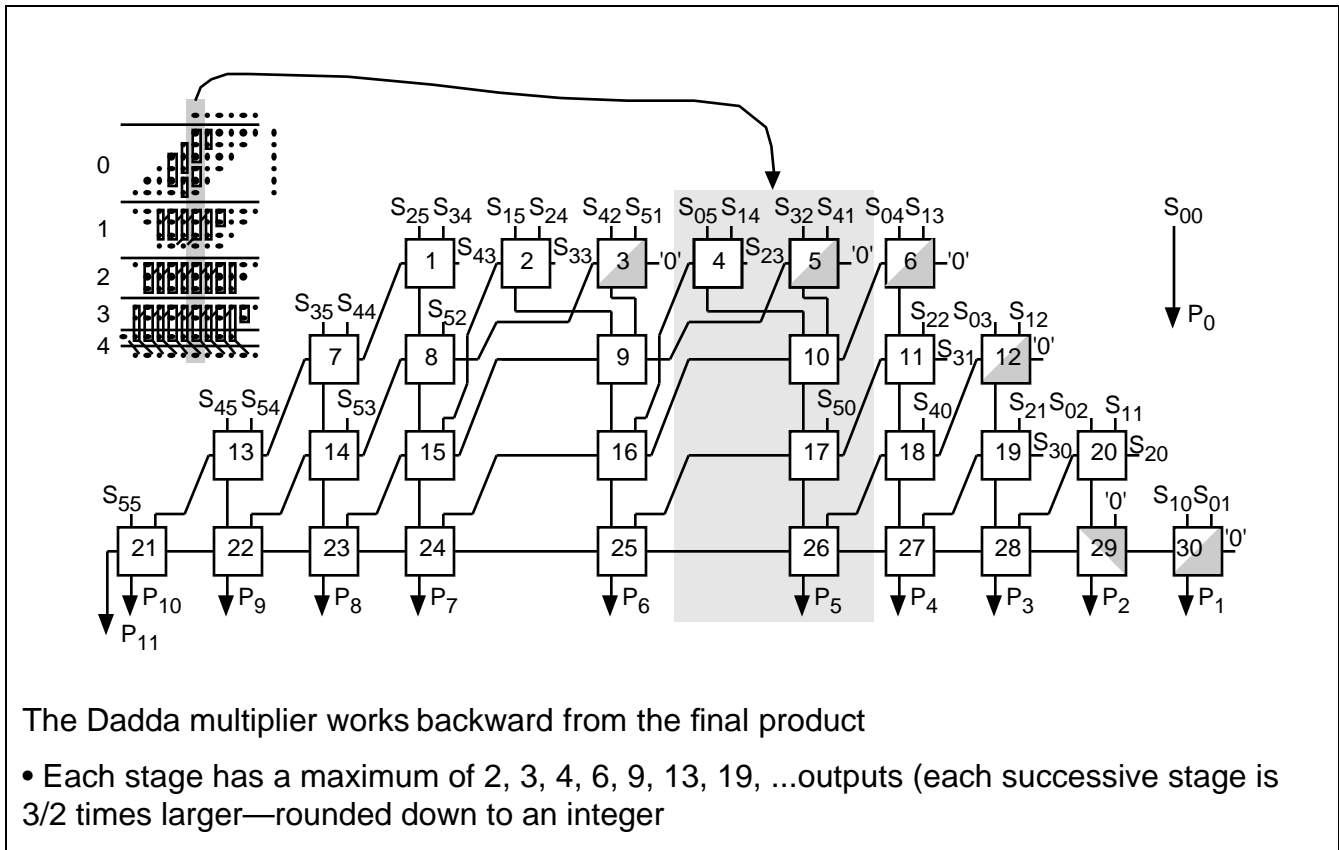
$B=101$ $E=\bar{1}1$



Tree-based multiplication – at each stage we have the following three choices:

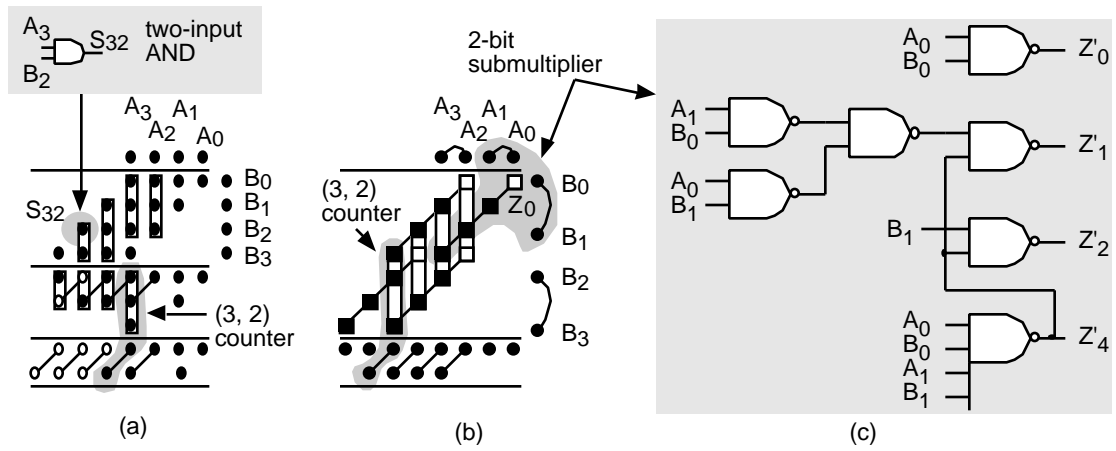
- (1) sum three outputs using a full adder
- (2) sum two outputs using a half adder
- (3) pass the outputs to the next stage





The number of stages and thus delay (in units of an FA delay—excluding the CPA) for an n -bit tree-based multiplier using (3, 2) counters is

$$\log_{1.5} n = \log_{10} n / \log_{10} 1.5 = \log_{10} n / 0.176$$



Ferrari-Stefanelli architecture “nests” multipliers

2.6.5 Other Arithmetic Systems

binary	decimal	redundant binary	CSD vector	
1010111	87	10101001	10101001	addend
+ 1100101	101	+ 11100111	+ 01100101	augend
		01001110	= 11001100	intermediate sum
		11000101	11000000	intermediate carry
= 10111100	= 188	111000100	101001100	sum

Redundant binary addition • redundant binary encoding avoids carry propagation					
A[i]	B[i]	A[i-1]	B[i-1]	Intermediate sum	Intermediate carry
$\bar{1}$	$\bar{1}$	x	x	0	$\bar{1}$
$\bar{1}$	0	A[i-1]=0/1 and B[i-1]=0/1		$\bar{1}$	0
0	$\bar{1}$	A[i-1]= $\bar{1}$ or B[i-1]= $\bar{1}$		1	$\bar{1}$
$\bar{1}$	1	x	x	0	0
1	$\bar{1}$	x	x	0	0
0	0	x	x	0	0
0	1	A[i-1]=0/1 and B[i-1]=0/1		$\bar{1}$	1
1	0	A[i-1]= $\bar{1}$ or B[i-1]= $\bar{1}$		1	0
1	1	x	x	0	1

- 101 (decimal) is **1100101** (in binary and CSD vector) or **1 $\bar{1}$ 100111**
- 188 (decimal) is **10111100** (in binary), **1 $\bar{1}$ 1000 $\bar{1}$ 00**, **10 $\bar{1}$ 00 $\bar{1}$ 100**, or **10 $\bar{1}$ 000 $\bar{1}$ 00** (CSD vector)
- **10 $\bar{1}$** is represented as 010010 (using sign magnitude) — rather wasteful

Residue number system

- 11 (decimal) is represented as [1, 2] residue (5, 3)
- $11R_5=11 \bmod 5=1$ and $11R_3=11 \bmod 3=2$
- The **size** of this system is $3 \times 5=15$
- We can now add, subtract, or multiply without using any carry

$$\begin{array}{rcl}
 4 & [4, 1] & 12 & [2, 0] & 3 & [3, 0] \\
 + 7 & + [2, 1] & - 4 & - [4, 1] & \times 4 & \times [4, 1] \\
 = 11 & = [1, 2] & = 8 & = [3, 2] & = 12 & = [2, 0]
 \end{array}$$

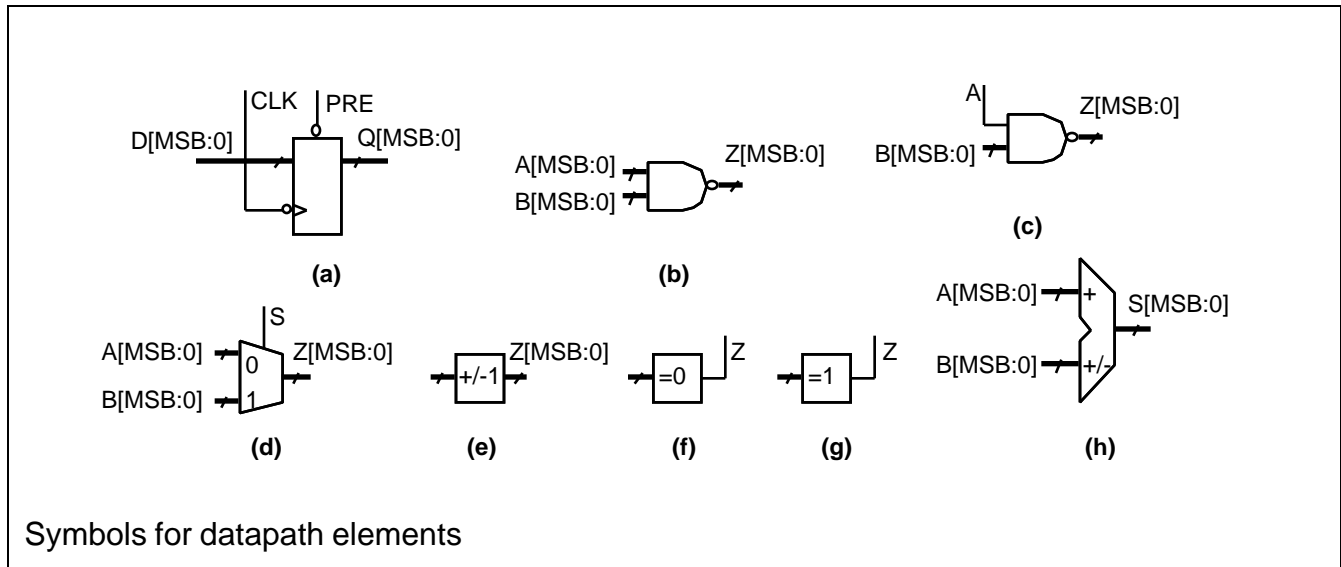
The 5, 3 residue number system								
n	residue 5	residue 3	n	residue 5	residue 3	n	residue 5	residue 3
0	0	0	5	0	2	10	0	1
1	1	1	6	1	0	11	1	2
2	2	2	7	2	1	12	2	0
3	3	0	8	3	2	13	3	1
4	4	1	9	4	0	14	4	2

2.6.6 Other Datapath Operators

Full subtracter

$$\begin{aligned} \text{DIFF} &= A \oplus \text{NOT}(B) \oplus (\text{BIN}) \\ &= \text{SUM}(A, \text{NOT}(B), \text{NOT}(\text{BIN})) \end{aligned}$$

$$\begin{aligned} \text{NOT}(\text{BOUT}) &= A \cdot \text{NOT}(B) + A \cdot \text{NOT}(\text{BIN}) + \text{NOT}(B) \cdot \text{NOT}(\text{BIN}) \\ &= \text{MAJ}(\text{NOT}(A), B, \text{NOT}(\text{BIN})) \end{aligned}$$

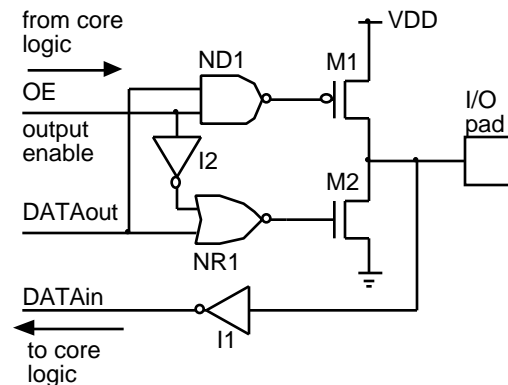


Keywords: adder/subtractor • barrel shifter • normalizer • denormalizer • leading-one detector • priority encoder • exponent correcter • accumulator • multiplier-accumulator (MAC) • incrementer • decrements • incrementer/decrementer • all-zeros detector • all-ones detector • register file • first-in first-out register (FIFO) • last-in first-out register (LIFO)

2.7 I/O Cells

Keywords: Tri-State[®] is a registered trademark of National Semiconductor) • **drivers** • **contention** • **bus keeper** or **bus-hold** cell (TI calls this Bus-Friendly logic) • **slew rate** • **power-supply bounce** • **simultaneously switching outputs (SSOs)** • **quiet-I/O** • **bidirectional I/O** • **open-drain** • **level shifter** • **electrostatic discharge, or ESD** • **electrical overstress (EOS)** • **ESD implant** • **human-body model (HBM)** • **machine model (MM)** • **charge-device model (CDM, also called device charge–discharge)** • **latch-up** • **undershoot** • **overshoot** • **guard rings**

A three-state bidirectional output buffer



2.8 Cell Compilers

Keywords: silicon compilers • RAM compiler • multiplier compiler • single-port RAM • dual-port RAMs • multiport RAMs • asynchronous • synchronous • model compiler • netlist compiler • correct by construction

2.9 Summary

- The use of transistors as switches
- The difference between a flip-flop and a latch
- The meaning of setup time and hold time

- Pipelines and latency
- The difference between datapath, standard-cell, and gate-array logic cells
- Strong and weak logic levels
- Pushing bubbles
- Ratio of logic
- Resistance per square of layers and their relative values in CMOS
- Design rules and

2.10 Problems

Suggested homework: 2.1, 2.2, 2.38, 2.39 (from *ASICs... the book*)

ASIC LIBRARY DESIGN

3

Key concepts: Tau, logical effort, and the prediction of delay • Sizes of cells, and their drive strengths • Cell importance • The difference between gate-array macros, standard cells, and datapath cells

ASIC design uses predefined and precharacterized cells from a library—so we need to design or buy a cell library. A knowledge of ASIC library design is not necessary but makes it easier to use library cells effectively.

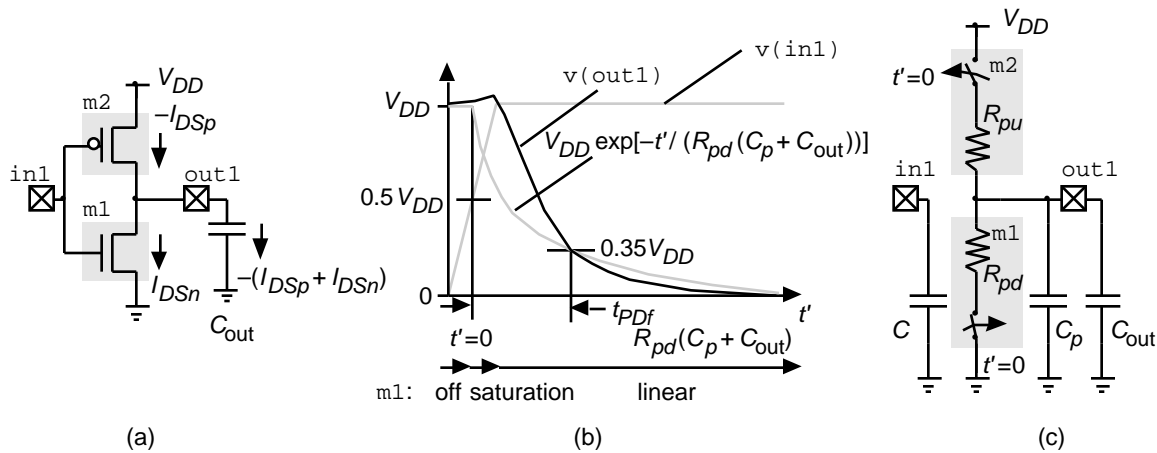
3.1 Transistors as Resistors

$$0.35V_{DD} = V_{DD} \exp \frac{-t_{PDf}}{R_{pd}(C_{out} + C_p)}$$

An output trip point of 0.35 is convenient because $\ln(1/0.35) = 1.04$ and thus

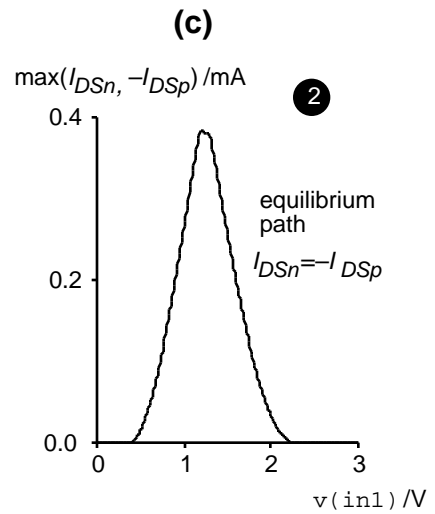
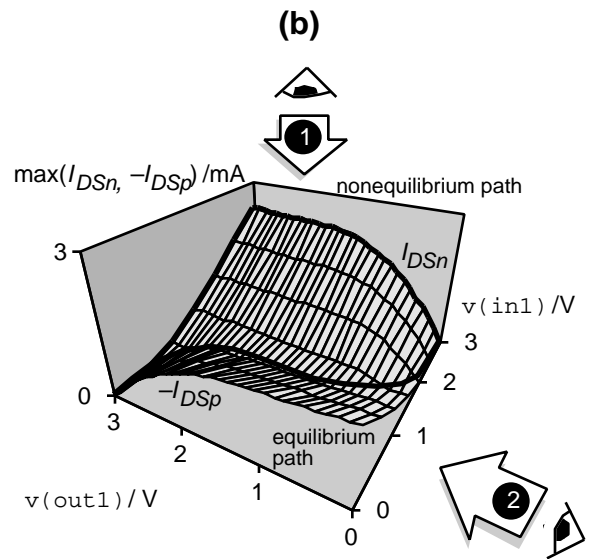
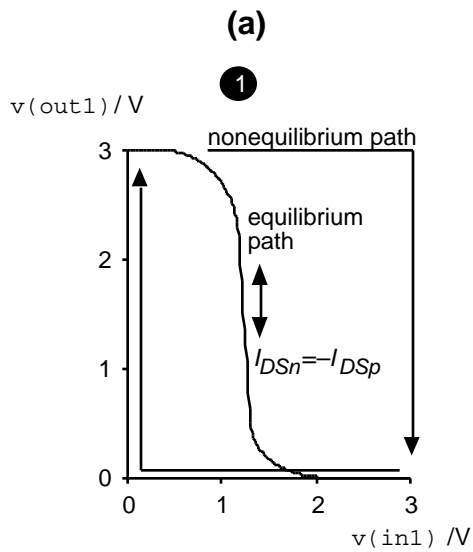
$$t_{PDf} = R_{pd}(C_{out} + C_p) \ln(1/0.35) = R_{pd}(C_{out} + C_p)$$

For output trip points of 0.1/0.9 we multiply by $-\ln(0.1) = 2.3$, because $\exp(-2.3) = 0.100$



A linear model for CMOS logic delay

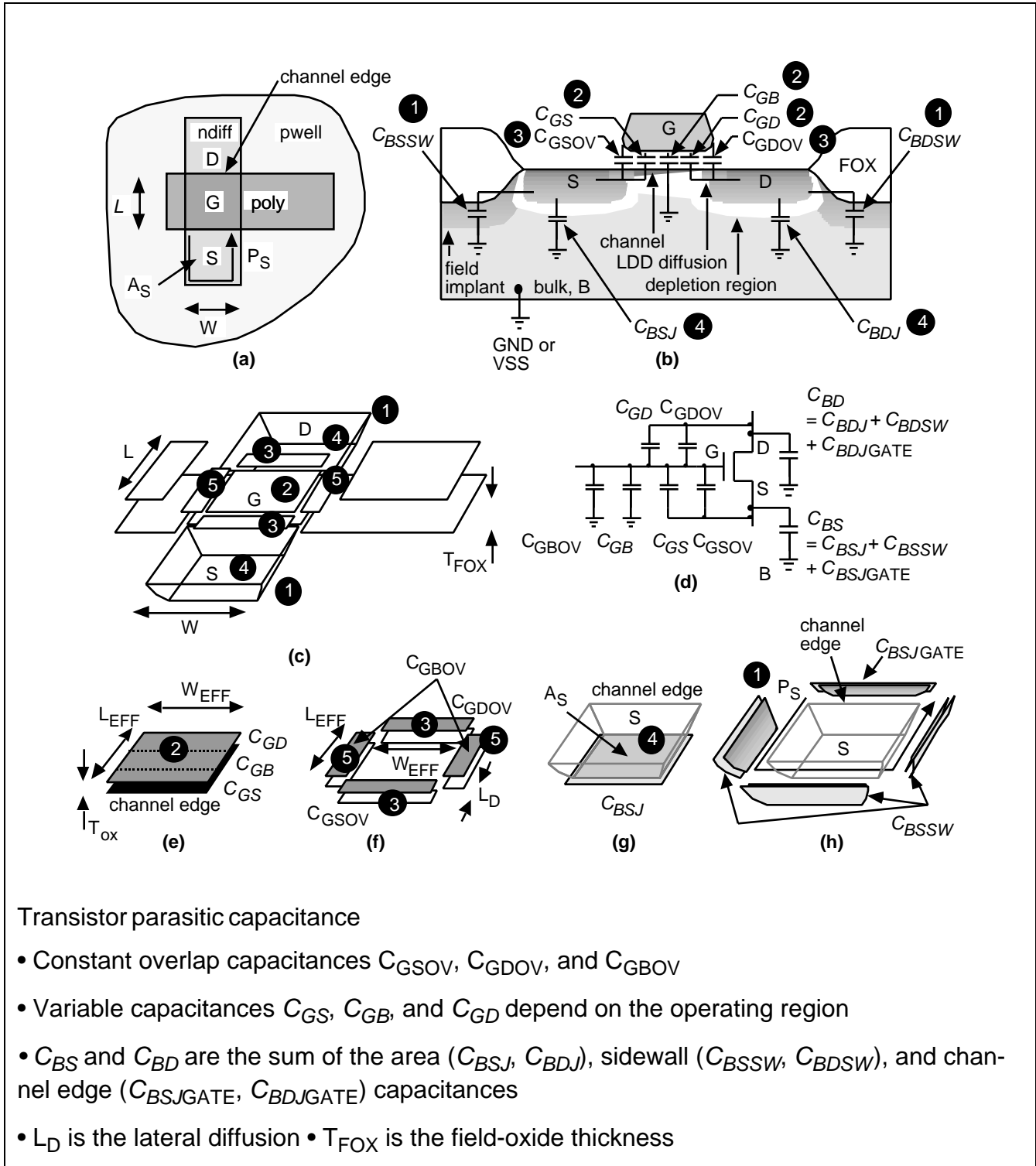
- Ideal switches = no delay • Resistance and capacitance causes delay
- Load capacitance, C_{out} • parasitic output capacitance, C_p • input capacitance, C
- Linearize the switch resistance • Pull-up resistance, R_{pu} • pull-down resistance, R_{pd}
- Measure and compare the input, $v(in1)$ and output, $v(out1)$
- Input trip point of 0.5 • output trip points are 0.35 (falling) and 0.65 (rising)
- The linear prop-ramp model: falling propagation delay, $t_{PDf} R_{pd}(C_p + C_{out})$



CMOS inverter characteristics

- Equilibrium switching
- Non-equilibrium switching
- Nonlinear switching resistance
- Switching current

3.2 Transistor Parasitic Capacitance



Transistor parasitic capacitance

- Constant overlap capacitances C_{GSOV} , C_{GDOV} , and C_{GBOV}
- Variable capacitances C_{GS} , C_{GB} , and C_{GD} depend on the operating region
- C_{BS} and C_{BD} are the sum of the area (C_{BSJ} , C_{BDJ}), sidewall (C_{BSSW} , C_{BDSW}), and channel edge ($C_{BSJGATE}$, $C_{BDJGATE}$) capacitances
- L_D is the lateral diffusion • T_{FOX} is the field-oxide thickness

NAME	m1	m2
MODEL	CMOSN	CMOSP
ID	7.49E-11	-7.49E-11
VGS	0.00E+00	-3.00E+00
VDS	3.00E+00	-4.40E-08
VBS	0.00E+00	0.00E+00
VTH	4.14E-01	-8.96E-01
VDSAT	3.51E-02	-1.78E+00
GM	1.75E-09	2.52E-11
GDS	1.24E-10	1.72E-03
GMB	6.02E-10	7.02E-12
CBD	2.06E-15	1.71E-14
CBS	4.45E-15	1.71E-14
CGSOV	1.80E-15	2.88E-15
CGDOV	1.80E-15	2.88E-15
CGBOV	2.00E-16	2.01E-16
CGS	0.00E+00	1.10E-14
CGD	0.00E+00	1.10E-14
CGB	3.88E-15	0.00E+00

- ID (I_{DS}), VGS, VDS, VBS, VTH (V_t), and VDSAT ($V_{DS(sat)}$) are DC parameters
- GM, GDS, and GMB are small-signal conductances (corresponding to I_{DS}/V_{GS} , I_{DS}/V_{DS} , and I_{DS}/V_{BS} , respectively)

Calculations of parasitic capacitances for an n-channel MOS transistor.		
PSpice	Equation	Values ¹ for $V_{GS}=0V, V_{DS}=3V, V_{SB}=0V$
CBD	$C_{BD} = C_{BDJ} + C_{BDSW}$ $C_{BDJ} + A_D C_J (1 + V_{DB}/V_B)^{-m_J} \quad (V_B = V_{PB})$ $C_{BDSW} = P_D C_{JSW} (1 + V_{DB}/V_B)^{-m_{JSW}}$ (P_D may or may not include channel edge)	$C_{BD} = 1.855 \times 10^{-13} + 2.04 \times 10^{-16} = 2.06 \times 10^{-13} \text{ F}$ $C_{BDJ} = (4.032 \times 10^{-15})(1 + (3/1))^{-0.56} = 1.86 \times 10^{-15} \text{ F}$ $C_{BDSW} = (4.2 \times 10^{-16})(1 + (3/1))^{-0.5} = 2.04 \times 10^{-16} \text{ F}$
CBS	$C_{BS} = C_{BSJ} + C_{BSSW}$ $C_{BSJ} + A_S C_J (1 + V_{SB}/V_B)^{-m_J}$ $C_{BSSW} = P_S C_{JSW} (1 + V_{SB}/V_B)^{-m_{JSW}}$	$C_{BS} = 4.032 \times 10^{-15} + 4.2 \times 10^{-16} = 4.45 \times 10^{-15} \text{ F}$ $A_S C_J = (7.2 \times 10^{-15})(5.6 \times 10^{-4}) = 4.03 \times 10^{-15} \text{ F}$ $P_S C_{JSW} = (8.4 \times 10^{-6})(5 \times 10^{-11}) = 4.2 \times 10^{-16} \text{ F}$
CGSOV	$C_{GSOV} = W_{EFF} C_{GSO} ; W_{EFF} = W - 2W_D$	$C_{GSOV} = (6 \times 10^{-6})(3 \times 10^{-10}) = 1.8 \times 10^{-16} \text{ F}$
CGDOV	$C_{GDOV} = W_{EFF} C_{GSO}$	$C_{GDOV} = (6 \times 10^{-6})(3 \times 10^{-10}) = 1.8 \times 10^{-15} \text{ F}$
CGBOV	$C_{GBOV} = L_{EFF} C_{GBO} ; L_{EFF} = L - 2L_D$	$C_{GDOV} = (0.5 \times 10^{-6})(4 \times 10^{-10}) = 2 \times 10^{-16} \text{ F}$
CGS	$C_{GS}/C_O = 0$ (off), 0.5 (lin.), 0.66 (sat.) C_O (oxide capacitance) = $W_{EF} L_{EFF} \epsilon_{ox} / T_{ox}$	$C_O = (6 \times 10^{-6})(0.5 \times 10^{-6})(0.00345) = 1.03 \times 10^{-14} \text{ F}$ $C_{GS} = 0.0 \text{ F}$
CGD	$C_{GD}/C_O = 0$ (off), 0.5 (lin.), 0 (sat.)	$C_{GD} = 0.0 \text{ F}$
CGB	$C_{GB} = 0$ (on), = C_O in series with C_{GS} (off)	$C_{GB} = 3.88 \times 10^{-15} \text{ F}$, C_S =depletion capacitance
¹ Input	.MODEL CMOSN NMOS LEVEL=3 PHI=0.7 TOX=10E-09 XJ=0.2U TPG=1 VTO=0.65 DELTA=0.7 + LD=5E-08 KP=2E-04 UO=550 THETA=0.27 RSH=2 GAMMA=0.6 NSUB=1.4E+17 NFS=6E+11 + VMAX=2E+05 ETA=3.7E-02 KAPPA=2.9E-02 CGDO=3.0E-10 CGSO=3.0E-10 CGBO=4.0E-10 + CJ=5.6E-04 MJ=0.56 CJSW=5E-11 MJSW=0.52 PB=1 m1 out1 in1 0 0 cmosn W=6U L=0.6U AS=7.2P AD=7.2P PS=8.4U PD=8.4U	

3.2.1 Junction Capacitance

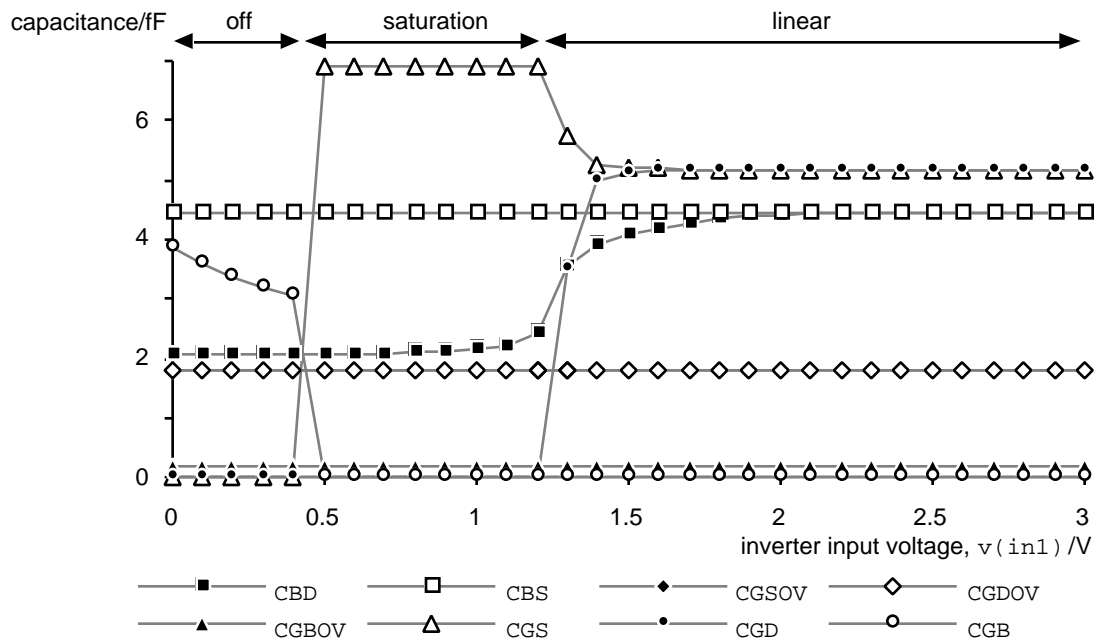
- Junction capacitances, C_{BD} and C_{BS} , consist of two parts: junction area and sidewall
- Both C_{BD} and C_{BS} have different physical characteristics with parameters: C_J and M_J for the junction, C_{JSW} and M_{JSW} for the sidewall, and P_B is common
- C_{BD} and C_{BS} depend on the voltage across the junction (V_{DB} and V_{SB})
- The sidewalls facing the channel ($C_{BSJGATE}$ and $C_{BDJGATE}$) are different from the sidewalls that face the field
- It is a mistake to exclude the gate edge assuming it is in the rest of the model—it is not
- In HSPICE there is a separate mechanism to account for the channel edge capacitance (using parameters ACM and $CJGATE$)

3.2.2 Overlap Capacitance

- The overlap capacitance calculations for C_{GSOV} and C_{GDOV} account for lateral diffusion
- SPICE parameter $LD=5E-08$ or $L_D=0.05\mu m$
- Not all SPICE versions use the equivalent parameter for width reduction, WD , in calculating C_{GDOV}
- Not all SPICE versions subtract W_D to form W_{EFF}

3.2.3 Gate Capacitance

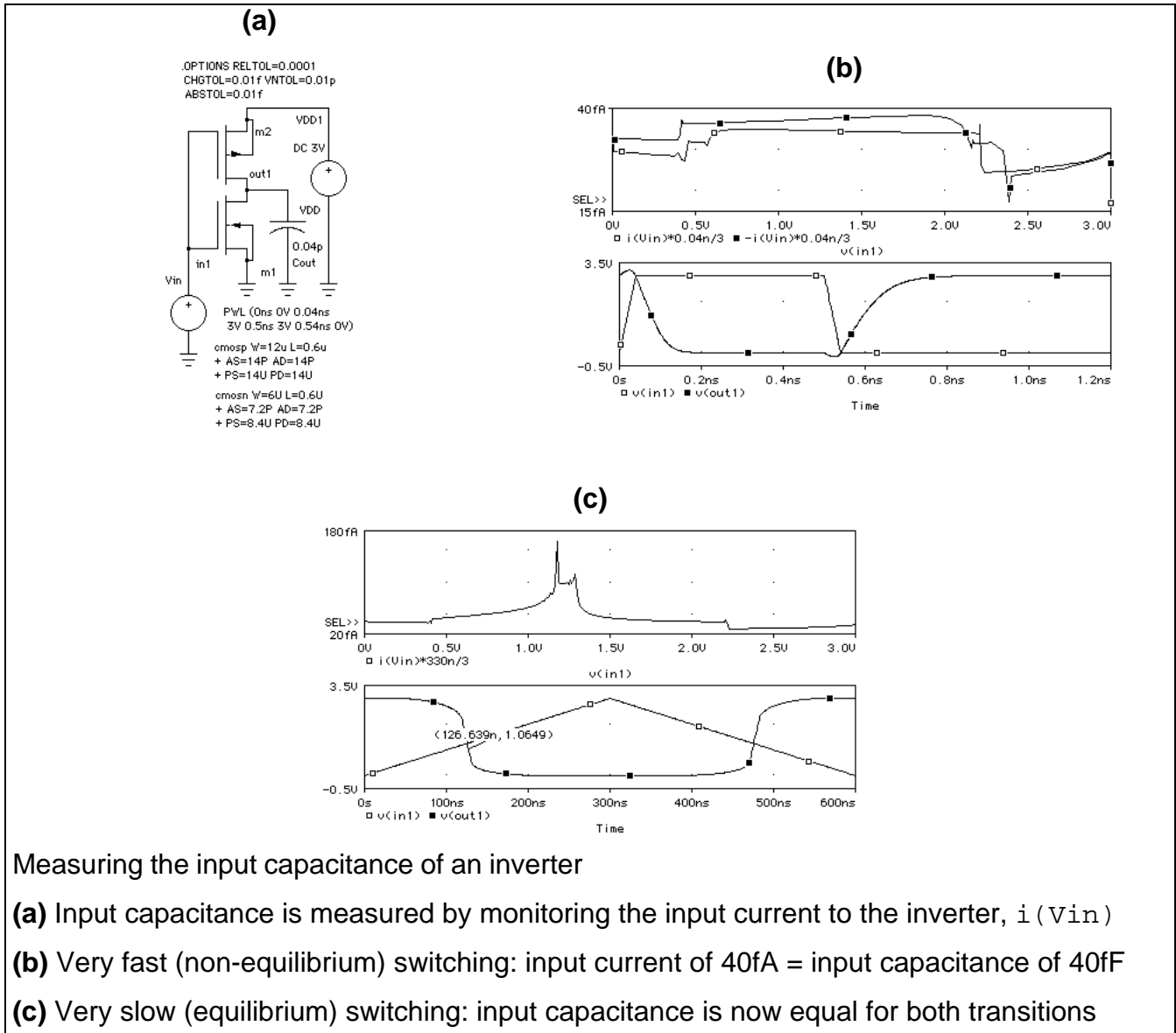
- The gate capacitance depends on the operating region
- The gate–source capacitance C_{GS} varies from zero (off) to $0.5C_O$ in the linear region to $(2/3)C_O$ in the saturation region
- The gate–drain capacitance C_{GD} varies from zero (off) to $0.5C_O$ (linear region) and back to zero (saturation region)
- The gate–bulk capacitance C_{GB} is two capacitors in series: the fixed gate-oxide capacitance, C_O , and the variable depletion capacitance, C_S
- As the transistor turns on the channel shields the bulk from the gate—and C_{GB} falls to zero
- Even with $V_{GS}=0V$, the depletion width under the gate is finite and thus C_{GB} is less than C_O

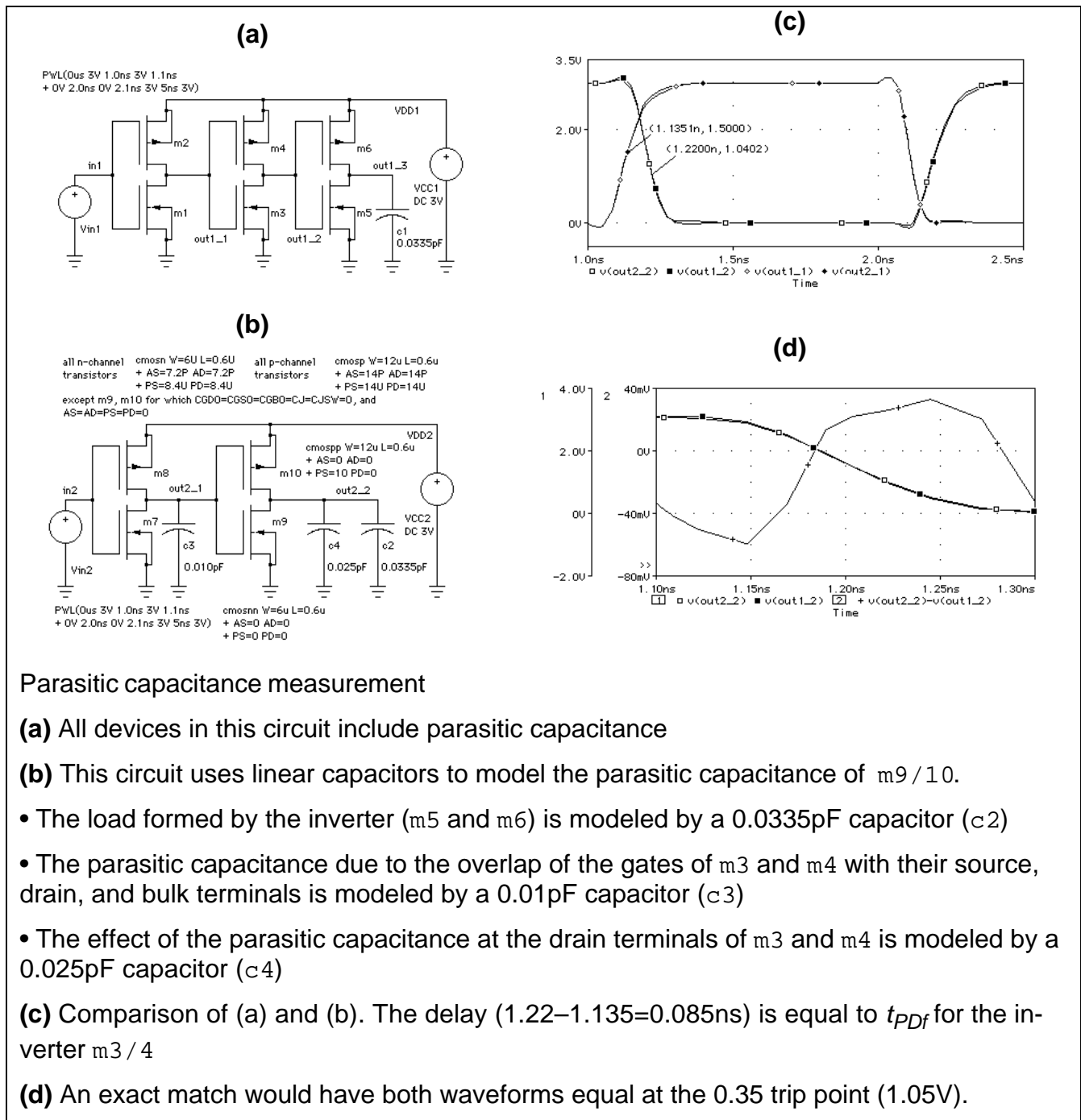


The variation of n-channel transistor parasitic capacitance

- PSpice v5.4 (LEVEL=3)
- Created by varying the input voltage, $v(in1)$, of an inverter
- Data points are joined by straight lines
- Note that $CGSOV=CGDOV$

3.2.4 Input Slew Rate





Parasitic capacitance measurement

(a) All devices in this circuit include parasitic capacitance

(b) This circuit uses linear capacitors to model the parasitic capacitance of m9 / 10.

- The load formed by the inverter (m5 and m6) is modeled by a 0.0335pF capacitor (c2)
- The parasitic capacitance due to the overlap of the gates of m3 and m4 with their source, drain, and bulk terminals is modeled by a 0.01pF capacitor (c3)
- The effect of the parasitic capacitance at the drain terminals of m3 and m4 is modeled by a 0.025pF capacitor (c4)

(c) Comparison of (a) and (b). The delay (1.22–1.135=0.085ns) is equal to t_{PDf} for the inverter m3 / 4

(d) An exact match would have both waveforms equal at the 0.35 trip point (1.05V).

3.3 Logical Effort

We extend the prop-ramp model with a “catch all” term, t_q , that includes:

- delay due to internal parasitic capacitance
- the time for the input to reach the switching threshold of the cell
- the dependence of the delay on the slew rate of the input waveform

$$t_{PD} = R(C_{out} + C_p) + t_q$$

We can **scale** any logic cell by a scaling factor s : $t_{PD} = (R/s) \cdot (C_{out} + sC_p) + st_q$

$$t_{PD} = RC \frac{C_{out}}{C_{in}} + RC_p + st_q$$

$$\text{Normalizing the delay: } d = \frac{(RC) (C_{out} / C_{in}) + RC_p + st_q}{RC} = f + p + q$$

The time constant $\tau_{inv} = R_{inv} C_{inv}$, is a basic property of any CMOS technology

The delay equation is the sum of three terms, $d = f + p + q$ or
 delay = **effort delay** + **parasitic delay** + **nonideal delay**

The effort delay f is the product of **logical effort**, g , and **electrical effort**, h : $f = gh$

Thus, delay = logical effort \times electrical effort + parasitic delay + nonideal delay

- R and C will change as we scale a logic cell, but the RC product stays the same
- Logical effort is independent of the size of a logic cell
- We can find logical effort by scaling a logic cell to have the same drive as a 1X minimum-size inverter
- Then the logical effort, g , is the ratio of the input capacitance, C_{in} , of the 1X logic cell to C_{inv}

1 Measure the input capacitance of a minimum-size inverter.
 $C_{inv} = 2 + 1 = 3$

2 Make the cell have the same drive strength as a minimum-size inverter.

3 Measure ratio of cell input capacitance to that of a minimum-size inverter.
 $g = C_{in}/C_{inv} = 4/3$

(a) (b) (c)

Logical effort • For a two-input NAND cell, the logical effort, $g=4/3$

(a) Find the input capacitance, C_{inv} , looking into the input of a minimum-size inverter in terms of the gate capacitance of a minimum-size device

(b) Size a logic cell to have the same drive strength as a minimum-size inverter (assuming a logic ratio of 2). The input capacitance looking into one of the logic-cell terminals is then C_{in}

(c) The logical effort of a cell is C_{in}/C_{inv}

The h depends only on the load capacitance C_{out} connected to the output of the logic cell and the input capacitance of the logic cell, C_{in} ; thus

electrical effort $h = C_{out}/C_{in}$

parasitic delay $p = RC_p/$ (the parasitic delay of a minimum-size inverter is: $p_{inv} = C_p/ C_{inv}$)

nonideal delay $q = st_q/$

Cell effort, parasitic delay, and nonideal delay (in units of) for single-stage CMOS cells				
Cell	Cell effort (logic ratio=2)	Cell effort (logic ratio=r)	Parasitic delay/	Nonideal delay/
inverter	1 (by definition)	1 (by definition)	p_{inv} (by definition)	q_{inv} (by definition)
n-input NAND	$(n+2)/3$	$(n+r)/(r+1)$	np_{inv}	nq_{inv}
n-input NOR	$(2n+1)/3$	$(nr+1)/(r+1)$	np_{inv}	nq_{inv}

3.3.1 Predicting Delay

- Example: predict the delay of a three-input NOR logic cell
- 2X drive
- driving a net with a fanout of four
- 0.3pF total load capacitance (input capacitance of cells we are driving plus the interconnect)
- $p=3p_{inv}$ and $q=3q_{inv}$ for this cell
- the input gate capacitance of a 1X drive, three-input NOR logic cell is equal to gC_{inv}
- for a 2X logic cell, $C_{in} = 2gC_{inv}$

$$gh = g \frac{C_{out}}{C_{in}} = \frac{g \cdot (0.3 \text{ pF})}{2gC_{inv}} = \frac{(0.3 \text{ pF})}{(2) \cdot (0.036 \text{ pF})} \quad (\text{Notice } g \text{ cancels out in this equation})$$

The delay of the NOR logic cell, in units of τ , is thus

$$d = gh + p + q = \frac{0.3 \times 10^{-12}}{(2) \cdot (0.036 \times 10^{-12})} + (3) \cdot (1) + (3) \cdot (1.7)$$

$$= 4.1666667 + 3 + 5.1$$

$$= 12.266667 \quad \text{equivalent to an absolute delay, } t_{PD} = 12.3 \times 0.06 \text{ ns} = 0.74 \text{ ns}$$

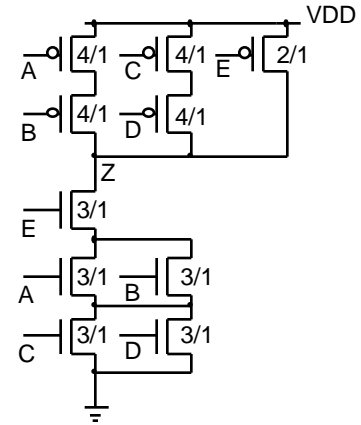
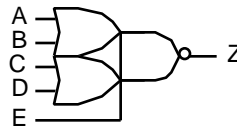
The delay for a 2X drive, three-input NOR logic cell is $t_{PD} = (0.03 + 0.72C_{out} + 0.60) \text{ ns}$

With $C_{out} = 0.3 \text{ pF}$, $t_{PD} = 0.03 + (0.72) \cdot (0.3) + 0.60 = 0.846 \text{ ns}$ compared to our prediction of 0.74ns

3.3.2 Logical Area and Logical Efficiency

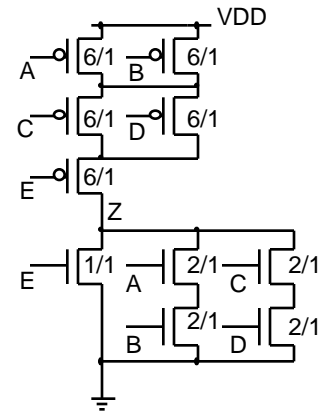
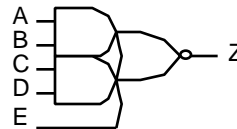
An OAI221 logic cell

- **Logical-effort vector $g=(7/3, 7/3, 5/3)$**
- **The logical area is 33 logical squares**



An AOI221 logic cell

- **$g=(8/3, 8/3, 7/3)$**
- **Logical area is 39 logical squares**
- **Less logically efficient than OAI221**



3.3.3 Logical Paths

$$\text{path delay } D = \sum_i g_i h_i + \sum_i (p_i + q_i)$$

3.3.4 Multistage Cells

(a)

$g_0=1$ $g_2=1.4$ $g_3=1.4$ AOI221
 $p_0=1$ $p_2=2$ $p_3=2$
 $q_0=1.7$ $q_2=3.4$ $q_3=3.4$
 $g_4=1$
 $p_4=1$
 $q_4=1.7$

$g_1=(2, 1.6)$
 $p_1=3$
 $q_1=5.4$

delay d_1
 $h_0=1.4$ $h_2=1.0$
 $h_3=0.7$
 $h_4=C_L$

$$d_1 = (g_0 h_0 + p_0 + q_0) + (g_2 h_2 + p_2 + q_2) + (g_3 h_3 + p_3 + q_3) + (g_4 h_4 + p_4 + q_4)$$

$$= (1 \times 1.4 + 1 + 1.7) + (1.4 \times 1 + 2 + 3.4) + (1.4 \times 0.7 + 2 + 3.4) + (1 \times C_L + 1 + 1.7) = 20 + C_L$$

(b)

$g_0=1$ $g_1=(2.6, 2.6, 2.2)$
 $p_0=1$ $p_1=5$
 $q_0=1.7$ $q_1=8.5$

delay d_1
 $h_1=C_L$

$$d_1 = (1 \times 2.6 + 1 + 1.7) + (1 \times C_L + 5 + 8.5) = 18.8 + C_L$$

(b) is slightly faster than (a)

Logical paths • Comparison of multistage and single-stage implementations

(a) An AOI221 logic cell constructed as a multistage cell, $d_1 = 20 + C_L$

(b) A single-stage AOI221 logic cell, $d_1 = 18.8 + C_L$

3.3.5 Optimum Delay

path logical effort $G = \prod_{i \text{ path}} g_i$

path electrical effort $H = \prod_{i \text{ path}} h_i \frac{C_{out}}{C_{in}}$

C_{out} is the load and C_{in} is the first input capacitance on the path

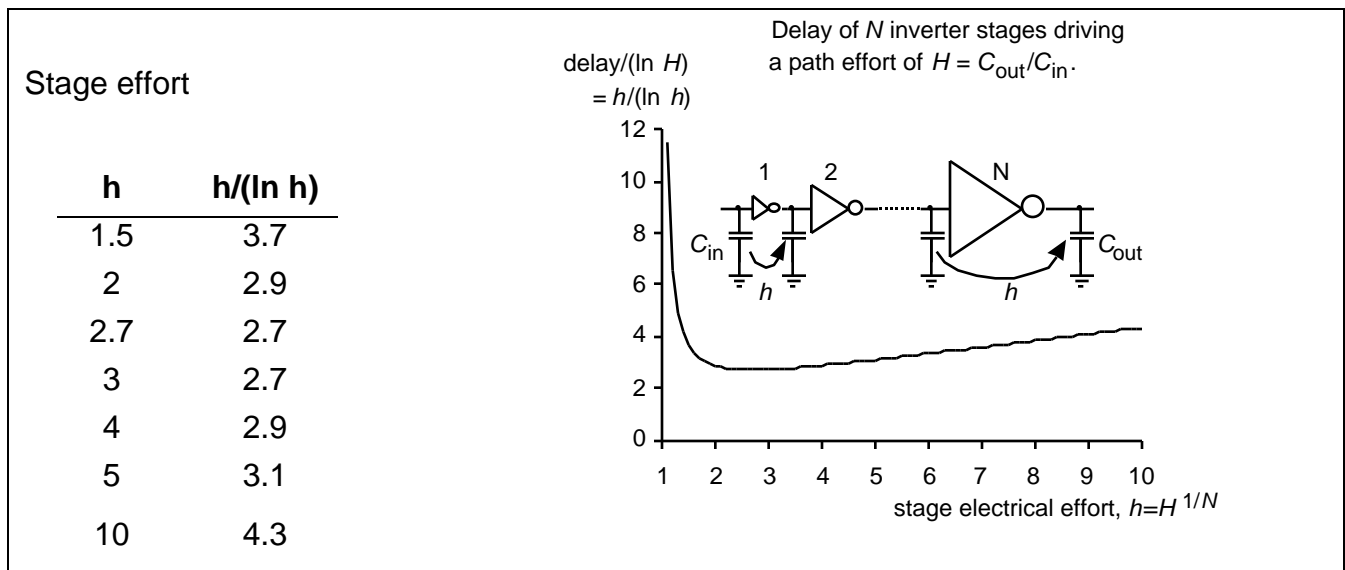
path effort $F = GH$

optimum effort delay $f_i = g_i h_i = F^{1/N}$

optimum path delay $D^* = NF^{1/N} = N(GH)^{1/N} + P + Q$

$$P + Q = \prod_{i \text{ path}} p_i + h_i$$

3.3.6 Optimum Number of Stages



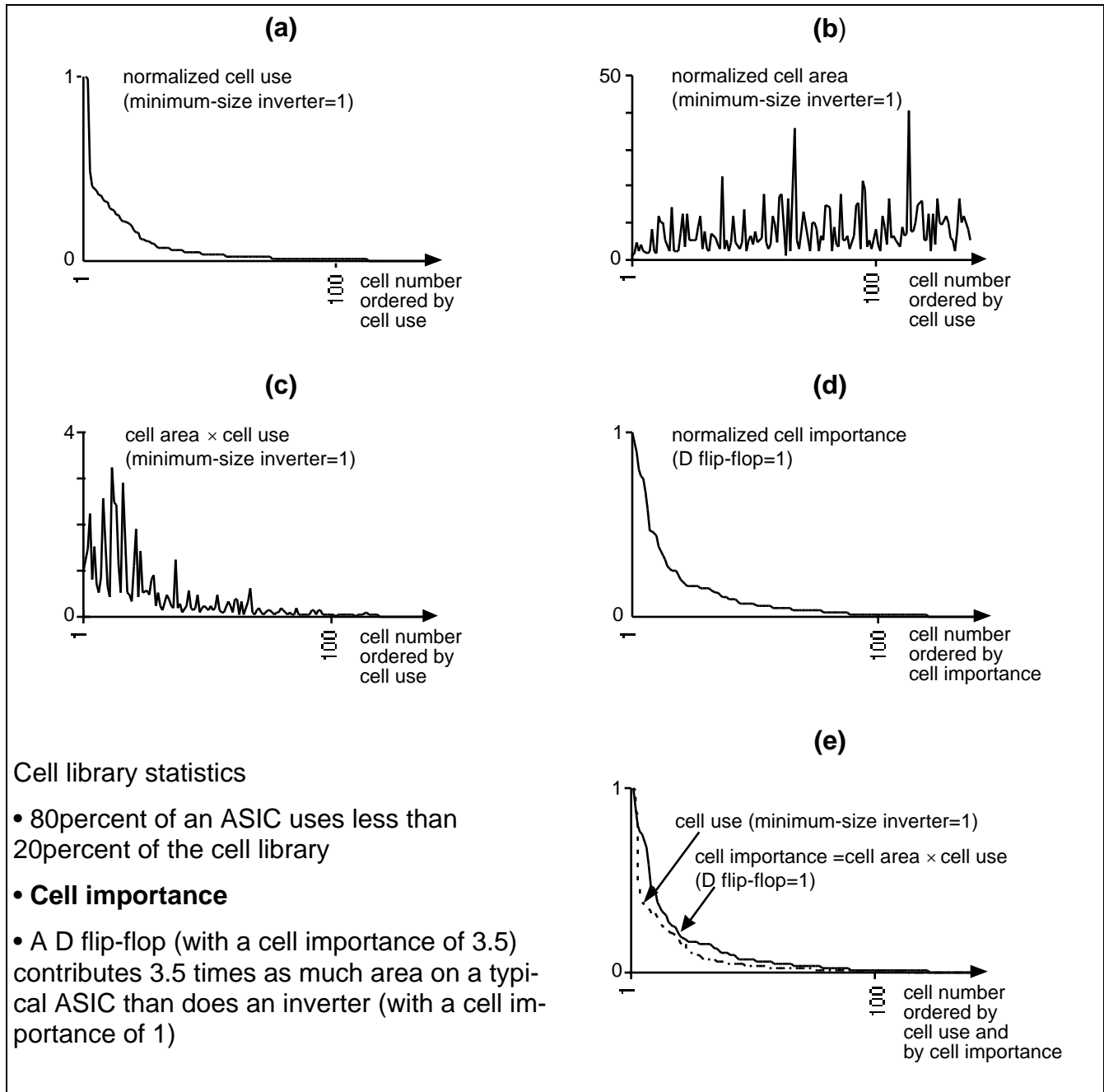
- Chain of N inverters each with equal stage effort, $f = gh$
- Total path delay is $Nf = Ngh = Nh$, since $g = 1$ for an inverter

- To drive a path electrical effort H , $h^N=H$, or $N\ln h=\ln H$
- Delay, $Nh = h\ln H/\ln h$
- Since $\ln H$ is fixed, we can only vary $h/\ln(h)$
- $h/\ln(h)$ is a shallow function with a minimum at $h=e \approx 2.718$
- Total delay is $Ne=e\ln H$

3.4 Library-Cell Design

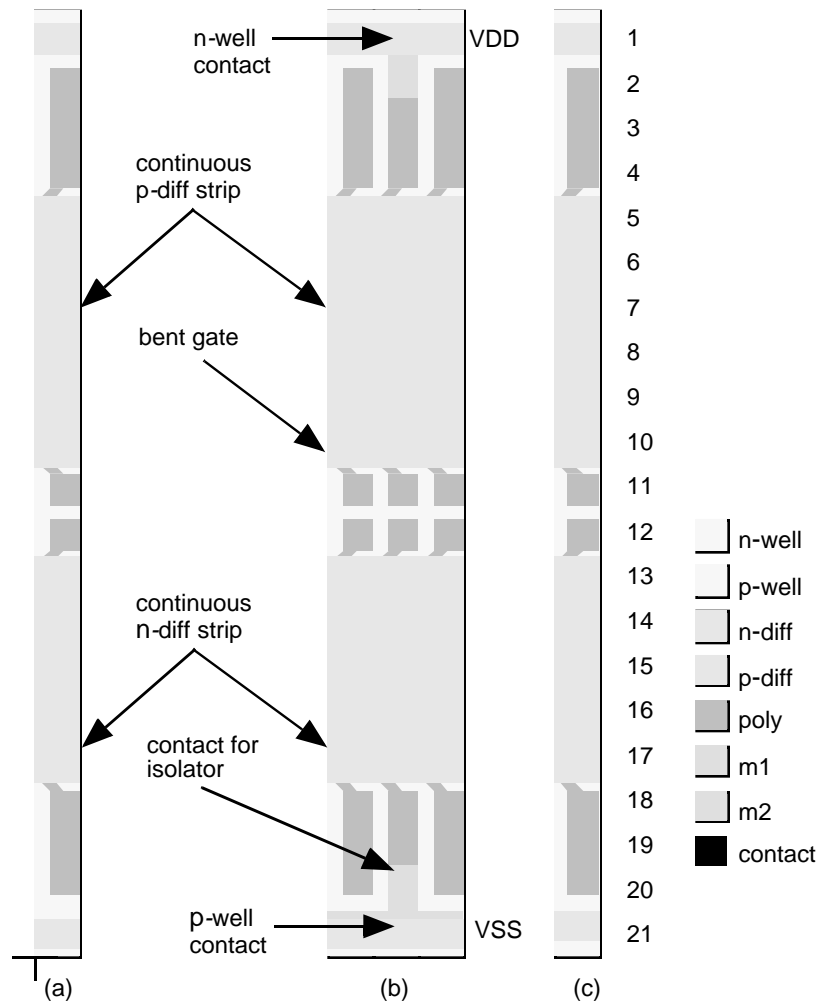
- A big problem in library design is dealing with design rules
- Sometimes we can **waive** design rules
- **Symbolic layout, sticks** or **logs** can decrease the library design time (9 months for Virtual Silicon—currently the most sophisticated standard-cell library)
- Mapping symbolic layout uses 10–20 percent more area (5–10 percent with compaction)
- Allowing 45° layout decreases silicon area (some companies do not allow 45° layout)

3.5 Library Architecture



3.6 Gate-Array Design

Key words: gate-array base cell (or base cell) • gate-array base (or base) • horizontal tracks • vertical track • gate isolation • isolator transistor • oxide isolation • oxide-isolated gate array

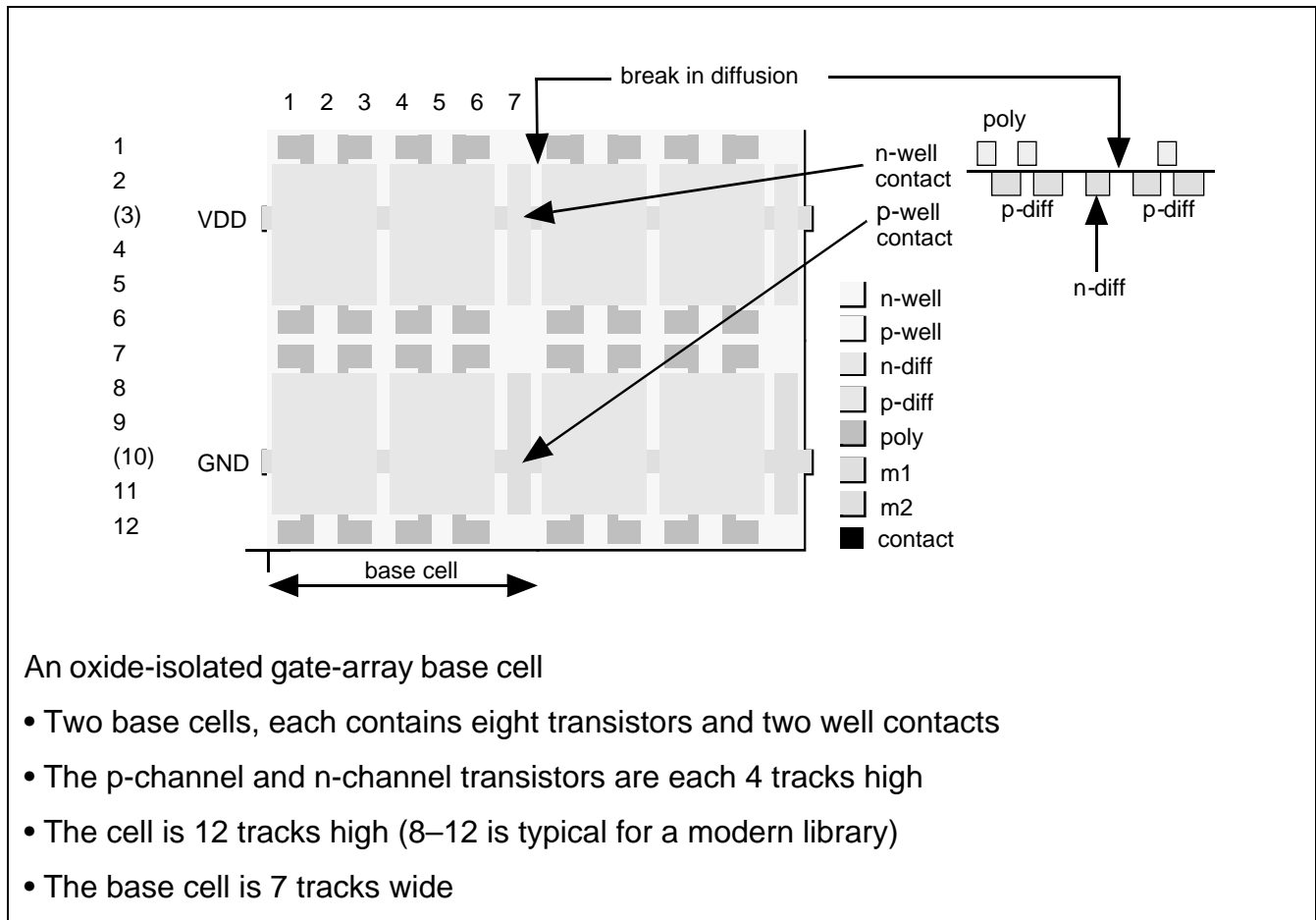


The construction of a gate-isolated gate array

(a) The one-track-wide base cell containing one p-channel and one n-channel transistor

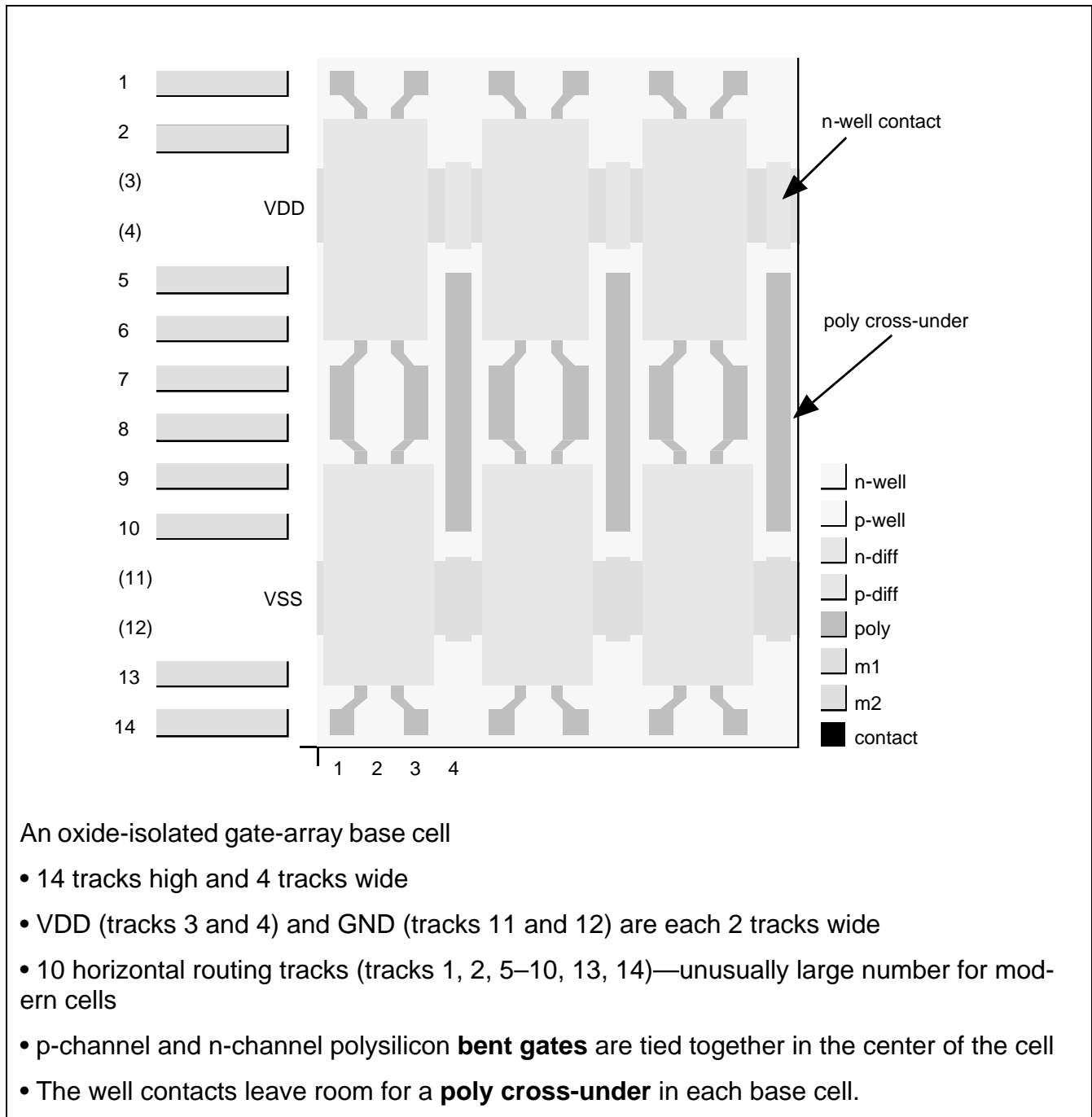
(b) The center base cell is isolating the base cells on either side from each other

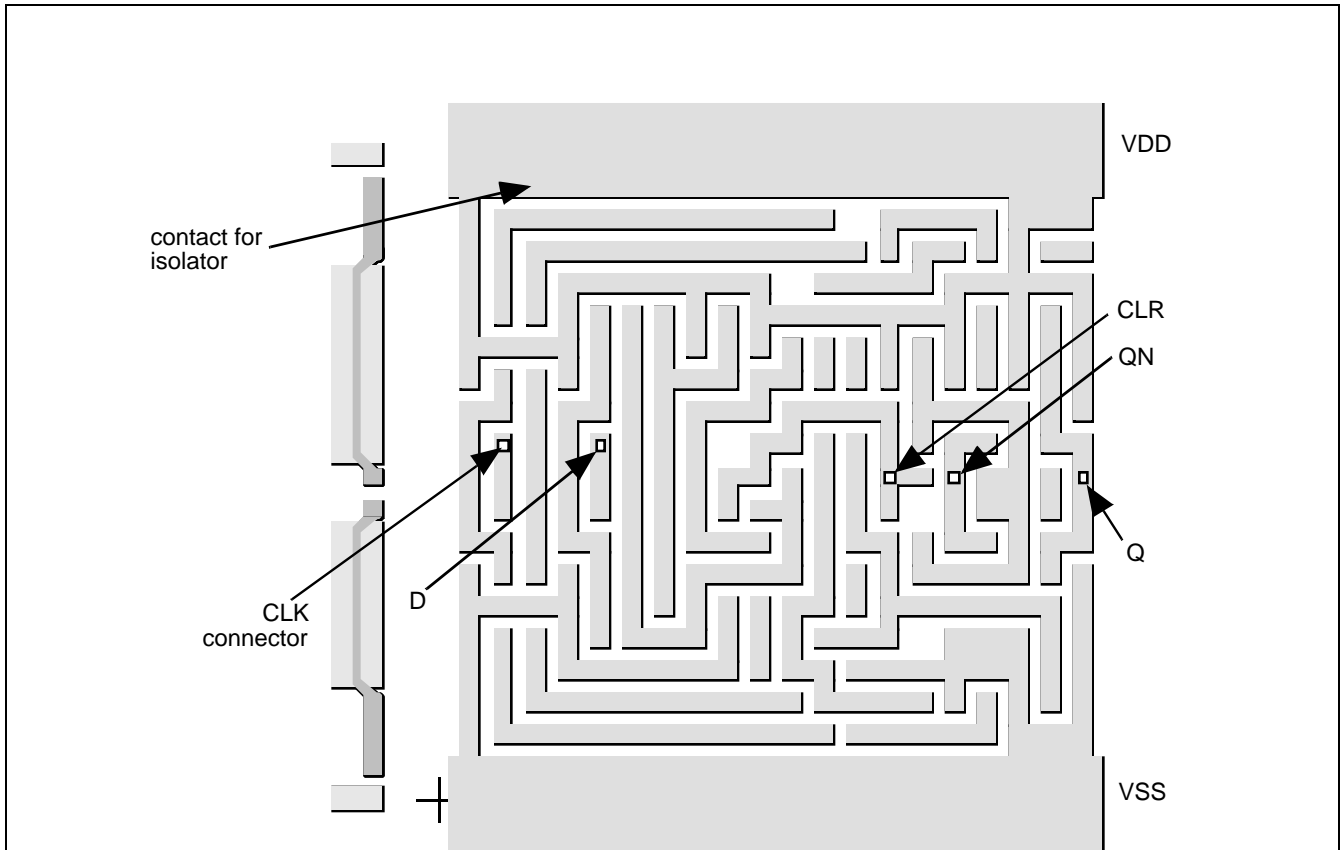
(c) The base cell is 21 tracks high (high for a modern cell library)



An oxide-isolated gate-array base cell

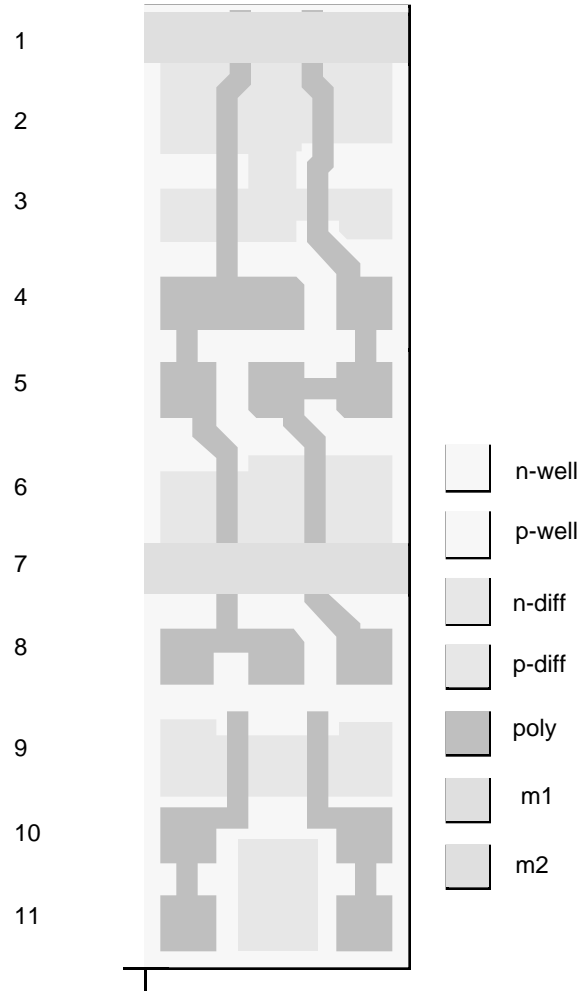
- Two base cells, each contains eight transistors and two well contacts
- The p-channel and n-channel transistors are each 4 tracks high
- The cell is 12 tracks high (8–12 is typical for a modern library)
- The base cell is 7 tracks wide





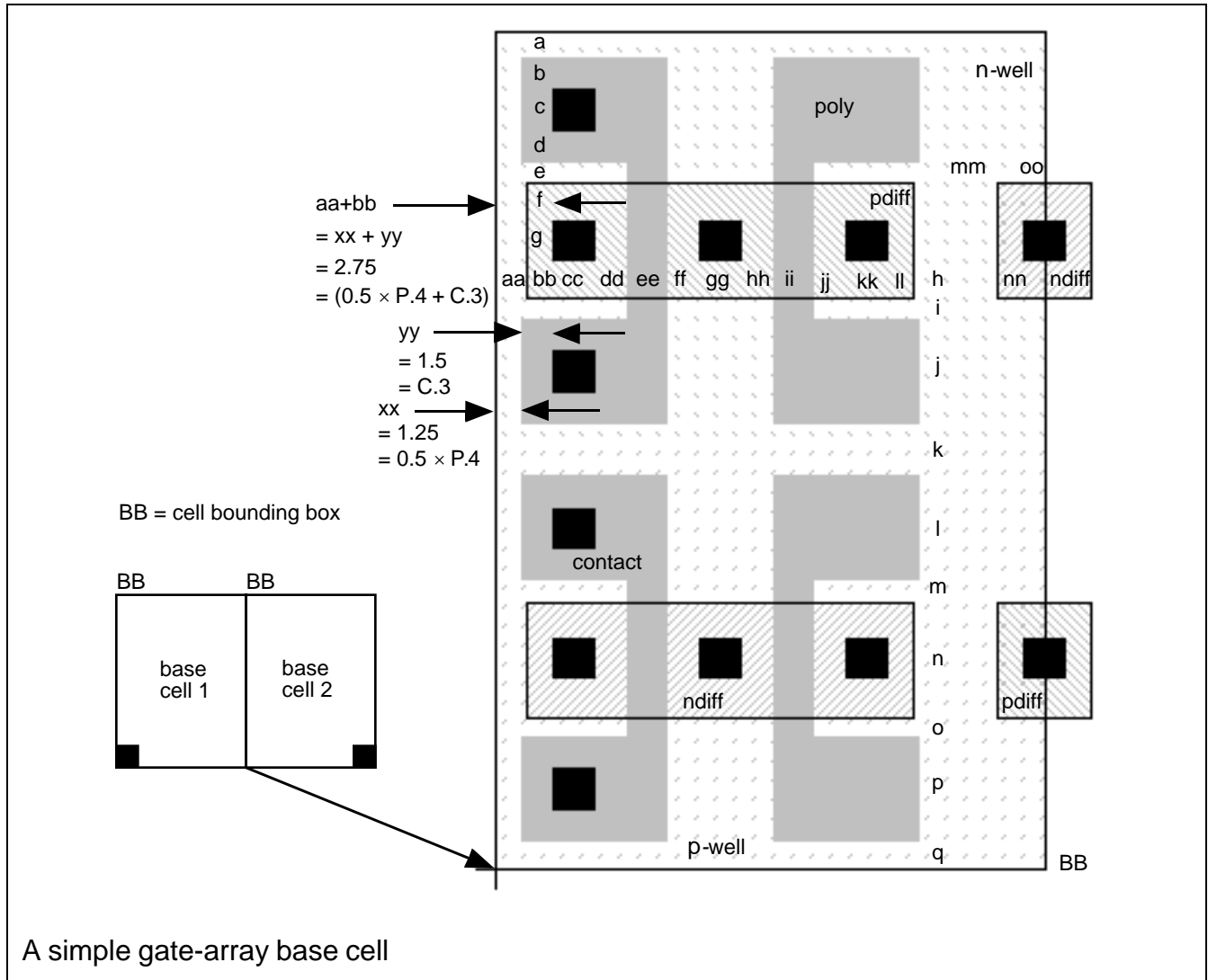
Flip-flop macro in a gate-isolated gate-array library

- Only the first-level metallization and contact pattern, the **personalization**, is shown, but this is enough information to derive the schematic
- This is an older topology for 2LM (cells for 3LM are shorter in height)

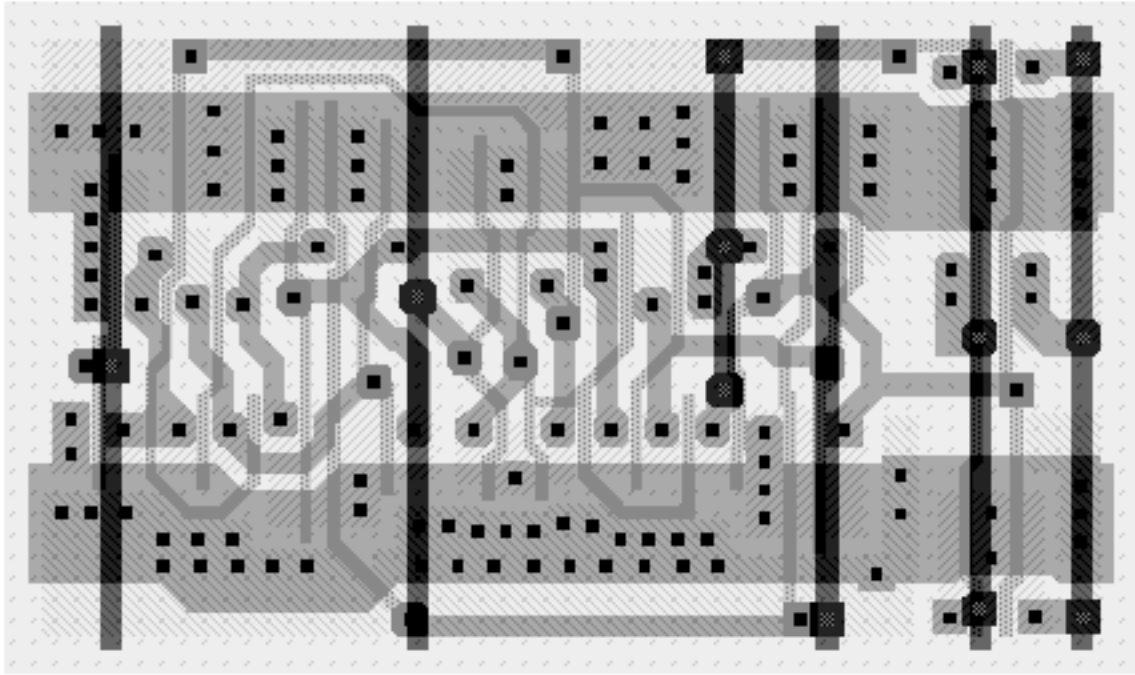


The SiARC/Synopsys cell-based array (CBA) basic cell

- This is CBA I for 2LM (CBA II is intended for 3LM and salicide proceses)

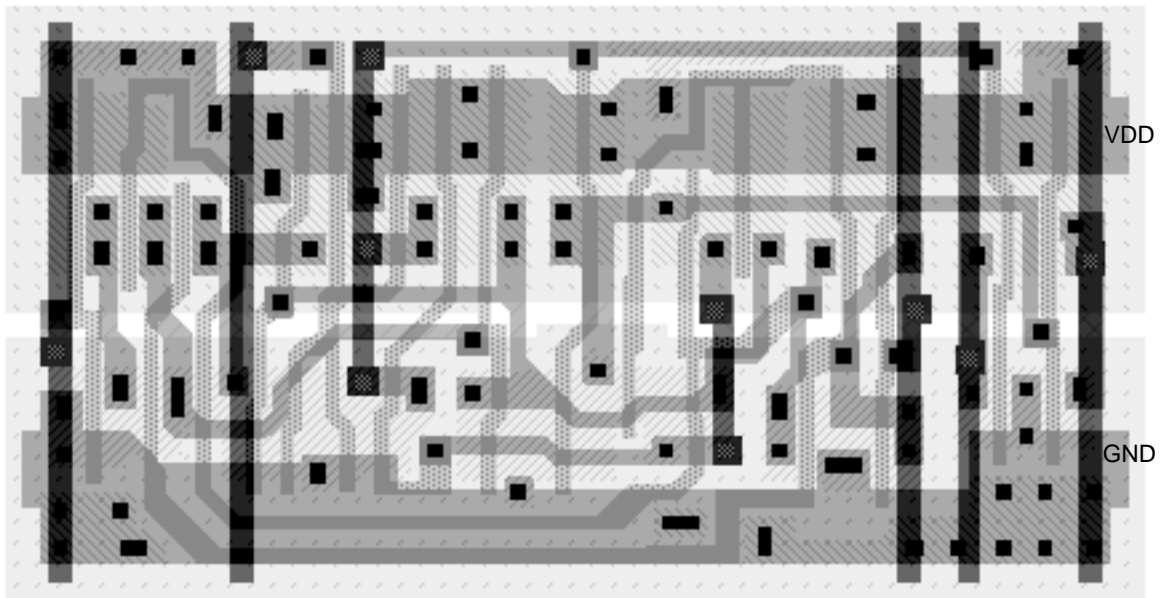


3.7 Standard-Cell Design

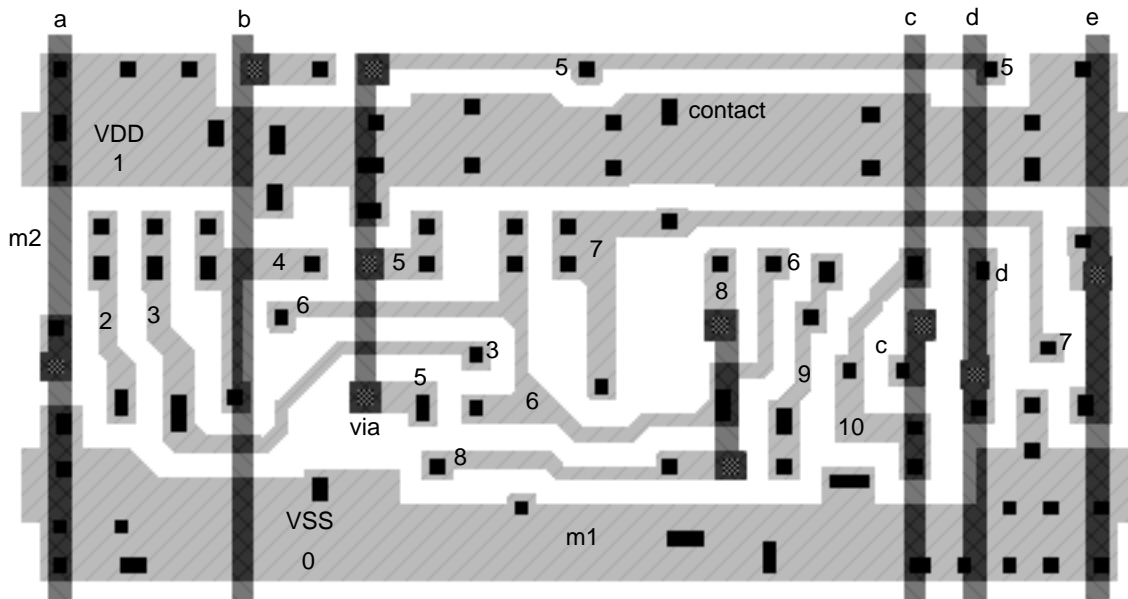
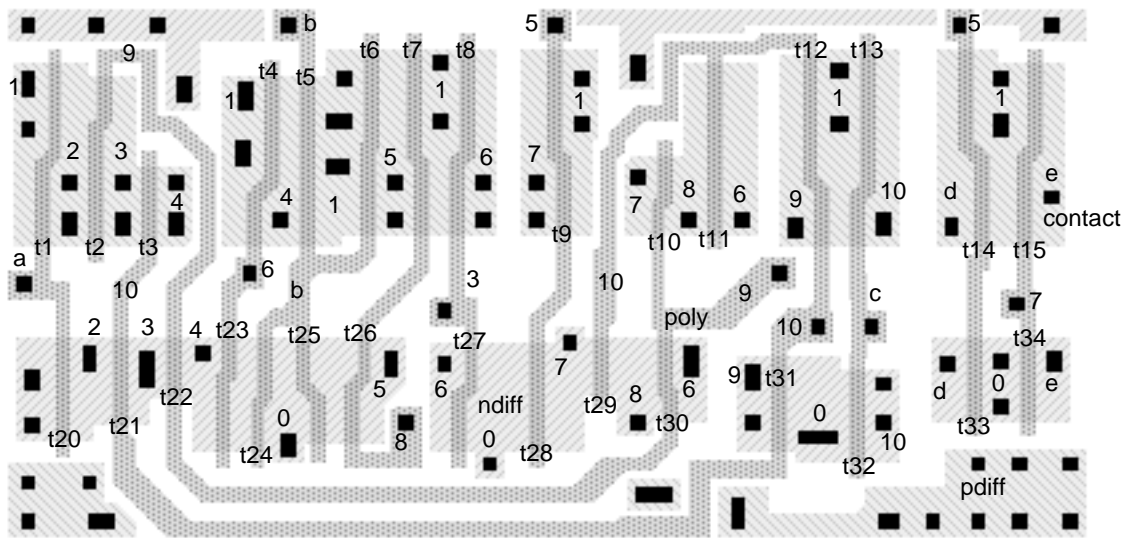


A D flip-flop standard cell

- **Performance-optimized library** • **Area-optimized library**
- Wide **power buses** and transistors for a performance-optimized cell
- **Double-entry cell** intended for a 2LM process and channel routing
- Five **connectors** run vertically through the cell on m2
- The extra short vertical metal line is an internal **crossover**
- **bounding box (BB)** • **abutment box (AB)** • **physical connector** • **abut**



A D flip-flop from a 1.0 μ m standard-cell library

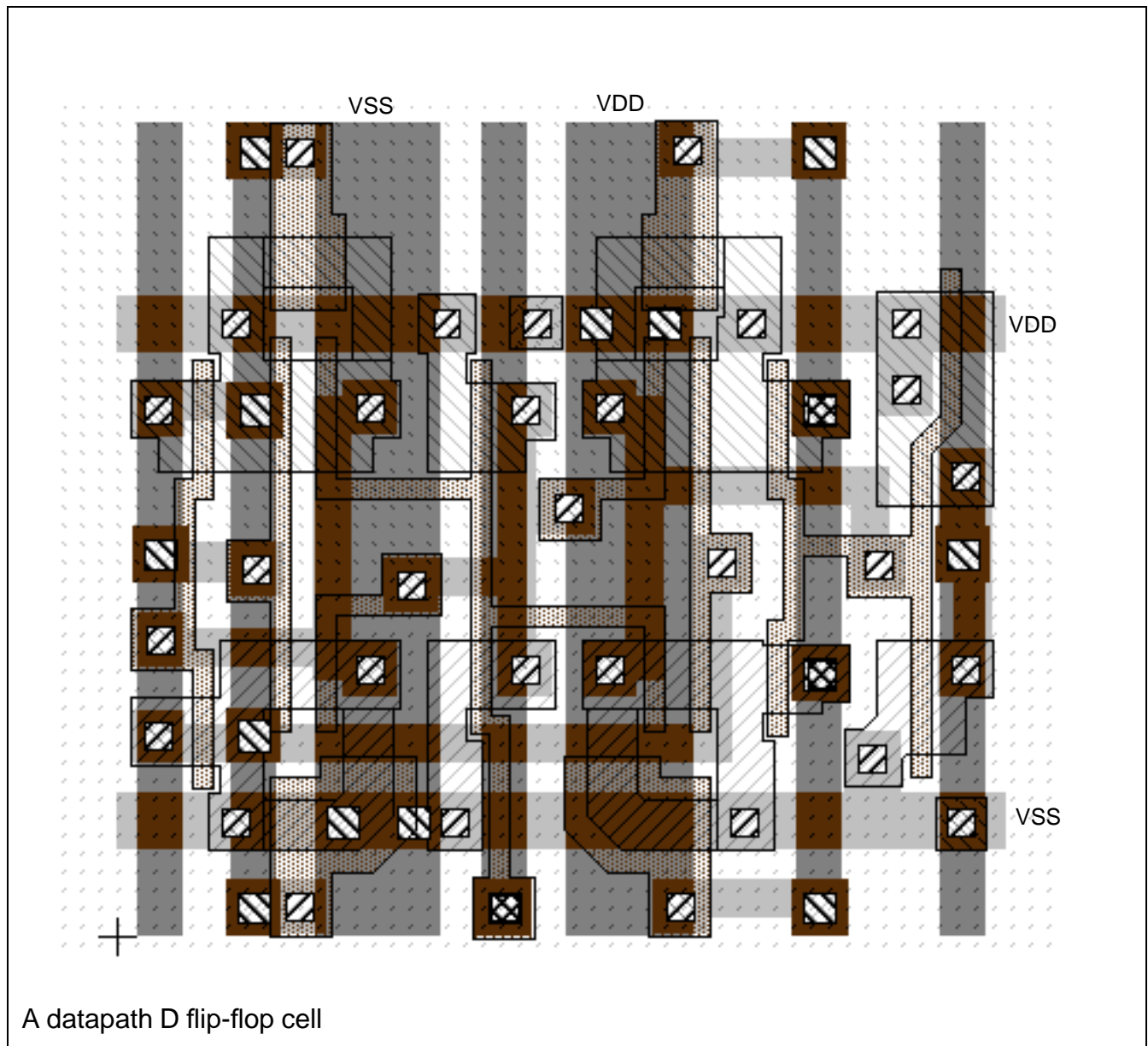


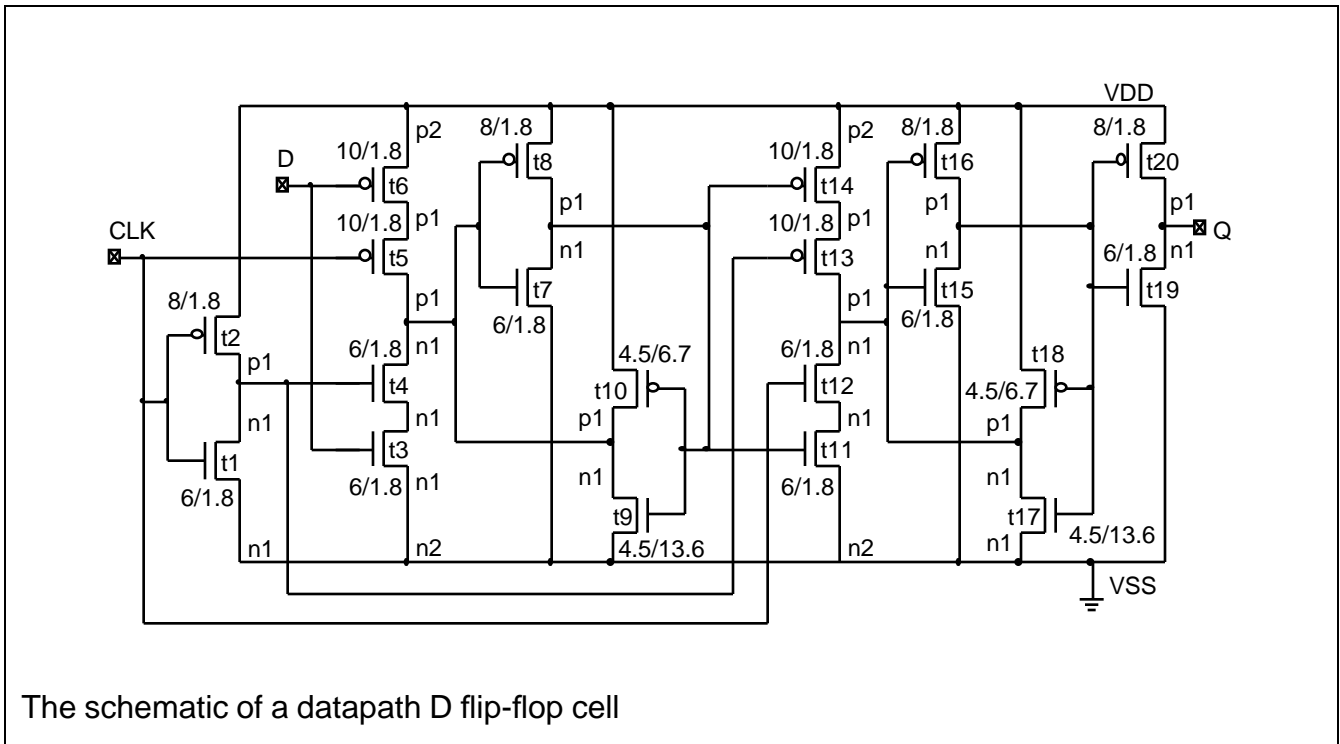
D flip-flop

(Top) n-diffusion, p-diffusion, poly, contact (n-well and p-well are not shown)

(Bottom) m1, contact, m2, and via layers

3.8 Datapath-Cell Design





A narrow datapath

(a) Implemented in a two-level metal process

(b) Implemented in a three-level metal process

3.9 Summary

Key concepts:

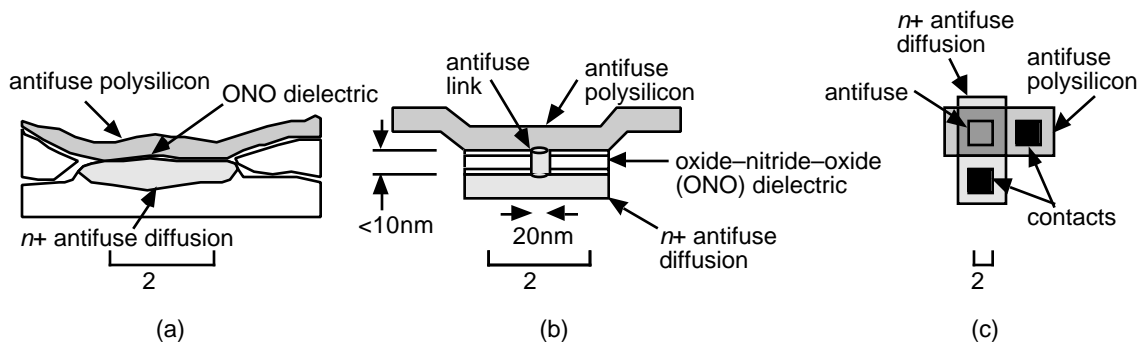
- Tau, logical effort, and the prediction of delay
- Sizes of cells, and their drive strengths
- Cell importance
- The difference between gate-array macros, standard cells, and datapath cells

PROGRAMMABLE ASICs

4

Key concepts: programmable logic devices (PLDs) • field-programmable gate arrays (FPGAs) • programming technology • basic logic cells • I/O logic cells • programmable interconnect • software to design and program the FPGA

4.1 The Antifuse

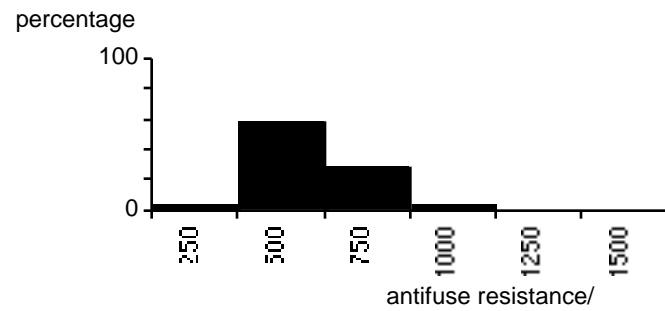


Actel antifuse

antifuse • programming current (about 5mA) • (PLICE') • oxide-nitride-oxide (ONO) dielectric • Activator • in-system programming (ISP) • gang programmers • one-time programmable (OTP) FPGAs

Number of antifuses on Actel FGAs

Device	Antifuses
A1010	112,000
A1020	186,000
A1225	250,000
A1240	400,000
A1280	750,000



The resistance of blown Actel antifuses

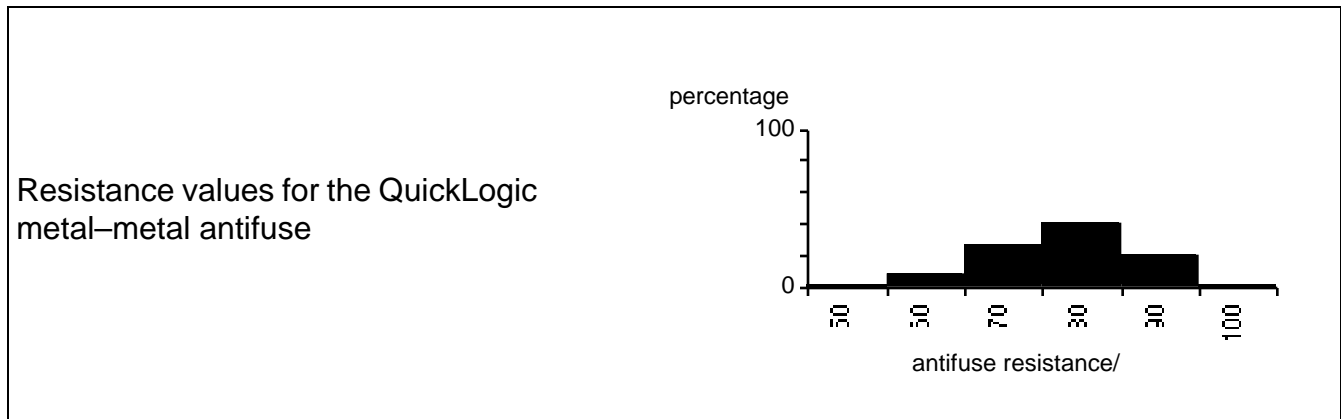
4.1.1 Metal–Metal Antifuse

(a) link, m2, SiO₂, m1, SiO₂, via, amorphous Si, 2

(b) link, m3, tungsten plug, SiO₂, m2, amorphous Si, 2

m2, 4, 4, 2, m3

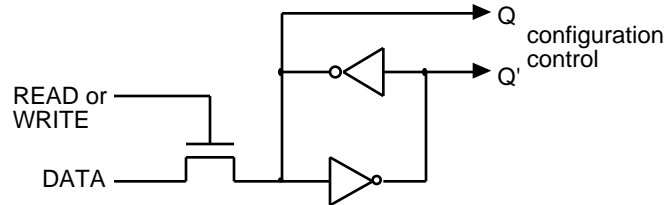
Metal–metal antifuse
 QuickLogic metal–metal antifuse (ViaLink') • alloy of tungsten, titanium, and silicon • bulk resistance of about 500m Ω



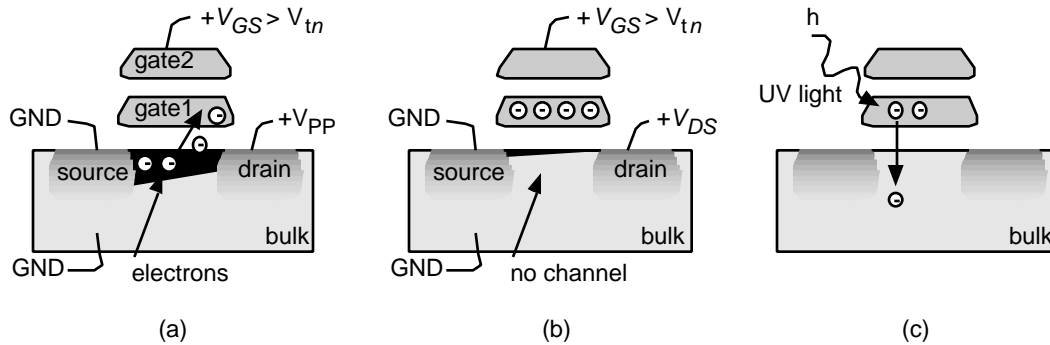
4.2 Static RAM

Xilinx SRAM (static RAM) configuration cell

- use in reconfigurable hardware
- use of programmable read-only memory or PROM to hold configuration



4.3 EPROM and EEPROM Technology



An EPROM transistor

(a) With a high ($>12\text{V}$) programming voltage, V_{PP} , applied to the drain, electrons gain enough energy to “jump” onto the floating gate (gate1)

(b) Electrons stuck on gate1 raise the threshold voltage so that the transistor is always off for normal operating voltages

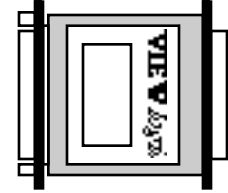
(c) UV light provides enough energy for the electrons stuck on gate1 to “jump” back to the bulk, allowing the transistor to operate normally

Facts and keywords: Altera MAX 5000 EPLDs and Xilinx EPLDs both use UV-erasable electrically programmable read-only memory (EPROM) • hot-electron injection or avalanche injection • floating-gate avalanche MOS (FAMOS)

4.4 Practical Issues

Hardware security key

computer-aided engineering (CAE) tools • PC vs. workstation • ease of use • cost of ownership



4.4.1 FPGAs in Use

- **inventory**
- **risk inventory** or safety supply
- **just-in-time (JIT)**
- **printed-circuit boards (PCBs)**
- **pin locking** or **I/O locking**

4.5 Specifications

- **qualification kit**
- **down-binning**

4.6 PREP Benchmarks

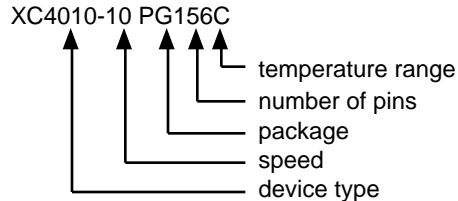
- **Programmable Electronics Performance Company (PREP)**
- <http://www.prep.org>

4.7 FPGA Economics

Xilinx part-naming convention

Not all parts are available in all packages

Some parts are packaged with fewer leads than I/Os



XC4010-10 PG156C

- temperature range
- number of pins
- package
- speed
- device type

Programmable ASIC part codes				
Item	Code	Description	Code	Description
Manufacturer's code	A	Actel	ATT	AT&T (Lucent)
	XC	Xilinx	isp	Lattice Logic
	EPM	Altera MAX	M5	AMD MACH 5 is on the device
	EPF CY7C	Altera FLEX Cypress	QL	QuickLogic
Package type	PL or PC	plastic J-leaded chip carrier, PLCC	VQ	very thin quad flatpack, VQFP
	PQ	plastic quad flatpack, PQFP	TQ	thin plastic flatpack, TQFP
	CQ or CB	ceramic quad flatpack, CQFP	PP	plastic pin-grid array, PPGA
	PG	ceramic pin-grid array, PGA	WB, PB	ball-grid array, BGA
Application	C	commercial	B	MIL-STD-883
	I	industrial	E	extended
	M	military		

1992 base Actel FPGA prices		1992 base Xilinx XC3000 FPGA prices	
Actel part	1H92 base price	Xilinx part	1H92 base price
A1010A-PL44C	\$23.25	XC3020-50PC68C	\$26.00
A1020A-PL44C	\$43.30	XC3030-50PC44C	\$34.20
A1225-PQ100C	\$105.00	XC3042-50PC84C	\$52.00
A1240-PQ144C	\$175.00	XC3064-50PC84C	\$87.00
A1280-PQ160C	\$305.00	XC3090-50PC84C	\$133.30

4.7.1 FPGA Pricing

“How much do FPGAs cost?” • “How much does a car cost?” • pricing matrix

Actel price adjustment factors					
Purchase quantity, all types					
(1–9)	(10–99)	(100–999)			
100%	96%	84%			
Purchase time, in (100–999) quantity					
1H92	2H92	93			
100%	80–95%	60–80%			
Qualification type, same package					
Commercial	Industrial	Military	883-B		
100%	120%	150%	230–300%		
Speed bin¹					
ACT 1-Std	ACT 1-1	ACT 1-2	ACT 2-Std	ACT 2-1	
100%	115%	140%	100%	120%	
Package type					
A1010:	PL44, 64, 84	PQ100	PG84		
	100%	125%	400%		
A1020:	PL44, 64, 84	PQ100	JQ44, 68, 84	PG84	CQ84
	100%	125%	270%	275%	400%
A1225:	PQ100	PG100			
	100%	175%			
A1240:	PQ144	PG132			
	100%	140%			
A1280:	PQ160	PG176	CQ172		
	100%	145%	160%		
¹ Actel bins: Std=standard speed grade; 1=medium speed grade; 2=fastest speed grade					

4.7.2 Pricing Examples

base prices and adjustment factors • “sticker price”

Example Actel part-price calculation		
Example: A1020A-2-PQ100 in (100–999) quantity, purchased 1H92.		
Factor	Example	Value
Base price	A1020A	\$43.30
Quantity	100–999	84%
Time	1H92	100%
Qualification type	Industrial (I)	120%
Speed bin ¹	2	140%
Package	PQ100	125%
Estimated price (1H92)		\$76.38
Actual Actel price (1H92)		\$75.60
¹ The speed bin is a manufacturer’s code (usually a number) that follows the family part number and indicates the maximum operating speed of the device		

- Marshall at <http://marshall.com>, carry Xilinx
- Hamilton-Avnet, at <http://www.hh.avnet.com>, carry Xilinx
- Wyle, at <http://www.wyle.com> carries Actel and Altera

4.8 Summary

Programmable ASIC technologies				
	Actel	Xilinx LCA¹	Altera EPLD	Xilinx EPLD
Programming technology	Poly-diffusion antifuse, PLICE	Erasable SRAM ISP	UV-erasable EPROM (MAX 5k) EEPROM (MAX 7/9k)	UV-erasable EPROM
Size of programming element	Small but requires contacts to metal	Two inverters plus pass and switch devices. Largest.	One n-channel EPROM device. Medium.	One n-channel EPROM device. Medium.
Process	Special: CMOS plus three extra masks.	Standard CMOS	Standard EPROM and EEPROM	Standard EPROM
Programming method	Special hardware	PC card, PROM, or serial port	ISP (MAX 9k) or EPROM programmer	EPROM programmer
	QuickLogic	Crosspoint	Atmel	Altera FLEX
Programming technology	Metal-metal antifuse, ViaLink	Metal-polysilicon antifuse	Erasable SRAM. ISP.	Erasable SRAM. ISP.
Size of programming element	Smallest	Small	Two inverters plus pass and switch devices. Largest.	Two inverters plus pass and switch devices. Largest.
Process	Special, CMOS plus ViaLink	Special, CMOS plus antifuse	Standard CMOS	Standard CMOS
Programming method	Special hardware	Special hardware	PC card, PROM, or serial port	PC card, PROM, or serial port
¹ Lucent (formerly AT&T) FPGAs have almost identical properties to the Xilinx LCA family				

All FPGAs have the following key elements:

- The programming technology
- The basic logic cells
- The I/O logic cells
- Programmable interconnect
- Software to design and program the FPGA

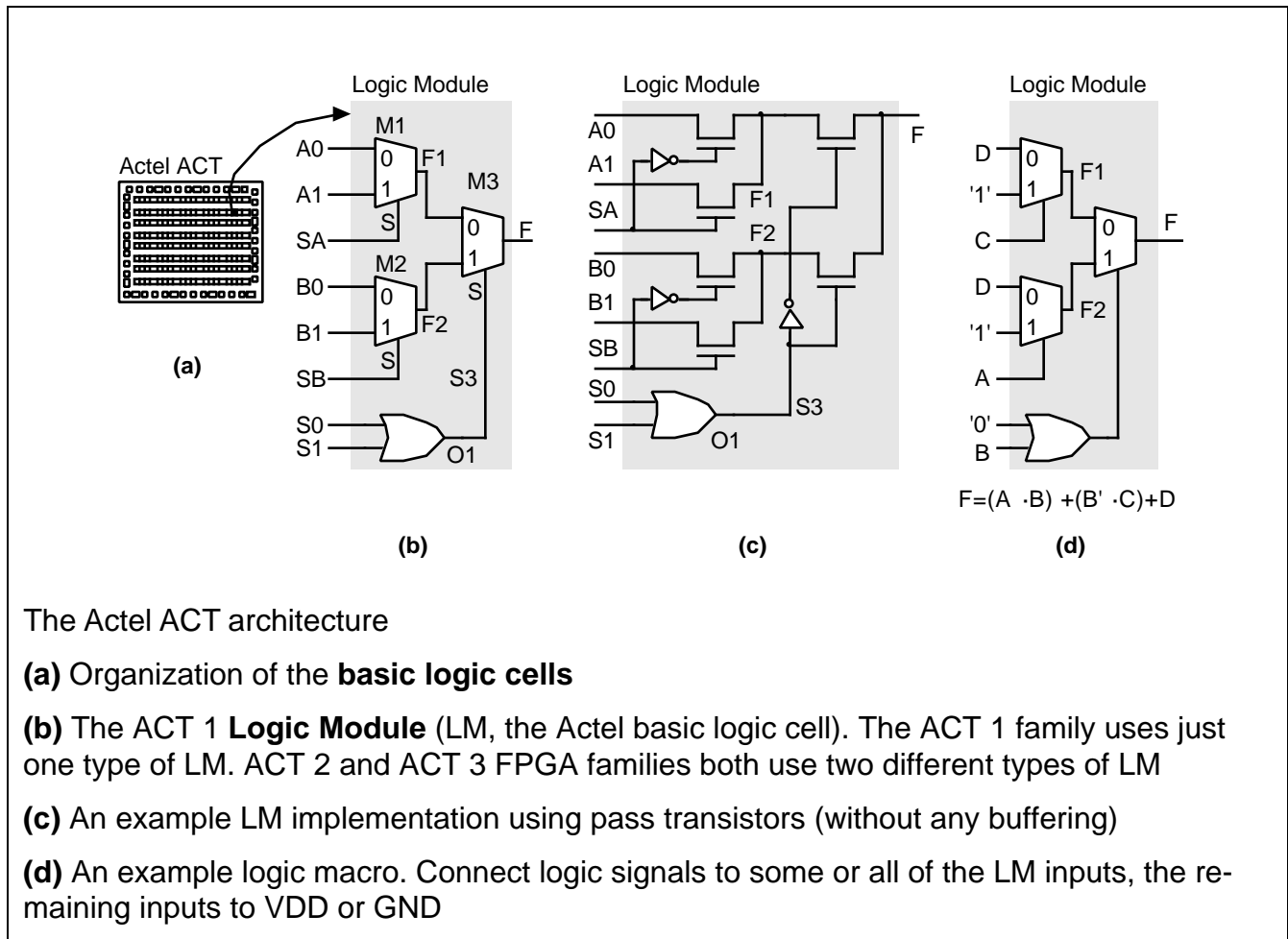
4.9 Problems

PROGRAMMABLE ASIC LOGIC CELLS

Key concepts: basic logic cell • multiplexer-based cell • look-up table (LUT) • programmable array logic (PAL) • influence of programming technology • timing • worst-case design

5.1 Actel ACT

5.1.1 ACT 1 Logic Module



The Actel ACT architecture

(a) Organization of the basic logic cells

(b) The ACT 1 Logic Module (LM, the Actel basic logic cell). The ACT 1 family uses just one type of LM. ACT 2 and ACT 3 FPGA families both use two different types of LM

(c) An example LM implementation using pass transistors (without any buffering)

(d) An example logic macro. Connect logic signals to some or all of the LM inputs, the remaining inputs to VDD or GND

5.1.2 Shannon's Expansion Theorem

- We can use the **Shannon expansion theorem** to **expand** $F = A \cdot F(A='1') + A' \cdot F(A='0')$

Example: $F = A' \cdot B + A \cdot B \cdot C' + A' \cdot B' \cdot C = A \cdot (B \cdot C') + A' \cdot (B + B' \cdot C)$

- $F(A='1') = B \cdot C'$ is the **cofactor** of F with respect to (**wrt**) A or F_A
- If we expand F *wrt* B , $F = A' \cdot B + A \cdot B \cdot C' + A' \cdot B' \cdot C = B \cdot (A' + A \cdot C') + B' \cdot (A' \cdot C)$
- Eventually we reach the unique **canonical form**, which uses only minterms
- (A **minterm** is a **product term** that contains all the variables of F —such as $A \cdot B' \cdot C$)

Another example: $F = (A \cdot B) + (B' \cdot C) + D$

- Expand F *wrt* B : $F = B \cdot (A + D) + B' \cdot (C + D) = B \cdot F_2 + B' \cdot F_1$
- $F = 2:1$ MUX, with B selecting between two inputs: $F(A='1')$ and $F(A='0')$
- F also describes the output of the ACT 1 LM
- Now we need to split up F_1 and F_2
- Expand F_2 *wrt* A , and F_1 *wrt* C : $F_2 = A + D = (A \cdot 1) + (A' \cdot D)$; $F_1 = C + D = (C \cdot 1) + (C' \cdot D)$
- A, B, C connect to the select lines and '1' and D are the inputs of the MUXes in the ACT 1 LM
- Connections: $A_0 = D, A_1 = '1', B_0 = D, B_1 = '1', S_A = C, S_B = A, S_0 = '0',$ and $S_1 = B$

5.1.3 Multiplexer Logic as Function Generators

The 16 logic functions of 2 variables:

- 2 of the 16 functions are not very interesting ($F='0'$, and $F='1'$)
- There are 10 functions that we can implement using just one 2:1 MUX
- 6 functions are useful: INV, BUF, AND, OR, AND1-1, NOR1-1

4 ways to arrange one '1'

6 ways to arrange two '1's

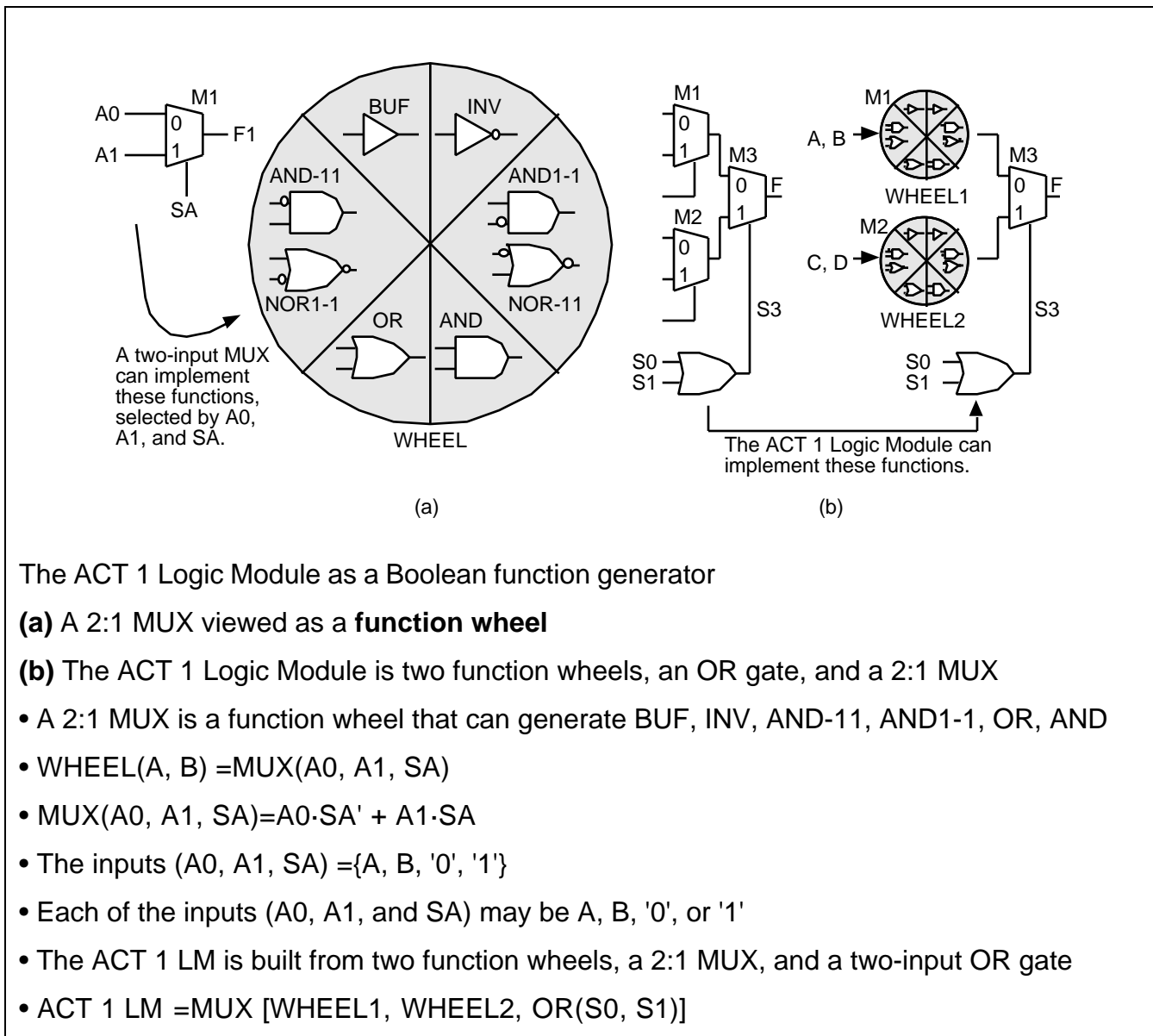
4 ways to arrange one '0'

14 functions of 2 variables (and $F='0'$, $F='1'$ makes 16)

Boolean functions using a 2:1 MUX								
Function, F	F=	Canonical form	Min-terms	Min-term code	Function number	M1		
						A0	A1	SA
1 '0'	'0'	'0'	none	0000	0	0	0	0
2 NOR1-1(A, B)	$(A+B)'$	$A' \cdot B$	1	0010	2	B	0	A
3 NOT(A)	A'	$A' \cdot B' + A' \cdot B$	0, 1	0011	3	0	1	A
4 AND1-1(A, B)	$A \cdot B'$	$A \cdot B'$	2	0100	4	A	0	B
5 NOT(B)	B'	$A' \cdot B' + A \cdot B'$	0, 2	0101	5	0	1	B
6 BUF(B)	B	$A' \cdot B + A \cdot B$	1, 3	1010	6	0	B	1
7 AND(A, B)	$A \cdot B$	$A \cdot B$	3	1000	8	0	B	A
8 BUF(A)	A	$A \cdot B' + A \cdot B$	2, 3	1100	9	0	A	1
9 OR(A, B)	$A+B$	$A' \cdot B + A \cdot B' + A \cdot B$	1, 2, 3	1110	13	B	1	A
10 '1'	'1'	$A' \cdot B' + A' \cdot B + A \cdot B' + A \cdot B$	0, 1, 2, 3	1111	15	1	1	1

Example of using the WHEEL functions to implement $F=\text{NAND}(A, B)=(A \cdot B)'$

1. First express F as the output of a 2:1 MUX: we do this by expanding F wrt A (or wrt B; since F is symmetric) $F=A \cdot (B') + A' \cdot ('1')$
2. Assign WHEEL1 to implement INV(B), and WHEEL2 to implement '1'
3. Set the select input to the MUX connecting WHEEL1 and WHEEL2, $S_0+S_1=A$. We can do this using $S_0=A$, $S_1='1'$



5.1.4 ACT 2 and ACT 3 Logic Modules

- ACT 1 requires 2 LMs per flip-flop: with unknown interconnect capacitance
- ACT 2 and ACT 3 use two types of LMs, one includes a D flip-flop
- ACT 2 **C-Module** is similar to the ACT 1 LM but can implement five-input logic functions
- *combinatorial* module implements *combinational* logic (blame MMI for the misuse of terms)
- ACT 2 **S-Module (sequential module)** contains a C-Module and a **sequential element**

5.1.5 Timing Model and Critical Path

Keywords and concepts: timing model • deals only with internal logic • estimates delays • before place-and-route step • nondeterministic architecture • find slowest register–register delay or critical path

Example of timing calculations (a rather complex examination of internal module timing):

- The setup and hold times, measured *inside* (not outside) the S-Module, are t'_{SUD} and t'_H (a prime denotes parameters that are measured inside the S-Module)
- The clock–Q propagation delay is t'_{CO}
- The parameters t'_{SUD} , t'_H , and t'_{CO} are measured using the *internal* clock signal CLK_i
- The propagation delay of the combinational logic *inside* the S-Module is t'_{PD}
- The delay of the combinational logic that drives the flip-flop clock signal is t'_{CLKD}
- From *outside* the S-Module, with reference to the outside clock signal CLK₁:

$$t_{SUD} = t'_{SUD} + (t'_{PD} - t'_{CLKD}), \quad t_H = t'_H + (t'_{PD} - t'_{CLKD}), \quad t_{CO} = t'_{CO} + t'_{CLKD}$$

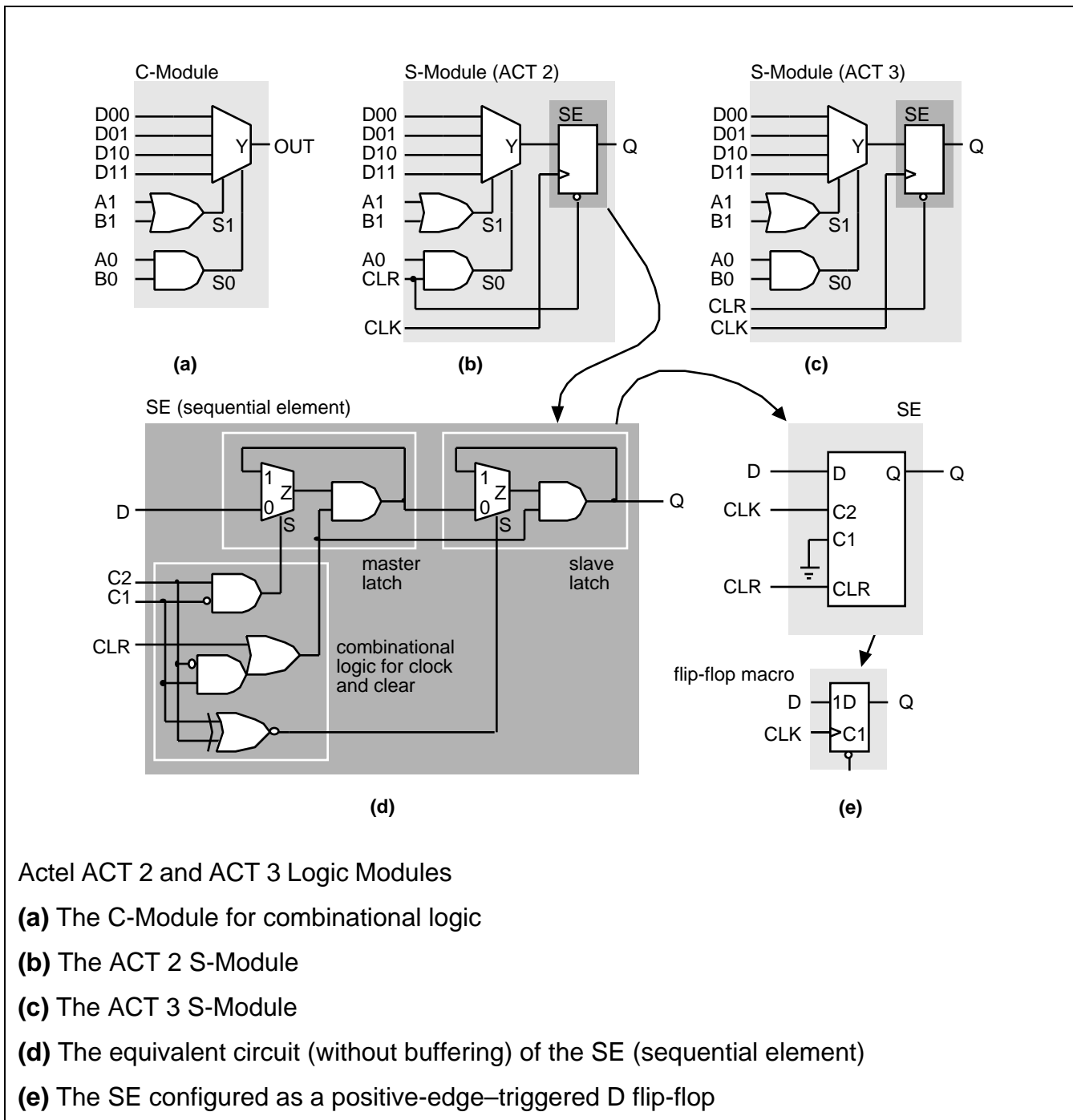
- We do not know the *internal* parameters t'_{SUD} , t'_H , and t'_{CO} , but assume reasonable values:

$$t'_{SUD} = 0.4\text{ns}, \quad t'_H = 0.1\text{ns}, \quad t'_{CO} = 0.4\text{ns}.$$

- t'_{PD} (combinational logic inside the S-Module) is equal to the C-Module delay, so $t'_{PD} = 3\text{ns}$ for the ACT 3
- We do not know t'_{CLKD} ; assume a value of $t'_{CLKD} = 2.6\text{ns}$ (the exact value does not matter)
- Thus the *external* S-Module parameters are: $t_{SUD} = 0.8\text{ns}$, $t_H = 0.5\text{ns}$, $t_{CO} = 3.0\text{ns}$
- These are the same as the ACT 3 S-Module parameters (I chose t'_{CLKD} so they would be)
- Of the 3.0ns combinational logic delay: 0.4ns increases the setup time and 2.6ns increases the clock–output delay, t_{CO}
- Actel says that the combinational logic delay is *buried* in the flip-flop setup time. But this is borrowed money—you have to pay it back.

5.1.6 Speed Grading

- **Speed grading** (or **speed binning**) uses a **binning circuit**
- Measure $t_{PD} = (t_{PLH} + t_{PHL})/2$ — and use the fact that properties match across a chip
- Actel speed grades are based on 'Std' speed grade



Actel ACT 2 and ACT 3 Logic Modules

(a) The C-Module for combinational logic

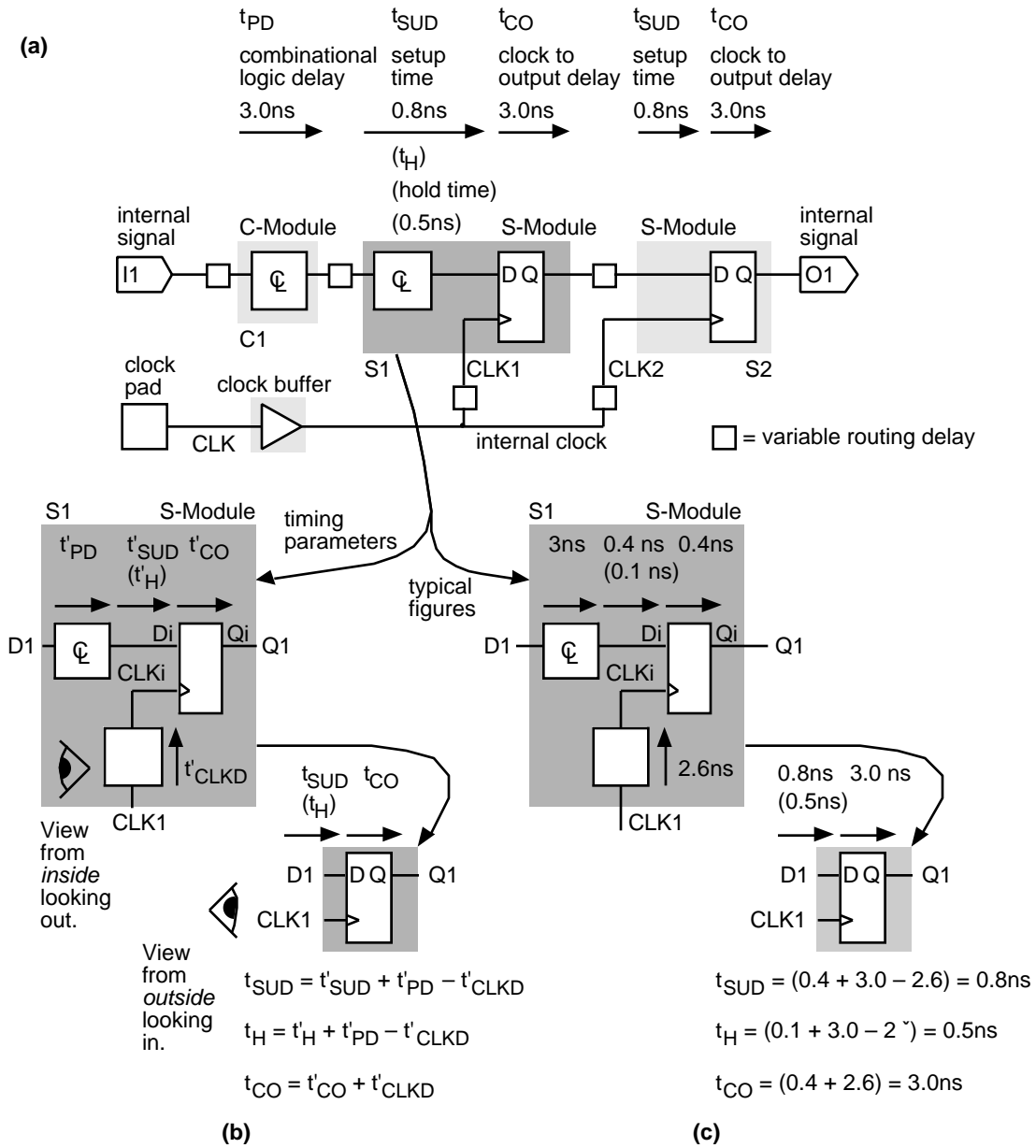
(b) The ACT 2 S-Module

(c) The ACT 3 S-Module

(d) The equivalent circuit (without buffering) of the SE (sequential element)

(e) The SE configured as a positive-edge-triggered D flip-flop

- '1' speed grade is approximately 15 percent faster than 'Std'
- '2' speed grade is approximately 25 percent faster than 'Std'
- '3' speed grade is approximately 35 percent faster than 'Std'.



Timing views from inside and outside the Actel ACT S-module

(a) Timing parameters for a 'Std' speed grade ACT 3

(b) Flip-flop timing

(c) An example of flip-flop timing based on ACT 3 parameters

5.1.7 Worst-Case Timing

Keywords and concepts: Using synchronous design you worry about how slow your circuit may be—not how fast • **ambient temperature**, T_A • package **case temperature**, T_C (military) • temperature of the chip, the **junction temperature**, T_J • nominal operating conditions: $V_{DD}=5.0V$, and $T_J=25^\circ C$ • **worst-case commercial** conditions: $V_{DD}=4.75V$, and $T_J=+70^\circ C$ • always design using **worst-case timing** • **derating factors** • **critical path delay** between registers • **process corner** (slow–slow • fast–fast • slow–fast • fast–slow) • Commercial. $V_{DD}=5V \pm 5\%$, T_A (ambient)=0 to $+70^\circ C$ • Industrial. $V_{DD}=5V \pm 10\%$, T_A (ambient)=-40 to $+85^\circ C$ • Military: $V_{DD}=5V \pm 10\%$, T_C (case)=-55 to $+125^\circ C$ • Military: Standard MIL-STD-883C Class B • Military extended: unmanned spacecraft

ACT 3 timing parameters

Family	Delay	Fanout				
		1	2	3	4	8
ACT 3-3 (data book)	t_{PD}	2.9	3.2	3.4	3.7	4.8
ACT3-2 (calculated)	$t_{PD}/0.85$	3.41	3.76	4.00	4.35	5.65
ACT3-1 (calculated)	$t_{PD}/0.75$	3.87	4.27	4.53	4.93	6.40
ACT3-Std (calculated)	$t_{PD}/0.65$	4.46	4.92	5.23	5.69	7.38

ACT 3 derating factors

V_{DD}/V	Temperature T_J (junction)/ $^\circ C$						
	-55	-40	0	25	70	85	125
4.5	0.72	0.76	0.85	0.90	1.04	1.07	1.17
4.75	0.70	0.73	0.82	0.87	1.00	1.03	1.12
5.00	0.68	0.71	0.79	0.84	0.97	1.00	1.09
5.25	0.66	0.69	0.77	0.82	0.94	0.97	1.06
5.5	0.63	0.66	0.74	0.79	0.90	0.93	1.01

5.1.8 Actel Logic Module Analysis

- Actel uses a **fine-grain architecture** which allows you to use almost all of the FPGA
- Synthesis can map logic efficiently to a fine-grain architecture

- Physical symmetry simplifies place-and-route (swapping equivalent pins on opposite sides of the LM to ease routing)
- Matched to small antifuse programming technology
- LMs balance efficiency of implementation and efficiency of utilization
- A simple LM reduces performance, but allows fast and robust place-and-route

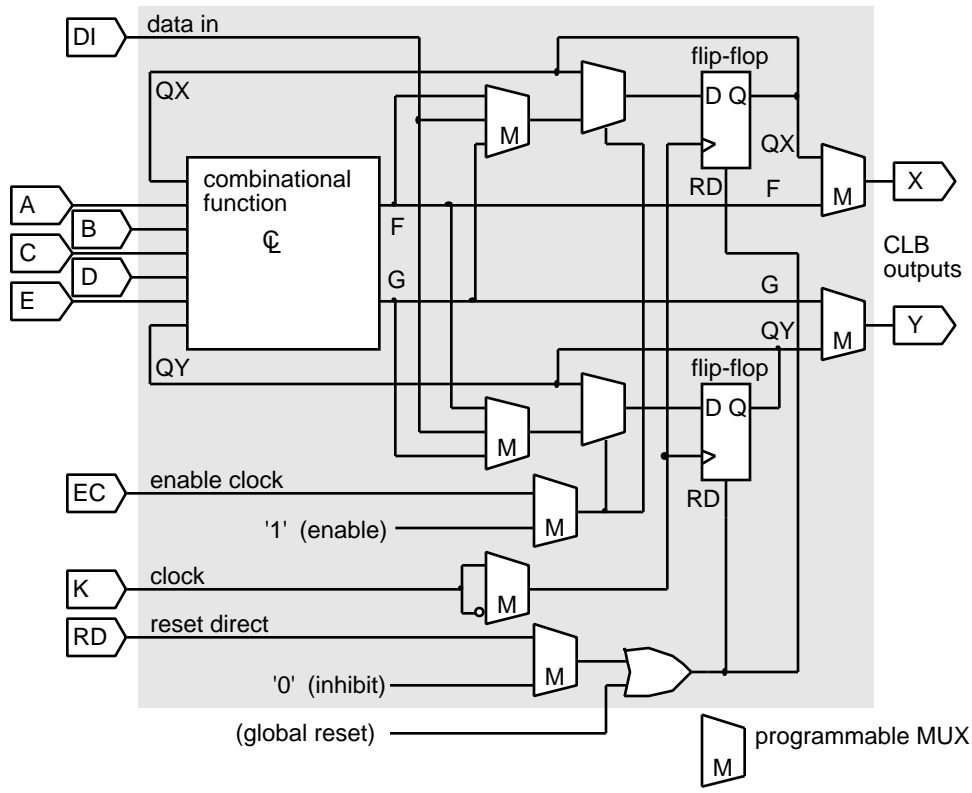
5.2 Xilinx LCA

Keywords and concepts: Xilinx LCA (a trademark, logic cell array) • **configurable logic block**
• **coarse-grain architecture**

5.2.1 XC3000 CLB

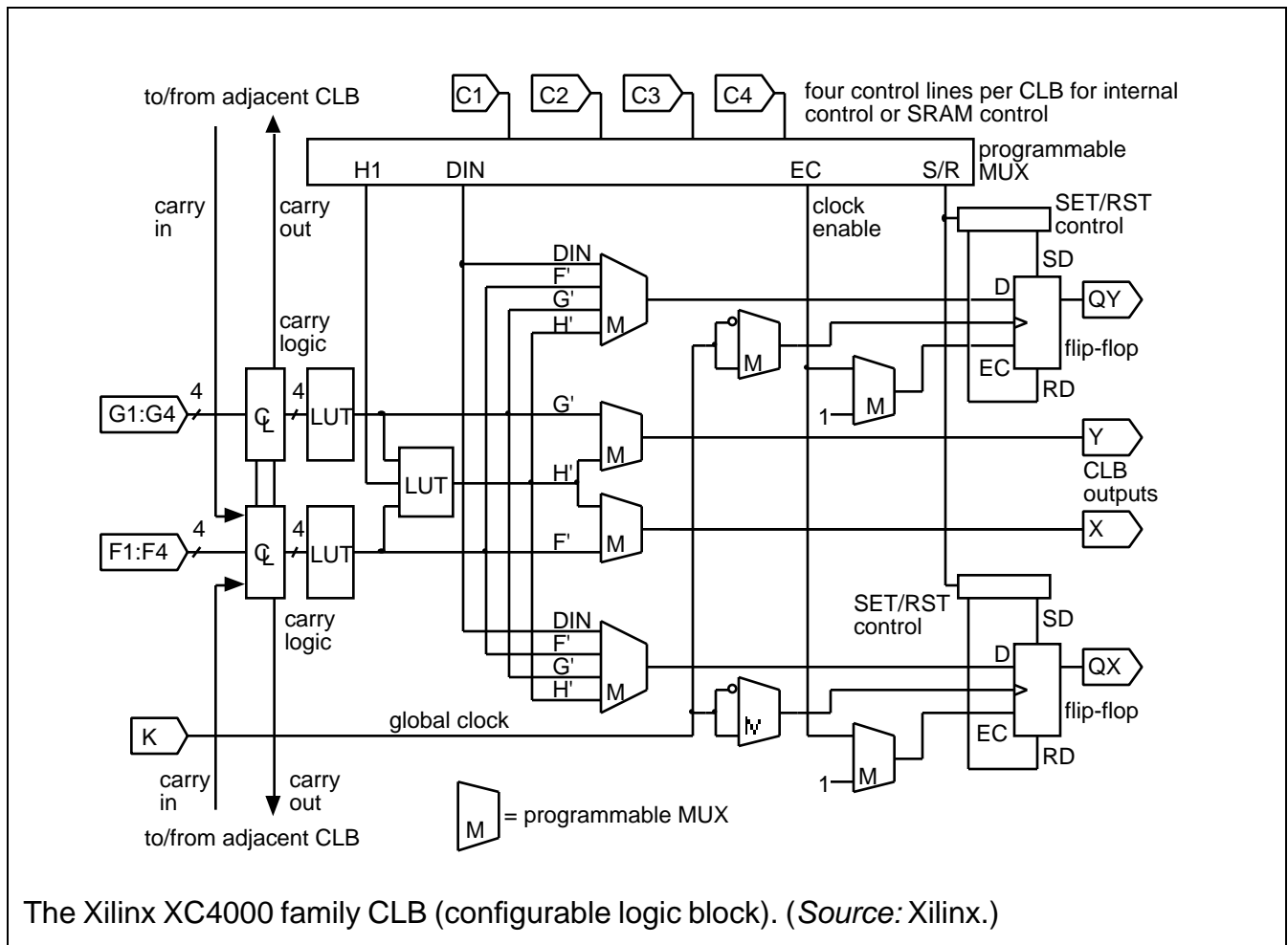
- A 32-bit **look-up table** (LUT)
- CLB propagation delay is fixed (the LUT access time) and independent of the logic function
- 7 inputs to the XC3000 CLB: 5 CLB inputs (A–E), and 2 flip-flop outputs (QX and QY)
- 2 outputs from the LUT (F and G). Since a 32-bit LUT requires only five variables to form a unique address ($32=2^5$), there are several ways to use the LUT:
- Use 5 of the 7 possible inputs (A–E, QX, QY) with the entire 32-bit LUT (the CLB outputs (F and G) are then identical)
- Split the 32-bit LUT in half to implement 2 functions of 4 variables each; choose 4 input variables from the 7 inputs (A–E, QX, QY). You have to choose 2 of the inputs from the 5 CLB inputs (A–E); then one function output connects to F and the other output connects to G.
- You can split the 32-bit LUT in half, using one of the 7 input variables as a select input to a 2:1 MUX that switches between F and G (to implement some functions of 6 and 7 variables).

5.2.2 XC4000 Logic Block

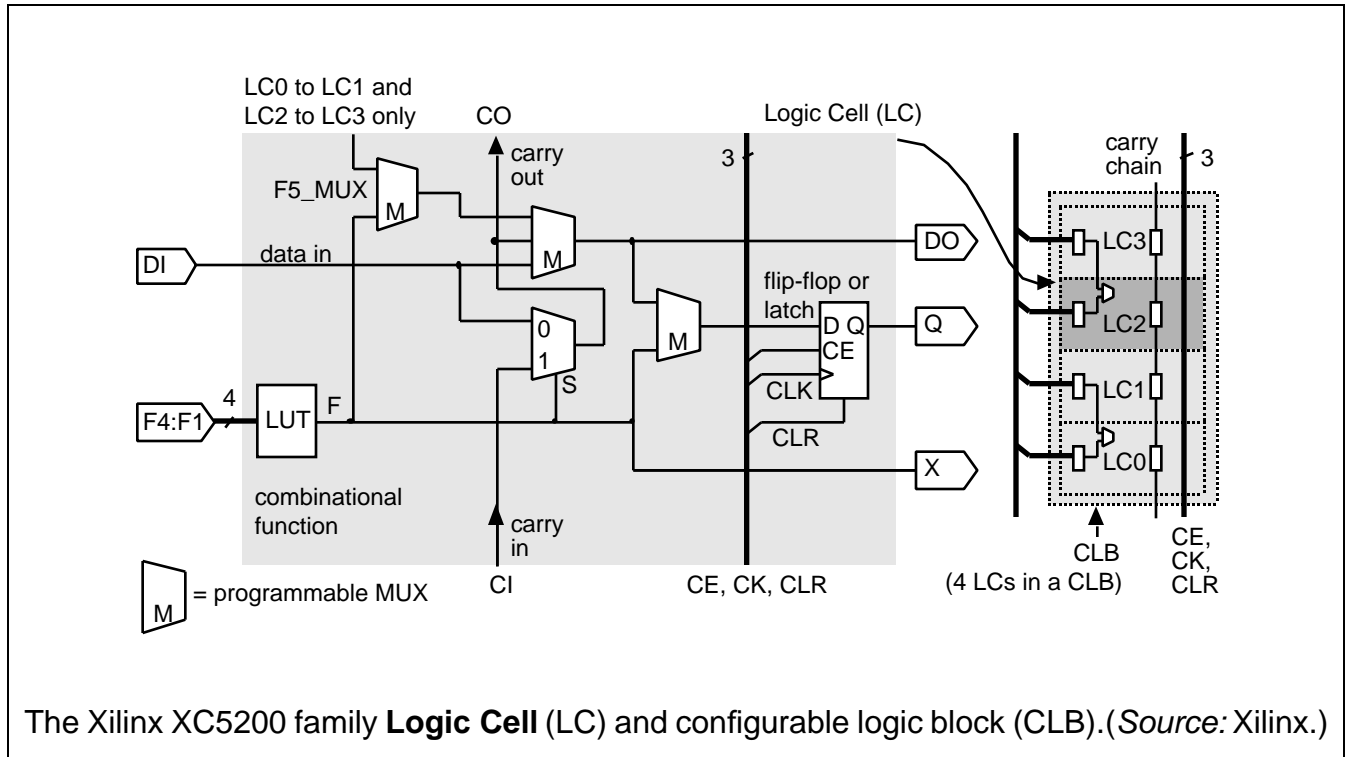


The Xilinx XC3000 CLB (configurable logic block)

(Source: Xilinx.)



5.2.3 XC5200 Logic Block



5.2.4 Xilinx CLB Analysis

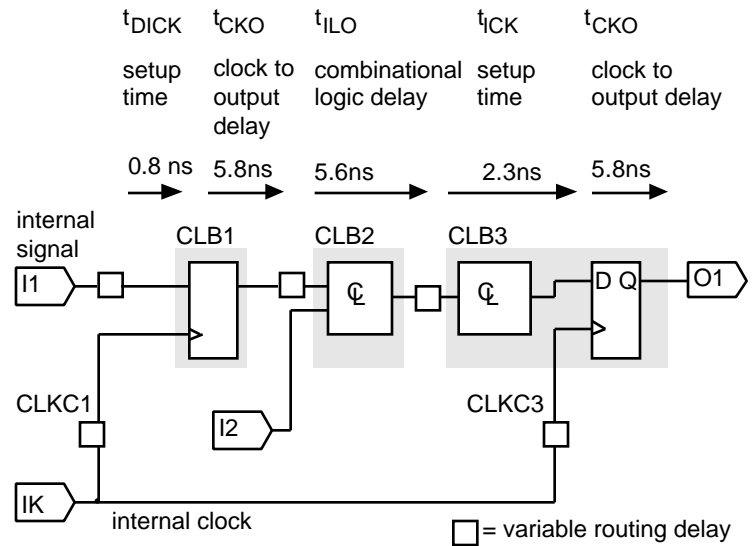
The use of a LUT has advantages and disadvantages:

- An inverter is as slow as a five-input NAND
- A LUT simplifies timing of synchronous logic
- Matched to large SRAM programming technology

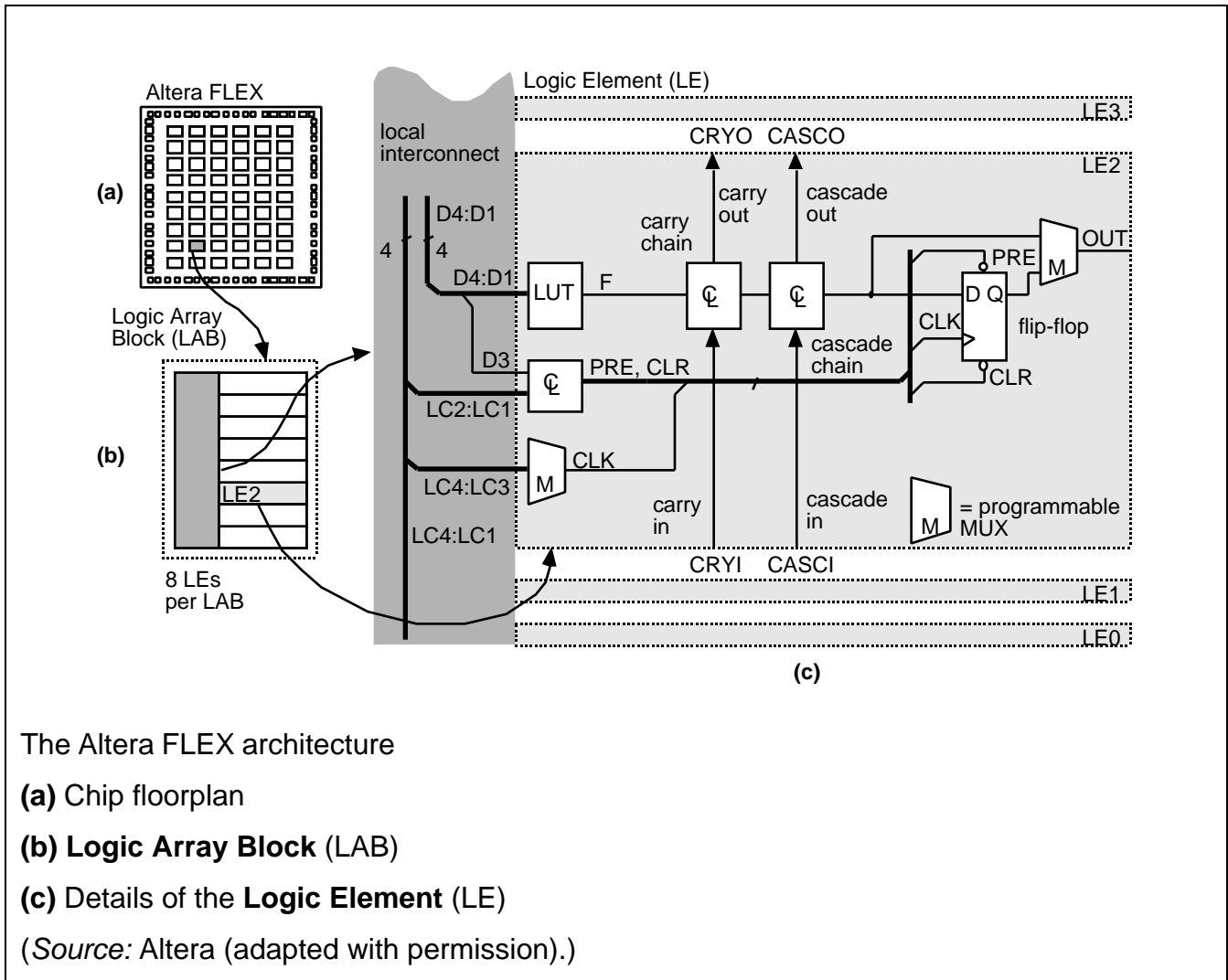
Xilinx uses two speed-grade systems:

- Maximum guaranteed toggle rate of a CLB flip-flop (in MHz) as a suffix—higher is faster
- Example: Xilinx XC3020-125 has a toggle frequency of 125MHz
- Delay time of the combinational logic in a CLB in ns—lower is faster
- Example: XC4010-6 has $t_{ILO} = 6.0\text{ns}$
- Correspondence between grade and t_{ILO} is fairly accurate for the XC2000, XC4000, and XC5200 but not for the XC3000

Xilinx LCA timing model (XC5210-6)
 (Source: Xilinx.)



5.3 Altera FLEX



The Altera FLEX architecture

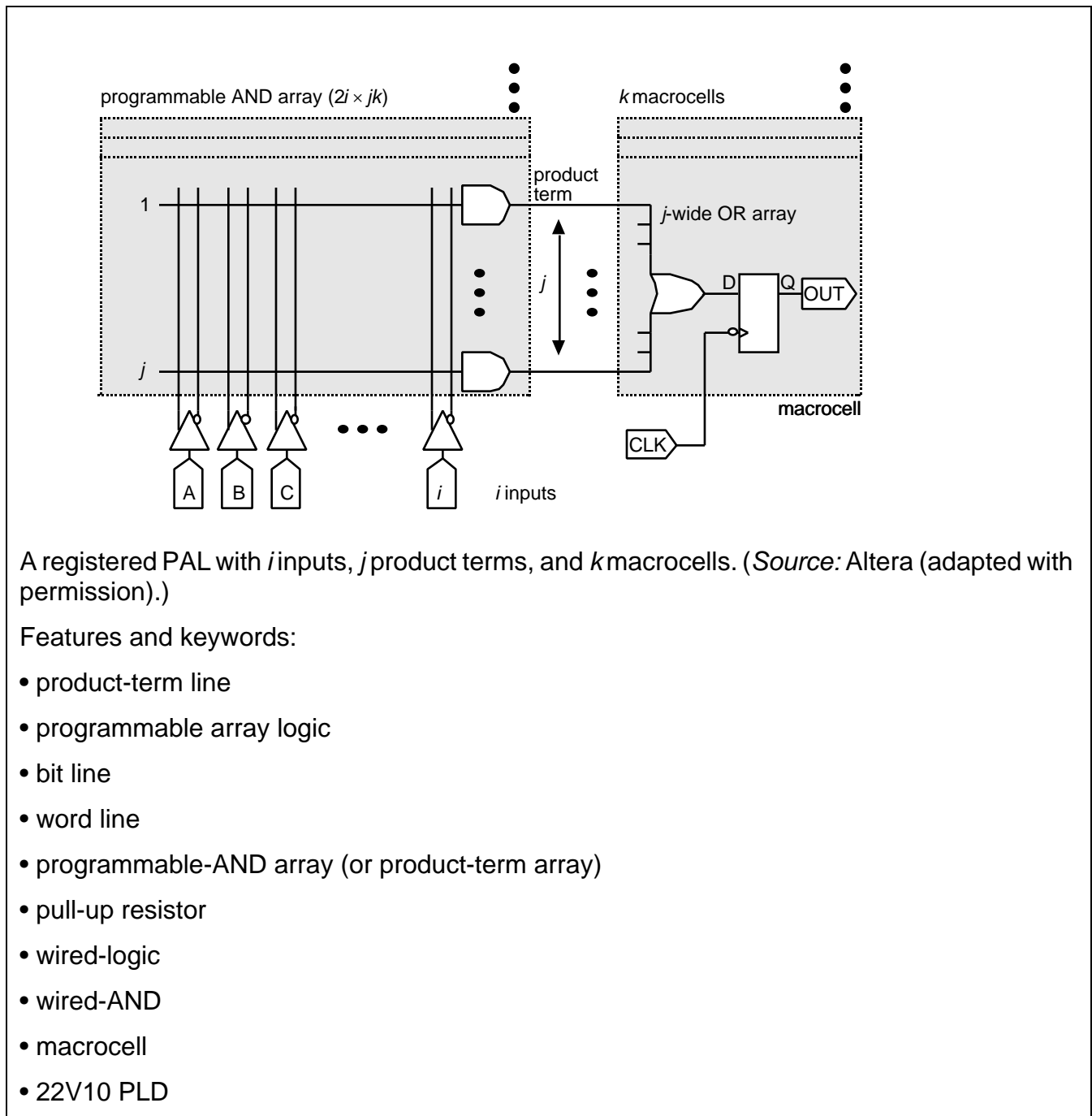
(a) Chip floorplan

(b) Logic Array Block (LAB)

(c) Details of the Logic Element (LE)

(Source: Altera (adapted with permission).)

5.4 Altera MAX

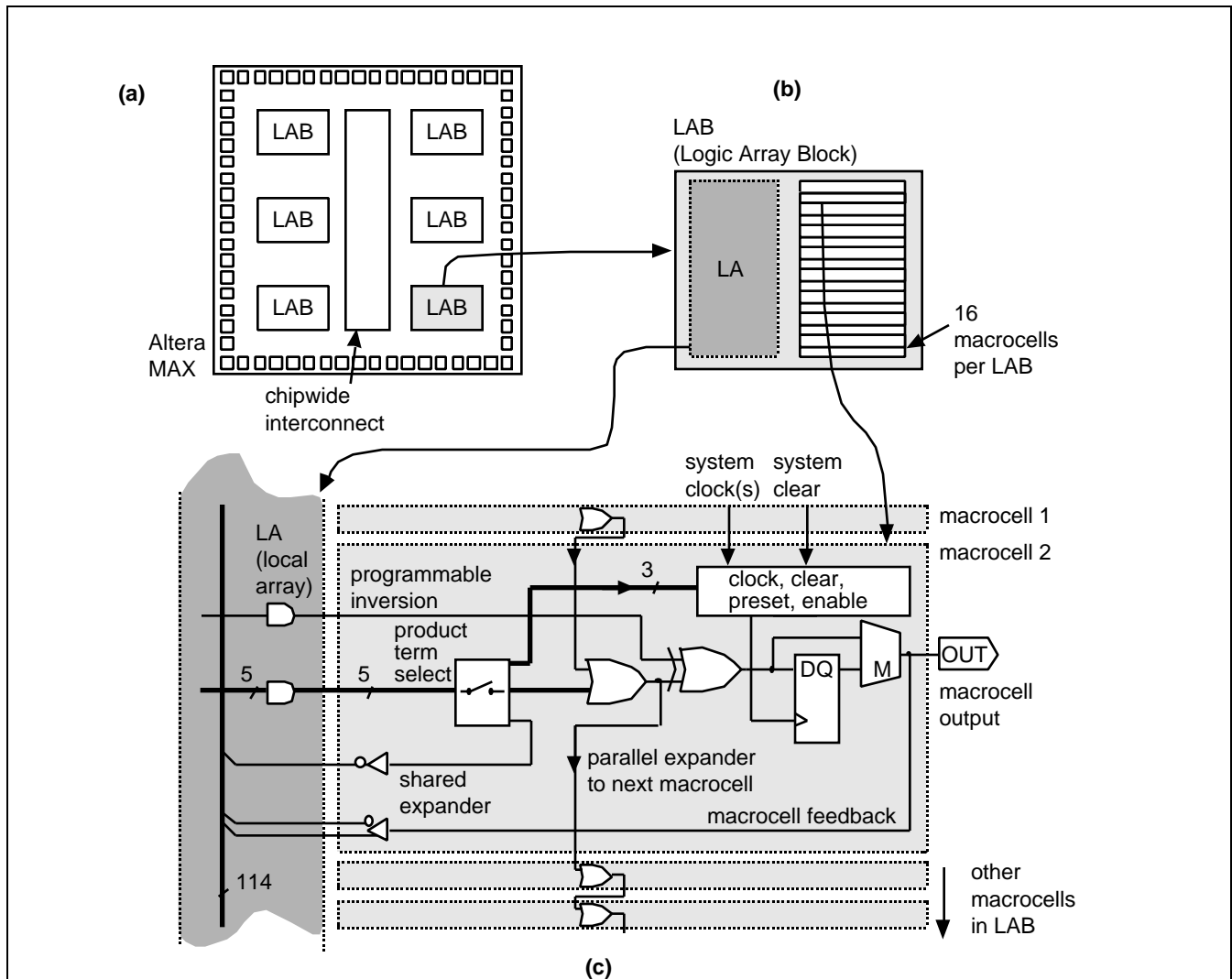


A registered PAL with i inputs, j product terms, and k macrocells. (Source: Altera (adapted with permission).)

Features and keywords:

- product-term line
- programmable array logic
- bit line
- word line
- programmable-AND array (or product-term array)
- pull-up resistor
- wired-logic
- wired-AND
- macrocell
- 22V10 PLD

5.4.1 Logic Expanders

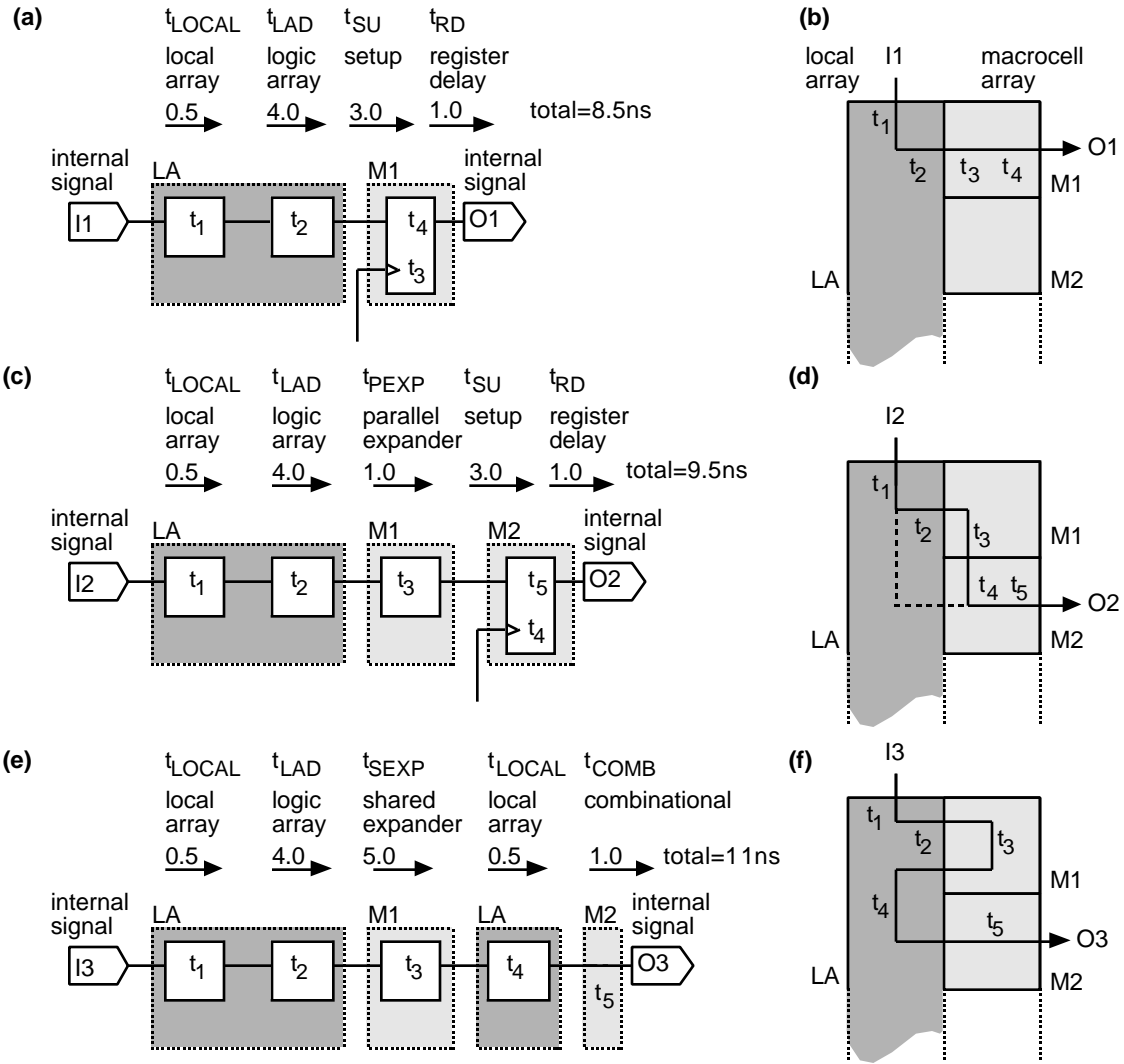


The Altera MAX architecture (the macrocell details vary between the MAX families—the functions shown here are closest to those of the MAX 9000 family macrocells) (Source: Altera (adapted with permission).) **(a)** Organization of logic and interconnect **(b)** LAB (Logic Array Block) **(c)** Macrocell

Features:

- Logic expanders and expander terms (helper terms) increase term efficiency
- Shared logic expander (shared expander, intranet) and parallel expander (internet)
- Deterministic architecture allows deterministic timing before logic assignment
- Any use of two-pass logic breaks deterministic timing
- Programmable inversion increases term efficiency

5.4.2 Timing Model



Altera MAX timing model (ns for the MAX 9000 series, '15' speed grade) (Source: Altera .)

- (a) A direct path through the logic array and a register
- (b) Timing for the direct path
- (c) Using a parallel expander
- (d) Parallel expander timing
- (e) Making two passes through the logic array to use a shared expander
- (f) Timing for the shared expander (there is no register in this path)

5.4.3 Power Dissipation in Complex PLDs

Key points: **static power • Turbo Bit**

5.5 Summary

Key points: The use of multiplexers, look-up tables, and programmable logic arrays • The difference between fine-grain and coarse-grain FPGA architectures • Worst-case timing design • Flip-flop timing • Timing models • Components of power dissipation in programmable ASICs • Deterministic and nondeterministic FPGA architectures

5.6 Problems

PROGRAMMABLE ASIC I/O CELLS

6

Key concepts:

Input/output cell (I/O cell) • I/O requirements • DC output • AC output • DC input • AC input • Clock input • Power input

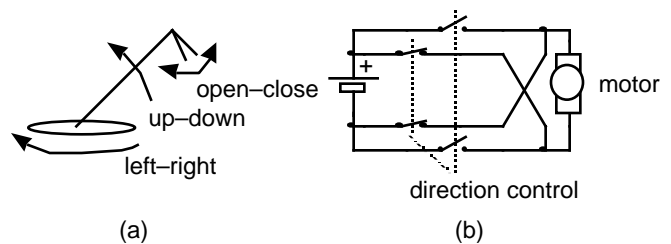
6.1 DC Output

A robot arm example

To design a system work from the outputs back to the inputs

(a) Three small DC motors drive the arm

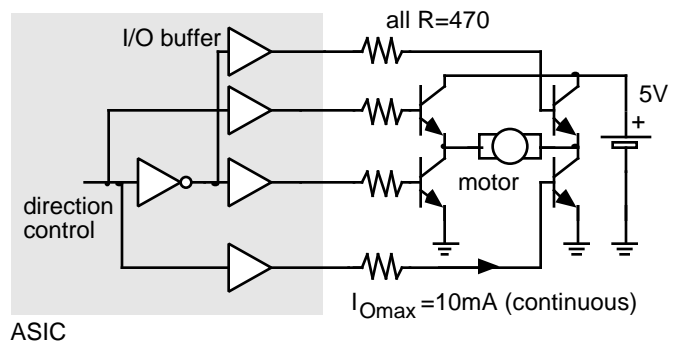
(b) Switches control each motor

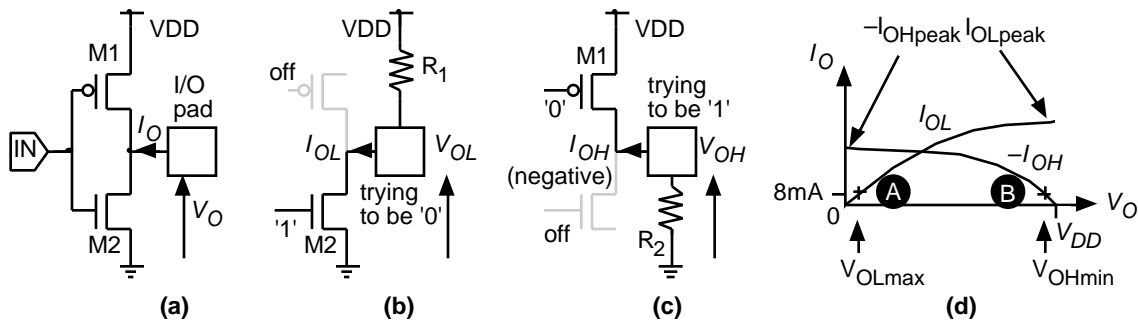


A circuit to drive a small electric motor (0.5A) using ASIC I/O buffers

Work from the outputs to the inputs

The 470 resistors drop up to 5V if an output buffer current approaches 10mA, reducing the drive to the output transistors





CMOS output buffer characteristics

(a) A CMOS complementary output buffer

(b) Transistor M2 (M1 off) sinks (to GND) a current I_{OL} through a pull-up resistor, R_1

(c) Transistor M1 (M2 off) sources (from VDD) a current $-I_{OH}$ (I_{OH} is negative) through a pull-down resistor, R_2

(d) Output characteristics:

- Data books specify characteristics at two points, A (V_{OHmin} , I_{OHmax}) and B (V_{OLmax} , I_{OLmax})

Example (Xilinx XC5200):

$V_{OLmax}=0.4V$, **low-level output voltage** at $I_{OLmax}=8.0mA$

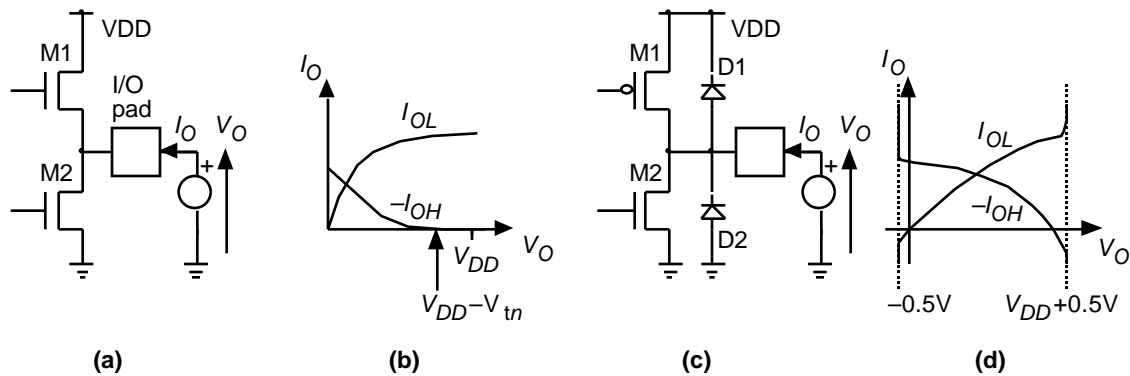
$V_{OHmin}=4.0V$, **high-level output voltage** at $I_{OHmax}=-8.0mA$

- **Output current**, I_O , is positive if it flows into the output
- Input current, if there is any, is positive if it flows into the input
- Output buffer can force the output pad to 0.4V or lower and **sink** no more than 8mA
- When the output is 4V, the buffer can **source** 8mA
- Specifying only $V_{OLmax}=0.4V$ and $V_{OHmin}=4.0V$ for a technology is strictly incorrect
- We do not know the value of I_{OLpeak} or I_{OHpeak} (typical values are 50–200mA)

6.1.1 Totem-Pole Output

Keywords: totem-pole output buffer • similar to TTL totem-pole output • two n-channel transistors in a stack • reduced output voltage swing

6.1.2 Clamp Diodes



Output buffer characteristics

(a) A CMOS totem-pole output stage (both M1 and M2 are n-channel transistors)

(b) Totem-pole output characteristics (notice the reduced signal swing)

(c) Clamp diodes, D1 and D2, in an output buffer (totem-pole or complementary) prevent the I/O pad from voltage excursions greater than V_{DD} and less than V_{SS}

(d) The clamp diodes conduct as the output voltage exceeds the supply voltage bounds

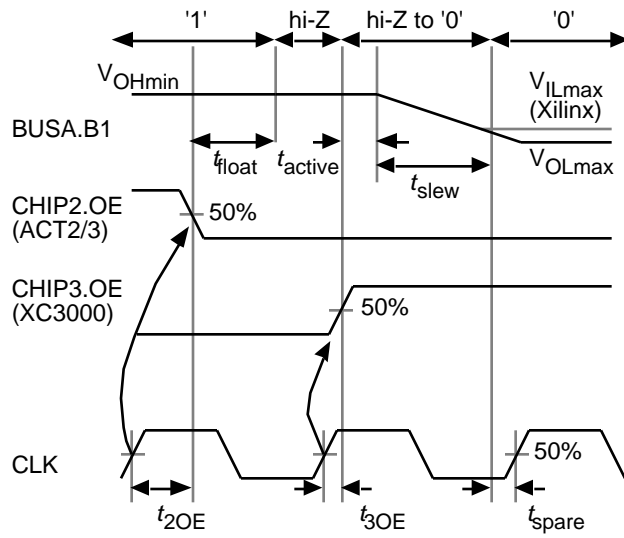
6.2 AC Output

Keywords: bus transceivers • bus transaction (a sequence of signals on a bus) • floating a bus • bus keeper • trip points • three-stated (high-impedance or hi-Z) • time to float • disable time, time to begin hi-Z, or time to turn off • slew • sustained three-state (s/t/s) • turnaround cycle

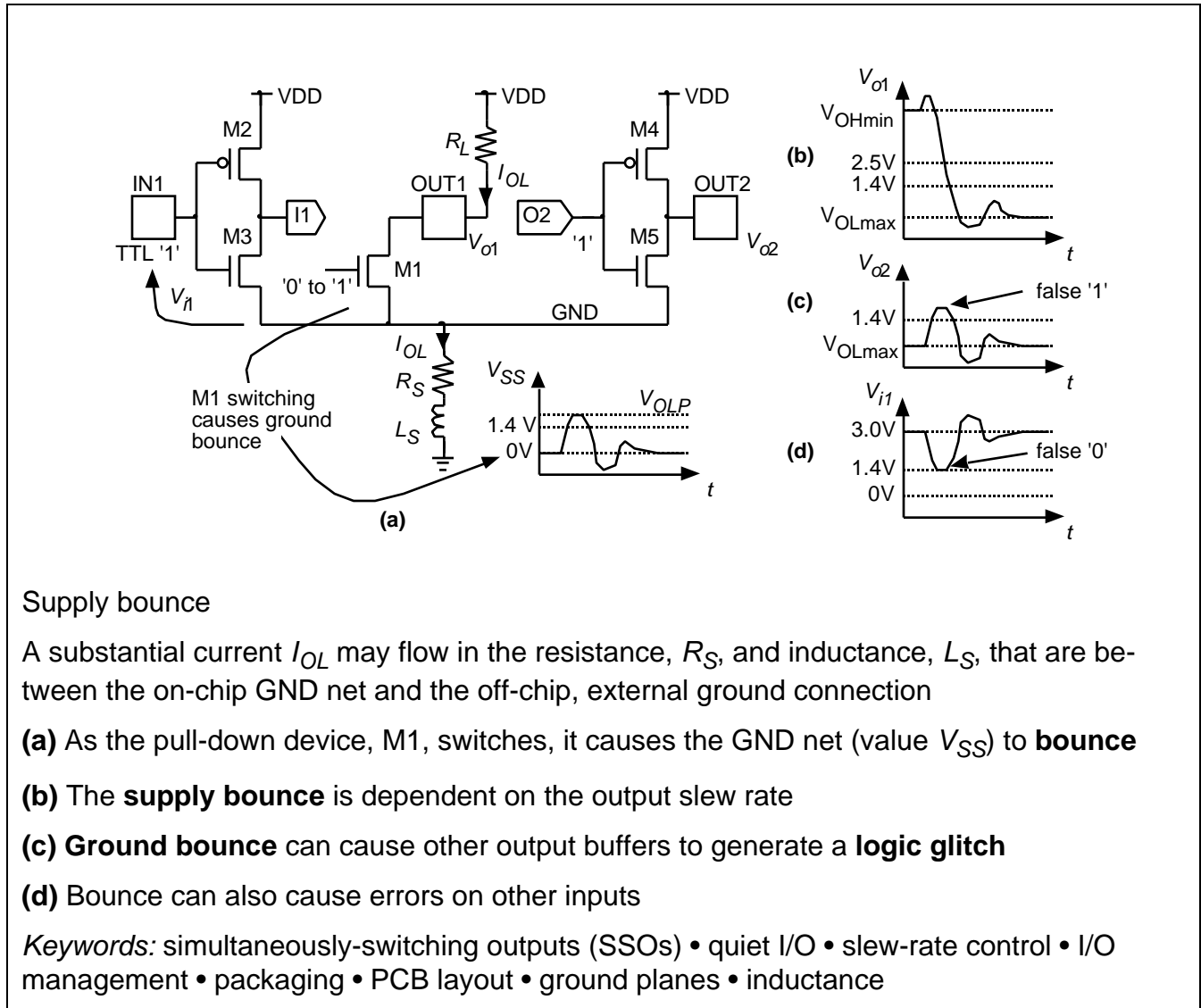
Three-state bus timing

The on-chip delays, t_{2OE} and t_{3OE} , for the logic that generates signals CHIP2.E1 and CHIP3.E1 are derived from the timing models

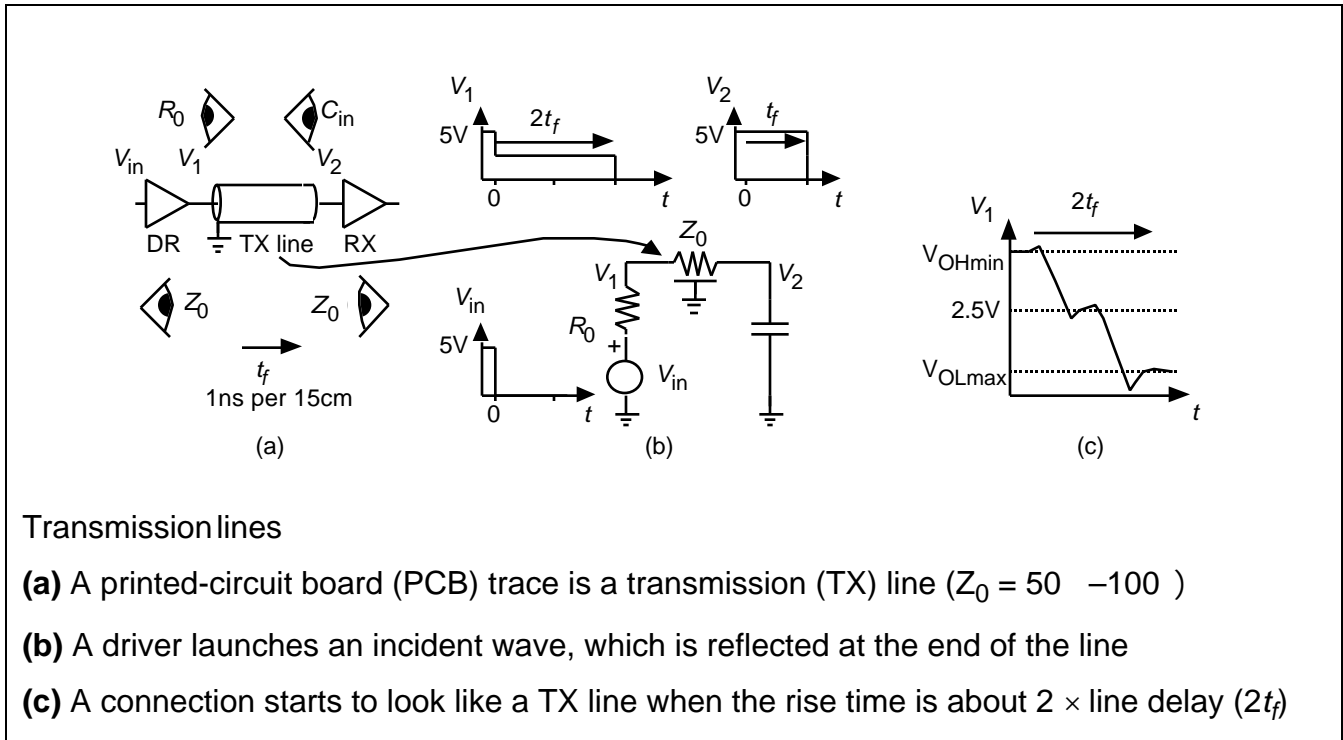
(The minimum values for each chip would be the clock-to-Q delay times)



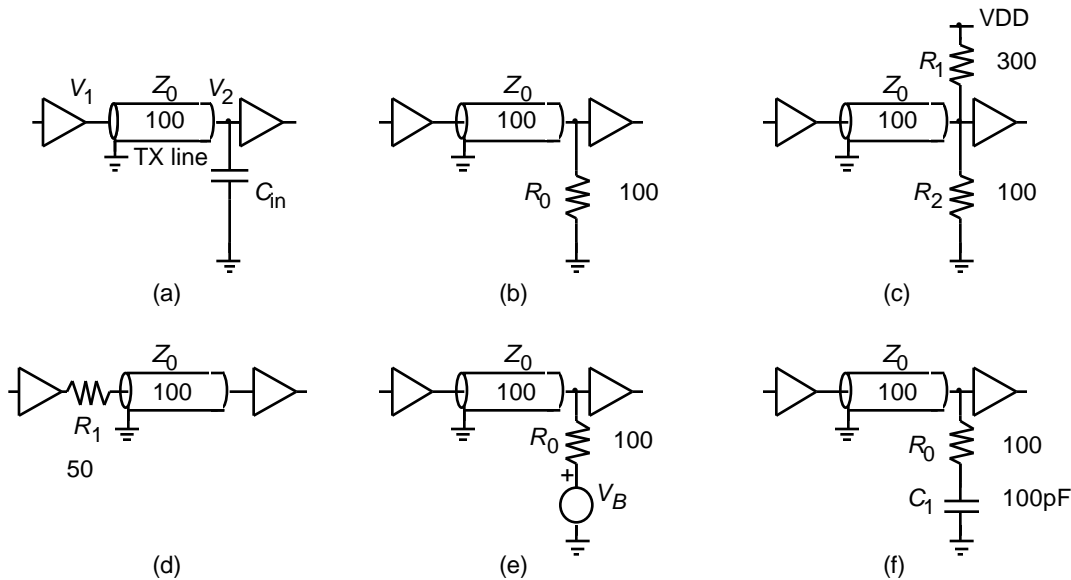
6.2.1 Supply Bounce



6.2.2 Transmission Lines



6.3 DC Input



Transmission line termination

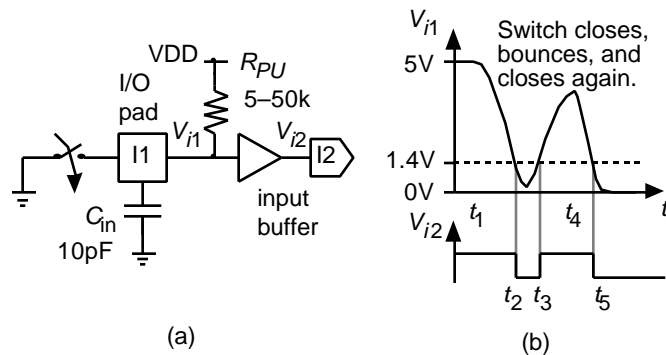
- (a) Open-circuit or capacitive termination
- (b) Parallel resistive termination
- (c) Thévenin termination
- (d) Series termination at the source
- (e) Parallel termination using a voltage bias
- (f) Parallel termination with a series capacitor

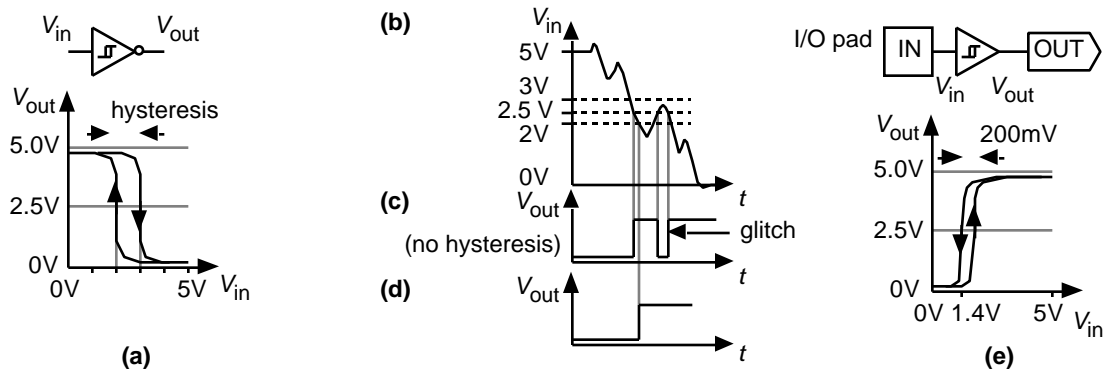
A switch input

(a) A pushbutton switch connected to an input buffer with a pull-up resistor

(b) As the switch bounces several pulses may be generated

We might have to **debounce** this signal using an SR flip-flop or small state machine





DC input

(a) A **Schmitt-trigger inverter** • lower switching threshold • upper switching threshold • difference between thresholds is the **hysteresis**

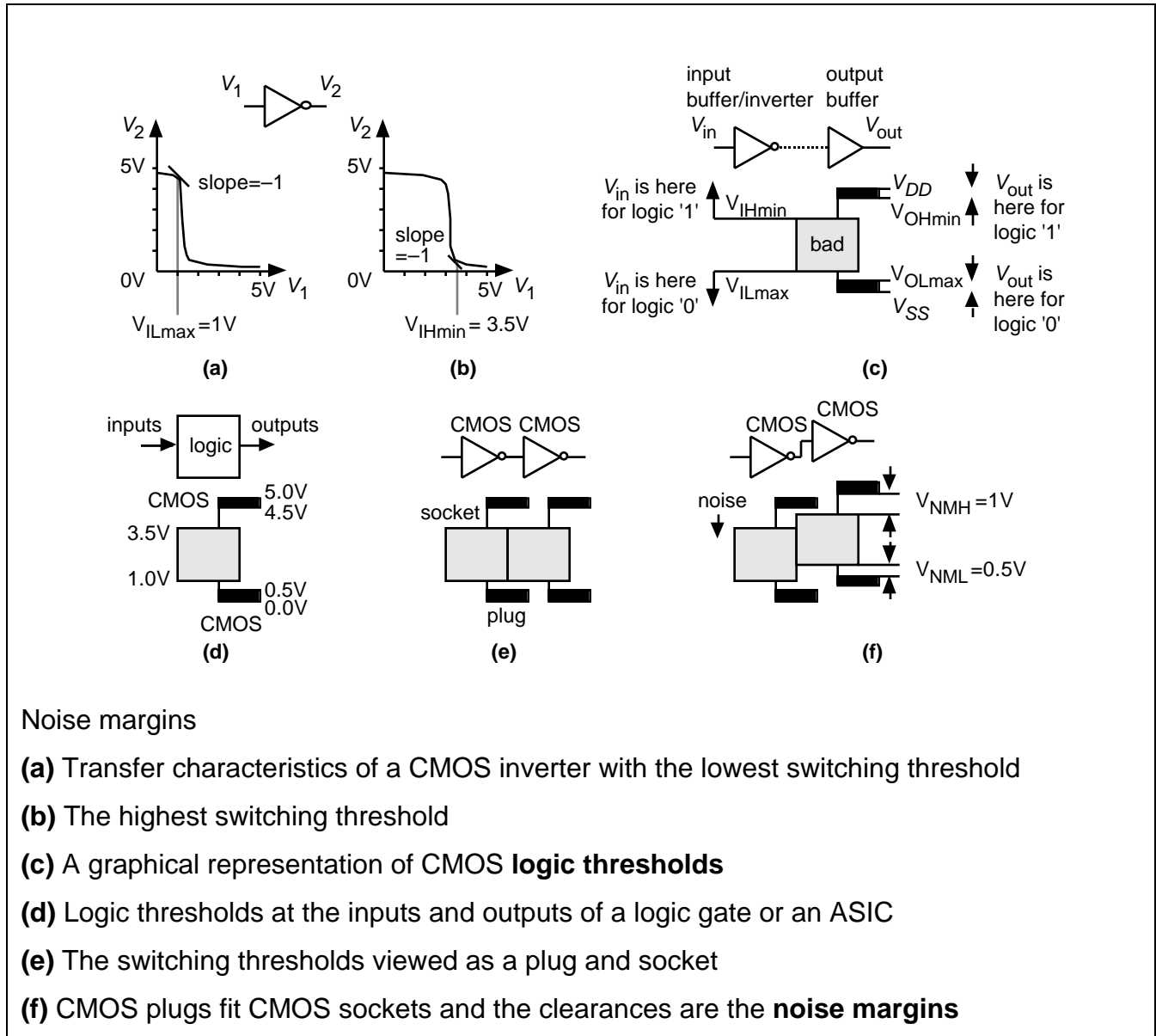
(b) A noisy input signal

(c) Output from an inverter with no hysteresis

(d) Hysteresis helps prevent **glitches**

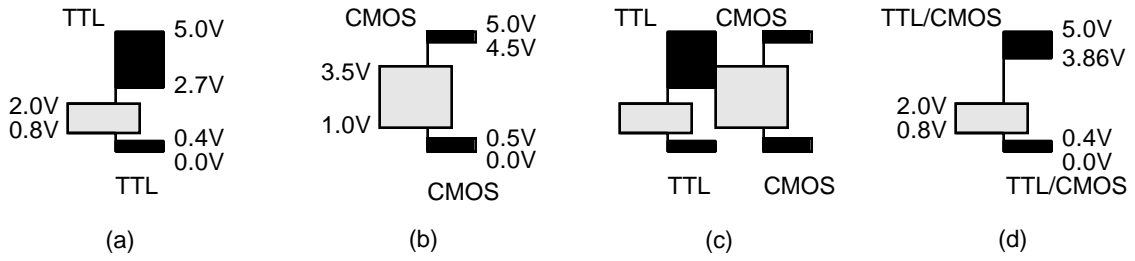
(e) A typical FPGA input buffer with a hysteresis of 200mV and a threshold of 1.4V

6.3.1 Noise Margins



Noise margins

- (a) Transfer characteristics of a CMOS inverter with the lowest switching threshold
- (b) The highest switching threshold
- (c) A graphical representation of CMOS **logic thresholds**
- (d) Logic thresholds at the inputs and outputs of a logic gate or an ASIC
- (e) The switching thresholds viewed as a plug and socket
- (f) CMOS plugs fit CMOS sockets and the clearances are the **noise margins**



TTL and CMOS logic thresholds

- (a) TTL logic thresholds
- (b) Typical CMOS logic thresholds
- (c) A TTL plug will not fit in a CMOS socket
- (d) Raising V_{OHmin} solves the problem

6.3.2 Mixed-Voltage Systems

FPGA logic thresholds												
	I/O options		Input levels		Output levels (high current)				Output levels (low current)			
XC3000	TTL		2.0	0.8	3.86	-4.0	0.40	4.0				
	CMOS		3.85	0.9	3.86	-4.0	0.40	4.0				
XC3000L			2.0	0.8	2.40	-4.0	0.40	4.0	2.80	-0.1	0.2	0.1
XC4000			2.0	0.8	2.40	-4.0	0.40	12.0				
XC4000H	TTL	TTL	2.0	0.8	2.40	-4.0	0.50	24.0				
	CMOS	CMOS	3.85	0.9	4.00	-1.0	0.50	24.0				
XC8100	TTL	R	2.0	0.8	3.86	-4.0	0.50	24.0				
	CMOS	C	3.85	0.9	3.86	-4.0	0.40	4.0				
ACT 2/3			2.0	0.8	2.4	-8.0	0.50	12.0	3.84	-4.0	0.33	6.0
FLEX10k	3V/5V		2.0	0.8	2.4	-4.0	0.45	12.0				

Mixed-voltage systems

(a) TTL levels

(b) Low-voltage CMOS levels • JEDEC 8 • $3.3 \pm 0.3V$

(c) Mixed-voltage ASIC • 5V-tolerant I/O • V_{DDint} and $V_{DDI/O}$

(d) A problem when connecting two chips with different supply voltages—caused by the input clamp diodes

The diagrams illustrate mixed-voltage systems. (a) shows TTL levels with a 5.0V supply and input levels of 2.0V (low) and 2.7V (high), and output levels of 0.4V (low) and 0.0V (high). (b) shows CMOS3V levels with a 3.3V supply and input levels of 2.0V (low) and 2.4V (high), and output levels of 0.4V (low) and 0.0V (high). (c) shows a mixed-voltage ASIC with a core and I/O blocks, with separate supply voltages V_{DDIO} and V_{DDINT} . (d) shows a circuit diagram of two chips, CHIP1 and CHIP2, connected. CHIP1 is powered by $V_{DD1} = 5.5V$ and has an output OUT1. CHIP2 is powered by $V_{DD2} = 3.0V$ and has an input IN2. The output of CHIP1 is connected to the input of CHIP2 through a 1k resistor R_{in} . Input clamp diodes D1, D2, D3, and D4 are shown on the input and output lines of both chips.

6.4 AC Input

Keywords and concepts: input bus • sampled data • clock frequency of 100kHz • FPGA • system clock • 10MHz • Data should be at the flip-flop input at least the flip-flop setup time before the clock edge. Unfortunately there is no way to guarantee this; the data clock and the system clock are completely independent

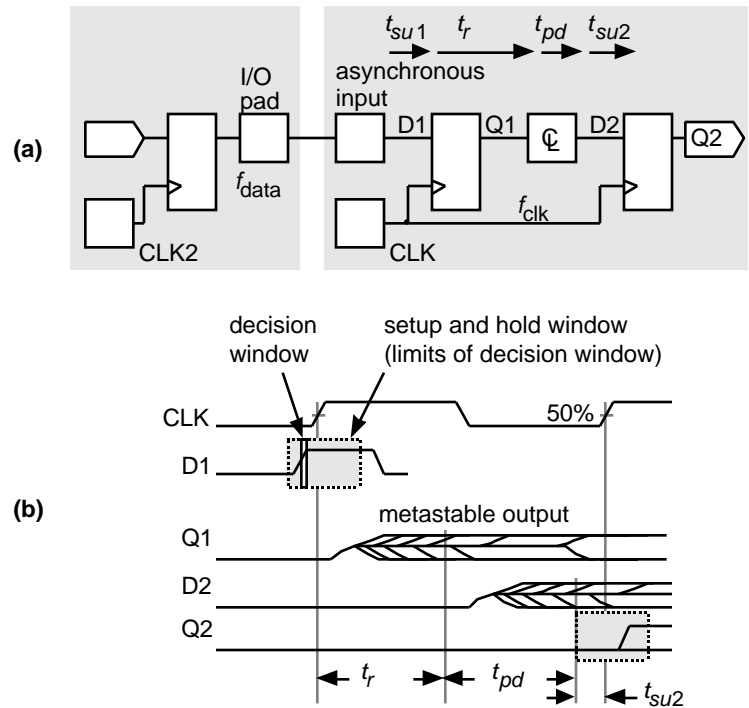
6.4.1 Metastability

Metastability

(a) Data coming from one clocked system is an **asynchronous** input to another

(b) A flip-flop (or latch, a **sampler**) has a very narrow decision window bounded by the setup and hold times to **resolve** the input

If the data input changes inside the decision window (a setup or hold-time **violation**) the output may be **metastable**—neither '1' or '0'—an **upset**



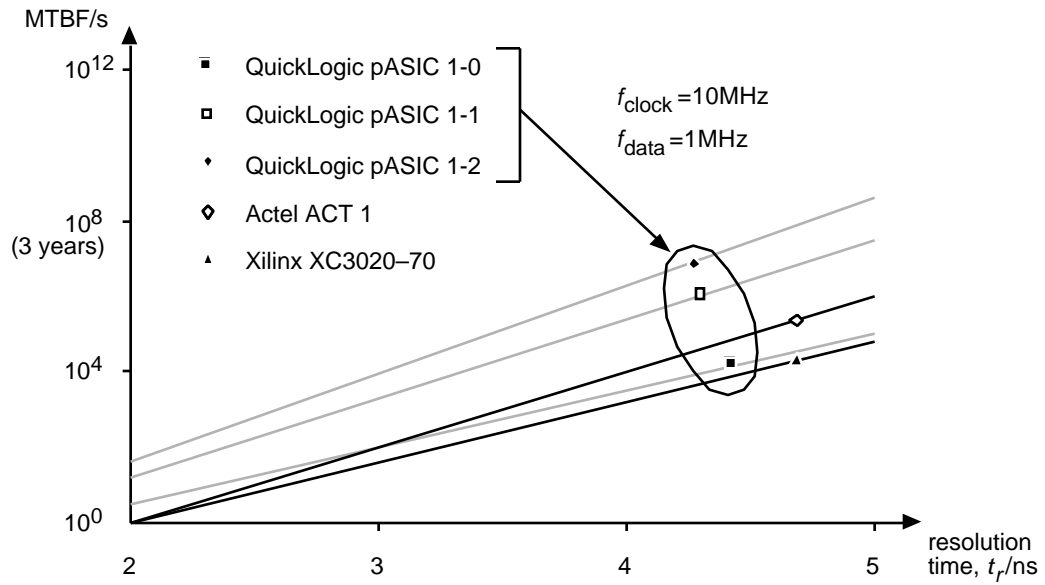
Metastability parameters for FPGA flip-flops (not guaranteed by the vendors)		
FPGA	T_0/s	τ_c/s
Actel ACT 1	1.0E-09	2.17E-10
Xilinx XC3020-70	1.5E-10	2.71E-10
QuickLogic QL12x16-0	2.94E-11	2.91E-10
QuickLogic QL12x16-1	8.38E-11	2.09E-10
QuickLogic QL12x16-2	1.23E-10	1.85E-10
Altera MAX 7000	2.98E-17	2.00E-10
Altera FLEX 8000	1.01E-13	7.89E-11

The **mean time between upsets (MTBU)** or MTBF is

$$\text{MTBU} = \frac{1}{p f_{\text{clock}} f_{\text{data}}} = \frac{\exp t_r / \tau_c}{f_{\text{clock}} f_{\text{data}}}$$

where f_{clock} is the clock frequency and f_{data} is the data frequency

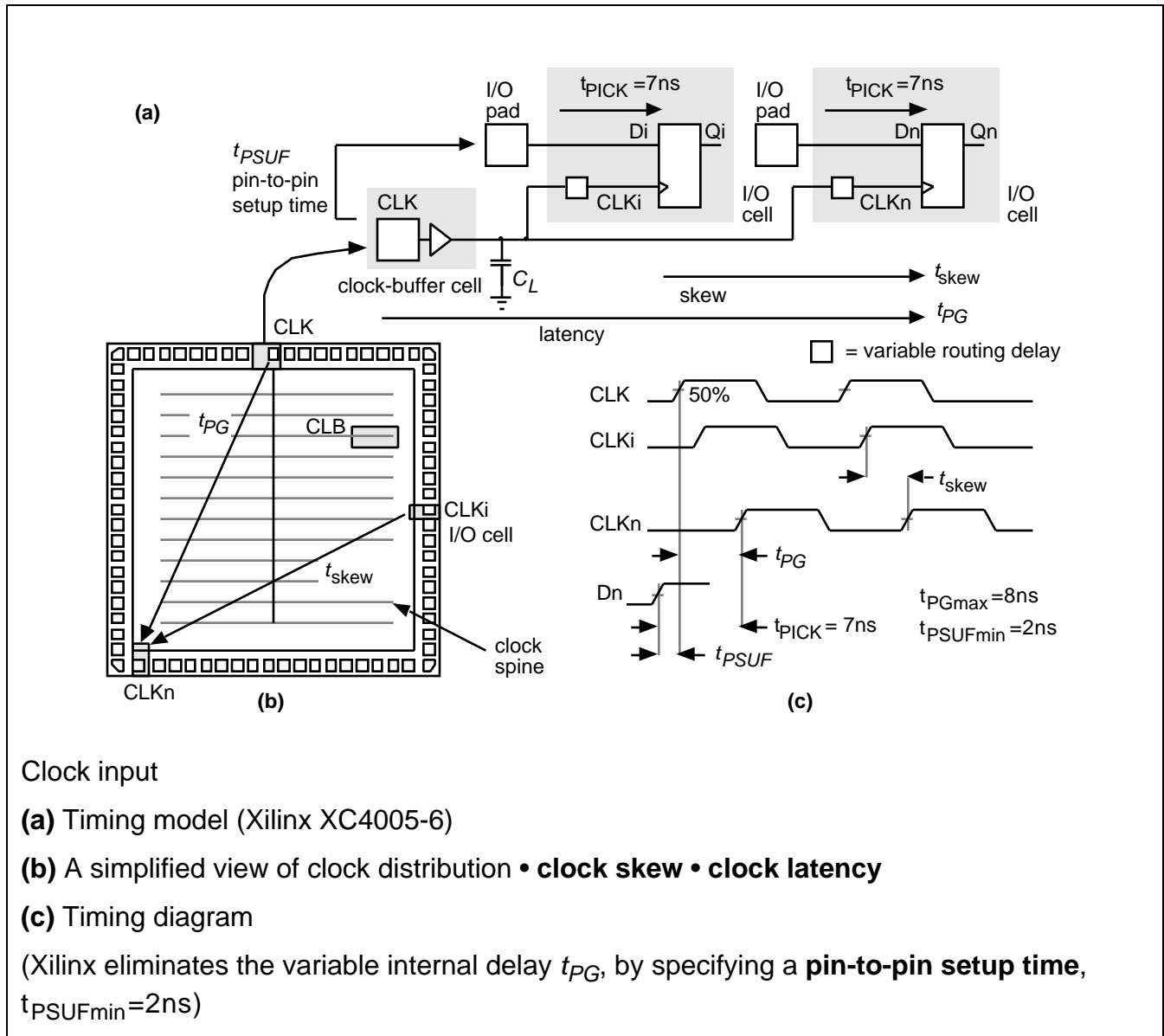
A **synchronizer** is built from two flip-flops in cascade, and greatly reduces the effective values of τ_c and T_0 over a single flip-flop. The penalty is an extra clock cycle of latency.



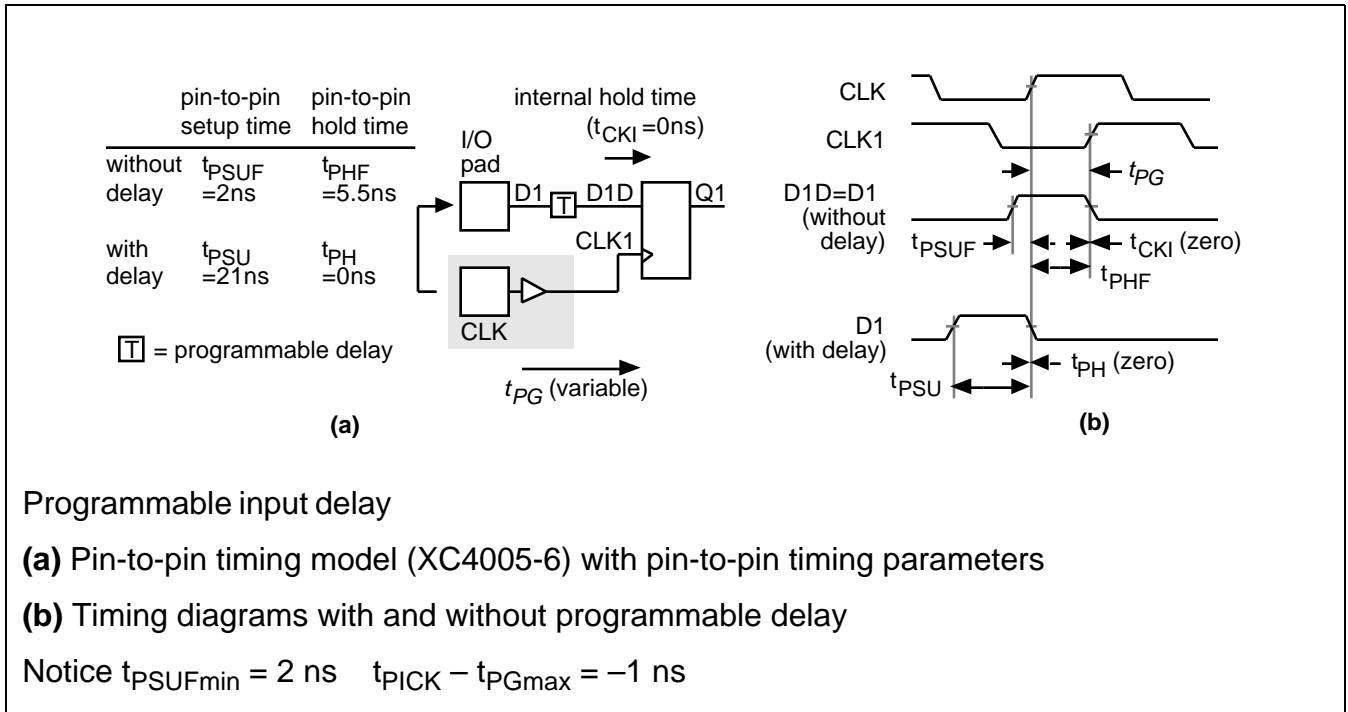
Mean time between failure (MTBF) as a function of resolution time

The data is from FPGA vendors' data books for a single flip-flop with clock frequency of 10MHz and a data input frequency of 1MHz

6.5 Clock Input

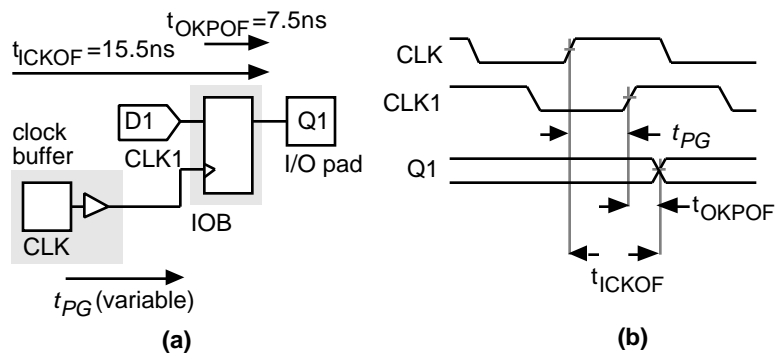


6.5.1 Registered Input



Registered output

- (a) Timing model with values for an XC4005-6 programmed with the fast slew-rate option
- (b) Timing diagram



6.6 Power Input

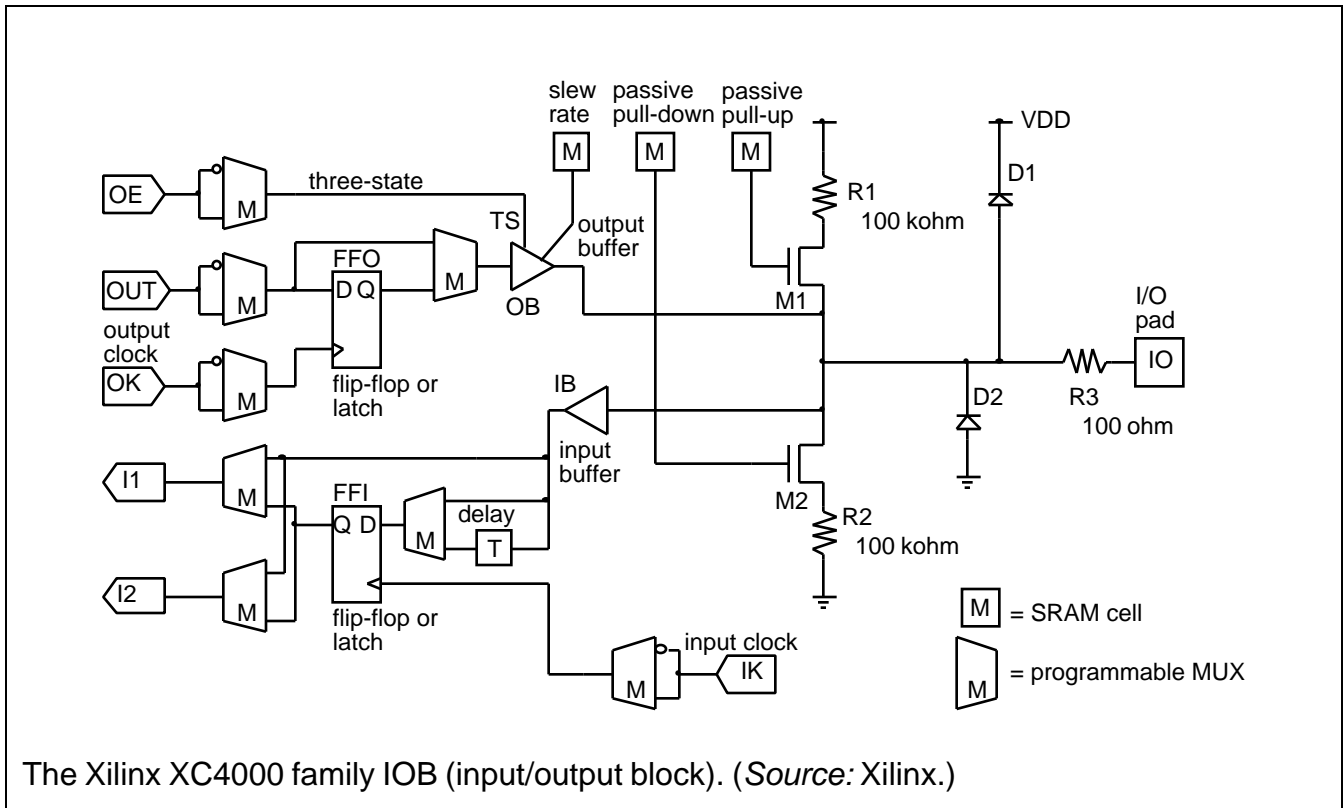
6.6.1 Power Dissipation

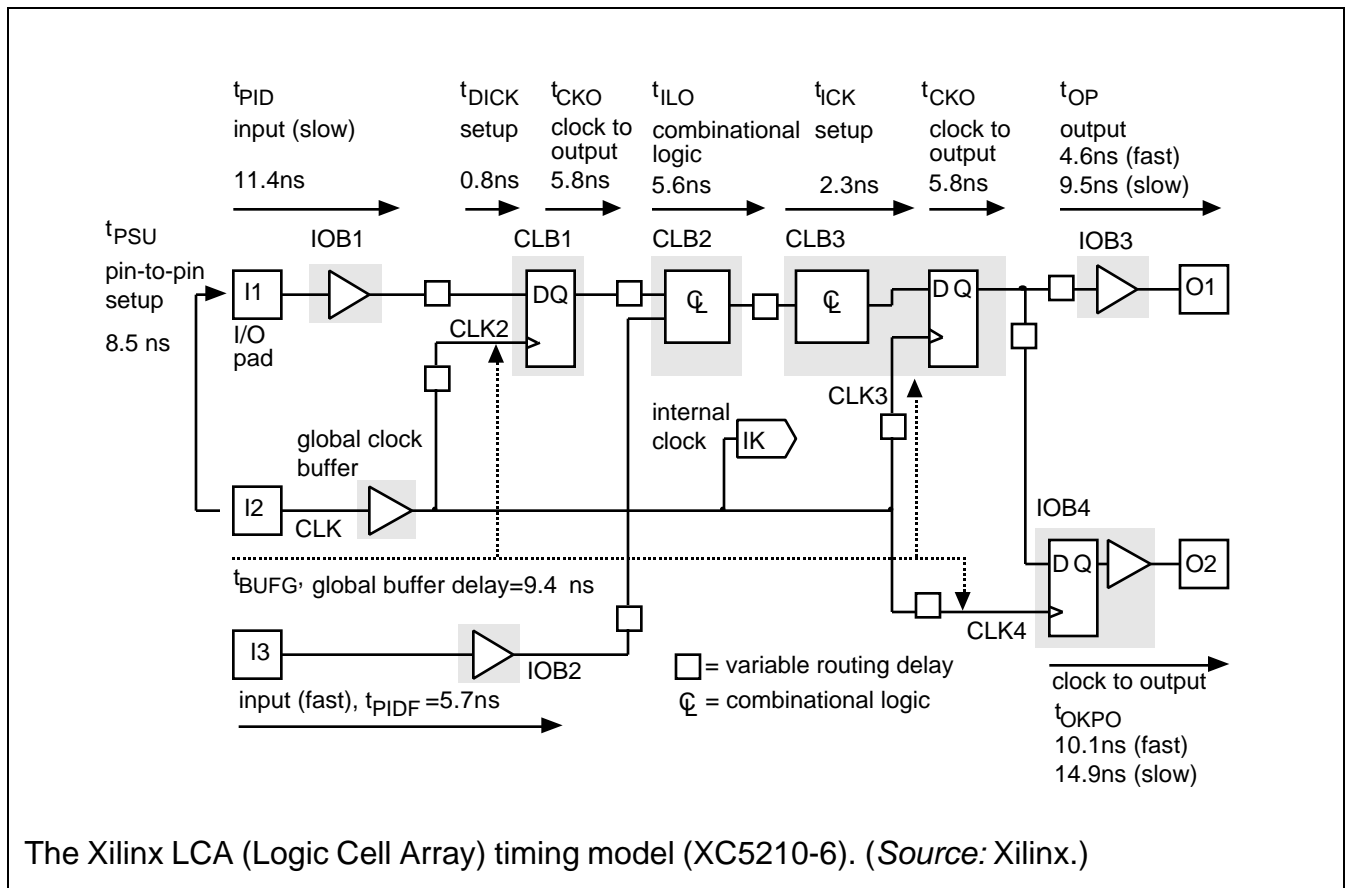
Thermal characteristics of ASIC packages				
Package	Pin count	Max. power P_{\max}/W	$J_A / ^\circ C W^{-1}$ (still air)	$J_A / ^\circ C W^{-1}$ (still air)
CPGA	84		33	32–38
CQFP	84		40	
CQFP	172		25	
VQFP	80		68	

6.6.2 Power-On Reset

Key concepts: Power-on reset sequence • Xilinx FPGAs configure all flip-flops (in either the CLBs or IOBs) as either SET or RESET • after chip programming is complete, the global SET/RESET signal forces all flip-flops on the chip to a known state • this may determine the initial state of a state machine, for example

6.7 Xilinx I/O Block





The Xilinx LCA (Logic Cell Array) timing model (XC5210-6). (Source: Xilinx.)

6.7.1 Boundary Scan

Key concepts: IEEE boundary-scan standard 1149.1 • Many FPGAs contain a standard boundary-scan test logic structure with a four-pin interface • **in-system programming (ISP)**

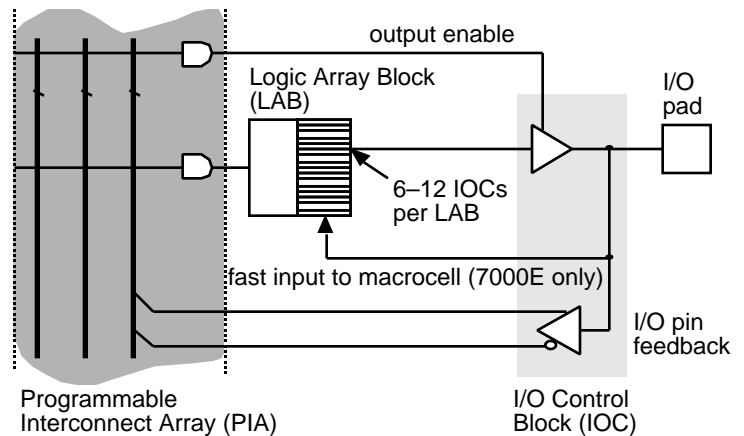
6.8 Other I/O Cells

A simplified block diagram of the Altera **I/O Control Block (IOC)** used in the MAX 5000 and MAX 7000 series

The **I/O pin feedback** allows the I/O pad to be isolated from the macrocell

It is thus possible to use a LAB without using up an I/O pad (as you often have to do using a PLD such as a 22V10)

The **PIA** is the chipwide interconnect

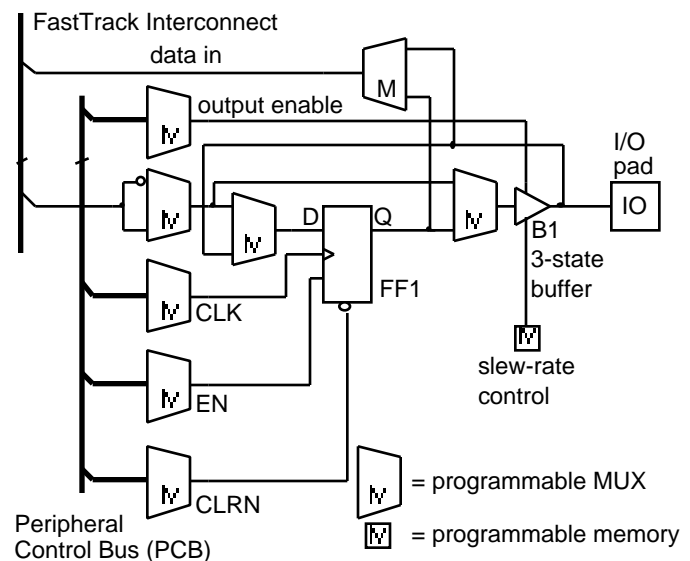


A simplified block diagram of the Altera **I/O Element (IOE)**, used in the FLEX 8000 and 10k series

The MAX 9000 IOC (I/O Cell) is similar

The FastTrack Interconnect bus is the chipwide interconnect

The **Peripheral Control Bus (PCB)** is used for control signals common to each IOE



6.9 Summary

Key concepts:

Outputs can typically source or sink 5–10mA continuously into a DC load

Outputs can typically source or sink 50–200mA transiently into an AC load

Input buffers can be CMOS (threshold at $0.5 V_{DD}$) or TTL (1.4V)

Input buffers normally have a small hysteresis (100–200mV)

CMOS inputs must never be left floating

Clamp diodes to GND and VDD are present on every pin

Inputs and outputs can be registered or direct

I/O registers can be in the I/O cell or in the core

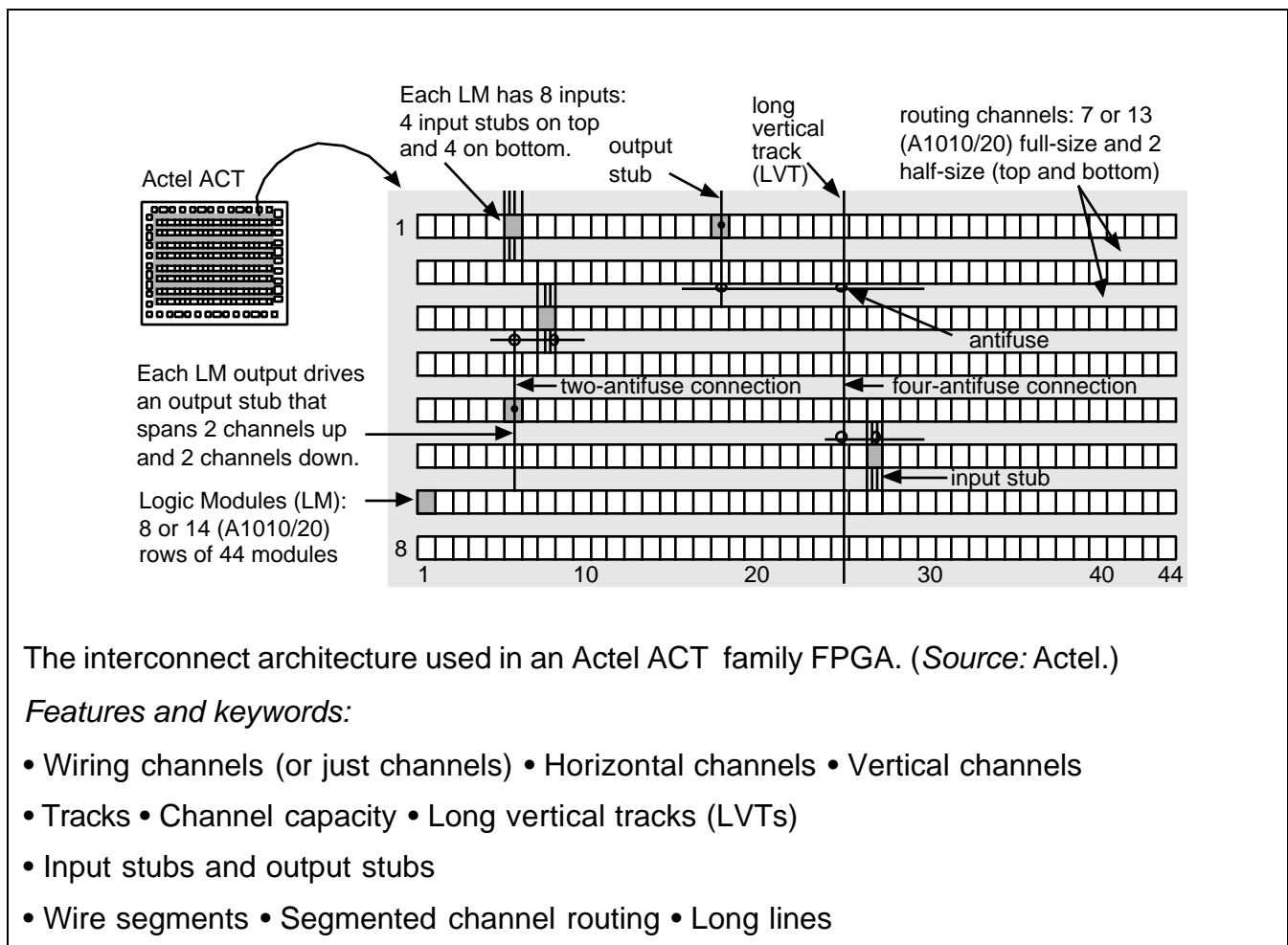
Metastability is a problem when working with asynchronous inputs

PROGRAMMABLE ASIC INTERCONNECT

7

Key concepts: programmable interconnect • raw materials: aluminum-based metallization and a line capacitance of 0.2pFcm^{-1}

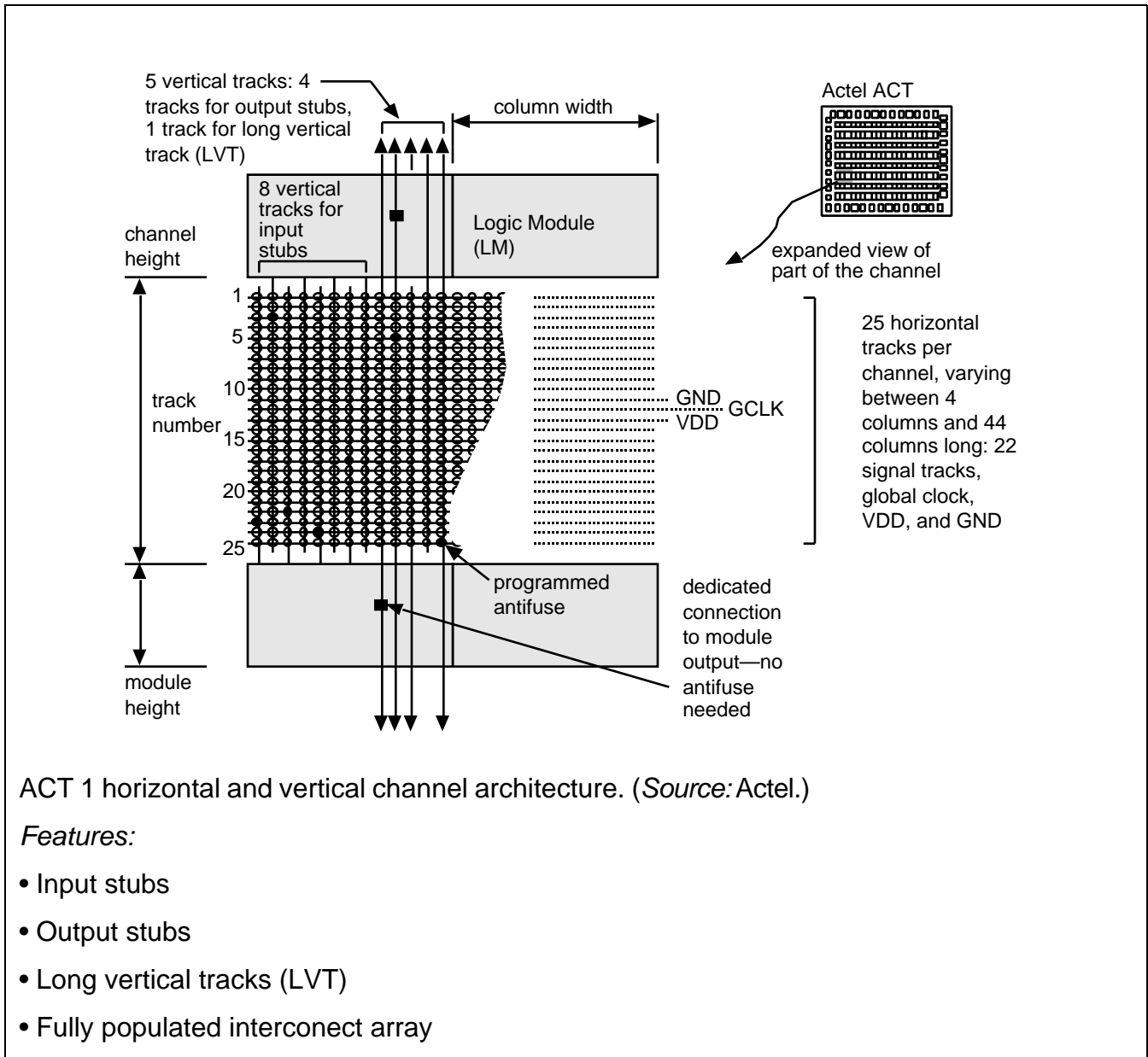
7.1 Actel ACT



The interconnect architecture used in an Actel ACT family FPGA. (Source: Actel.)

Features and keywords:

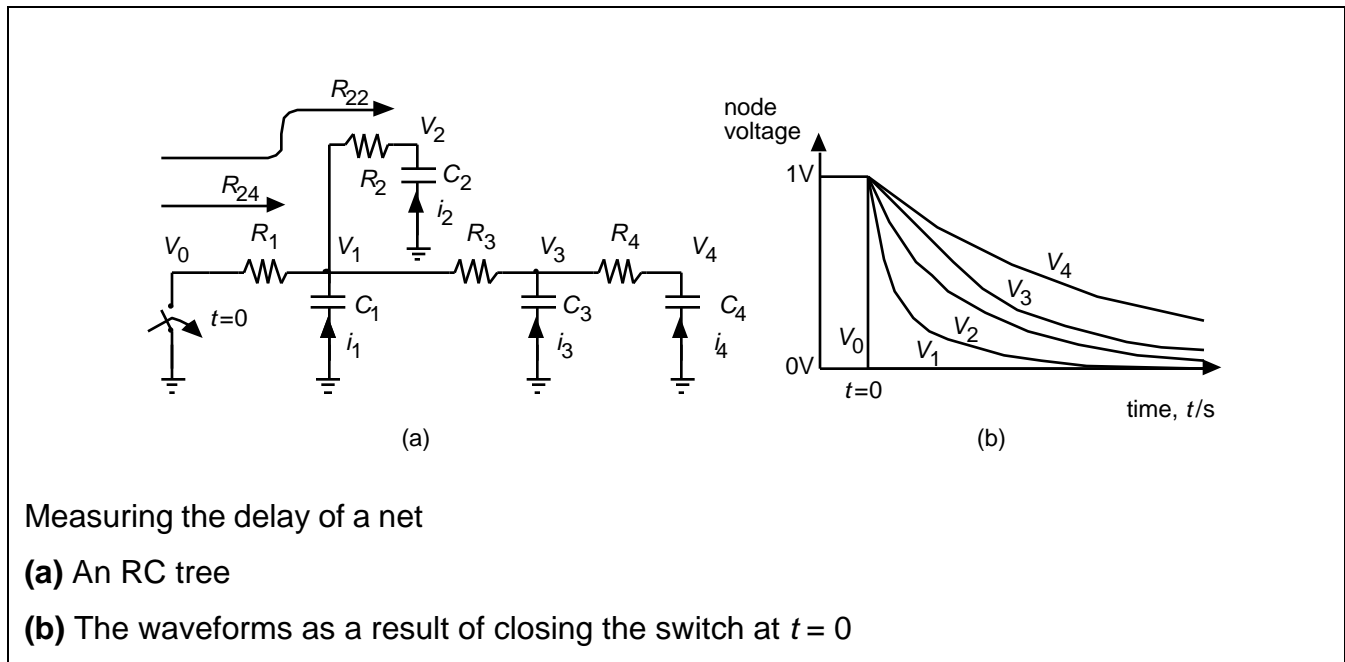
- Wiring channels (or just channels) • Horizontal channels • Vertical channels
- Tracks • Channel capacity • Long vertical tracks (LVTs)
- Input stubs and output stubs
- Wire segments • Segmented channel routing • Long lines



7.1.1 Routing Resources

Actel FPGA routing resources						
	Horizontal tracks per channel, H	Vertical tracks per column, V	Rows, R	Columns, C	Total antifuses on each chip	H×V×R × C
A1010	22	13	8	44	112,000	100,672
A1020	22	13	14	44	186,000	176,176
A1225A	36	15	13	46	250,000	322,920
A1240A	36	15	14	62	400,000	468,720
A1280A	36	15	18	82	750,000	797,040

7.1.2 Elmore's Constant

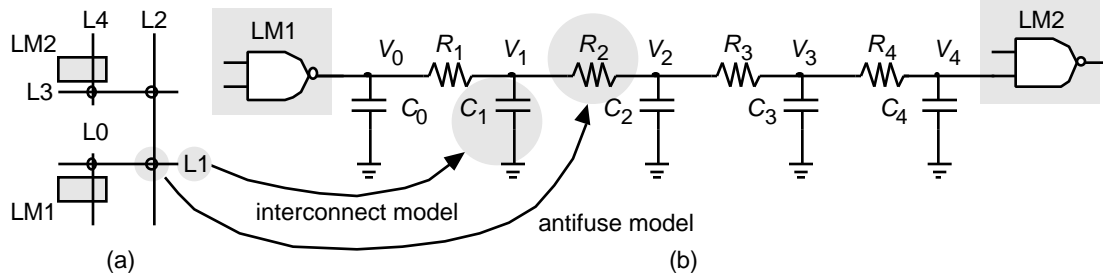


$$V_j(t) = \exp(-t/D_j) \quad ; \quad D_j = \sum_{k=1}^n R_{ki}C_k$$

The time constant D_j is often called the **Elmore delay** and is different for each node.

I call D_j the **Elmore time constant** as a reminder that, if we approximate V_j by an exponential waveform, the delay of the RC tree using 0.35/0.65 trip points is approximately D_j seconds.

7.1.3 RC Delay in Antifuse Connections



Actel routing model

(a) A four-antifuse connection. L0 is an output stub, L1 and L3 are horizontal tracks, L2 is a long vertical track (LVT), and L4 is an input stub

(b) An RC-tree model. Each antifuse is modeled by a resistance and each interconnect segment is modeled by a capacitance.

$$D_4 = R_{14}C_1 + R_{24}C_2 + R_{14}C_1 + R_{44}C_4$$

$$= (R_1 + R_2 + R_3 + R_4)C_4 + (R_1 + R_2 + R_3)C_3 + (R_1 + R_2)C_2 + R_1C_1$$

$$D_4 = 4RC_4 + 3RC_3 + 2RC_2 + RC_1$$

- Two antifuses will generate a $3RC$ time constant
- Three antifuses a $6RC$ time constant
- Four antifuses gives a $10RC$ time constant
- Interconnect delay grows quadratically (n^2) as we increase the interconnect length and the number of antifuses, n

7.1.4 Antifuse Parasitic Capacitance

7.1.5 ACT 2 and ACT 3 Interconnect channel density • fast fuse

Actel interconnect parameters		
Parameter	A1010/A1020	A1010B/A1020B
Technology	2.0 μ m, =1.0 μ m	1.2 μ m, =0.6 μ m
Die height (A1010)	240mil	144mil
Die width (A1010)	360mil	216mil
Die area (A1010)	86,400mil ² =56M ²	31,104mil ² =56M ²
Logic Module (LM) height (Y1)	180 μ m=180	108 μ m=180
LM width (X)	150 μ m=150	90 μ m=150
LM area (X \times Y1)	27,000 μ m ² =27k ²	9,720 μ m ² =27k ²
Channel height (Y2)	25 tracks=287 μ m	25 tracks=170 μ m
Channel area per LM (X \times Y2)	43,050 μ m ² =43k ²	15,300 μ m ² =43k ²
LM and routing area (X \times Y1+X \times Y2)	70,000 μ m ² =70k ²	25,000 μ m ² =70k ²
Antifuse capacitance	—	10 fF
Metal capacitance	0.2pFmm ⁻¹	0.2pFmm ⁻¹
Output stub length (spans 3 LMs + 4 channels)	4 channels=1688 μ m	4 channels=1012 μ m
Output stub metal capacitance	0.34pF	0.20pF
Output stub antifuse connections	100	100
Output stub antifuse capacitance	—	1.0pF
Horiz. track length	4–44 cols.= 600–6600 μ m	4–44 cols.= 360–3960 μ m
Horiz. track metal capacitance	0.1–1.3pF	0.07–0.8pF
Horiz. track antifuse connections	52–572 antifuses	52–572 antifuses
Horiz. track antifuse capacitance	—	0.52–5.72 pF
Long vertical track (LVT)	8–14 channels=3760–6580 μ m	8–14 channels=2240–3920 μ m
LVT metal capacitance	0.08–0.13pF	0.45–0.8pF
LVT track antifuse connections	200–350 antifuses	200–350 antifuses
LVT track antifuse capacitance		2–3.5pF
Antifuse resistance (ACT 1)		0.5k (typ.), 0.7k (max.)

Actel interconnect:

An input stub (1 channel) connects to 25 antifuses

An output stub (4 channels) connects to 100 (25×4) antifuses

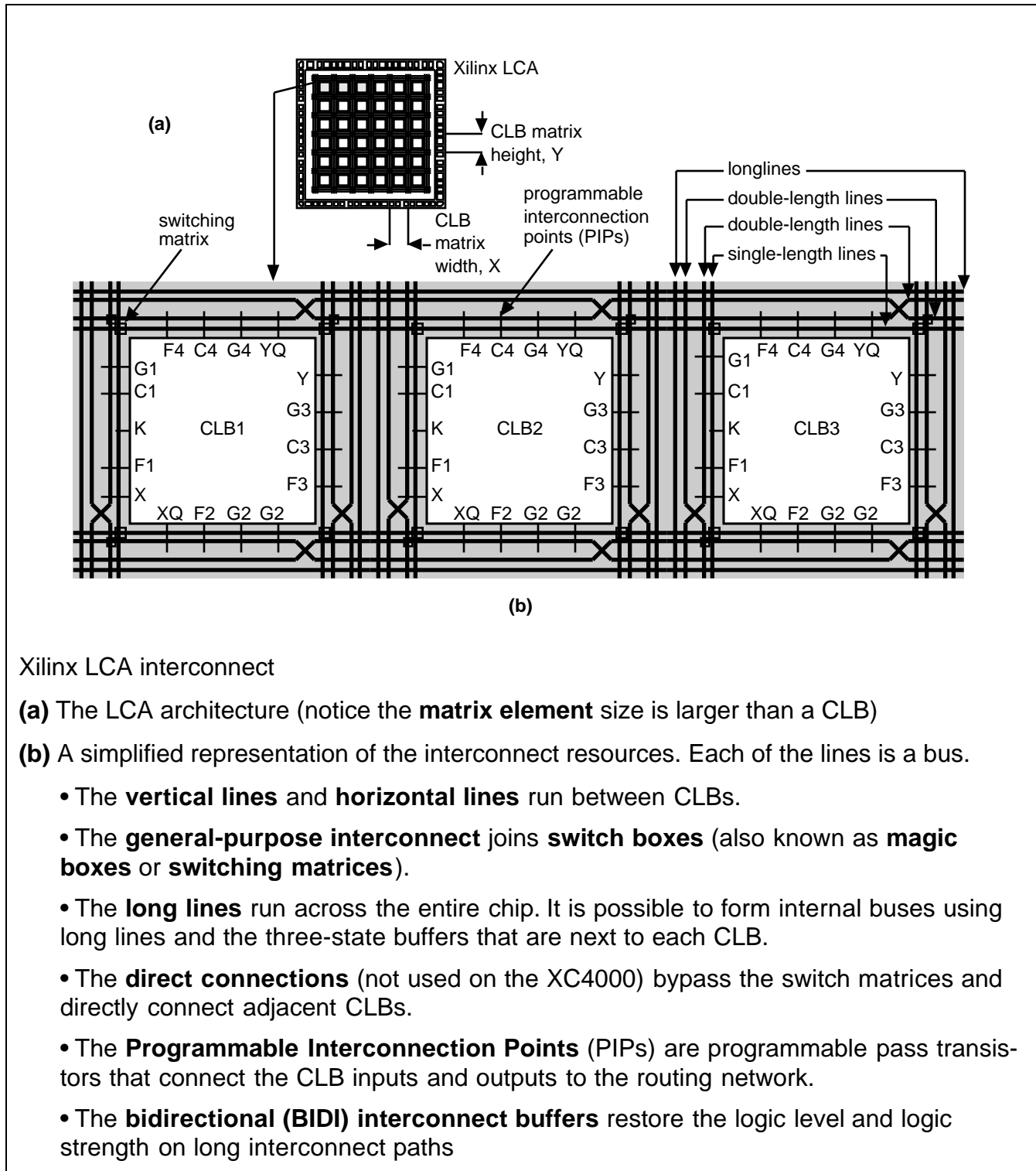
An LVT (1010, 8 channels) connects to 200 (25×8) antifuses

An LVT (1020, 14 channels) connects to 350 (25×14) antifuses

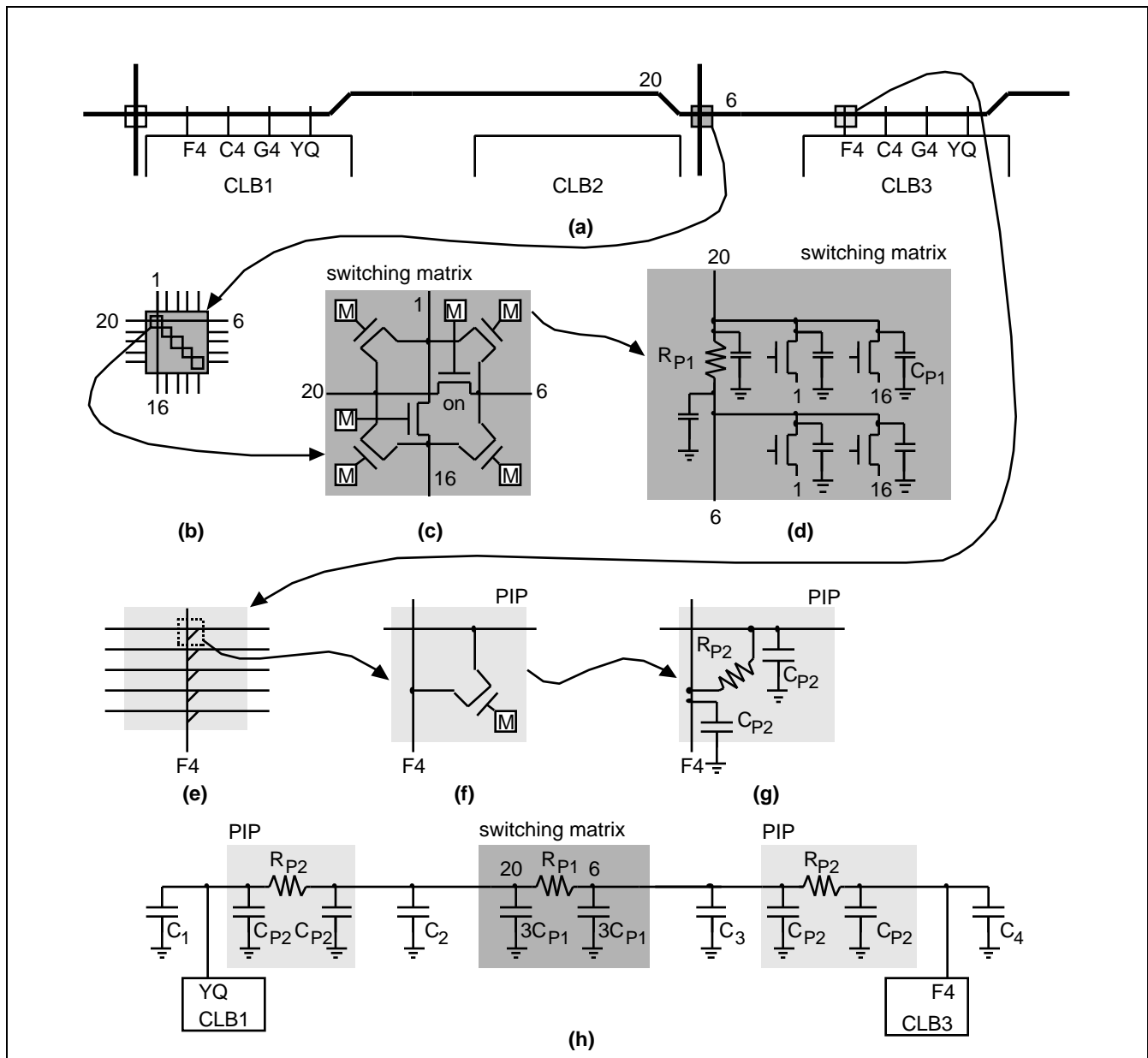
A four-column horizontal track connects to 52 (13×4) antifuses

A 44-column horizontal track connects to 572 (13×44) antifuses

7.2 Xilinx LCA



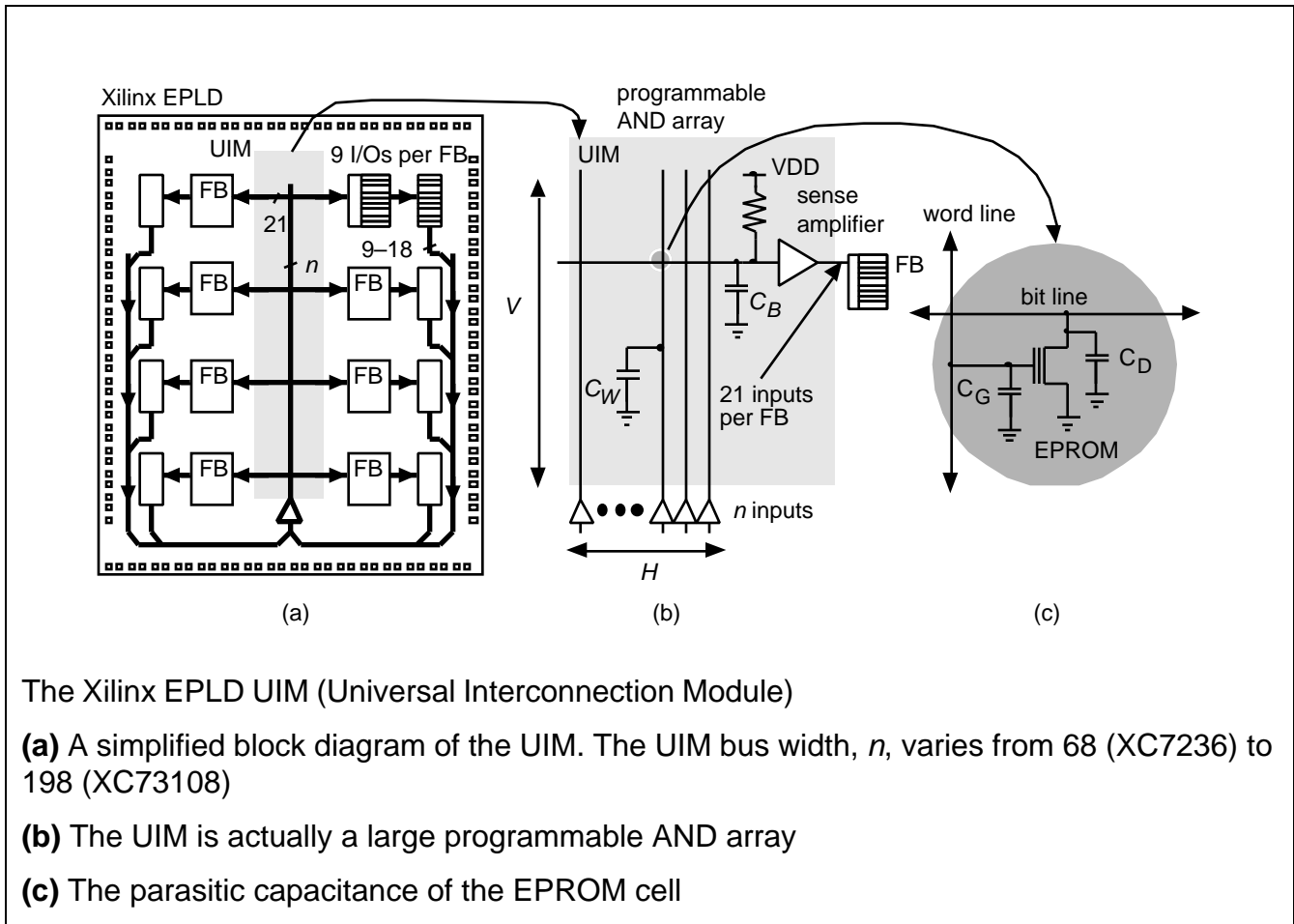
XC3000 interconnect parameters	
Parameter	XC3020
Technology	1.0 μm , =0.5 μm
Die height	220mil
Die width	180mil
Die area	39,600mil ² =102M ²
CLB matrix height (Y)	480 μm =960
CLB matrix width (X)	370 μm =740
CLB matrix area (X \times Y)	17,600 μm^2 =710k ²
Matrix transistor resistance, R _{P1}	0.5–1k
Matrix transistor parasitic capacitance, C _{P1}	0.01–0.02pF
PIP transistor resistance, R _{P2}	0.5–1k
PIP transistor parasitic capacitance, C _{P2}	0.01–0.02pF
Single-length line (X, Y)	370 μm , 480 μm
Single-length line capacitance: C _{LX} , C _{LY}	0.075pF, 0.1pF
Horizontal Longline (8X)	8 cols.=2960 μm
Horizontal Longline metal capacitance, C _{LL}	0.6pF



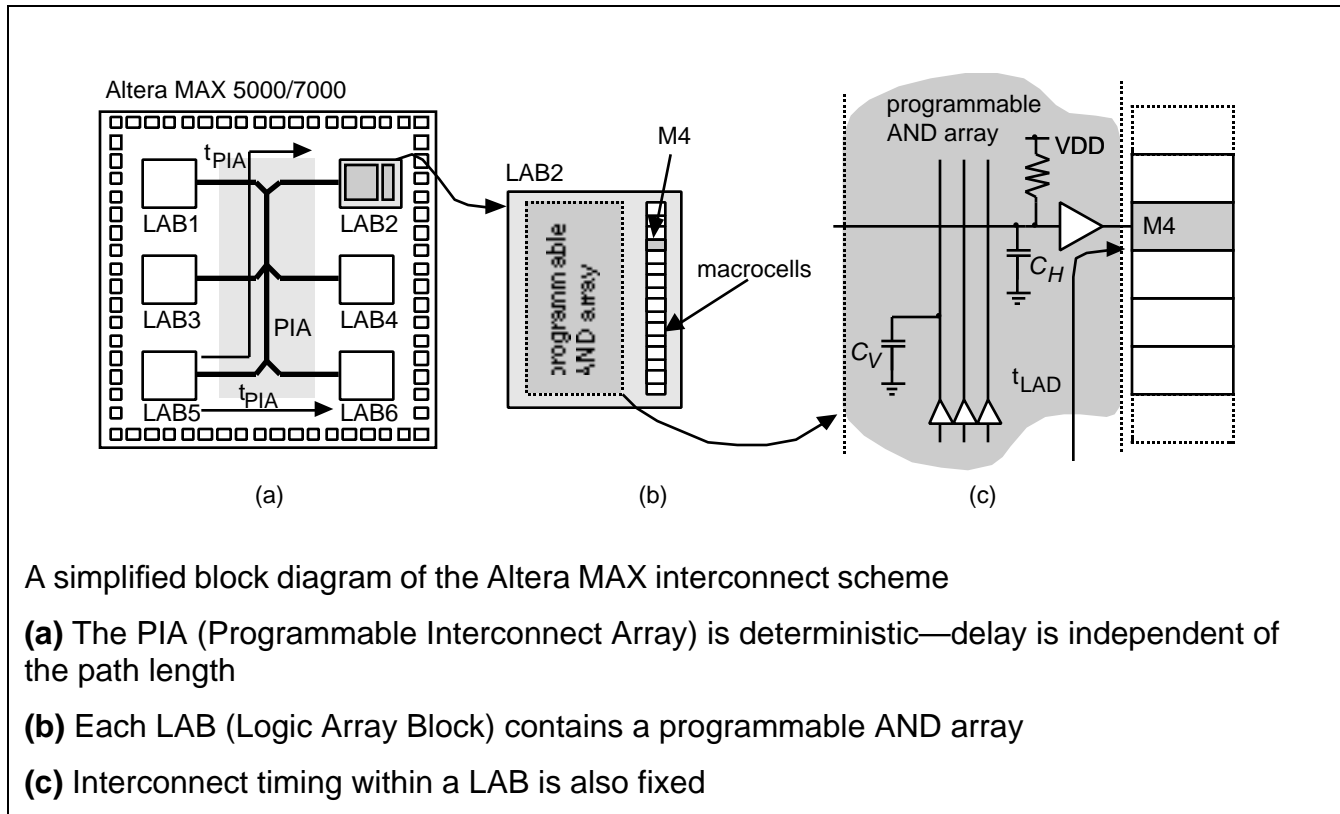
Components of interconnect delay in a Xilinx LCA array

- (a) A portion of the interconnect around the CLBs
- (b) A switching matrix
- (c) A detailed view inside the switching matrix showing the pass-transistor arrangement
- (d) The equivalent circuit for the connection between nets 6 and 20 using the matrix
- (e) A view of the interconnect at a Programmable Interconnection Point (PIP)
- (f) and (g) The equivalent schematic of a PIP connection
- (h) The complete RC delay path

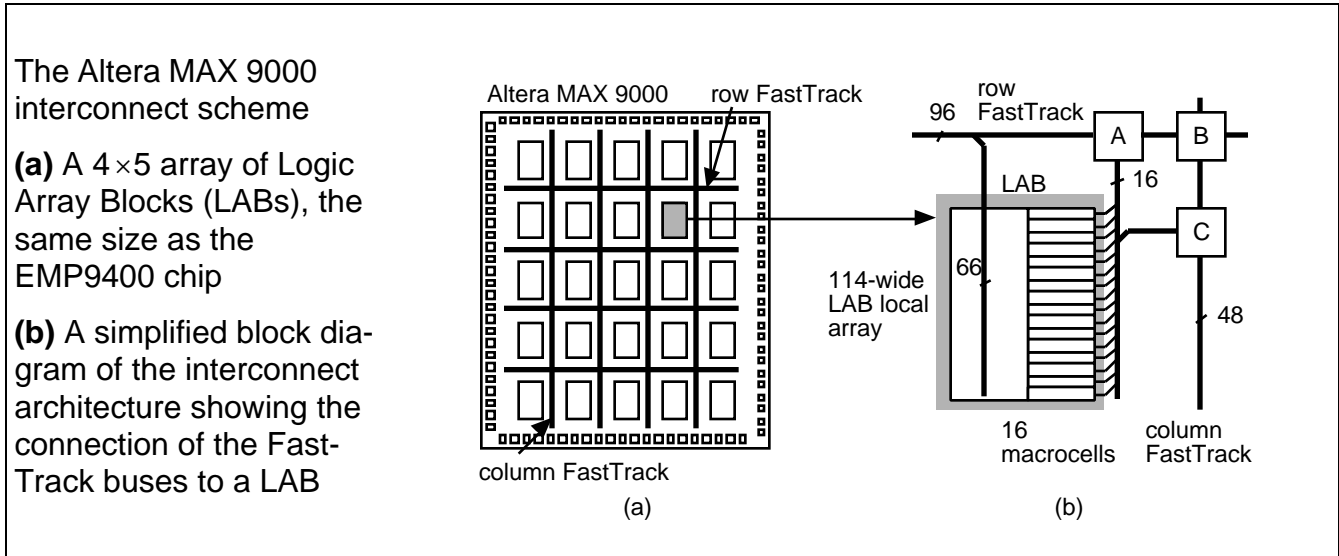
7.3 Xilinx EPLD



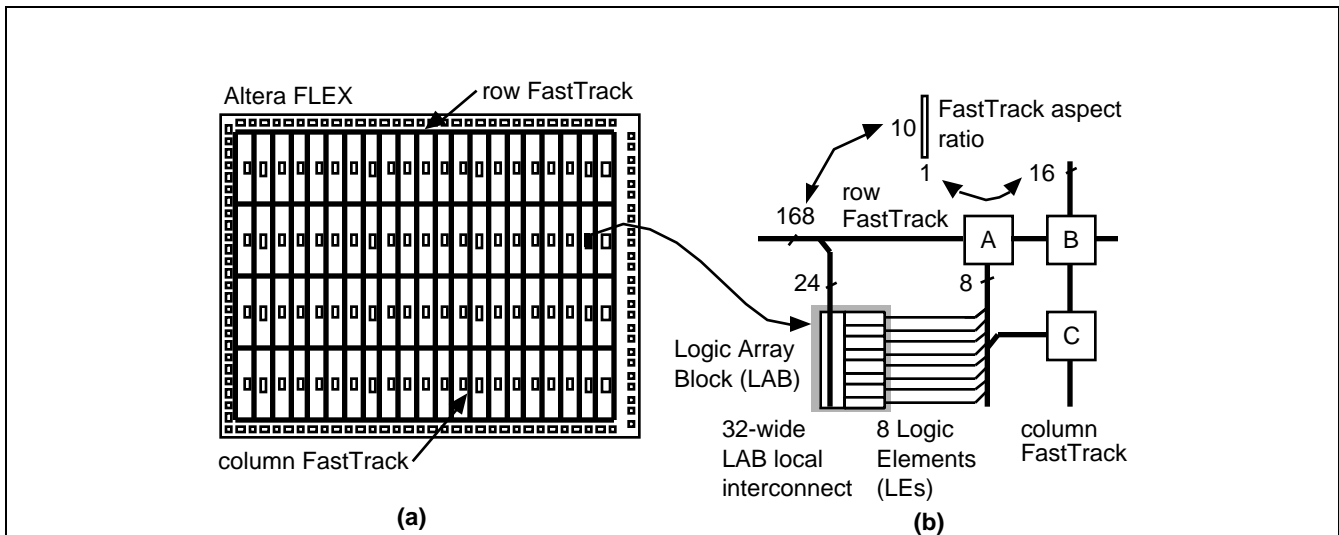
7.4 Altera MAX 5000 and 7000



7.5 Altera MAX 9000



7.6 Altera FLEX



The Altera FLEX interconnect scheme

(a) The row and column FastTrack interconnect. The chip shown, with 4 rows x 21 columns, is the same size as the EPF8820

(b) A simplified diagram of the interconnect architecture showing the connections between the FastTrack buses and a LAB. Boxes A, B, and C represent the bus-to-bus connections

7.7 Summary

The RC product of the parasitic elements of an antifuse and a pass transistor are not too different. However, an SRAM cell is much larger than an antifuse which leads to coarser interconnect architectures for SRAM-based programmable ASICs. The EPROM device lends itself to large wired-logic structures.

These differences in programming technology lead to different architectures:

- The antifuse FPGA architectures are dense and regular.
- The SRAM architectures contain nested structures of interconnect resources.
- The complex PLD architectures use long interconnect lines but achieve deterministic routing.

Key points:

- The difference between deterministic and nondeterministic interconnect
- Estimating interconnect delay
- Elmore's constant

7.8 Problems

PROGRAMMABLE ASIC DESIGN SOFTWARE



Key concepts: There are five components of a programmable ASIC or FPGA :

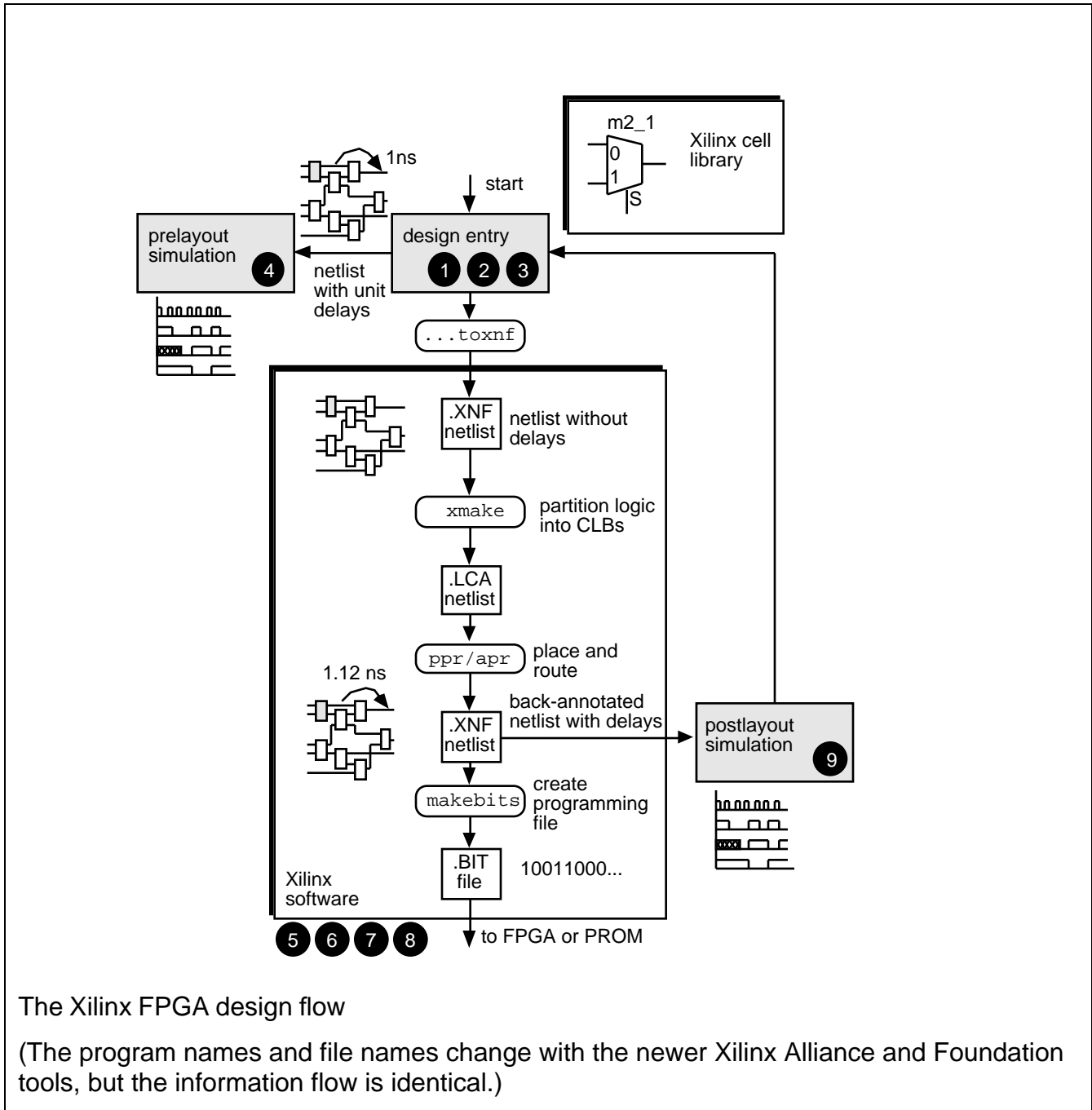
- (1) the programming technology
- (2) the basic logic cell
- (3) the I/O cell
- (4) the interconnect
- (5) the **design software** that allows you to program the ASIC

The design software is much more closely tied to the FPGA architecture than is the case for other types of ASICs

8.1 Design Systems

Keywords: design kits • original equipment manufacturer (OEM) • generic cell library • hardware description languages (HDLs) • ABEL (pronounced “able”) • CUPL (“cupple”) • PALASM (“pal-azzam”) • VHDL • Verilog • logic simulator • back-annotation • postlayout timing information • postlayout netlist (also called a back-annotated netlist) • postlayout timing simulation • timing-analysis • timing constraint • timing violation • forward-annotation

8.1.1 Xilinx



The Xilinx FPGA design flow

(The program names and file names change with the newer Xilinx Alliance and Foundation tools, but the information flow is identical.)

8.1.2 Actel

File types used by Actel design software (an example—these change often)

ADL	Main design netlist
IPF	Partial or complete pin assignment for the design
CRT	Net criticality
VALIDATED	Audit information
COB	List of macros removed from design
VLD	Information, warning, and error messages
PIN	Complete pin assignment for the design
DFR	Information about routability and I/O assignment quality
LOC	Placement of non-I/O macros, pin swapping, and freeway assignment
PLI	Feedback from placement step
SEG	Assignment of horizontal routing segments
STF	Back-annotation timing
RTI	Feedback from routing step
FUS	Fuse coordinates (column-track, row-track)
DEL	Delays for input pins, nets, and I/O modules
AVI	Fuse programming times and currents for last chip programmed

FPGA state-machine language (an example of “third-party” tools)**LOG/iC state-machine language****PALASM version**

```

*IDENTIFICATION
sequence detector
LOG/iC code
*X-NAMES
X; !input
*Y-NAMES
D; !output, D = 1 when three 1's
appear on X
*FLOW-TABLE
;State, X input, Y output, next
state
  S1,  X1,  Y0,  F2;
  S1,  X0,  Y0,  F1;
  S2,  X1,  Y0,  F3;
  S2,  X0,  Y0,  F1;
  S3,  X1,  Y0,  F4;
  S3,  X0,  Y0,  F1;
  S4,  X1,  Y1,  F4;
  S4,  X0,  Y0,  F1;
*STATE-ASSIGNMENT
BINARY;
*RUN-CONTROL
PROGFORMAT = P-EQUATIONS;
*END

```

```

TITLE sequence detector
CHIP MEALY USER
CLK Z QQ2 QQ1 X
EQUATIONS
Z = X * QQ2 * QQ1
QQ2 := X * QQ1 + X * QQ2
QQ1 := X * QQ2 + X * /QQ1

```

8.1.3 Altera

Altera uses a self-contained design system, **MAX+plus** (as well as an interface to EDIF for third-party schematic entry or logic synthesis).

- The interconnect scheme in Altera complex PLDs is nearly **deterministic**, simplifying the physical-design software as well as eliminating the need for back-annotation and a postlayout simulation.
- As Altera FPGAs become larger and more complex, some cases require signals to make more than one pass through the routing structures or travel large distances across the Altera **FastTrack interconnect**. It is possible to tell if this will be the case only by trying to place and route an Altera device.

8.2 Logic Synthesis

It is easier to write $A = B + C$ than to draw an FPGA schematic for a 32-bit adder at the gate level

Key concepts, facts, and terms: logic synthesis • logic minimization • **mapping** • fine-grain architecture • coarse-grain architecture • vendor independence • Synplicity • Synopsys FPGA Express • FPGA Compiler • Design Compiler • Exemplar • X-BLOX • LPM • IP cores

8.2.1 FPGA Synthesis

The VHDL code for a sequence detector

```

entity detector is port (X, CLK: in BIT; Z : out BIT); end;

architecture behave of detector is
  type states is (S1, S2, S3, S4);
  signal current, next: states;
begin
  combinational: process begin
    case current is
      when S1 =>
        if X = '1' then Z <= '0'; next <= S3; else Z <= '0'; next <=
S1; end if;
      when S2 =>
        if X = '1' then Z <= '0'; next <= S2; else Z <= '0'; next <=
S1; end if;
      when S3 =>
        if X = '1' then Z <= '0'; next <= S2; else Z <= '0'; next <=
S1; end if;
      when S4 =>
        if X = '1' then Z <= '1'; next <= S4; else Z <= '0'; next <=
S1; end if
    end case;
  end process
  sequential: process begin
    wait until CLK'event and CLK = '1'; current <= next ;
  end process;
end behave;

```

A Synopsys script

```


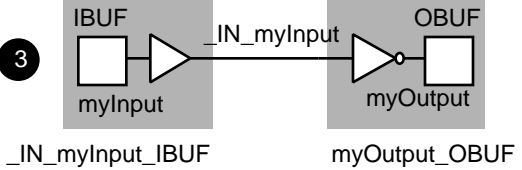
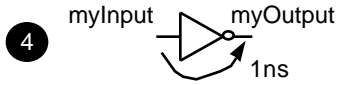
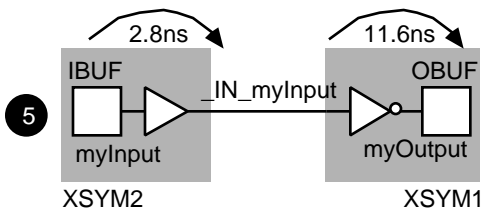
/design checking/
search_path = .
/use the TI cell libraries/
link_library = tpc10.db
target_library = tpc10.db
symbol_library = tpc10.sdb
read -f vhd1 detector.vhd
current_design = detector
write -n -f db -hierarchy -0
detector.db
check_design > detector.rpt

report_design > detector.rpt
/optimize for area/
max_area 0.0
compile
write -h -f db -o detector_opt.db
report -area -cell -timing >
detector.rpt
free -all
/write EDIF netlist/
write -h -f edif -0
exit

```

8.3 The Halfgate ASIC

8.3.1 Xilinx

Design flow for the Xilinx implementation of the halfgate ASIC	Design flow
Script (using Compass tools as an example)	Design flow
<pre> # halfgate.xilinx.inp shell setdef path working xc4000d xblox cmosch000x quit asic open [v]halfgate synthesize save [nls]halfgate_p quit fpga set tag xc4000 set opt area optimize [nls]halfgate_p quit qtv open [nls]halfgate_p trace critical print trace [txt]halfgate_p quit shell vuterm exec xnfmerge -p 4003PC84 halfgate_p > /dev/null exec xnfprep halfgate_p > /dev/null exec ppr halfgate_p > /dev/null exec makebits -w halfgate_p > /dev/null exec lca2xnf -g -v halfgate_p halfgate_b > /dev/null quit manager notice utility netlist open [xnf]halfgate_b save [nls]halfgate_b save [edf]halfgate_b quit qtv open [nls]halfgate_b trace critical print trace [txt]halfgate_b quit </pre>	<p>1 myOutput = ~myInput</p> <p>2 myInput  myOutput</p> <p>3 </p> <p>4 </p> <p>5 </p>

The Xilinx files for the halfgate ASIC**Verilog file** (halfgate.v)

```
module halfgate(myInput, myOutput); input myInput; output
myOutput; wire myOutput;
  assign myOutput = ~myInput;
endmodule
```

Preroute XNF file (halfgate_p.xnf)

```
LCANET, 5                                END
USER, FPGA-Optimizer, 4.1,              EXT, myInput, I,
Date:960710 , Option: Area              SYM,
PROG, FPGA-Optimizer, 4.1,              myOutput_obuf, OBUF, LIBVER=
"Lib=4000"                               2.0.0,
PART, 4010PG191                          PIN, I, I, _IN_myInput,,
PWR, 0, GND                               INV
PWR, 1, VCC                               PIN, O, O, myOutput,
SYM, _IN_myInput_IBUF, IBUF, LIB        END
VER = 2.0.0                               EXT, myOutput, O,
PIN, I, I, myInput,                      EOF
PIN, O, O, _IN_myInput,
```

LCA file (halfgate_p.lca)

```

;: halfgate_p.lca (4003PC84-      Editblk PAD61
4), makebits 5.2.0, Tue Jul 16   Base IO
20:09:43 1996                   Config INFF: I1: I2:I 0:
Version 2                        OUT: PAD: TRI:
Design 4003PC84 4 0             Endblk
Speed -4                         Editblk PAD1
Addnet PAD_myInput PAD61.I2     Base IO
PAD1.0                           Config INFF: I1: I2: 0:
Netdelay PAD_myInput PAD1.0     OUT:0:NOT PAD: TRI:
3.1                               Endblk
Program PAD_myInput {65G521}    Nameblk PAD61 myInput
{65G287} {65G50} {63G50}       Nameblk PAD1 myOutput
{52G50} {45G50}               Intnet myOutput PAD
NProgram PAD_myInput           myOutput
col.B.long.3:PAD1.0           Intnet myInput PAD myInput
col.B.long.3:row.G.local.1     System FGG 0 VERS 2 !
col.B.long.3:row.M.local.5-s   System FGG 1 GD0 0 !
MB.
40.1.14 MB.40.1.35
row.M.local.5:PAD61.I2

```

Postroute XNF file (halfgate_b.xnf)

```

LCANET, 4                        EXT, myOutput, 0, 10
PROG, LCA2XNF, 5.2.0, "COMMAND  EXT, myInput, I, 29
= -g -v halfgate_p halfgate_b   EOF
TIME = Tue Jul 16 21:53:31
1996"
PART, 4003PC84-4
SYM, XSYM1, OBUF, SLOW
    PIN, 0, 0, myOutput, 3.0
    PIN, I, I, _IN_myInput,
8.6, INV
END
SYM, XSYM2, IBUF
    PIN, 0, 0, _IN_myInput,
2.8
    PIN, I, I, myInput
END

```

8.3.2 Actel

The Actel files for the halfgate ASIC	
ADL file	STF file
<pre> ; HEADER ; FILEID ADL ./halfgate_io.adl 85e8053b ; CHECKSUM 85e8053b ; PROGRAM certify ; VERSION 23/1 ; ALSMAJORREV 2 ; ALSMINORREV 3 ; ALSPATCHREV .1 ; NODEID 72705192 ; VAR FAMILY 1400 ; ENDHEADER DEF halfgate_io; myInput, myOutput. USE ADLIB:INBUF; INBUF_2. USE ADLIB:OUTBUF; OUTBUF_3. USE ADLIB:INV; u2. NET DEF_NET_8; u2:A, INBUF_2:Y. NET DEF_NET_9; myInput, INBUF_2:PAD. NET DEF_NET_11; OUTBUF_3:D, u2:Y. NET DEF_NET_12; myOutput, OUTBUF_3:PAD. END. </pre>	<pre> ; HEADER ; FILEID STF ./halfgate_io.stf c96ef4d8 ... lines omitted ... (126 lines total) DEF halfgate_io. USE ; INBUF_2/U0; TPADH:'11:26:37', TPADL:'13:30:41', TPADE:'12:29:41', TPADD:'20:48:70', TYH:'8:20:27', TYL:'12:28:39'. PIN u2:A; RDEL:'13:31:42', FDEL:'11:26:37'. USE ; OUTBUF_3/U0; TPADH:'11:26:37', TPADL:'13:30:41', TPADE:'12:29:41', TPADD:'20:48:70', TYH:'8:20:27', TYL:'12:28:39'. PIN OUTBUF_3/U0:D; RDEL:'14:32:45', FDEL:'11:26:37'. END. </pre>

8.3.3 Altera

EDIF netlist in Altera format for the halfgate ASIC

```

(edif halfgate_p          (direction          (portRef
(edifVersion 2 0 0) OUTPUT)) myInput)
(edifLevel 0)            (designator          (portRef IN
(keywordMap              "@@Label"))))      (instanceRef
(keywordLevel 0))      (library working    B1_i1)))
(status                  (edifLevel 0)      (net myOutput
(written                 (technology        (joined
(timestamp 1996 7       (numberDefinition (portRef
10 23 55 8)            ) myOutput)
(program "COMPASS      (simulationInfo     (portRef OUT
Design Automation --   (logicValue H)      (instanceRef
EDIF Interface"       (logicValue         B1_i1))))
(version "v9r1.2 L)) (net VDD
last updated 26-Mar-   (cell halfgate_p   (joined )
96"))                 (cellType          (property
(author                GENERIC)          global
"mikes"))             (view              (string
(library flex8kd      COMPASS_nls_view   "vcc")))
(edifLevel 0)         (viewType          (net VSS
(technology           NETLIST)            (joined )
(numberDefinition     (interface          (property
)                       (port myInput    global
(simulationInfo       (direction          (string
(logicValue H) INPUT)) "gnd"))))))
(logicValue           (port myOutput   (design halfgate_p
L))                   (direction       (cellRef halfgate_p
(cell not             OUTPUT))          (libraryRef
(cellType             (designator      working))))
GENERIC)              "@@Label"))
(view                 (contents
COMPASS_mde_view     (instance B1_i1
(viewType            (viewRef
NETLIST)             COMPASS_mde_view
(interface           (cellRef not
(port IN              (libraryRef
(direction           flex8kd))))
INPUT))              (net myInput
(port OUT            (joined

```

Report for the halfgate ASIC fitted to an Altera MAX 7000 complex PLD

** INPUTS **

Pin	LC	LAB	Primitive	Code	Shareable		Fan-In		Fan-Out		Name	
					Total	Shared	n/a	INP	FBK	OUT		FBK
43	-	-	INPUT		0	0	0	0	0	0	1	myInput

** OUTPUTS **

Pin	LC	LAB	Primitive	Code	Shareable		Fan-In		Fan-Out		Name	
					Total	Shared	n/a	INP	FBK	OUT		FBK
41	17	B	OUTPUT	t	0	0	0	1	0	0	0	myOutput

** LOGIC CELL INTERCONNECTIONS **

Logic Array Block 'B':

```

      +- LC17 myOutput
      |
LC    | | A B | Name

```

Pin

```

43   -> * | - * | myInput

```

* = The logic cell or pin is an input to the logic cell (or LAB) through the PIA.

- = The logic cell or pin is not an input to the logic cell (or LAB).

The structural postlayout files generated by the Altera MAX+plus software:

```

// halfgate_p (EPM7032LC44) MAX+plus II Version 5.1 RC6 10/03/94
// Wed Jul 17 04:07:10 1996
`timescale 100 ps / 100 ps

```

```

module TRI_halfgate_p( IN, OE, OUT );input IN; input OE; output OUT;
bufif1 ( OUT, IN, OE );
specify
  specparam TTRI = 40; specparam TTXZ = 60; specparam TTZX = 60;
  (IN => OUT) = (TTRI,TTRI);
  (OE => OUT) = (0,0, TTXZ, TTZX, TTXZ, TTZX);
endspecify
endmodule

```

```

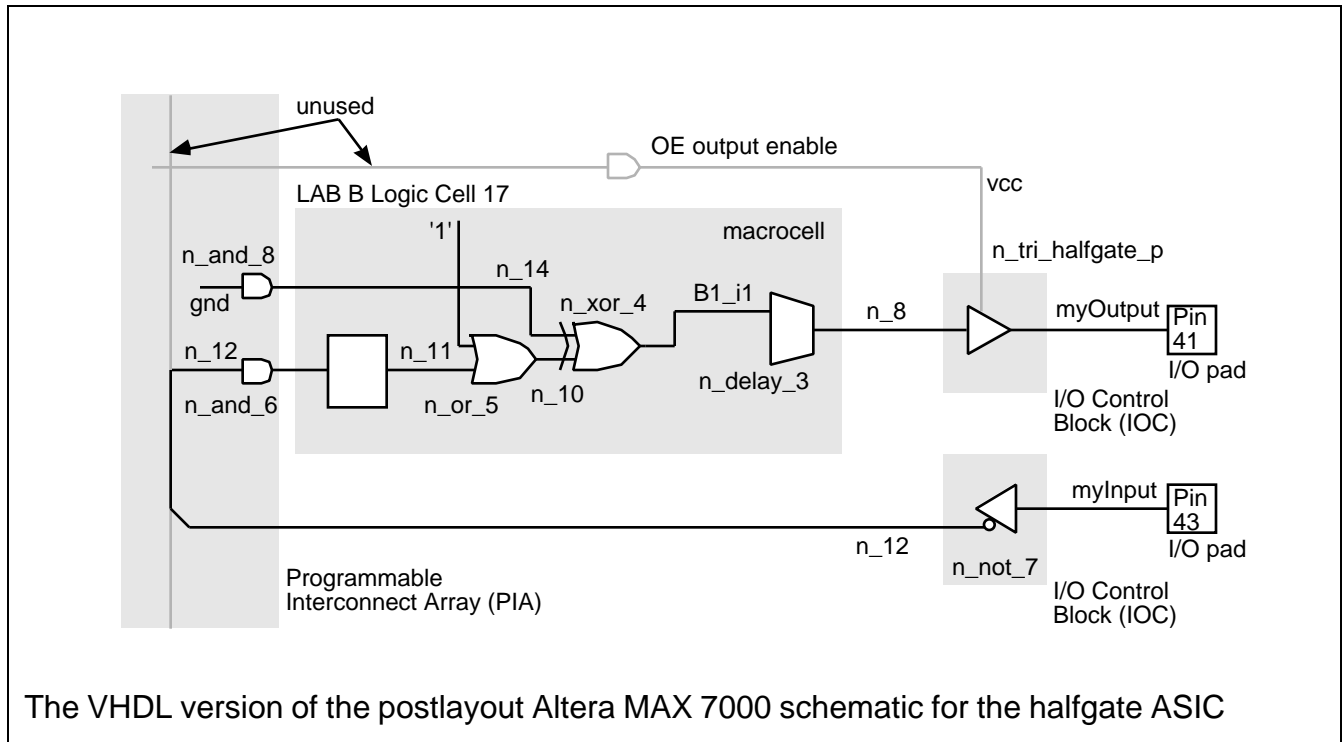
module halfgate_p (myInput, myOutput);
  input myInput; output myOutput; supply0 gnd; supply1 vcc;
  wire B1_i1, myInput, myOutput, N_8, N_10, N_11, N_12, N_14;
  TRI_halfgate_p tri_2 ( .OUT(myOutput), .IN(N_8), .OE(vcc) );
  TRANSPORT transport_3 ( N_8, N_8_A );
  defparam transport_3.DELAY = 10;
  and delay_3 ( N_8_A, B1_i1 );

```



```
xor xor2_4 ( B1_i1, N_10, N_14 );
or or1_5 ( N_10, N_11 );
TRANSPORT transport_6 ( N_11, N_11_A );
defparam transport_6.DELAY = 60;
and and1_6 ( N_11_A, N_12 );
TRANSPORT transport_7 ( N_12, N_12_A );
defparam transport_7.DELAY = 40;
not not_7 ( N_12_A, myInput );
TRANSPORT transport_8 ( N_14, N_14_A );
defparam transport_8.DELAY = 60;
and and1_8 ( N_14_A, gnd );
endmodule
```

```
// MAX+plus II Version 5.1 RC6 10/03/94 Wed Jul 17 04:07:10 1996
`timescale 100 ps / 100 ps
module TRANSPORT( OUT, IN ); input IN; output OUT; reg OUTR;
wire OUT = OUTR; parameter DELAY = 0;
`ifdef ZeroDelaySim
    always @IN OUTR <= IN;
`else
    always @IN OUTR <= #DELAY IN;
`endif
`ifdef Silos
    initial #0 OUTR = IN;
`endif
endmodule
```



The VHDL version of the postlayout Altera MAX 7000 schematic for the halfgate ASIC

8.3.4 Comparison

- Xilinx XC4000, a nondeterministic coarse-grained FPGA
- Actel ACT 3, a nondeterministic fine-grained FPGA
- Altera MAX 7000, a deterministic complex PLD

The differences:

1. The Xilinx LCA architecture does not permit an accurate timing analysis until after place and route. This is because of the coarse-grained nondeterministic architecture.
2. The Actel ACT architecture is nondeterministic, but the fine-grained structure allows fairly accurate preroute timing prediction.
3. The Altera MAX CPLD requires logic to be fitted to the product steering and programmable array logic. The Altera MAX 7000 has an almost deterministic architecture, which allows accurate preroute timing.

8.4 Summary

Key concepts:

- FPGA design flow: design entry, simulation, physical design, and programming
- Schematic entry, hardware design languages, logic synthesis
- PALASM as a common low-level hardware description
- EDIF, Verilog, and VHDL as vendor-independent netlist standards

LOW-LEVEL DESIGN ENTRY

9

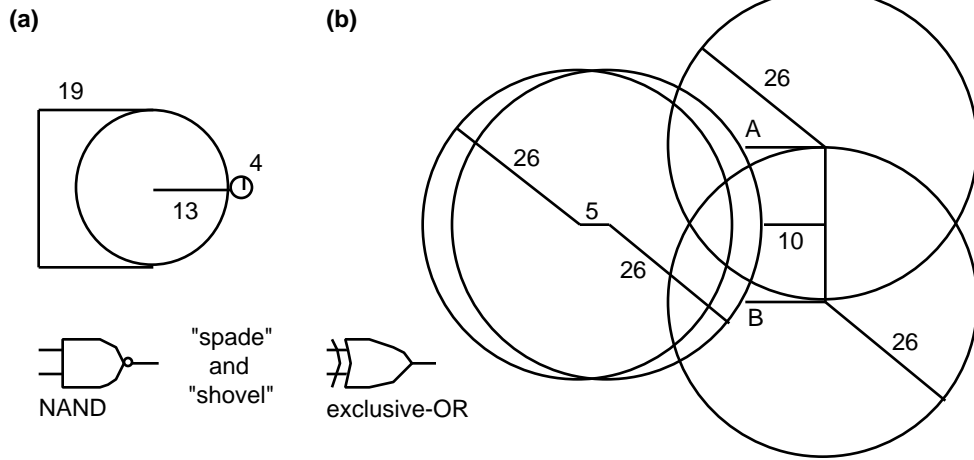
Key concepts: design entry • electronic-design automation (EDA) • schematic • connectivity • schematic entry • schematic capture • netlist • documentation • hardware description language (HDL) • logic synthesis • low-level design-entry

9.1 Schematic Entry

Key terms and concepts: graphical design entry • transforms an idea to a computer file • an “old” method that periodically regains popularity • schematic sheets • frame • border • “spades” and “shovels” • component or device • low-cost

ANSI (American National Standards Institute) and ISO (International Standards Organization) schematic sheet sizes

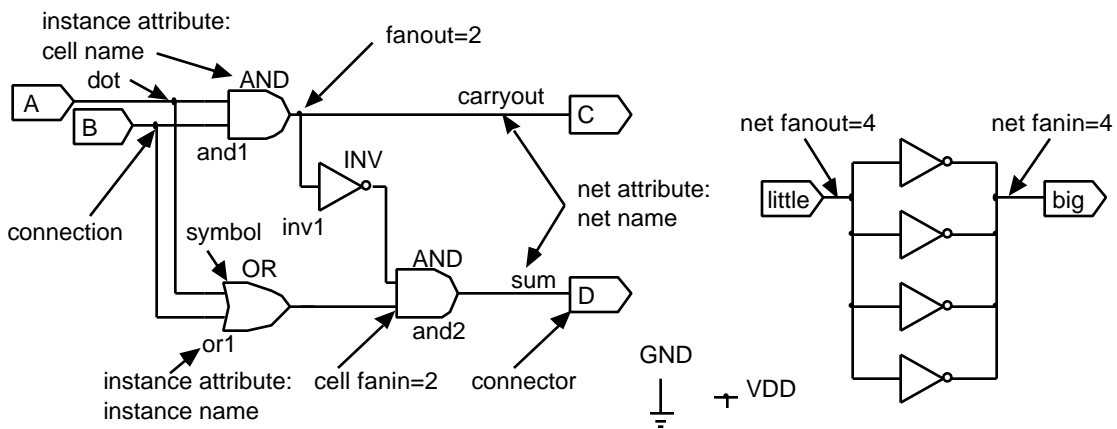
ANSI sheet	Size (inches)	ISO sheet	Size (cm)
A	8.5 × 11	A5	21.0 × 14.8
B	11 × 17	A4	29.7 × 21.0
C	17 × 22	A3	42.0 × 29.7
D	22 × 34	A2	59.4 × 42.0
E	34 × 44	A1	84.0 × 59.4
		A0	118.9 × 84.0



IEEE-recommended dimensions and their construction for logic-gate symbols

(a) NAND gate

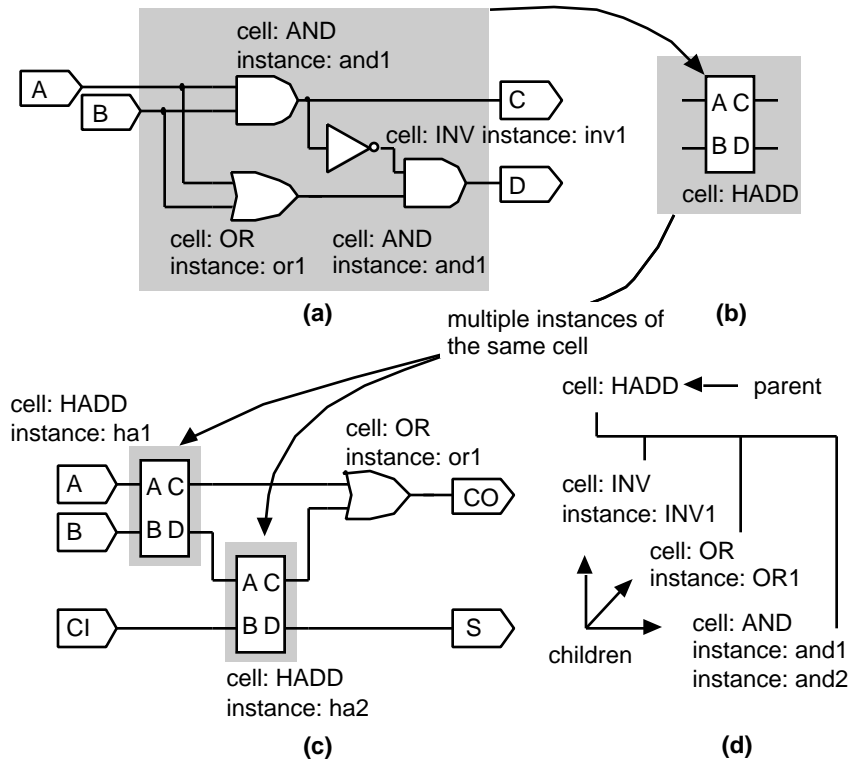
(b) exclusive-OR gate (an OR gate is a subset)



Terms used in circuit schematics

9.1.1 Hierarchical Design

Key terms and concepts: use of hierarchy to hide complexity • hierarchical design • subschematic • child • parent • flat design • flat netlist



Schematic example showing hierarchical design

- (a) The schematic of a half-adder, the subschematic of cell HADD
- (b) A schematic symbol for the half adder
- (c) A schematic that uses the half-adder cell
- (d) The hierarchy of cell HADD

9.1.2 The Cell Library

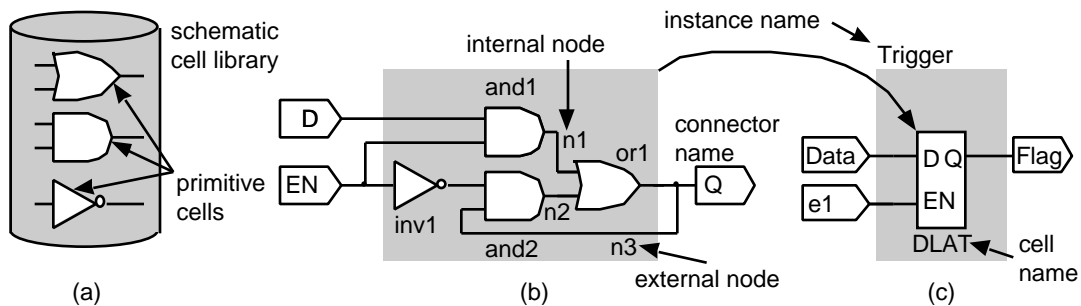
Key terms: modules (cells, gates, macros, books) • schematic library (vendor-dependent) • retargeting • porting a design • primitive cells or cells (flip-flops or transistors?) • hard macro (placement) • soft macro (connection)

9.1.3 Names

Key terms: cell name • cell instance • instance name • icon (picture) • symbol • name spaces • case sensitivity • hierarchical names

9.1.4 Schematic Icons and Symbols

Key terms: derived icon • derived symbol • subcell • vectored instance • cardinality

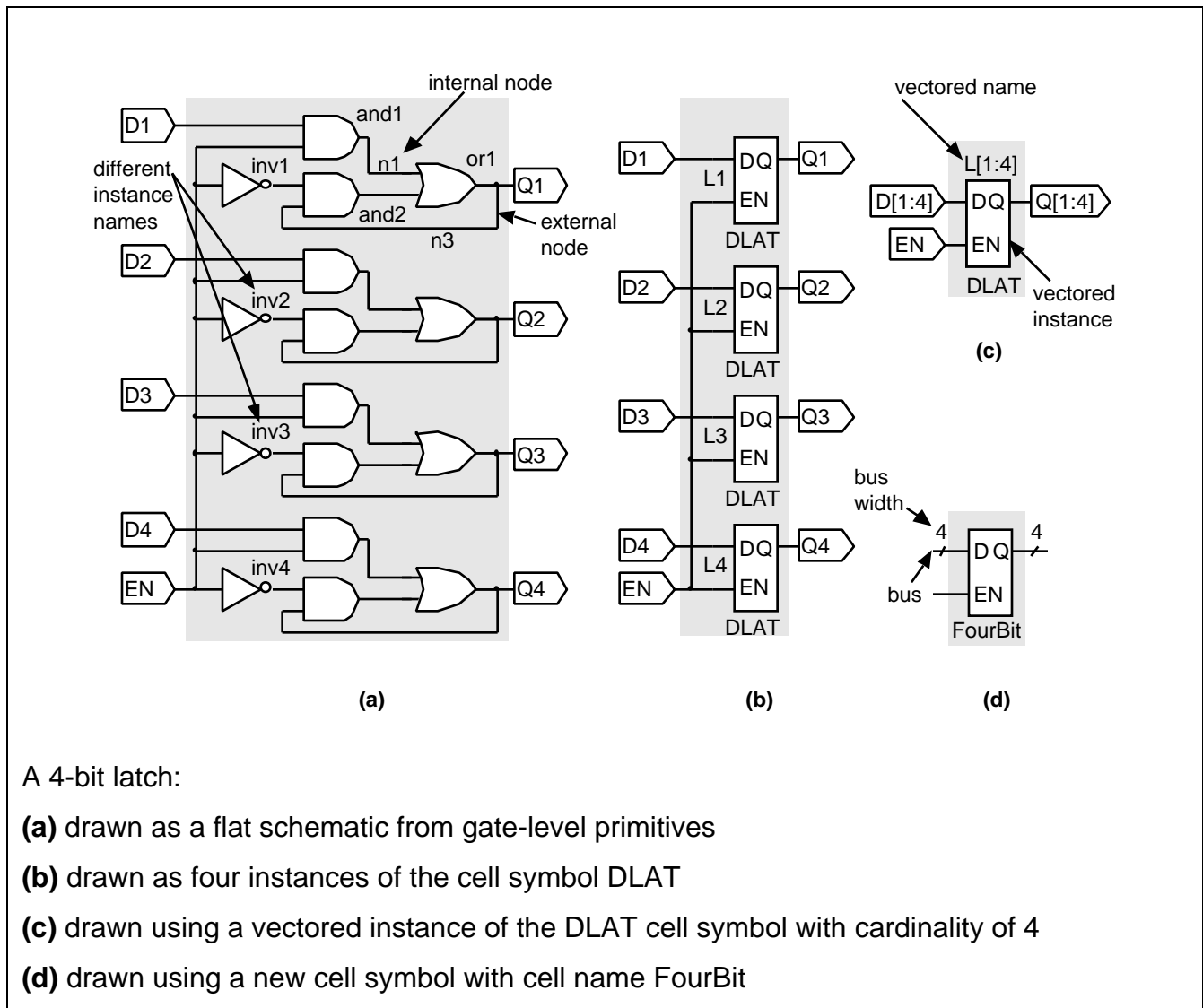


A cell and its subschematic

(a) A schematic library containing icons for the primitive cells

(b) A subschematic for a cell, DLAT, showing the instance names for the primitive cells

(c) A symbol for cell DLAT



9.1.5 Nets

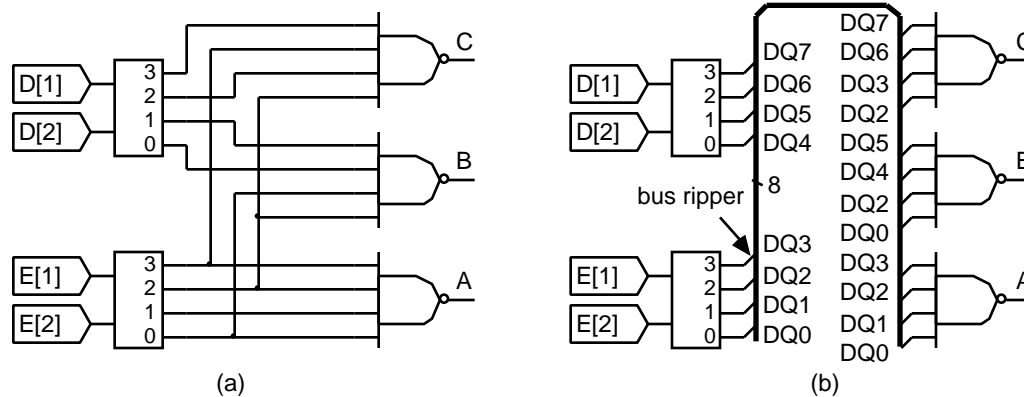
Key terms: local nets • external nets • delimiter • Verilog and VHDL naming

9.1.6 Schematic Entry for ASICs and PCBs

Key terms: component • TTL SN74LS00N • Quad 2-input NAND • component parts • reference designator • R99 • pin number • part assignment

9.1.7 Connections

Key terms: terminals • pins, connectors, or signals • wire segments or nets • bus or buses (not busses) • bundle or array • breakout • ripper (EDIF) • extractor • swizzle (Compass datapath)

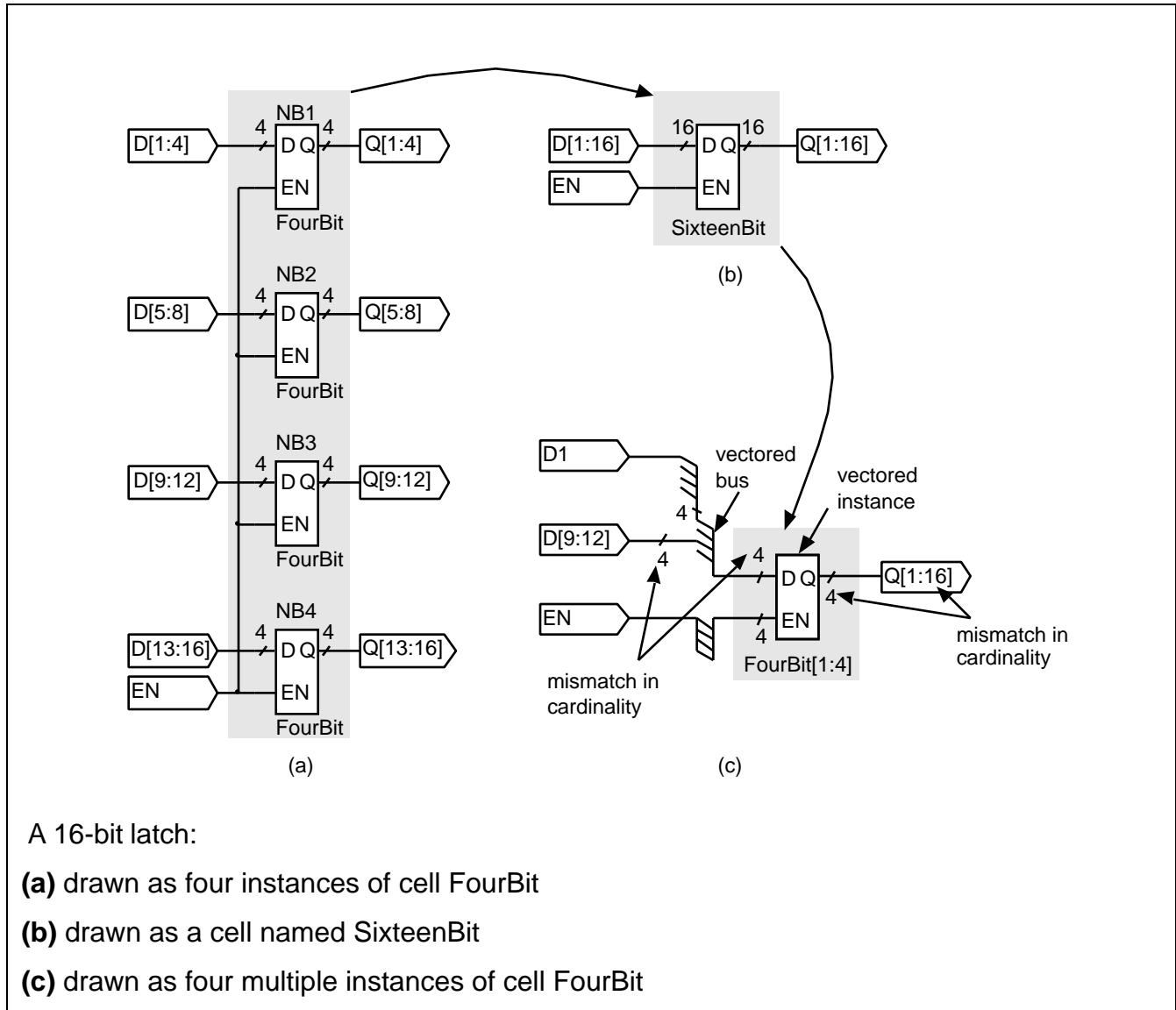


An example of the use of a bus to simplify a schematic

(a) An address decoder without using a bus

(b) A bus with bus rippers simplifies the schematic and reduces the possibility of making a mistake in creating and reading the schematic

9.1.8 Vectored Instances and Buses



9.1.9 Edit-in-Place

Key terms: edit-in-place • alias • dictionary of names

9.1.10 Attributes

Key terms: name • identifier • label • attribute • property • NFS filenames (28 characters)

9.1.11 Netlist Screener

Key terms: schematic or netlist screener catches errors at an early stage • handle (to find components) • snap to grid • wildcard matching • automatic naming • datapath (multiple instances) • vectored cell instance • vectored instance • cell cardinality • cardinality • terminal polarity • terminal direction • fanout • fanin • standard load

9.1.12 Schematic-Entry Tools

Key terms: icon edit-in-place • timestamp or datestamp • versions • version number • design manager or library manager • version history • check-out • undo • rubber banding • global nets • connectors • off-page connector • multipage connector • fanout • fanin • standard load

9.1.13 Back-Annotation

Key terms: logical design • prelayout simulation • physical design • parasitic capacitance • interconnect delay • back-annotation • postlayout simulation

9.2 Low-Level Design Languages

Key terms and concepts: changes to a schematic are tedious • no standards for schematics
 • PLD design entry • a design language is better than schematic entry • a low-level design language is not as powerful as logic synthesis • legacy code

9.2.1 ABEL

ABEL

Statement	Example	Comment																		
Module	<code>module MyModule</code>	You can have multiple modules.																		
Title	<code>title 'Title in a String'</code>	A string is a character series between quotes.																		
Device	<code>MYDEV device '22V10' ;</code>	MYDEV is Device ID for documentation. 22V10 is checked by the compiler.																		
Comment	<code>"comments go between double quotes"</code> <code>"end of line is end of comment"</code>	The end of a line signifies the end of a comment; there is no need for an end quote.																		
@ALTER-NATE	@ALTERNATE "use alternate symbols	<table border="1"> <thead> <tr> <th>operator</th> <th>alternate</th> <th>default</th> </tr> </thead> <tbody> <tr> <td>AND</td> <td>*</td> <td>&</td> </tr> <tr> <td>OR</td> <td>+</td> <td>#</td> </tr> <tr> <td>NOT</td> <td>/</td> <td>!</td> </tr> <tr> <td>XOR</td> <td>:+:</td> <td>\$</td> </tr> <tr> <td>XNOR</td> <td>:*:</td> <td>!\$</td> </tr> </tbody> </table>	operator	alternate	default	AND	*	&	OR	+	#	NOT	/	!	XOR	:+:	\$	XNOR	:*:	!\$
operator	alternate	default																		
AND	*	&																		
OR	+	#																		
NOT	/	!																		
XOR	:+:	\$																		
XNOR	:*:	!\$																		
Pin declaration	<code>MYINPUT pin 2; I3, I4 pin 3, 4 ;</code> <code>/MYOUTPUT pin 22; IO3,IO4 pin 21,20 ;</code>	Pin 22 is the IO for input on pin 2 for a 22V10. MYOUTPUT is active-low at the chip pin. Signal names must start with a letter.																		
Equations	<code>equations</code> <code>IO4 = HELPER ; HELPER = /I4 ;</code>	Defines combinational logic. Two-pass logic																		
Assignments	<code>MYOUTPUT = /MYINPUT ;</code>	Equals '=' is unlocked assignment.																		

	<code>IO3 := I4 ;</code>	Clocked assignment operator (registered IO)
Signal sets	<code>D = [D0, D1, D2, D3] ;</code> <code>Q = [Q0, Q1, Q2, Q3];</code>	A signal set, an ABEL bus
Suffix	<code>Q := D ;</code> <code>MYOUTPUT.RE = CLR ;</code> <code>MYOUTPUT.PR = PRE ;</code>	4-bit-wide register Register reset Register preset
Addition	<code>COUNT = [D0, D1, D2];</code> <code>COUNT := COUNT + 1;</code>	Can't use @ALTERNATE if you use '+' to add.
Enable	<code>ENABLE IO3 = IO2;</code> <code>IO3 = MYINPUT;</code>	Three-state enable (ENABLE is a keyword). IO3 must be a three-state pin.
Constants	<code>K = [1, 0, 1] ;</code>	K is 5.
Relational	<code>IO# = D == K5 ;</code>	Operators: == != < > <= >=
End	<code>end MyModule</code>	Last statement in module

Example:

```

module MUX4
title '4:1 MUX'
MyDevice device 'P16L8' ;
@ALTERNATE
"inputs
A, B, /P1G1, /P1G2 pin 17,18,1,6 "LS153 pins 14,2,1,15
P1C0, P1C1, P1C2, P1C3 pin 2,3,4,5 "LS153 pins 6,5,4,3
P2C0, P2C1, P2C2, P2C3 pin 7,8,9,11 "LS153 pins 10,11,12,13
"outputs
P1Y, P2Y pin 19, 12 "LS153 pins 7,9
equations
  P1Y = P1G*(/B*/A*P1C0 + /B*A*P1C1 + B*/A*P1C2 + B*A*P1C3);
  P2Y = P1G*(/B*/A*P1C0 + /B*A*P1C1 + B*/A*P1C2 + B*A*P1C3);
end MUX4

```

9.2.2 CUPL

Key terms and concepts: CUPL is a PLD design language from Logical Devices • CUPL 4.0 extension • fitter • Atmel ATV2500B • complex PLD • “buried” features • pin-number tables • skeleton headers and pin declarations

```
SEQUENCE BayBridgeTollPlaza {
  PRESENT red
    IF car NEXT green OUT go; /* conditional synchronous output */
    DEFAULT NEXT red; /* default next state */
  PRESENT green
    NEXT red; /* unconditional next state */
}
```

CUPL statements for state-machine entry

Statement		Description
IF	NEXT	Conditional next state transition
IF	NEXT OUT	Conditional next state transition with synchronous output
	NEXT	Unconditional next state transition
	NEXT OUT	Unconditional next state transition with asynchronous output
	OUT	Unconditional asynchronous output
IF	OUT	Conditional asynchronous output
DEFAULT	NEXT	Default next state transition
DEFAULT	OUT	Default asynchronous output
DEFAULT	NEXT OUT	Default next state transition with synchronous output

You may encode state machines as truth tables in CUPL:

```
FIELD input = [in1..0];
FIELD output = [out3..0];
TABLE input => output {00 => 01; 01 => 02; 10 => 04; 11 => 08; }
```

CUPL file for a 4-bit counter (for an ATMEL PLD) that illustrates extensions:

```
Name 4BIT; Device V2500B;
/* inputs */
```

```
pin 1 = CLK; pin 3 = LD_; pin 17 = RST_;
pin [18,19,20,21] = [I0,I1,I2,I3];
/* outputs */
pin [4,5,6,7] = [Q0,Q1,Q2,Q3];
field CNT = [Q3,Q2,Q1,Q0];
/* equations */
Q3.T = (!Q2 & !Q1 & !Q0) & LD_ & RST_ /* count down */
      # Q3 & !RST_ /* ReSeT */
      # (Q3 $ I3) & !LD_; /* Load*/
Q2.T = (!Q1 & !Q0) & LD_ & RST_ # Q2 & !RST_ # (Q2 $ I2) & !LD_;
Q1.T = !Q0 & LD_ & RST_ # Q1 & !RST_ # (Q1 $ I1) & !LD_;
Q0.T = LD_ & RST_ # Q0 & !RST_ # (Q0 $ I0) & !LD_;
CNT.CK = CLK; CNT.OE = 'h'F; CNT.AR = 'h'0; CNT.SP = 'h'0;
```

CUPL extensions guide the **logic fitter**, for example:

```
output.ext = (Boolean expression);
```

.OE is output enable

.CK marks the clock

.T configures sequential logic as T flip-flops

.OE (wired high) is an output enable

.AR (wired low) is an asynchronous reset

.SP (wired low) is an synchronous preset

CUPL 4.0 extensions

Extension	Explanation	Extension	Explanation
D	L D input to a D register	DFB	R D register feedback of combinational output
L	L L input to a latch	LFB	R Latched feedback of combinational output
J, K	L J-K-input to a J-K register	TFB	R T register feedback of combinational output
S, R	L S-R input to an S-R register	INT	R Internal feedback
T	L T input to a T register	IO	R Pin feedback of registered output
DQ	R D output of an input D register	IOD/T	R D/T register on pin feedback path selection
LQ	R Q output of an input latch	IOL	R Latch on pin feedback path selection
AP, AR	L Asynchronous preset/reset	IOAP, IOAR	L Asynchronous preset/reset of register on feedback path
SP, SR	L Synchronous preset/reset	IOSP, IOSR	L Synchronous preset/reset of register on feedback path
CK	L Product clock term (async.)	IOCK	L Clock for pin feedback register
OE	L Product-term output enable	APMUX, ARMUX	L Asynchronous preset/reset multiplexor selection
CA	L Complement array	CKMUX	L Clock multiplexor selector
PR	L Programmable preload	LEMUX	L Latch enable multiplexor selector
CE	L CE input of a D-CE register	OEMUX	L Output enable multiplexor selector
LE	L Product-term latch enable	IMUX	L Input multiplexor selector of two pins
OBS	L Programmable observability of buried nodes	TEC	L Technology-dependent fuse selection
BYP	L Programmable register bypass	T1	L T1 input of 2-T register

ABEL and CUPL pin declarations for an ATMEL ATV2500B

ABEL	CUPL
device_id device 'P2500B';	
"device_id used for JEDEC	
filename	device V2500B;
I1,I2,I3,I17,I18 pin 1,2,3,17,18;	pin [1,2,3,17,18] =
O4,O5 pin 4,5 istype	[I1,I2,I3,I17,I18];
'reg_d,buffer';	pin [7,6,5,4] = [O7,O6,O5,O4];
O6,O7 pin 6,7 istype 'com';	pinnode [41,65,44] =
O4Q2,O7Q2 node 41,44 istype	[O4Q2,O4Q1,O7Q2];
'reg_d';	pinnode [43,68] = [O6Q2,O7Q1];
O6F2 node 43 istype 'com';	
O7Q1 node 220 istype 'reg_d';	

9.2.3 PALASM

Key terms and concepts: PALASM is a PLD design language from AMD/MMI • PALASM 2 • ordering of the pin numbers is important • DEVICE • often need manufacturer's data sheet

PALASM 2

Statement	Example	Comment
Chip	CHIP abc 22V10	Specific PAL type
	CHIP xyz USER	Free-form equation entry
Pinlist	CLK /LD D0 D1 D2 D3 D4 GND NC Q4 Q3 Q2 Q1 Q0 /RST VCC	Part of CHIP statement; PAL pins in numerical order starting with pin 1
String	STRING string_name 'text'	Before EQUATIONS statement
Equations	EQUATIONS	After CHIP statement
	A = /B	Logical negation
	A = B * C	Logical AND
	A = B + C	Logical OR
	A = B :+ : C	Logical exclusive-OR
	A = B :* : C	Logical exclusive-NOR
Polarity inversion	/A = /(B + C)	Same as A = B + C
Assignment	A = B + C	Combinational assignment
	A := B + C	Registered assignment
Comment	A = B + C ; comment	Comment
Functional equation	name.TRST	Output enable control
	name.CLKF	Register clock control
	name.RSTF	Register reset control
	name.SETF	Register set control

Example:

```
TITLE video ; shift register
CHIP video PAL20X8
CK /LD D0 D1 D2 D3 D4 D5 D6 D7 CURS GND NC REV Q7 Q6 Q5 Q4 Q3 Q2 Q1
Q0 /RST VCC
STRING Load 'LD*/REV*/CURS*RST' ; load data
STRING LoadInv 'LD*REV*/CURS*RST' ; load inverted of data
```

STRING Shift '/LD*/CURS*/RST' ; shift data from MSB to LSB

EQUATIONS

/Q0 := /D0*Load+D0*LoadInv:++/Q1*Shift+RST

/Q1 := /D1*Load+D1*LoadInv:++/Q2*Shift+RST

/Q2 := /D2*Load+D2*LoadInv:++/Q3*Shift+RST

/Q3 := /D3*Load+D3*LoadInv:++/Q4*Shift+RST

/Q4 := /D4*Load+D4*LoadInv:++/Q5*Shift+RST

/Q5 := /D5*Load+D5*LoadInv:++/Q6*Shift+RST

/Q6 := /D6*Load+D6*LoadInv:++/Q7*Shift+RST

/Q7 := /D7*Load+D7*LoadInv:++:Shift+RST;

9.3 PLATools

Key terms and concepts: developed at UC Berkeley • eqntott input format • espresso logic-minimization program • widely used tools in the 1980s • important stepping stones to modern logic synthesis software

A PLA tools example

Input (6 minterms): $F1 = A|B|!C$; $F2 = !B\&C$; $F3 = A\&B|C$;

A	B	C	F1	F2	F3	eqntott output	espresso output
0	0	0	1	0	0		.i 3
0	0	1	0	1	1	.i 3	.o 3
0	1	0	1	0	0	.o 3	.p 6
0	1	1	1	0	1	.p 6	1-- 100
1	0	0	1	0	0	--0 100	11- 001
1	0	1	1	1	1	--1 001	--0 100
1	1	0	1	0	1	-01 010	-01 011
1	1	1	1	0	1	-1- 100	-11 101
						1-- 100	.e
						11- 001	
1	1	1	1	0	1	.e	

Output (5 minterms): $F1 = A|!C|(B\&C)$; $F2 = !B\&C$; $F3 = A\&B|(!B\&C)|(B\&C)$;

The format of the input and output files used by the PLA design tool *espresso*

Expression	Explanation
# comment	# must be first character on a line
[d]	Decimal number
[s]	Character string
.i [d]	Number of input variables
.o [d]	Number of output variables
.p [d]	Number of product terms
.ilb [s1] [s2]... [sn]	Names of the binary-valued variables must be after .i and .o
.ob [s1] [s2]... [sn]	Names of the output functions must be after .i and .o
.type f	Following table describes the ON set; DC set is empty
.type fd	Following table describes the ON set and DC set
.type fr	Following table describes the ON set and OFF set
.type fdr	Following table describes the ON set, OFF set, and DC set.
.e	Optional, marks the end of the PLA description.

The format of the plane part of the input and output files for *espresso*

Plane	Character	Explanation
I	1	The input literal appears in the product term
I	0	The input literal appears complemented in the product term
I	-	The input literal does not appear in the product term
O	1 or 4	This product term appears in the ON set
O	0	This product term appears in the OFF set
O	2 or -	This product term appears in the don't care set
O	3 or ~	No meaning for the value of this function

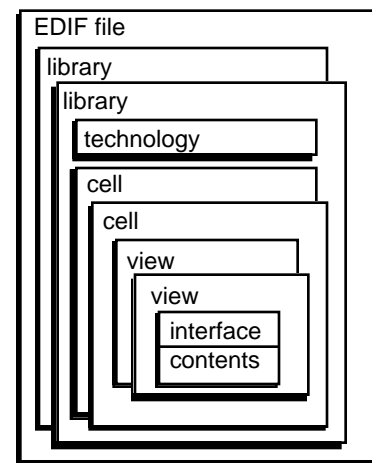
9.4 EDIF

Key terms: electronic design interchange format (EDIF) • EDIF version 2 0 0 • EDIF 3 0 0 handles buses, bus rippers, and buses across schematic pages • EDIF 4 0 0 includes new extensions for PCB and multichip module (MCM) data • Library of Parameterized Modules (LPM) • Electronic Industries Association (EIA) • ANSI/EIA Standard 548-1988

9.4.1 EDIF Syntax

Key terms: EDIF looks like Lisp or Postscript • a “write-only” language • (keywordName {form}) • keywords • forms • “define before use” • identifiers • &clock, Clock, and clock are the same • (e 14 -1) is 1.4 • scale factor • technology section • numberDefinition • scale • "A quote is % 34 %" is a string with an embedded double-quote character

The hierarchical nature of an EDIF file



9.4.2 An EDIF Netlist Example

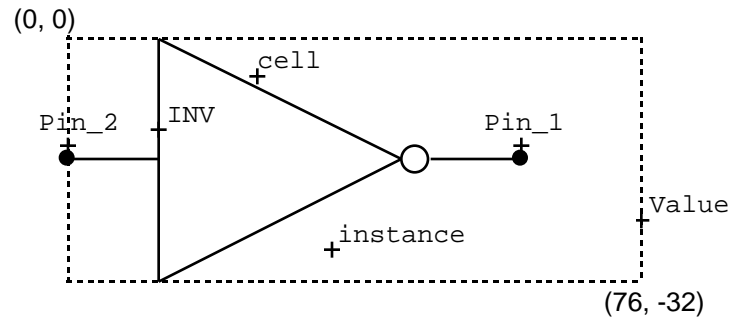
EDIF file for the halfgate netlist

```

(edif halfgate_p      (viewType NETLIST)      (viewRef
(edifVersion 2 0 0)  (interface             COMPASS_mde_view
(edifLevel 0)       (port I                 (cellRef INV
(keywordMap         (direction              (libraryRef
  (keywordLevel 0)) INPUT))                xc4000d))))
(status            (port O                 (net myInput
  (written         (direction              (joined
    (timeStamp 1996 7 OUTPUT))                (portRef
10 22          (designator                 myInput)
5 10)          "@@Label"))))                (portRef I
  (program "COMPASS (library working        (instanceRef
Design Automation -- (edifLevel 0)                B1_i1))))
EDIF Interface"    (technology            (net myOutput
  (version "v9r1.2 (numberDefinition )      (joined
last updated 26-Mar- (simulationInfo        (portRef
96"))            (logicValue H)          myOutput)
  (author         (logicValue L))        (portRef O
"mikes"))        (cell                    (instanceRef
  (library xc4000d (rename HALFGATE_P      B1_i1))))
  (edifLevel 0)   "halfgate_p")          (net VDD
  (technology     (cellType GENERIC)      (joined ))
  (numberDefinition (view                (net VSS
)                COMPASS_nls_view        (joined ))))))
  (simulationInfo (viewType NETLIST)      (design HALFGATE_P
  (logicValue H) (interface              (cellRef HALFGATE_P
  (logicValue    (port myInput          (libraryRef
L))              (direction              working))))
  (cell          INPUT))
  (rename INV    (port myOutput
"inv")          (direction
  (cellType     OUTPUT))
GENERIC)        (designator
  (view         "@@Label"))
COMPASS_mde_view (contents
                  (instance B1_i1

```

9.4.3 An EDIF Schematic Icon



An EDIF view of an inverter icon

The coordinates shown are in EDIF units. The crosses that show the text location origins and the dotted bounding box do not print as part of the icon.

9.4.4 An EDIF Example

EDIF file for a standard-cell schematic icon

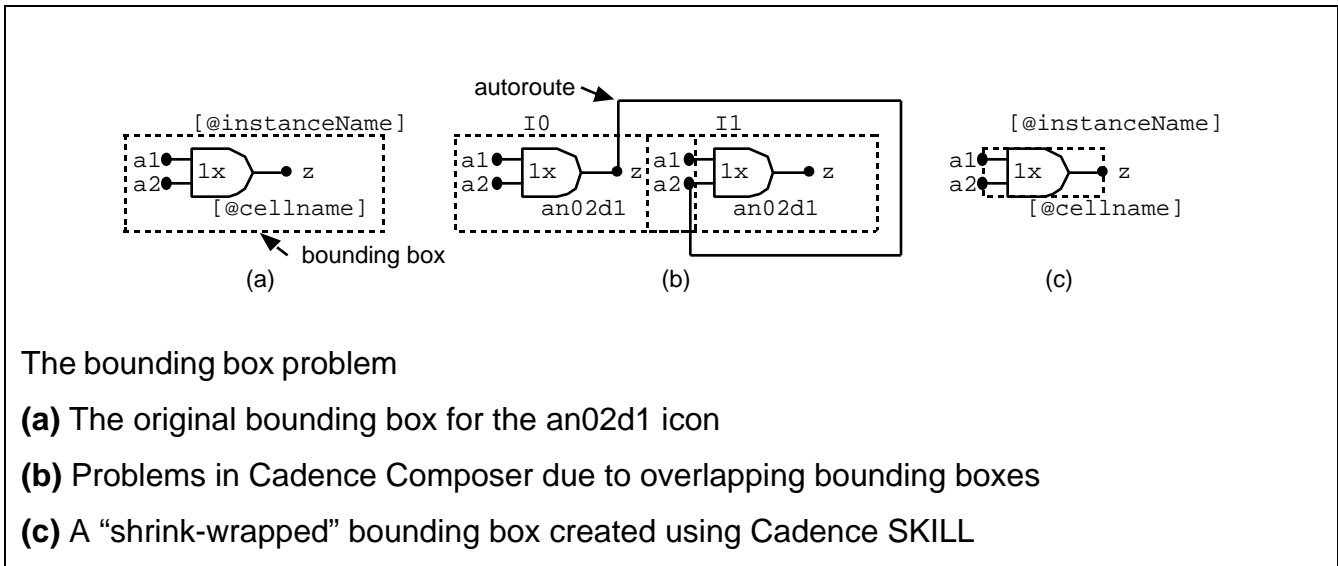
```

(edif pvsc370d
  (edifVersion 2 0 0)
  (edifLevel 0)
  (keywordMap
    (keywordLevel 0))
  (status
    (written
      (timeStamp 1993 2 9 22
38 36)
      (program "COMPASS"
        (version "v8"))
        (author "mikes")))
  (library pvsc370d
    (edifLevel 0)
    (technology
      (numberDefinition )
      (figureGroup
connector_FG
        (color 100 100 100)
        (textHeight 30)
        (visible
          (true )))
        (figureGroup icon_FG
          (color 100 100 100)
          (textHeight 30)
          (visible
            (true )))
          (figureGroup
instance_FG
            (color 100 100 100)
            (textHeight 30)
            (visible
              (true )))
            (figureGroup net_FG
              (color 100 100 100)
              (textHeight 30)
              (visible
                (true )))
              (figureGroup bus_FG
                (color 100 100 100)
                (textHeight 30)
                (visible
                  (true ))
                (pathWidth 4)))
            (cell an02d1
              (cellType GENERIC)
              (view Icon_view
                (viewType SCHEMATIC)
                (interface
                  (port A2
                    (direction INPUT))
                  (port A1
                    (direction INPUT))
                  (port Z
                    (direction OUTPUT))
                  (property label
                    (string ""))
                  (symbol
                    (portImplementation
                      (name A2
                        (display
connector_FG
                          (origin
                            (pt -5 1))))
                        (connectLocation
connector_FG
                          (figure
connector_FG
                            (dot
                              (pt 0 0))))))
                      (portImplementation
                        (name A1
                          (display
connector_FG
                            (origin
                              (pt -5 21))))
                          (connectLocation
connector_FG
                            (figure
connector_FG
                              (dot
                                (pt 0 20))))))
                      (portImplementation
                        (name Z
                          (display
connector_FG
                            (origin
                              (pt 60 15))))
                          (connectLocation
connector_FG
                            (figure
connector_FG
                              (dot
                                (pt 60 10))))))
                    (figure icon_FG
                      (path
                        (pointList
                          (pt 0 20)
                          (pt 10 20)))
                      (path
                        (pointList
                          (pt 0 0)
                          (pt 10 0)))
                      (path
                        (pointList
                          (pt 10 -5)
                          (pt 10 25)))
                      (path
                        (pointList
                          (pt 10 -5)
                          (pt 30 -5)))
                      (path
                        (pointList
                          (pt 10 25)
                          (pt 30 25)))
                      (path
                        (pointList
                          (pt 45 10)
                          (pt 60 10)))
                      (openShape
                        (curve
                          (arc
                            (pt 30 -5)
                            (pt 45 10)
                            (pt 30 25))))))
                    (boundingBox
                      (rectangle
                        (pt -15 -28)
                        (pt 134 27)))
                    (keywordDisplay
instance
                      (display icon_FG
                        (origin
                          (pt 20 29))))
                    (propertyDisplay
label
                      (display icon_FG
                        (origin

```

Compass and corresponding Cadence figureGroupnames

Compass name	Cadence name	Compass name	Cadence name
connector_FG	pin	net_FG	wire
icon_FG	device	bus_FG	not used
instance_FG	instance		

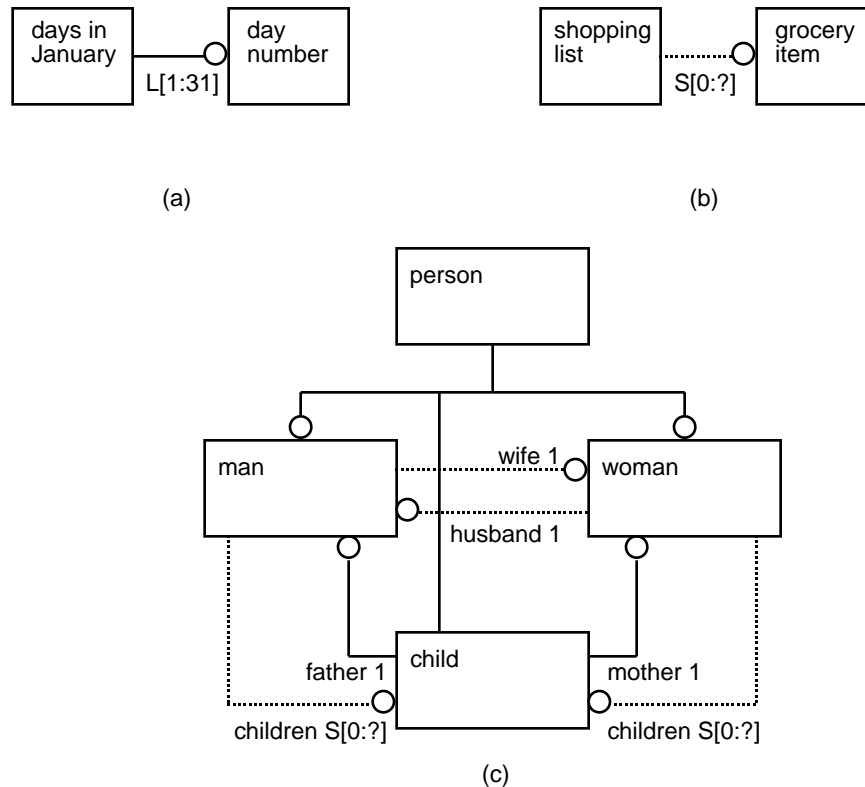


9.5 CFI Design Representation

Key terms: CAD Framework Initiative (CFI) • design representation (DR) • information model (IM) • CFI started as an attempt to standardize schematic entry • CFI ended up as an attempt to close the stable door after the horse had bolted

9.5.1 CFI Connectivity Model

Key terms: EXPRESS language • EXPRESS-G • schema • Base Connectivity Model (BCM) • five-box model • an elegant method to represent complex notions



Examples of EXPRESS-G

(a) Each day in January has a number from 1 to 31

(b) A shopping list may contain a list of items

(c) An EXPRESS-G model for a family:

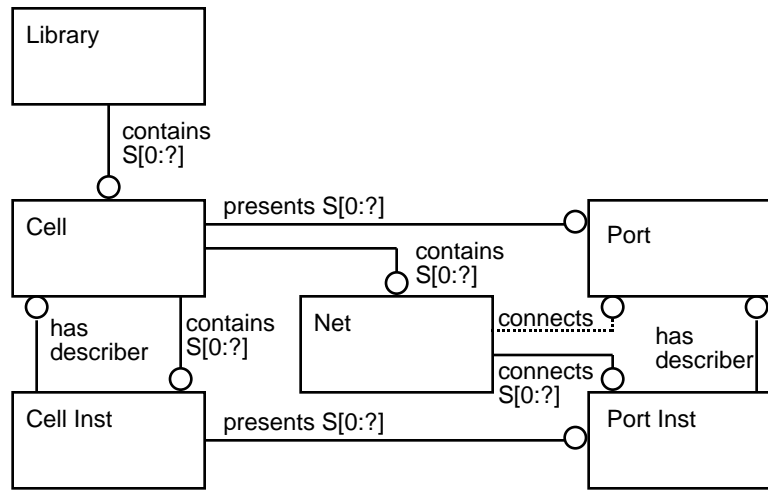
“Men, women, and children are people.”

“A man can have one woman as a wife, but does not have to.”

“A wife can have one man as a husband, but does not have to.”

“A man or a woman can have several children.”

“A child has one father and one mother.”



The original “five-box” model of electrical connectivity. (There are actually six boxes or types in this figure; the Library type was added later.)

“A library contains cells.”

“Cells have ports, contain nets, and can contain other cells.”

“Cell instances are copies of a cell and have port instances.”

“A port instance is a copy of the port in the library cell.”

“You connect to a port using a net.”

“Nets connect port instances together.”

```
SCHEMA family_model;
  ENTITY person
    ABSTRACT SUPERTYPE OF (ONEOF (man, woman, child));
    name: STRING;
    date of birth: STRING;
  END_ENTITY;

  ENTITY man
    SUBTYPE OF (person);
    wife: SET[0:1] OF woman;
    children: SET[0:?] OF child;
  END_ENTITY;

  ENTITY woman
    SUBTYPE OF (person);
    husband: SET[0:1] OF man;
    children: SET[0:?] OF child;
  END_ENTITY;

  ENTITY child
    SUBTYPE OF (person);
    father: man;
    mother: woman;
  END_ENTITY;
END_SCHEMA;
```

9.6 Summary

Key concepts:

Schematic entry using a cell library

Cells and cell instances, nets and ports

Bus naming, vectored instances in datapath

Hierarchy

Editing cells

PLD languages: ABEL, PALASM, and CUPL

Logic minimization

The functions of EDIF

CFI representation of design information

VHDL

10

Key terms and concepts: syntax and semantics • identifiers (names) • entity and architecture • package and library • interface (ports) • types • sequential statements • operators • arithmetic • concurrent statements • execution • configuration and specification

History: U.S. Department of Defense (DoD) • **VHDL** (VHSIC hardware description language) • VHSIC (very high-speed IC) program • Institute of Electrical and Electronics Engineers (IEEE) • IEEE Standard 1076-1987 and 1076-1993 • MIL-STD-454 • **Language Reference Manual (LRM)**

10.1 A Counter

Key terms and concepts: VHDL keywords • parallel programming language • VHDL is a hardware description language • **analysis** (the VHDL word for “compiled”) • logic description, simulation, and synthesis

```
entity Counter_1 is end; -- declare a "black box" called Counter_1
library STD; use STD.TEXTIO.all; -- we need this library to print
architecture Behave_1 of Counter_1 is -- describe the "black box"
-- declare a signal for the clock, type BIT, initial value '0'
    signal Clock : BIT := '0';
-- declare a signal for the count, type INTEGER, initial value 0
    signal Count : INTEGER := 0;
begin
    process begin -- process to generate the clock
        wait for 10 ns; -- a delay of 10 ns is half the clock cycle
        Clock <= not Clock;
        if (now > 340 ns) then wait; end if; -- stop after 340 ns
    end process;
-- process to do the counting, runs concurrently with other processes
    process begin
-- wait here until the clock goes from 1 to 0
        wait until (Clock = '0');
-- now handle the counting
```

```
    if (Count = 7) then Count <= 0;
    else Count <= Count + 1;
    end if;
end process;
process (Count) variable L: LINE; begin -- process to print
    write(L, now); write(L, STRING'" Count=");
    write(L, Count); writeline(output, L);
end process;
end;
```

```
> vlib work
> vcom Counter_1.vhd
Model Technology VCOM V-System VHDL/Verilog 4.5b
-- Loading package standard
-- Compiling entity counter_1
-- Loading package textio
-- Compiling architecture behave_1 of counter_1
> vsim -c counter_1
# Loading ../std.standard
# Loading ../std.textio(body)
# Loading work.counter_1(behave_1)
VSIM 1> run 500
# 0 ns Count=0
# 20 ns Count=1
(...15 lines omitted...)
# 340 ns Count=1
VSIM 2> quit
>
```

10.2 A 4-bit Multiplier

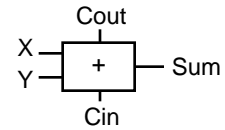
- An example to motivate the study of the syntax and semantics of VHDL
- We will multiply two 4-bit numbers by shifting and adding
- We need: two shift-registers, an 8-bit adder, and a state-machine for control
- This is an inefficient algorithm, but will illustrate how VHDL is “put together”
- We would not build/synthesize a real multiplier like this!

10.2.1 An 8-bit Adder

A full adder

```

entity Full_Adder is
  generic (TS : TIME := 0.11 ns; TC : TIME := 0.1 ns);
  port (X, Y, Cin: in BIT; Cout, Sum: out BIT);
end Full_Adder;
architecture Behave of Full_Adder is
begin
  Sum <= X xor Y xor Cin after TS;
  Cout <= (X and Y) or (X and Cin) or (Y and Cin) after TC;
end;
    
```



Timing:

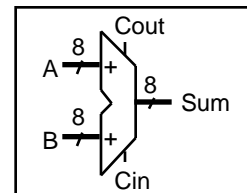
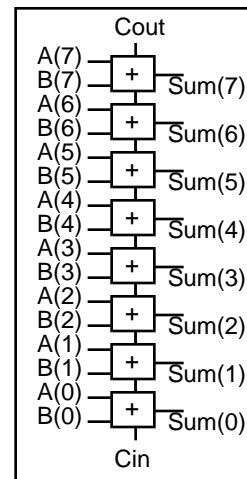
TS (Input to Sum) = 0.11 ns

TC (Input to Cout) = 0.1 ns

An 8-bit ripple-carry adder

```

entity Adder8 is
  port (A, B: in BIT_VECTOR(7 downto 0);
        Cin: in BIT; Cout: out BIT;
        Sum: out BIT_VECTOR(7 downto 0));
end Adder8;
architecture Structure of Adder8 is
  component Full_Adder
  port (X, Y, Cin: in BIT; Cout, Sum: out BIT);
  end component;
  signal C: BIT_VECTOR(7 downto 0);
begin
  Stages: for i in 7 downto 0 generate
    LowBit: if i = 0 generate
      FA:Full_Adder port map (A(0),B(0),Cin,C(0),Sum(0));
    end generate;
    OtherBits: if i /= 0 generate
      FA:Full_Adder port map
        (A(i),B(i),C(i-1),C(i),Sum(i));
    end generate;
  end generate;
  Cout <= C(7);
end;
    
```



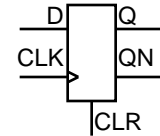
10.2.2 A Register Accumulator

Positive-edge-triggered D flip-flop with asynchronous clear

```

entity DFFClr is
  generic (TRQ : TIME := 2 ns; TCQ : TIME := 2 ns);
  port (CLR, CLK, D : in BIT; Q, QB : out BIT);
end;
architecture Behave of DFFClr is
  signal Qi : BIT;
begin
  QB <= not Qi; Q <= Qi;
  process (CLR, CLK) begin
    if CLR = '1' then Qi <= '0' after TRQ;
    elsif CLK'EVENT and CLK = '1'
      then Qi <= D after TCQ;
    end if;
  end process;
end;

```



Timing:

TRQ (CLR to Q/QN) = 2ns

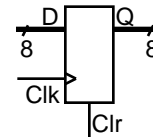
TCQ (CLK to Q/QN) = 2ns

An 8-bit register

```

entity Register8 is
  port (D : in BIT_VECTOR(7 downto 0);
        Clk, Clr: in BIT ; Q : out BIT_VECTOR(7 downto 0));
end;
architecture Structure of Register8 is
  component DFFClr
    port (Clr, Clk, D : in BIT; Q, QB : out BIT);
  end component;
begin
  STAGES: for i in 7 downto 0 generate
    FF: DFFClr port map (Clr, Clk, D(i), Q(i), open);
  end generate;
end;

```



8-bit register. Uses

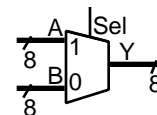
DFFClr positive edge-triggered flip-flop model.

An 8-bit multiplexer

```

entity Mux8 is
  generic (TPD : TIME := 1 ns);
  port (A, B : in BIT_VECTOR (7 downto 0);
        Sel : in BIT := '0'; Y : out BIT_VECTOR (7 downto 0));
end;
architecture Behave of Mux8 is
begin
  Y <= A after TPD when Sel = '1' else B after TPD;
end;

```



Eight 2:1 MUXs with single select input.

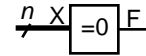
Timing:

TPD(input to Y)=1ns

10.2.3 Zero Detector

A zero detector

```
entity AllZero is
  generic (TPD : TIME := 1 ns);
  port (X : BIT_VECTOR; F : out BIT );
end;
architecture Behave of AllZero is
begin process (X) begin F <= '1' after TPD;
  for j in X'RANGE loop
    if X(j) = '1' then F <= '0' after TPD; end if;
  end loop;
end process;
end;
```



Variable-width zero detector.

Timing:

TPD(X to F) = 1ns

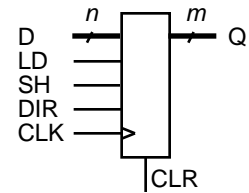
10.2.4 A Shift Register

A variable-width shift register

```

entity ShiftN is
  generic (TCQ : TIME := 0.3 ns; TLQ : TIME := 0.5 ns;
    TSQ : TIME := 0.7 ns);
  port(CLK, CLR, LD, SH, DIR: in BIT;
    D: in BIT_VECTOR; Q: out BIT_VECTOR);
  begin assert (D'LENGTH <= Q'LENGTH)
    report "D wider than output Q" severity Failure;
end ShiftN;
architecture Behave of ShiftN is
  begin Shift: process (CLR, CLK)
  subtype InB is NATURAL range D'LENGTH-1 downto 0;
  subtype OutB is NATURAL range Q'LENGTH-1 downto 0;
  variable St: BIT_VECTOR(OutB);
  begin
    if CLR = '1' then
      St := (others => '0'); Q <= St after TCQ;
    elsif CLK'EVENT and CLK='1' then
      if LD = '1' then
        St := (others => '0');
        St(InB) := D;
        Q <= St after TLQ;
      elsif SH = '1' then
        case DIR is
          when '0' => St := '0' & St(St'LEFT-1 downto 0);
          when '1' => St := St(St'LEFT-1 downto 0) & '0';
        end case;
        Q <= St after TSQ;
      end if;
    end if;
  end process;
end;

```



CLK	Clock
CLR	Clear, active high
LD	Load, active high
SH	Shift, active high
DIR	Direction, 1 = left
D	Data in
Q	Data out

Variable-width shift register. Input width must be less than output width. Output is left-shifted or right-shifted under control of DIR. Unused MSBs are zero-padded during load. Clear is asynchronous. Load is synchronous.

Timing:

TCQ (CLR to Q) = 0.3ns

TLQ (LD to Q) = 0.5ns

TSQ (SH to Q) = 0.7ns

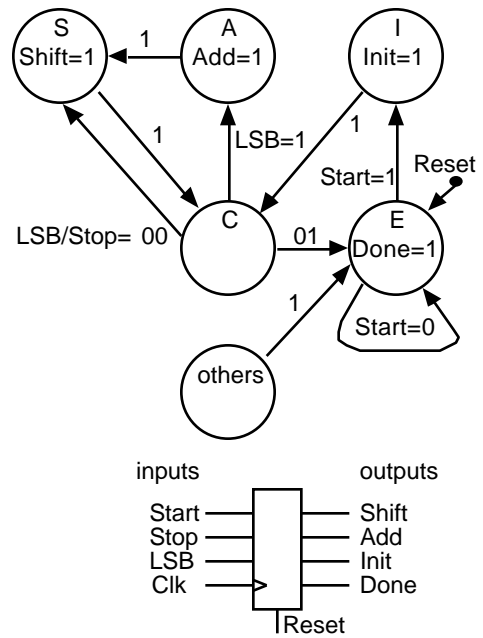
10.2.5 A State Machine

A Moore state machine for the multiplier

```

entity SM_1 is
  generic (TPD : TIME := 1 ns);
  port(Start, Clk, LSB, Stop, Reset: in BIT;
        Init, Shift, Add, Done : out BIT);
end;
architecture Moore of SM_1 is
  type STATETYPE is (I, C, A, S, E);
  signal State: STATETYPE;
begin
  Init <= '1' after TPD when State = I
    else '0' after TPD;
  Add <= '1' after TPD when State = A
    else '0' after TPD;
  Shift <= '1' after TPD when State = S
    else '0' after TPD;
  Done <= '1' after TPD when State = E
    else '0' after TPD;
  process (CLK, Reset) begin
    if Reset = '1' then State <= E;
    elsif CLK'EVENT and CLK = '1' then
      case State is
        when I => State <= C;
        when C =>
          if LSB = '1' then State <= A;
          elsif Stop = '0' then State <= S;
          else State <= E;
          end if;
        when A => State <= S;
        when S => State <= C;
        when E =>
          if Start = '1' then State <= I; end if;
        end case;
      end if;
    end process;
end;

```



State and function

E End of multiply cycle.

I Initialize: clear output register and load input registers.

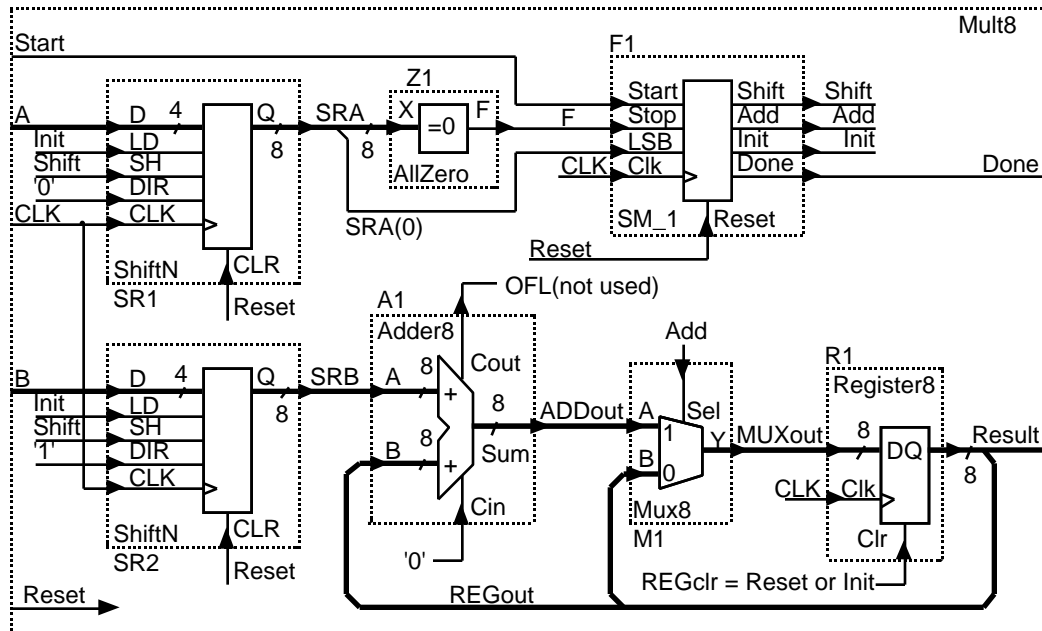
C Check if LSB of register A is zero.

A Add shift register B to accumulator.

S Shift input register A right and input register B left.

10.2.6 A Multiplier

A 4-bit by 4-bit multiplier



```

entity Mult8 is
port (A, B: in BIT_VECTOR(3 downto 0); Start, CLK, Reset: in BIT;
Result: out BIT_VECTOR(7 downto 0); Done: out BIT); end Mult8;
architecture Structure of Mult8 is use work.Mult_Components.all;
signal SRA, SRB, ADDout, MUXout, REGout: BIT_VECTOR(7 downto 0);
signal Zero, Init, Shift, Add, Low:BIT := '0'; signal High:BIT := '1';
signal F, OFL, REGclr: BIT;
begin
REGclr <= Init or Reset; Result <= REGout;
SR1 : ShiftN port map
(CLK=>CLK, CLR=>Reset, LD=>Init, SH=>Shift, DIR=>Low , D=>A, Q=>SRA);
SR2 : ShiftN port map
(CLK=>CLK, CLR=>Reset, LD=>Init, SH=>Shift, DIR=>High, D=>B, Q=>SRB);
Z1 : AllZero port map (X=>SRA, F=>Zero);
A1 : Adder8 port map (A=>SRB, B=>REGout, Cin=>Low, Cout=>OFL, Sum=>ADDout);
M1 : Mux8 port map (A=>ADDout, B=>REGout, Sel=>Add, Y=>MUXout);
R1 : Register8 port map (D=>MUXout, Q=>REGout, Clk=>CLK, Clr=>REGclr);
F1 : SM_1 port map (Start, CLK, SRA(0), Zero, Reset, Init, Shift, Add, Done);
end;

```

10.2.7 Packages and Testbench

```

package Mult_Components is --1
component Mux8 port (A,B:BIT_VECTOR(7 downto 0); --2
  Sel:BIT;Y:out BIT_VECTOR(7 downto 0));end component; --3
component AllZero port (X : BIT_VECTOR; --4
  F:out BIT );end component; --5
component Adder8 port (A,B:BIT_VECTOR(7 downto 0);Cin:BIT; --6
  Cout:out BIT;Sum:out BIT_VECTOR(7 downto 0));end component; --7
component Register8 port (D:BIT_VECTOR(7 downto 0); --8
  Clk,Clr:BIT; Q:out BIT_VECTOR(7 downto 0));end component; --9
component ShiftN port (CLK,CLR,LD,SH,DIR:BIT;D:BIT_VECTOR; --10
  Q:out BIT_VECTOR);end component; --11
component SM_1 port (Start,CLK,LSB,Stop,Reset:BIT; --12
  Init,Shift,Add,Done out BIT);end component; --13
end; --14

```

Utility code to help test the multiplier:

```

package Clock_Utills is --1
procedure Clock (signal C: out Bit; HT, LT:TIME); --2
end Clock_Utills; --3

package body Clock_Utills is --4
procedure Clock (signal C: out Bit; HT, LT:TIME) is --5
begin --6
  loop C<='1' after LT, '0' after LT + HT; wait for LT + HT; --7
  end loop; --8
end; --9
end Clock_Utills; --10

```

Two functions for testing—to convert an array of bits to a number and vice versa:

```

package Utills is --1
  function Convert (N,L: NATURAL) return BIT_VECTOR; --2
  function Convert (B: BIT_VECTOR) return NATURAL; --3
end Utills; --4

package body Utills is --5
  function Convert (N,L: NATURAL) return BIT_VECTOR is --6
    variable T:BIT_VECTOR(L-1 downto 0); --7
    variable V:NATURAL:= N; --8
    begin for i in T'RIGHT to T'LEFT loop --9
      T(i) := BIT'VAL(V mod 2); V:= V/2; --10
    end loop; return T; --11
  end; --12
  function Convert (B: BIT_VECTOR) return NATURAL is --13
    variable T:BIT_VECTOR(B'LENGTH-1 downto 0) := B; --14

```

```

    variable V:NATURAL:= 0; --15
    begin for i in T'RIGHT to T'LEFT loop --16
        if T(i) = '1' then V:= V + (2**i); end if; --17
        end loop; return V; --18
    end; --19
end Utils; --20

```

The following **testbench** exercises the multiplier model:

```

entity Test_Mult8_1 is end; -- runs forever, use break!! --1
architecture Structure of Test_Mult8_1 is --2
use Work.Utils.all; use Work.Clock_Utils.all; --3
    component Mult8 port --4
        (A, B : BIT_VECTOR(3 downto 0); Start, CLK, Reset : BIT; --5
        Result : out BIT_VECTOR(7 downto 0); Done : out BIT); --6
    end component; --7
signal A, B : BIT_VECTOR(3 downto 0); --8
signal Start, Done : BIT := '0'; --9
signal CLK, Reset : BIT; --10
signal Result : BIT_VECTOR(7 downto 0); --11
signal DA, DB, DR : INTEGER range 0 to 255; --12
begin --13
C: Clock(CLK, 10 ns, 10 ns); --14
UUT: Mult8 port map (A, B, Start, CLK, Reset, Result, Done); --15
DR <= Convert(Result); --16
Reset <= '1', '0' after 1 ns; --17
process begin --18
    for i in 1 to 3 loop for j in 4 to 7 loop --19
        DA <= i; DB <= j; --20
        A<=Convert(i,A'Length);B<=Convert(j,B'Length); --21
        wait until CLK'EVENT and CLK='1'; wait for 1 ns; --22
        Start <= '1', '0' after 20 ns; wait until Done = '1'; --23
        wait until CLK'EVENT and CLK='1'; --24
    end loop; end loop; --25
    for i in 0 to 1 loop for j in 0 to 15 loop --26
        DA <= i; DB <= j; --27
        A<=Convert(i,A'Length);B<=Convert(j,B'Length); --28
        wait until CLK'EVENT and CLK='1'; wait for 1 ns; --29
        Start <= '1', '0' after 20 ns; wait until Done = '1'; --30
        wait until CLK'EVENT and CLK='1'; --31
    end loop; end loop; --32
    wait; --33
end process; --34
end; --35

```


10.3 Syntax and Semantics of VHDL

Key terms: syntax rules • Backus–Naur form (BNF) • constructs • semantic rules • lexical rules

```
sentence ::= subject verb object.  
subject  ::= The|A noun  
object   ::= [article] noun {, and article noun}  
article  ::= the|a  
noun     ::= man|shark|house|food  
verb     ::= eats|paints
```

::= means "can be replaced by"

| means "or"

[] means "contents optional"

{ } means "contents can be left out, used once, or repeated"

The following two sentences are correct according to the syntax rules:

A shark eats food.

The house paints **the** shark, **and the** house, **and a** man.

Semantic rules tell us that the second sentence does not make much sense.

10.4 Identifiers and Literals

Key terms: nouns of VHDL • identifiers • **literals** • VHDL is not case sensitive • static (known at analysis) • abstract literals (decimal or based) • decimal literals (integer or real) • character literals • bit-string literals

```
identifier ::=
    letter {[underline] letter_or_digit}
    | \graphic_character{graphic_character}\
```

```
s -- A simple name.
S -- A simple name, the same as s. VHDL is notcase sensitive.
a_name -- Imbedded underscores are OK.
-- Successive underscores are illegal in names: Ill__egal
-- Names can't start with underscore: _Illegal
-- Names can't end with underscore: Illegal_
Too_Good -- Names must start with a letter.
-- Names can't start with a number: 2_Bad
\74LS00\ -- Extended identifier to break rules (VHDL-93 only).
VHDL \vhdl\ \VHDL\ -- Three different names (VHDL-93 only).
s_array(0) -- A static indexed name (known at analysis time).
s_array(i) -- A non-static indexed name, if i is a variable.
```

```
entity Literals_1 is end;
architecture Behave of Literals_1 is
begin process
    variable I1 : integer; variable R1 : real;
    variable C1 : CHARACTER; variable S16 : STRING(1 to 16);
    variable BV4: BIT_VECTOR(0 to 3);
    variable BV12 : BIT_VECTOR(0 to 11);
    variable BV16 : BIT_VECTOR(0 to 15);
begin
-- Abstract literals are decimal or based literals.
-- Decimal literals are integer or real literals.
-- Integer literal examples (each of these is the same):
    I1 := 120000; Int := 12e4; Int := 120_000;
-- Based literal examples (each of these is the same):
    I1 := 2#1111_1111#; I1 := 16#FFFF#;
-- Base must be an integer from 2 to 16:
    I1 := 16:FFFF:; -- you may use a : if you don't have #
```

```

-- Real literal examples (each of these is the same):
  R1 := 120000.0; R1 := 1.2e5; R1 := 12.0E4;
-- Character literal must be one of the 191 graphic characters.
-- 65 of the 256 ISO Latin-1 set are non-printing control characters
  C1 := 'A'; C1 := 'a'; -- different from each other
-- String literal examples:
  S16 := " string" & " literal"; -- concatenate long strings
  S16 := ""Hello,"" I said!"; -- doubled quotes
  S16 := % string literal%; -- can use % instead of "
  S16 := %Sale: 50%% off!!!%; -- doubled %
-- Bit-string literal examples:
  BV4 := B"1100"; -- binary bit-string literal
  BV12 := O"7777"; -- octal bit-string literal
  BV16 := X"FFFF"; -- hex bit-string literal
wait; end process; -- the wait prevents an endless loop
end;

```

10.5 Entities and Architectures

Key terms: design file (bookshelf) • design units • library units (book) • **library** (collection of bookshelves) • primary units • secondary units (c.f. Table of Contents) • **entity declaration** (black box) • formal ports (or formals) • **architecture body** (contents of black box) • visibility • component declaration • structural model • local ports (or locals) • instance names • actual ports (or actuals) • binding • configuration declaration (a “shopping list”) • design entity (entity–architecture pair)

```

design_file ::=
  {library_clause|use_clause} library_unit
  {{library_clause|use_clause} library_unit}

```

```

library_unit ::= primary_unit|secondary_unit

```

```

primary_unit ::=
  entity_declaration|configuration_declaration|package_declaration

```

```

secondary_unit ::= architecture_body|package_body

```

```

entity_declaration ::=
entity identifier is
    [generic (formal_generic_interface_list);]
    [port (formal_port_interface_list);]
    {entity_declarative_item}
    [begin
        {[label:] [postponed] assertion ;
        | [label:] [postponed] passive_procedure_call ;
        | passive_process_statement}]
    end [entity] [entity_identifier] ;

```

```

entity Half_Adder is
    port (X, Y : in BIT := '0'; Sum, Cout : out BIT); -- formals
end;

```

```

architecture_body ::=
    architecture identifier of entity_name is
        {block_declarative_item}
        begin
            {concurrent_statement}
        end [architecture] [architecture_identifier] ;

```

```

architecture Behave of Half_Adder is
    begin Sum <= X xor Y; Cout <= X and Y;
end Behave;

```

Components:

```

component_declaration ::=
    component identifier [is]
        [generic (local_generic_interface_list);]
        [port (local_port_interface_list);]
    end component [component_identifier];

```

```

architecture Netlist of Half_Adder is
component MyXor port (A_Xor,B_Xor : in BIT; Z_Xor : out BIT);
end component; -- component with locals
component MyAnd port (A_And,B_And : in BIT; Z_And : out BIT);
end component; -- component with locals

```

```

begin
  Xor1: MyXor port map (X, Y, Sum);      -- instance with actuals
  And1 : MyAnd port map (X, Y, Cout);   -- instance with actuals
end;

```

These design entities (entity–architecture pairs) would be part of a technology library:

```

entity AndGate is
  port (And_in_1, And_in_2 : in BIT; And_out : out BIT); -- formals
end;

```

```

architecture Simple of AndGate is
  begin And_out <= And_in_1 and And_in_2;
end;

```

```

entity XorGate is
  port (Xor_in_1, Xor_in_2 : in BIT; Xor_out : out BIT); -- formals
end;

```

```

architecture Simple of XorGate is
  begin Xor_out <= Xor_in_1 xor Xor_in_2;
end;

```

```

configuration_declaration ::=
  configuration identifier of entity_name is
    {use_clause|attribute_specification|group_declaration}
    block_configuration
  end [configuration] [configuration_identifier] ;

```

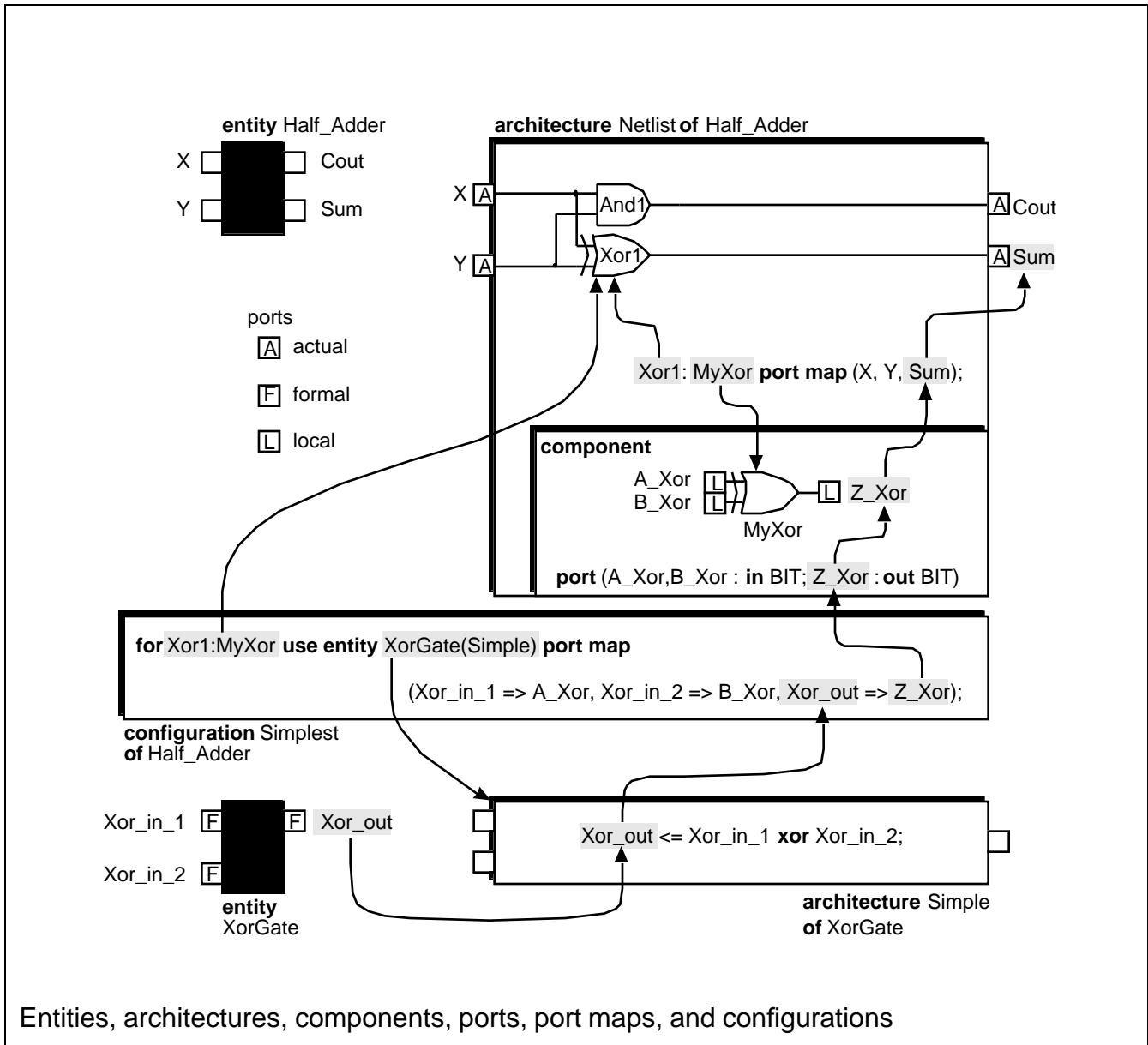
```

configuration Simplest of Half_Adder is
use work.all;
for Netlist
  for And1 : MyAnd use entity AndGate(Simple)
    port map -- association: formals => locals
      (And_in_1 => A_And, And_in_2 => B_And, And_out => Z_And);
  end for;
  for Xor1 : MyXor use entity XorGate(Simple)
    port map
      (Xor_in_1 => A_Xor, Xor_in_2 => B_Xor, Xor_out => Z_Xor);
  end for;
end configuration;

```

```

end for;
end for;
end;
    
```



10.6 Packages and Libraries

Key terms: design library (the current working library or a resource library) • **working library** (work) • **package** • package body • package visibility • **library clause** • **use clause**

```
package_declaration ::=
package identifier is
{subprogram_declaration | type_declaration | subtype_declaration
 | constant_declaration | signal_declaration | file_declaration
 | alias_declaration | component_declaration
 | attribute_declaration | attribute_specification
 | disconnection_specification | use_clause
 | shared_variable_declaration | group_declaration
 | group_template_declaration}
end [package] [package_identifier] ;
```

```
package_body ::=
package body package_identifier is
{subprogram_declaration | subprogram_body
 | type_declaration | subtype_declaration
 | constant_declaration | file_declaration | alias_declaration
 | use_clause
 | shared_variable_declaration | group_declaration
 | group_template_declaration}
end [package body] [package_identifier] ;
```

```
library MyLib; -- library clause
use MyLib.MyPackage all; -- use clause
-- design unit (entity + architecture, etc.) follows:
```

10.6.1 Standard Package

Key terms: STANDARD package (defined in the LRM) • TIME • INTEGER • REAL • STRING • CHARACTER • I use uppercase for standard types • ISO 646-1983 • ASCII character set • character codes • graphic symbol (glyph) • ISO 8859-1:1987(E) • ISO Latin-1

```
package Part_STANDARD is
type BOOLEAN is (FALSE, TRUE); type BIT is ('0', '1');
```

```

type SEVERITY_LEVEL is (NOTE, WARNING, ERROR, FAILURE);
subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;
subtype POSITIVE is INTEGER range 1 to INTEGER'HIGH;
type BIT_VECTOR is array (NATURAL range <>) of BIT;
type STRING is array (POSITIVE range <>) of CHARACTER;
-- the following declarations are VHDL-93 only:
attribute FOREIGN: STRING; -- for links to other languages
subtype DELAY_LENGTH is TIME range 0 fs to TIME'HIGH;
type FILE_OPEN_KIND is (READ_MODE,WRITE_MODE,APPEND_MODE);
type FILE_OPEN_STATUS is
(OPEN_OK,STATUS_ERROR,NAME_ERROR,MODE_ERROR);
end Part_STANDARD;

```

```

type TIME is range implementation_defined -- and varies with software
  units fs; ps = 1000 fs; ns = 1000 ps; us = 1000 ns; ms = 1000 us;
  sec = 1000 ms; min = 60 sec; hr = 60 min;end units;

```

```

type Part_CHARACTER is ( -- 128 ASCII characters in VHDL-87
NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL, -- 33 control characters
  BS, HT, LF, VT, FF, CR, SO, SI, -- including:
DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB, -- format effectors:
CAN, EM, SUB, ESC, FSP, GSP, RSP, USP, -- horizontal tab = HT
' ', '!', '"', '#', '$', '%', '&', '\'', -- line feed = LF
'(', ')', '*', '+', ',', '-', '.', '/', -- vertical tab = VT
'0', '1', '2', '3', '4', '5', '6', '7', -- form feed = FF
'8', '9', ':', ';', '<', '=', '>', '?', -- carriage return = CR
'@', 'A', 'B', 'C', 'D', 'E', 'F', 'G', -- and others:
'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', -- FSP, GSP, RSP, USP use P
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', -- suffix to avoid conflict
'X', 'Y', 'Z', '[', '\', ']', '^', '_', -- with TIME units
'`', 'a', 'b', 'c', 'd', 'e', 'f', 'g',
'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w',
'x', 'y', 'z', '{', '|', '}', '~', DEL -- delete = DEL

```

```

-- VHDL-93 includes 96 more Latin-1 characters, like ¥ (Yen) and
-- 32 more control characters, better not to use any of them.
);

```


10.6.2 Std_logic_1164 Package

Key terms: **logic-value system** • BIT • '0' and '1' • 'X' (unknown) • 'Z' (high-impedance) • metalogical value (simbits) • Std_logic_1164 package • MVL9—multivalued logic nine • driver • resolve • resolution function • resolved subtype STD_LOGIC • unresolved type STD_ULOGIC • subtypes are compatible with types • **overloading** • STD_LOGIC_VECTOR • STD_ULOGIC_VECTOR • don't care logic value '-' (hyphen)

```

type MVL4 is ('X', '0', '1', 'Z'); -- example of a four-value logic
system

library IEEE; use IEEE.std_logic_1164all; -- to use the IEEE package

package Part_STD_LOGIC_1164 is --1
type STD_ULOGIC is --2
( 'U', -- Uninitialized --3
  'X', -- Forcing Unknown --4
  '0', -- Forcing 0 --5
  '1', -- Forcing 1 --6
  'Z', -- High Impedance --7
  'W', -- Weak Unknown --8
  'L', -- Weak 0 --9
  'H', -- Weak 1 --10
  '-' -- Don't Care); --11
type STD_ULOGIC_VECTOR is array (NATURAL range <>) of STD_ULOGIC; --12
function resolved (s : STD_ULOGIC_VECTOR) return STD_ULOGIC; --13
subtype STD_LOGIC is resolved STD_ULOGIC; --14
type STD_LOGIC_VECTOR is array (NATURAL range <>) of STD_LOGIC; --15
subtype X01 is resolved STD_ULOGIC range 'X' to '1'; --16
subtype X01Z is resolved STD_ULOGIC range 'X' to 'Z'; --17
subtype UX01 is resolved STD_ULOGIC range 'U' to '1'; --18
subtype UX01Z is resolved STD_ULOGIC range 'U' to 'Z'; --19

-- Vectorized overloaded logical operators: --20
function "and" (L : STD_ULOGIC; R : STD_ULOGIC) return UX01; --21
-- Logical operators not, and, nand, or, nor, xor, xnor (VHDL-93), --22
-- overloaded for STD_ULOGIC STD_ULOGIC_VECTOR STD_LOGIC_VECTOR. --23

-- Strength strippers and type conversion functions: --24
-- function To_T (X : F) return T; --25
-- defined for types, T and F, where --26
-- F=BIT BIT_VECTOR STD_ULOGIC STD_ULOGIC_VECTOR STD_LOGIC_VECTOR --27
-- T=types F plus types X01 X01Z UX01 (but not type UX01Z) --28

-- Exclude _'s in T in name: TO_STDULOGIC not TO_STD_ULOGIC --29
-- To_X01 : L->0, H->1 others->X --30

```

```

-- To_X01Z: Z->Z, others as To_X01 --31
-- To_UX01: U->U, others as To_X01 --32
-- Edge detection functions: --33
function rising_edge (signal s: STD_ULOGIC) return BOOLEAN; --34
function falling_edge (signal s: STD_ULOGIC) return BOOLEAN; --35
-- Unknown detection (returns true if s = U, X, Z, W): --36
-- function Is_X (s : T) return BOOLEAN; --37
-- defined for T = STD_ULOGIC STD_ULOGIC_VECTOR STD_LOGIC_VECTOR. --38
end Part_STD_LOGIC_1164; --39

```

10.6.3 Textio Package

```

package Part_TEXTIO is -- VHDL-93 version.
type LINE is access STRING; -- LINE is a pointer to a STRING value.
type TEXT is file of STRING; -- File of ASCII records.
type SIDE is (RIGHT, LEFT); -- for justifying output data.
subtype WIDTH is NATURAL; -- for specifying widths of output
fields.
file INPUT : TEXT open READ_MODE is "STD_INPUT"; -- Default input
file.
file OUTPUT : TEXT open WRITE_MODE is "STD_OUTPUT"; -- Default
output.

-- The following procedures are defined for types, T, where
-- T = BIT BIT_VECTOR BOOLEAN CHARACTER INTEGER REAL TIME STRING
-- procedure READLINE(file F : TEXT; L : out LINE);
-- procedure READ(L : inout LINE; VALUE : out T);
-- procedure READ(L : inout LINE; VALUE : out T; GOOD: out
BOOLEAN);
-- procedure WRITELINE(F : out TEXT; L : inout LINE);
-- procedure WRITE(
-- L : inout LINE;
-- VALUE : in T;
-- JUSTIFIED : in SIDE:= RIGHT;
-- FIELD:in WIDTH := 0;
-- DIGITS:in NATURAL := 0; -- for T = REAL only

```

```
--      UNIT:in TIME:= ns);      -- for T = TIME only
-- function ENDFILE(F : in TEXT) return BOOLEAN;
```

```
end Part_TEXTIO;
```

Example:

```
library std; use std.textio.all; entity Text is end;
architecture Behave of Text is signal count : INTEGER := 0;
begin count <= 1 after 10 ns, 2 after 20 ns, 3 after 30 ns;
process (count) variable L: LINE; begin
if (count > 0) then
    write(L, now);          -- Write time.
    write(L, STRING'(" count="));-- STRING' is a type qualification.
    write(L, count); writeline(output, L);
end if; end process; end;
```

```
10 ns count=1
20 ns count=2
30 ns count=3
```

10.6.4 Other Packages

Key terms: arithmetic packages • Synopsys std_arith • (mis)use of IEEE library • math packages [IEEE 1076.2, 1996] • synthesis packages • component packages

10.6.5 Creating Packages

Key terms: packaged constants • linking the VHDL world and the real world

```
package Adder_Pkg is -- a package declaration
    constant BUSWIDTH : INTEGER := 16;
end Adder_Pkg;
```

```
use work.Adder_Pkg.all; -- a use clause
entity Adder is end Adder;
architecture Flexible of Adder is -- work.Adder_Pkg is visible here
    begin process begin
```

```
    MyLoop : for j in 0 to BUSWIDTH loop -- adder code goes here
    end loop; wait; -- the wait prevents an endless cycle
end process;
end Flexible;
```

```
package GLOBALS is
    constant HI : BIT := '1'; constant LO: BIT := '0';
end GLOBALS;
```

```
library MyLib; -- use MyLib.Add_Pkg.all; -- use all the package
use MyLib.Add_Pkg_Fn.add; -- just function 'add' from the package
```

```
entity Lib_1 is port (s : out BIT_VECTOR(3 downto 0) := "0000"); end;
architecture Behave of Lib_1 is begin process
begin s <= add ("0001", "0010", "1000");wait; end process; end;
```

There are three common methods to create the links between the file and directory names:

- Use a UNIX environment variable (SETENV MyLib ~/MyDirectory/MyLibFile for example).
- Create a separate file that establishes the links between the filename known to the operating system and the library name known to the VHDL software.
- Include the links in an initialization file (often with an '.ini' suffix).

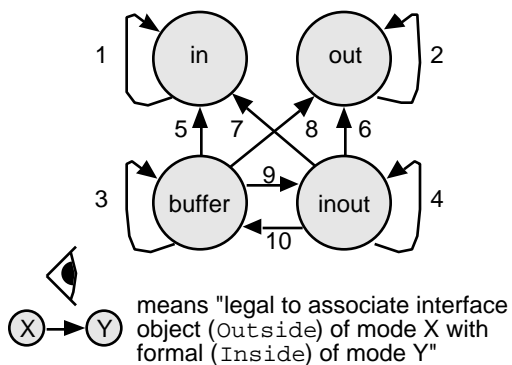
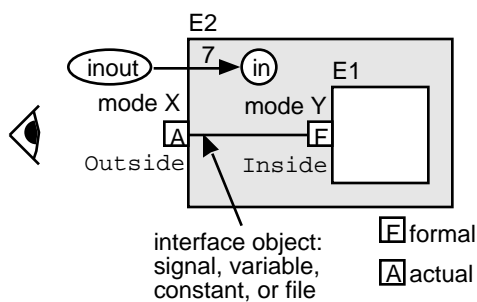
10.7 Interface Declarations

Key terms: interface declaration • formals • locals • actuals • interface objects (constants, **signals**, **variables**, or files) • interface constants (generics of a design entity, a component, or a block, or parameters of subprograms) • interface signals (ports of a design entity, component, or block, and parameters of subprograms) • interface variables and interface files (parameters of subprograms) • interface object **mode** (in, the default, out, inout, buffer, linkage) • read • update • interface object rules (“i before e”), there are also mode rules (“except after c”)

Modes of interface objects and their properties

```
entity E1 is port (Inside : in BIT); end; architecture Behave of E1 is begin end;
entity E2 is port (Outside : inout BIT := '1'); end; architecture Behave of E2 is
component E1 port (Inside: in BIT); end component; signal UpdateMe : BIT; begin
I1 : E1 port map (Inside => Outside); -- formal/local (mode in) => actual (mode
inout)
UpdateMe <= Outside; -- OK to read Outside (mode inout)
Outside <= '0' after 10 ns; -- and OK to update Outside (mode inout)
end;
```

Possible modes of interface object, Outside	in (default)	out	inout	buffer
Can you read Outside (RHS of assignment)?	Yes	No	Yes	Yes
Can you update Outside (LHS of assignment)?	No	Yes	Yes	Yes
Modes of Inside that Outside may connect to (see below)	in	out	any	any



10.7.1 Port Declaration

Key terms: **ports** (connectors) • port interface declaration • formals • locals • actuals • implicit signal declaration • **port mode** • signal kind • default value • default expression • open • **port map** • positional association • named association • default binding

Properties of ports

Example entity declaration:

```
entity E is port (F_1:BIT; F_2:out BIT; F_3:inout BIT; F_4:buffer BIT); end; -- formals
```

Example component declaration:

```
component C port (L_1:BIT; L_2:out BIT; L_3:inout BIT; L_4:buffer BIT); -- locals
end component;
```

Example component instantiation:

```
I1 : C port map
(L_1 => A_1,L_2 => A_2,L_3 => A_3,L_4 => A_4); -- locals => actuals
```

Example configuration:

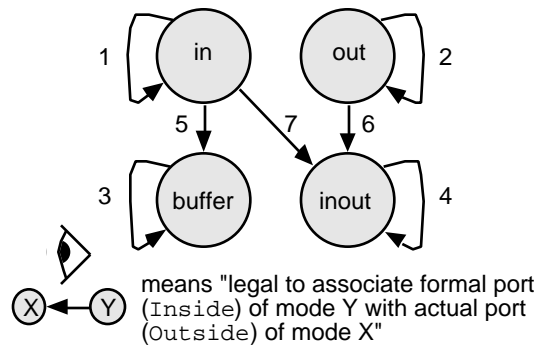
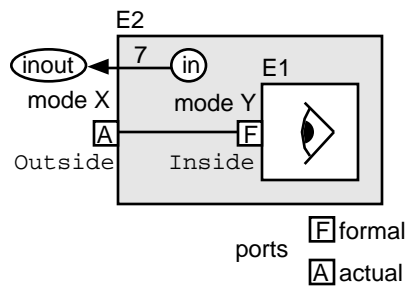
```
for I1 : C use entity E(Behave) port map
(F_1 => L_1,F_2 => L_2,F_3 => L_3,F_4 => L_4); -- formals => locals
```

Interface object, port F	F_1	F_2	F_3	F_4
Mode of F	in (default)	out	inout	buffer
Can you read attributes of F?	Yes, but not the attributes:	Yes, but not the attributes:	Yes, but not the attributes:	Yes
[VHDL LRM4.3.2]	'STABLE 'QUIET 'DELAYED 'TRANSACTION	'STABLE 'QUIET 'DELAYED 'TRANSACTION 'EVENT 'ACTIVE 'LAST_EVENT 'LAST_ACTIVE 'LAST_VALUE	'STABLE 'QUIET 'DELAYED 'TRANSACTION	

Connection rules for port modes

```
entity E1 is port (Inside : in BIT); end; architecture Behave of E1 is begin end;
entity E2 is port (Outside : inout BIT := '1'); end; architecture Behave of E2 is
component E1 port (Inside : in BIT); end component; begin
I1 : E1 port map (Inside => Outside); -- formal/local (mode in) => actual (mode
inout)
end;
```

Possible modes of interface object, Inside	in (default)	out	inout	buffer
Modes of Outside that Inside may connect to (see below)	in inout buffer	out inout	inout ¹	buffer ²



¹A signal of mode `inout` can be updated by any number of sources.

²A signal of mode `buffer` can be updated by at most one source.

```
port (port_interface_list)
```

```
interface_list ::=
  port_interface_declaration {; port_interface_declaration}
```

```
interface_declaration ::=
  [signal]
  identifier {, identifier} : in | out | inout | buffer | linkage
  subtype_indication bus] [ := static_expression]
```

```
entity Association_1 is
  port (signal X, Y : in BIT := '0'; Z1, Z2, Z3 : out BIT);
end;
```

```

use work.all; -- makes analyzed design entity AndGate(Simple)
visible.
architecture Netlist of Association_1 is
-- The formal port clause for entity AndGate looks like this:
-- port (And_in_1, And_in_2: in BIT; And_out : out BIT); -- Formals.
component AndGate port
  (And_in_1, And_in_2 : in BIT; And_out : out BIT); -- Locals.
end component;
begin
-- The component and entity have the same names: AndGate.
-- The port names are also the same: And_in_1, And_in_2, And_out,
-- so we can use default binding without a configuration.
-- The last (and only) architecture for AndGate will be used: Simple.
A1:AndGate port map (X, Y, Z1); -- positional association
A2:AndGate port map (And_in_2=>Y, And_out=>Z2, And_in_1=>X);-- named
A3:AndGate port map (X, And_out => Z3, And_in_2 => Y);-- both
end;

```

```

entity ClockGen_1 is port (Clock : out BIT); end;
architecture Behave of ClockGen_1 is
begin process variable Temp : BIT := '1';
  begin
-- Clock <= not Clock; -- Illegal, you cannot read Clock (mode out),
  Temp := not Temp;      -- use a temporary variable instead.
  Clock <= Temp after 10 ns; wait for 10 ns;
  if (now > 100 ns) then wait; end if; end process;
end;

```

10.7.2 Generics

Key terms: **generic** (similar to a port) • ports (signals) carry changing information between entities • generics carry constant, static information • generic interface list

```

entity AndT is
  generic (TPD : TIME := 1 ns);
  port (a, b : BIT := '0'; q: out BIT);
end;
architecture Behave of AndT is
  begin q <= a and b after TPD;
end;

```

```
entity AndT_Test_1 is end;
architecture Netlist_1 of AndT_Test_1 is
  component MyAnd
    port (a, b : BIT; q : out BIT);
  end component;
  signal a1, b1, q1 : BIT := '1';
  begin
    And1 : MyAnd port map (a1, b1, q1);
end Netlist_1;

configuration Simplest_1 of AndT_Test_1 is use work.all;
  for Netlist_1 for And1 : MyAnd
    use entity AndT(Behave) generic map (2 ns);
  end for; end for;
end Simplest_1;
```

10.8 Type Declarations

Key terms and concepts: **type** of an object • VHDL is strongly typed • you cannot add a temperature of type Centigrade to a temperature of type Fahrenheit • **type declaration** • **range** • precision • subtype • subtype declaration • composite type (**array type**) • aggregate notation • record type

There are four **type classes**: scalar types, composite types, access types, file types

1. Scalar types: integer type, floating-point type, physical type, enumeration type

(integer and **enumeration types** are discrete types)

(integer, floating-point, and physical types are numeric types)

(physical types correspond to time, voltage, current, and so on and have dimensions)

2. Composite types include **array types** (and record types)

3. Access types are pointers, good for abstract data structures, less so in ASIC design

4. File types are used for file I/O, not ASIC design

```

type_declaration ::=
    type identifier ;
| type identifier is
(identifier|'graphic_character' {, identifier|'graphic_character'}) ;
| range_constraint ;          | physical_type_definition ;
| record_type_definition ;    | access subtype_indication ;
| file of type_name ;         | file of subtype_name ;
| array index_constraint of element_subtype_indication ;
| array
  (type_name|subtype_name range <>
   {, type_name|subtype_name range <>}) of
  element_subtype_indication ;

```

```

entity Declaration_1 is end; architecture Behave of Declaration_1 is
type F is range 32 to 212; -- Integer type, ascending range.
type C is range 0 to 100; -- Range 0 to 100 is the range constraint.
subtype G is INTEGER range 9 to 0; -- Base type INTEGER, descending.
-- This is illegal: type Bad100 is INTEGER range 0 to 100;
-- don't use INTEGER in declaration of type (but OK in subtype).
type Rainbow is (R, O, Y, G, B, I, V); -- An enumeration type.
-- Enumeration types always have an ascending range.
type MVL4 is ('X', '0', '1', 'Z');

```

```
-- Note that 'X' and 'x' are different character literals.
-- The default initial value is MVL4'LEFT = 'X'.
-- We say '0' and '1' (already enumeration literals
-- for predefined type BIT) are overloaded.
-- Illegal enumeration type: type Bad4 is ("X", "0", "1", "Z");
-- Enumeration literals must be character literals or identifiers.
begin end;
```

```
entity Arrays_1 is end; architecture Behave of Arrays_1 is
type Word is array (0 to 31) of BIT; -- a 32-bit array, ascending
type Byte is array (NATURAL range 7 downto 0) of BIT; -- descending
type BigBit is array (NATURAL range <>) of BIT;
-- We call <> a box, it means the range is undefined for now.
-- We call BigBit an unconstrained array.
-- This is OK, we constrain the range of an object that uses
-- type BigBit when we declare the object, like this:
subtype Nibble is BigBit(3 downto 0);
type T1 is array (POSITIVE range 1 to 32) of BIT;
-- T1, a constrained array declaration, is equivalent to a type T2
-- with the following three declarations:
subtype index_subtype is POSITIVE range 1 to 32;
type array_type is array (index_subtype range <>) of BIT;
subtype T2 is array_type (index_subtype);
-- We refer to index_subtype and array_type as being
-- anonymous subtypes of T1 (since they don't really exist).
begin end;
```

```
entity Aggregate_1 is end; architecture Behave of Aggregate_1 is
type D is array (0 to 3) of BIT; type Mask is array (1 to 2) of BIT;
signal MyData : D := ('0', others => '1'); -- positional aggregate
signal MyMask : Mask := (2 => '0', 1 => '1'); -- named aggregate
begin end;
```

```
entity Record_2 is end; architecture Behave of Record_2 is
type Complex is record real : INTEGER; imag : INTEGER; end record;
signal s1 : Complex := (0, others => 1); signal s2 : Complex;
begin s2 <= (imag => 2, real => 1); end;
```

10.9 Other Declarations

Key concepts: (we already covered entity, configuration, component, package, interface, type, and subtype declarations)

- objects: constant, variable, signal, file
- alias (user-defined “monikers”)
- attributes (user-defined and tool-vendor defined)
- subprograms: functions and procedures
- groups and group templates are new to VHDL-93 and hardly used in ASIC design

```

declaration ::=
  type_declaration          | subtype_declaration | object_declaration
| interface_declaration    | alias_declaration   | attribute_declaration
| component_declaration   | entity_declaration
| configuration_declaration | subprogram_declaration
| package_declaration
| group_template_declaration | group_declaration

```

10.9.1 Object Declarations

Key terms and concepts: class of an **object** • declarative region (before the first begin) • declare a type with (explicit) initial value • (implicit) default initial value is T'LEFT • explicit signal declarations • shared variable

There are four object classes: constant, variable, signal, file

You use a constant declaration, signal declaration, variable declaration, or file declaration together with a type

Signals represent real wires in hardware

Variables are memory locations in a computer

```

entity Initial_1 is end; architecture Behave of Initial_1 is
type Fahrenheit is range 32 to 212; -- Default initial value is 32.
type Rainbow is (R, O, Y, G, B, I, V); -- Default initial value is R.
type MVL4 is ('X', '0', '1', 'Z'); -- MVL4'LEFT = 'X'.
begin end;

```

```
constant_declaration ::= constant  
identifier {, identifier}:subtype_indication [:= expression] ;
```

```
signal_declaration ::= signal  
identifier {, identifier}:subtype_indication register|bus  
[:=expression];
```

```
entity Constant_2 is end;  
library IEEE; use IEEE.STD_LOGIC_1164 all;  
architecture Behave of Constant_2 is  
constant Pi : REAL := 3.14159;          -- A constant declaration.  
signal B : BOOLEAN; signal s1, s2: BIT;  
signal sum : INTEGER range 0 to 15;  -- Not a new type.  
signal SmallBus : BIT_VECTOR(15downto 0);      -- 16-bit bus.  
signal GBus : STD_LOGIC_VECTOR(31downto 0) bus; -- A guarded signal.  
begin end;
```

```
variable_declaration ::= [shared] variable  
identifier {, identifier}:subtype_indication [:= expression] ;
```

```
library IEEE; use IEEE.STD_LOGIC_1164 all; entity Variables_1 is end;  
architecture Behave of Variables_1 is begin process  
    variable i : INTEGER range 1 to 10 := 10; -- Initial value = 10.  
    variable v : STD_LOGIC_VECTOR (0 to 31) := (others => '0');  
    begin wait; end process; -- The wait stops an endless cycle.  
end;
```

10.9.2 Subprogram Declarations

Key terms and concepts: subprogram • **function** • **procedure** • subprogram declaration: a function declaration or a procedure declaration • formal parameters (or formals) • subprogram invocation • actual parameters (or actuals) • impure function (now) • pure function (default) • subprogram specification • subprogram body • conform • private

Properties of subprogram parameters				
Example subprogram declarations:				
<code>function my_function(Ff) return BIT is -- Formal function parameter, Ff.</code>				
<code>procedure my_procedure(Fp); -- Formal procedure parameter, Fp.</code>				
Example subprogram calls:				
<code>my_result := my_function(Af); -- Calling a function with an actual parameter, Af.</code>				
<code>MY_LABEL:my_procedure(Ap); -- Using a procedure with an actual parameter, Ap.</code>				
Mode of Ff or Fp (formals)	in	out	inout	No mode
Permissible classes for Af (function actual parameter)	constant (default)	Not allowed	Not allowed	file
	signal			
Permissible classes for Ap (procedure actual parameter)	constant (default)	constant	constant	file
	variable	variable (default)	variable (default)	
	signal	signal	signal	
Can you read attributes of Ff or Fp (formals)?	Yes, except: 'STABLE 'QUIET 'DELAYED 'TRANSACTION of a signal	Yes, except: 'STABLE T 'DELAYED 'TRANSACTION 'EVENT E 'LAST_EVENT 'LAST_ACTIVE 'LAST_VALUE of a signal	'QUIE 'STABLE 'QUIET 'DELAYED 'TRANSACTION 'ACTIV of a signal	

```
subprogram_declaration ::= subprogram_specification ; ::=
procedure
  identifier | string_literal [ parameter_interface_list ]
```

```
| [pure|impure] function
  identifier|string_literal [parameter_interface_list]
return type_name|subtype_name;
```

```
function add(a, b, c : BIT_VECTOR(3 downto 0)) return BIT_VECTOR is
-- A function declaration, a function can't modify a, b, or c.
```

```
procedure Is_A_Eq_B (signal A, B : BIT; signal Y : out BIT);
-- A procedure declaration, a procedure can change Y.
```

```
subprogram_body ::=
  subprogram_specification is
  {subprogram_declaration|subprogram_body
  |type_declaration|subtype_declaration
  |constant_declaration|variable_declaration|file_declaration
  |alias_declaration|attribute_declaration|attribute_specification
  |use_clause|group_template_declaration|group_declaration}
  begin
  {sequential_statement}
  end [procedure|function] [identifier|string_literal] ;
```

```
function subset0(sout0 : in BIT) return BIT_VECTOR -- declaration
-- Declaration can be separate from the body.
```

```
function subset0(sout0 : in BIT) return BIT_VECTOR is -- body
variable y : BIT_VECTOR(2 downto 0);
begin
if (sout0 = '0') then y := "000"; else y := "100"; end if;
return result;
end;
```

```
procedure clockGen (clk : out BIT) -- Declaration
```

```
procedure clockGen (clk : out BIT) is -- Specification
begin -- Careful this process runs forever:
  process begin wait for 10 ns; clk <= not clk; end process;
end;
```

```

entity F_1 is port (s : out BIT_VECTOR(3 downto 0) := "0000"); end;
architecture Behave of F_1 is begin process
function add(a, b, c : BIT_VECTOR(3 downto 0)) return BIT_VECTOR is
begin return a xor b xor c; end;
begin s <= add("0001", "0010", "1000");wait; end process; end;

```

```

package And_Pkg is
  procedure V_And(a, b : BIT; signal c : out BIT);
  function V_And(a, b : BIT) return BIT;
end;

```

```

package body And_Pkg is
  procedure V_And(a,b : BIT;signal c : out BIT) is
    begin c <= a and b; end;
  function V_And(a,b : BIT) return BIT is
    begin return a and b; end;
end And_Pkg;

```

```

entity F_2 is port (s: out BIT := '0'); end;
use work.And_Pkg.all; -- use package already analyzed
architecture Behave of F_2 is begin process begin
s <= V_And('1', '1');wait; end process; end;

```

10.9.3 Alias and Attribute Declarations

```

alias_declaration ::=
alias
  identifier|character_literal|operator_symbol [ :subtype_indication]
is name [signature];

```

```

entity Alias_1 is end; architecture Behave of Alias_1 is
begin process variable Nmbr: BIT_VECTOR (31 downto 0);
-- alias declarations to split Nmbr into 3 pieces :
alias Sign : BIT is Nmbr(31);
alias Mantissa : BIT_VECTOR (23downto 0) is Nmbr (30 downto 7);
alias Exponent : BIT_VECTOR ( 6 downto 0) is Nmbr ( 6 downto 0);
begin wait; end process; end; -- the wait prevents an endless cycle

```



```
attribute_declaration ::=  
  attribute identifier:type_name ; | attribute identifier:subtype_name  
;
```

```
entity Attribute_1 is end; architecture Behave of Attribute_1 is  
begin process type COORD is record X, Y : INTEGER; end record;  
attribute LOCATION : COORD; -- the attribute declaration  
begin wait ; -- the wait prevents an endless cycle  
end process; end;
```

You define the attribute properties in an **attribute specification**:

```
attribute LOCATION of adder1 : label is (10,15);  
positionOfComponent := adder1'LOCATION;
```

10.9.4 Predefined Attributes

Predefined attributes for signals				
Attribute	Kind ¹	Parameter T ²	Result type ³	Result/restrictions
S'DELAYED [(T)]	S	TIME	base(S)	S delayed by time T
S'STABLE [(T)]	S	TIME	BOOLEAN	TRUE if no event on S for time T
S'QUIET [(T)]	S	TIME	BOOLEAN	TRUE if S is quiet for time T
S'TRANSACTION	S		BIT	Toggles each cycle if S becomes active
S'EVENT	F		BOOLEAN	TRUE when event occurs on S
S'ACTIVE	F		BOOLEAN	TRUE if S is active
S'LAST_EVENT	F		TIME	Elapsed time since the last event on S
S'LAST_ACTIVE	F		TIME	Elapsed time since S was active
S'LAST_VALUE	F		base(S)	Previous value of S, before last event ⁴
S'DRIVING	F		BOOLEAN	TRUE if every element of S is driven ⁵
S'DRIVING_VALUE	F		base(S)	Value of the driver for S in the current process ⁵

¹ F=function, S=signal.

²Time T ≥ 0 ns. The default, if T is not present, is T=0 ns.

³base(S)=base type of S.

⁴VHDL-93 returns last value of each signal in array separately as an aggregate, VHDL-87 returns the last value of the composite signal.

⁵VHDL-93 only.

Predefined attributes for scalar and array types					
Attribute	Kind¹	Prefix T, A, E²	Parameter X or N³	Result type⁴	Result
T'BASE	T	any		base(T)	base(T), use only with other attribute
T'LEFT	V	scalar		T	Left bound of T
T'RIGHT	V	scalar		T	Right bound of T
T'HIGH	V	scalar		T	Upper bound of T
T'LOW	V	scalar		T	Lower bound of T
T'ASCENDING	V	scalar		BOOLEAN	True if range of T is ascending ⁵
T'IMAGE(X)	F	scalar	base(T)	STRING	String representation of X in T ⁴
T'VALUE(X)	F	scalar	STRING	base(T)	Value in T with representation X ⁴
T'POS(X)	F	discrete	base(T)	UI	Position number of X in T (starts at 0)
T'VAL(X)	F	discrete	UI	base(T)	Value of position X in T
T'SUCC(X)	F	discrete	base(T)	base(T)	Value of position X in T plus one
T'PRED(X)	F	discrete	base(T)	base(T)	Value of position X in T minus one
T'LEFTOF(X)	F	discrete	base(T)	base(T)	Value to the left of X in T
T'RIGHTOF(X)	F	discrete	base(T)	base(T)	Value to the right of X in T
A'LEFT[(N)]	F	array	UI	T(Result)	Left bound of index N of array A
A'RIGHT[(N)]	F	array	UI	T(Result)	Right bound of index N of array A
A'HIGH[(N)]	F	array	UI	T(Result)	Upper bound of index N of array A
A'LOW[(N)]	F	array	UI	T(Result)	Lower bound of index N of array A
A'RANGE[(N)]	R	array	UI	T(Result)	Range A'LEFT(N) to A'RIGHT(N) ⁶
A'REVERSE_RANGE[(N)]	R	array	UI	T(Result)	Opposite range to A'RANGE[(N)]
A'LENGTH[(N)]	V	array	UI	UI	Number of values in index N of array A
A'ASCENDING[(N)]	V	array	UI	BOOLEAN	True if index N of A is ascending ⁴
E'SIMPLE_NAME	V	name		STRING	Simple name of E ⁴
E'INSTANCE_NAME	V	name		STRING	Path includes instantiated entities ⁴
E'PATH_NAME	V	name		STRING	Path excludes instantiated entities ⁴

¹T=Type, F=Function, V=Value, R=Range.

²any=any type or subtype, scalar=scalar type or subtype, discrete=discrete or physical type or subtype, name=entity name=identifier, character literal, or operator symbol.

³base(T)=base type of T, T=type of T, UI= universal_integer, T(Result)=type of object described in result column.

⁴base(T)=base type of T, T=type of T, UI= universal_integer, T(Result)=type of object described in result column.

⁵Only available in VHDL-93. For 'ASCENDING all enumeration types are ascending.

⁶Or reverse for descending ranges.

10.10 Sequential Statements

```

sequential_statement ::=
  wait_statement | assertion_statement
| signal_assignment_statement
| variable_assignment_statement | procedure_call_statement
| if_statement | case_statement | loop_statement
| next_statement | exit_statement
| return_statement | null_statement | report_statement

```

10.10.1 Wait Statement

Key terms and concepts: **suspending** (stopping) a process or procedure • sensitivity to events (changes) on **static** signals • **sensitivity clause** contains **sensitivity list** after **on** • process **resumes** at event on signal in the **sensitivity set** • **condition clause** after **until** • **timeout** (after **for**)

wait on light

makes you wait until a traffic light changes (any change)

wait until light = green

makes you wait (even at a green light) until the traffic signal changes to green

if light = (red **or** yellow) **then wait until** light = green; **end if**;

describes the basic rules at a traffic intersection

```

wait_statement ::= [label:] wait [sensitivity_clause]
  [condition_clause] [timeout_clause] ;
sensitivity_clause ::= on sensitivity_list
sensitivity_list ::= signal_name { , signal_name }
condition_clause ::= until condition
condition ::= boolean_expression
timeout_clause ::= for time_expression

```

```

wait_statement ::= [label:] wait
  [on signal_name { , signal_name }]
  [until boolean_expression]
  [for time_expression] ;

```

```
entity DFF is port (CLK, D : BIT; Q : out BIT); end;
```

--1

```
architecture Behave of DFF is
```

--2

```

process begin wait until Clk = '1'; Q <= D ; end process;      --3
end;                                                            --4

```

```

entity Wait_1 is port (Clk, s1, s2 :in BIT); end;
architecture Behave of Wait_1 is
signal x : BIT_VECTOR (0 to 15);
begin process variable v : BIT; begin
wait; -- Wait forever, stops simulation.
wait on s1 until s2 = '1'; -- Legal, but s1, s2 are signals so
-- s1 is in sensitivity list, and s2 is not in the sensitivity set.
-- Sensitivity set is s1 and process will not resume at event on
s2.
wait on s1, s2; -- resumes at event on signal s1 or s2.
wait on s1 for 10 ns; -- resumes at event on s1 or after 10
ns.
wait on x; -- resumes when any element of array x
-- has an event.
-- wait on x(1 to v); -- Illegal, nonstatic name, since v is a
variable.
end process;
end;

```

```

entity Wait_2 is port (Clk, s1, s2:in BIT); end;
architecture Behave of Wait_2 is
begin process variable v : BIT; begin
wait on Clk; -- resumes when Clk has an event: rising or falling.
wait until Clk = '1'; -- resumes on rising edge.
wait on Clk until Clk = '1'; -- equivalent to the last statement.
wait on Clk until v = '1';
-- The above is legal, but v is a variable so
-- Clk is in sensitivity list, v is not in the sensitivity set.
-- Sensitivity set is Clk and process will not resume at event on
v.
wait on Clk until s1 = '1';
-- The above is legal, but s1 is a signal so
-- Clk is in sensitivity list, s1 is not in the sensitivity set.
-- Sensitivity set is Clk, process will not resume at event on s1.
end process;
end;

```

10.10.2 Assertion and Report Statements

```
assertion_statement ::= [label:] assert
boolean_expression [report expression] [severity expression] ;
```

```
report_statement
::= [label:] report expression [severity expression] ;
```

```
entity Assert_1 is port (I:INTEGER:=0); end;
architecture Behave of Assert_1 is
  begin process begin
    assert (I > 0) report "I is negative or zero";wait;
  end process;
end;
```

10.10.3 Assignment Statements

Key terms and concepts: A **variable assignment statement** updates immediately • A **signal assignment statement** schedules a future assignment • **simulation cycle** • **delta cycle** • **delta time** • **delta**, • **event** • delay models: transport and inertial delay (the default) • pulse rejection limit

```
variable_assignment_statement ::=
  [label:] name|aggregate := expression ;
```

```
entity Var_Assignment is end;
architecture Behave of Var_Assignment is
  signal s1 : INTEGER := 0;
  begin process variable v1,v2 : INTEGER := 0;begin
    assert (v1/=0) report "v1 is 0" severity note ; -- this prints
    v1 := v1 + 1; -- after this statement v1 is 1
    assert (v1=0) report "v1 isn't 0" severity note ; -- this prints
    v2 := v2 + s1; -- signal and variable types must match
  wait;
  end process;
end;
```

```
signal_assignment_statement ::=
  [label:] target <=
  [transport | [reject time_expression] inertial] waveform ;
```

```
entity Sig_Assignment_1 is end;
architecture Behave of Sig_Assignment_1 is
  signal s1,s2,s3 : INTEGER := 0;
  begin process variable v1 : INTEGER := 1; begin
    assert (s1 /= 0) report "s1 is 0" severity note ; -- this prints.
    s1 <= s1 + 1; -- after this statement s1 is still 0.
    assert (s1 /= 0) report "s1 still 0" severity note ; -- this
prints.
    wait;
  end process;
end;
```

```
entity Sig_Assignment_2 is end;
architecture Behave of Sig_Assignment_2 is
  signal s1, s2, s3 : INTEGER := 0;
  begin process variable v1 : INTEGER := 1; begin
    -- s1, s2, s3 are initially 0; now consider the following:
    s1 <= 1 ; -- schedules updates to s1 at end of 0 ns cycle.
    s2 <= s1; -- s2 is 0, not 1.
    wait for 1 ns;
    s3 <= s1; -- now s3 will be 1 at 1 ns.
    wait;
  end process;
end;
```

```
entity Transport_1 is end;
architecture Behave of Transport_1 is
  signal s1, SLOW, FAST, WIRE : BIT := '0';
  begin process begin
    s1 <= '1' after 1 ns, '0' after 2 ns, '1' after 3 ns ;
    -- schedules s1 to be '1' at t+1 ns, '0' at t+2 ns, '1' at t+3 ns
    wait; end process;
  -- inertial delay: SLOW rejects pulsewidths less than 5ns:
  process (s1) begin SLOW <= s1 after 5 ns ; end process;
  -- inertial delay: FAST rejects pulsewidths less than 0.5ns:
  process (s1) begin FAST <= s1 after 0.5 ns ; end process;
  -- transport delay: WIRE passes all pulsewidths...
```



```
process (s1) begin WIRE <= transport s1 after 5 ns ; end process;
end;
```

```
process (s1) begin RJCT <= reject 2 ns s1 after 5 ns ; end process;
```

10.10.4 Procedure Call

```
procedure_call_statement ::=
  [label:] procedure_name [(parameter_association_list)];
```

```
package And_Pkg is
  procedure V_And(a, b : BIT; signal c : out BIT);
  function V_And(a, b : BIT) return BIT;
end;
```

```
package body And_Pkg is
  procedure V_And(a, b : BIT; signal c: out BIT) is
    begin c <= a and b; end;
  function V_And(a, b: BIT) return BIT is
    begin return a and b; end;
end And_Pkg;
```

```
use work.And_Pkg.all; entity Proc_Call_1 is end;
architecture Behave of Proc_Call_1 is signal A, B, Y: BIT := '0';
  begin process begin V_And (A, B, Y); wait; end process;
end;
```

10.10.5 If Statement

```
if_statement ::=
  [if_label:] if boolean_expression then {sequential_statement}
  {elsif boolean_expression then {sequential_statement}}
  [else {sequential_statement}]
  end if [if_label];
```

```
entity If_Then_Else_1 is end;
architecture Behave of If_Then_Else_1 is signal a, b, c: BIT := '1';
```

```

begin process begin
  if c = '1' then c <= a ; else c <= b; end if; wait;
end process;
end;

```

```

entity If_Then_1 is end;
architecture Behave of If_Then_1 is signal A, B, Y : BIT := '1';
  begin process begin
    if A = B then Y <= A; end if; wait;
  end process;
end;

```

10.10.6 Case Statement

```

case_statement ::=
[case_label:] case expression is
  when choice { | choice } => {sequential_statement}
  {when choice { | choice } => {sequential_statement}}
end case [case_label];

```

```

library IEEE; use IEEE.STD_LOGIC_1164 all; --1
entity sm_mealy is --2
  port (reset, clock, i1, i2 : STD_LOGIC; o1, o2 :out STD_LOGIC); --3
end sm_mealy; --4
architecture Behave of sm_mealy is --5
  type STATES is (s0, s1, s2, s3); signal current, new : STATES; --6
begin --7
  synchronous : process (clock, reset) begin --8
    if To_X01(reset) = '0' then current <= s0; --9
    elsif rising_edge(clock) then current <= new; end if; --10
  end process; --11
  combinational : process (current, i1, i2) begin --12
  case current is --13
    when s0 => --14
      if To_X01(i1) = '1' then o2 <='0'; o1 <='0'; new <= s2; --15
      else o2 <= '1'; o1 <= '1'; new <= s1;end if; --16
    when s1 => --17
      if To_X01(i2) = '1' then o2 <='1'; o1 <='0'; new <= s1; --18
      else o2 <='0'; o1 <='1'; new <= s3;end if; --19
    when s2 => --20
      if To_X01(i2) = '1' then o2 <='0'; o1 <='1'; new <= s2; --21

```

```
    else o2 <= '1'; o1 <= '0'; new <= s0;end if;           --22
  when s3 => o2 <= '0'; o1 <= '0'; new <= s0;           --23
  when others => o2 <= '0'; o1 <= '0'; new <= s0;     --24
end case;                                           --25
end process;                                       --26
end Behave;                                         --27
```

10.10.7 Other Sequential Control Statements

```

loop_statement ::=
[loop_label:]
[while boolean_expression|for identifier in discrete_range]
loop
  {sequential_statement}
end loop [loop_label];

```

```

package And_Pkg is function V_And(a, b : BIT) return BIT; end;

```

```

package body And_Pkg is function V_And(a, b : BIT) return BIT is
  begin return a and b; end; end And_Pkg;

```

```

entity Loop_1 is port (x, y : in BIT := '1'; s : out BIT := '0');
end;
use work.And_Pkg.all;
architecture Behave of Loop_1 is
  begin loop
    s <= V_And(x, y); wait on x, y;
  end loop;
end;

```

The **next statement** [VHDL LRM8.10] forces completion of current loop iteration:

```

next_statement ::=
[label:] next [loop_label] [when boolean_expression];

```

An **exit statement** [VHDL LRM8.11] forces an exit from a loop.

```

exit_statement ::=
[label:] exit [loop_label] [when condition] ;

```

```
loop wait on Clk; exit when Clk = '0'; end loop;  
-- equivalent to: wait until Clk = '0';
```

The **return statement** [VHDL LRM8.12] completes execution of a procedure or function:

```
return_statement ::= [label:] return [expression];
```

A **null statement** [VHDL LRM8.13] does nothing:

```
null_statement ::= [label:] null;
```

10.11 Operators

VHDL predefined operators (listed by increasing order of precedence)

logical_operator ::=	and or nand nor xor <u>xnor</u>
relational_operator ::=	= /= < <= > >=
shift_operator ::=	<u>sll</u> <u>srl</u> <u>sla</u> <u>sra</u> <u>rol</u> <u>ror</u>
adding_operator ::=	+ - &
sign ::=	+ -
multiplying_operator ::=	* / mod rem
miscellaneous_operator ::=	** abs not

```

entity Operator_1 is end; architecture Behave of Operator_1 is           --1
begin process                                                         --2
variable b : BOOLEAN; variable bt : BIT := '1'; variable i : INTEGER; --3
variable pi : REAL := 3.14; variable epsilon : REAL := 0.01;         --4
variable bv4 : BIT_VECTOR (3 downto 0) := "0001";                   --5
variable bv8 : BIT_VECTOR (0 to 7);                                   --6
begin                                                                   --7

b := "0000" < bv4; -- b is TRUE, "0000" treated as BIT_VECTOR.      --8
b := 'f' > 'g';      -- b is FALSE, 'dictionary' comparison.        --9
bt := '0' and bt;   -- bt is '0', analyzer knows '0' is BIT.      --10
bv4 := not bv4;     -- bv4 is now "1110".                             --11
i := 1 + 2;         -- Addition, must be compatible types.          --12
i := 2 ** 3;       -- Exponentiation, exponent must be integer.    --13
i := 7/3;          -- Division, L/R rounded towards zero, i=2.     --14
i := 12 rem 7;     -- Remainder, i=5. In general:                   --15
                    -- L rem R = L-((L/R)*R).                       --16
i := 12 mod 7;    -- modulus, i=5. In general:                      --17
                    -- L mod R = L-(R*N) for an integer N.         --18

-- shift := sll | srl | sla | sra | rol | ror (VHDL-93 only)       --19
bv4 := "1001" srl 2; -- Shift right logical, now bv4="0100".      --20
-- Logical shift fills with T'LEFT.                                 --21
bv4 := "1001" sra 2; -- Shift right arithmetic, now bv4="0111".  --22
-- Arithmetic shift fills with element at end being vacated.     --23
bv4 := "1001" ror 2; -- Rotate right, now bv4="0110".           --24
-- Rotate wraps around.                                           --25
-- Integer argument to any shift operator may be negative or zero.--26

```

```

if (pi*2.718)/2.718 = 3.14then wait; end if; -- This is unreliable.--27
if (abs((pi*2.718)/2.718)-3.14)<epsilon)then wait; end if; --
Better. --28
bv8 := bv8(1 to 7) & bv8(0); -- Concatenation, a left rotation. --29
wait; end process; --30
end; --31

```

10.12 Arithmetic

Key terms and concepts: **type checking** • **range checking** • **type conversion** between closely related types • **type_mark(expression)** type qualification and disambiguation (to persuade the analyzer) • **type_mark'(expression)**

```

entity Arithmetic_1 is end; architecture Behave of Arithmetic_1 is --1
  begin process
    variable i : INTEGER := 1; variable r : REAL := 3.33; --2
    variable b : BIT := '1'; --3
    variable bv4 : BIT_VECTOR (3 downto 0) := "0001"; --4
    variable bv8 : BIT_VECTOR (7 downto 0) := B"1000_0000"; --5
    begin --6
--      i := r; -- you can't assign REAL to INTEGER. --7
--      bv4 := bv4 + 2; -- you can't add BIT_VECTOR and INTEGER. --8
--      bv4 := '1'; -- you can't assign BIT to BIT_VECTOR. --9
--      bv8 := bv4; -- an error, the arrays are different sizes. --10
r := REAL(i); -- OK, uses a type conversion. --11
i := INTEGER(r); -- OK (0.5 rounds up or down). --12
bv4 := "001" & '1'; -- OK, you can mix an array and a scalar. --13
bv8 := "0001" & bv4; -- OK, if arguments are correct lengths. --14
wait; end process; end; --15

entity Arithmetic_2 is end; architecture Behave of Arithmetic_2 is --1
type TC is range 0 to 100; -- Type INTEGER. --2
type TF is range 32 to 212; -- Type INTEGER. --3
subtype STC is INTEGER range 0 to 100; -- Subtype of type INTEGER. --4
subtype STF is INTEGER range 32 to 212; -- Base type is INTEGER. --5
begin process --6
variable t1 : TC := 25; variable t2 : TF := 32; --7
variable st1 : STC := 25; variable st2 : STF := 32; --8
begin --9
--      t1 := t2; -- Illegal, different types. --10
--      t1 := st1; -- Illegal, different types and subtypes. --11

```

```

    st2 := st1;           -- OK to use same base types.           --12
    st2 := st1 + 1;      -- OK to use subtype and base type.    --13
-- st2 := 213;          -- Error, outside range at analysis time. --14
-- st2 := 212 + 1;     -- Error, outside range at analysis time. --15
    st1 := st1 + 100;   -- Error, outside range at initialization. --16
wait; end process; end;

```

```

entity Arithmetic_3 is end; architecture Behave of Arithmetic_3 is --1
type TYPE_1 is array (INTEGER range 3 downto 0) of BIT; --2
type TYPE_2 is array (INTEGER range 3 downto 0) of BIT; --3
subtype SUBTYPE_1 is BIT_VECTOR (3 downto 0); --4
subtype SUBTYPE_2 is BIT_VECTOR (3 downto 0); --5
begin process --6
variable bv4 : BIT_VECTOR (3 downto 0) := "0001"; --7
variable st1 : SUBTYPE_1 := "0001"; variable t1 : TYPE_1 := "0001"; --8
variable st2 : SUBTYPE_2 := "0001"; variable t2 : TYPE_2 := "0001"; --9
begin --10
    bv4 := st1;           -- OK, compatible type and subtype.    --11
-- bv4 := t1;           -- Illegal, different types.            --12
    bv4 := BIT_VECTOR(t1); -- OK, type conversion.              --13
    st1 := bv4;          -- OK, compatible subtype & base type.  --14
-- st1 := t1;          -- Illegal, different types.            --15
    st1 := SUBTYPE_1(t1); -- OK, type conversion.              --16
-- t1 := st1;          -- Illegal, different types.            --17
-- t1 := bv4;          -- Illegal, different types.            --18
    t1 := TYPE_1(bv4);   -- OK, type conversion.              --19
-- t1 := t2;           -- Illegal, different types.            --20
    t1 := TYPE_1(t2);   -- OK, type conversion.              --21
    st1 := st2;         -- OK, compatible subtypes.            --22
wait; end process; end; --23

```

10.12.1 IEEE Synthesis Packages

```

package Part_NUMERIC_BIT is
type UNSIGNED is array (NATURAL range <> ) of BIT;
type SIGNED is array (NATURAL range <> ) of BIT;
function "+" (L, R : UNSIGNED) return UNSIGNED;
-- other function definitions that overload +, -, =, >, and so on.
end Part_NUMERIC_BIT;

```

```

package body Part_NUMERIC_BIT is
constant NAU : UNSIGNED(0 downto 1) := (others => '0'); -- Null array.

```



```

constant NAS : SIGNED(0 downto 1):=(others => '0'); -- Null array.
constant NO_WARNING : BOOLEAN := FALSE; -- Default to emit warnings.

```

```

function MAX (LEFT, RIGHT : INTEGER)return INTEGER is
begin -- Internal function used to find longest of two inputs.
if LEFT > RIGHT then return LEFT; else return RIGHT; end if; end MAX;

```

```

function ADD_UNSIGNED (L, R : UNSIGNED; C: BIT)return UNSIGNED is
constant L_LEFT : INTEGER := L'LENGTH-1; -- L, R must be same length.
alias XL : UNSIGNED(L_LEFTdownto 0) is L; -- Descending alias,
alias XR : UNSIGNED(L_LEFTdownto 0) is R; -- aligns left ends.
variable RESULT : UNSIGNED(L_LEFTdownto 0); variable CBIT : BIT :=
C;
begin for I in 0 to L_LEFT loop -- Descending alias allows loop.
RESULT(I) := CBIT xor XL(I) xor XR(I); -- CBIT = carry, initially =
C.
CBIT := (CBIT and XL(I)) or (CBIT and XR(I)) or (XL(I) and XR(I));
end loop; return RESULT; end ADD_UNSIGNED;

```

```

function RESIZE (ARG : UNSIGNED; NEW_SIZE : NATURAL)return UNSIGNED
is
constant ARG_LEFT : INTEGER := ARG'LENGTH-1;
alias XARG : UNSIGNED(ARG_LEFTdownto 0) is ARG; -- Descending range.
variable RESULT : UNSIGNED(NEW_SIZE-1downto 0) := (others => '0');
begin -- resize the input ARG to length NEW_SIZE
if (NEW_SIZE < 1) then return NAU; end if; -- Return null array.
if XARG'LENGTH = 0 then return RESULT; end if; -- Null to empty.
if (RESULT'LENGTH < ARG'LENGTH) then -- Check lengths.
RESULT(RESULT'LEFTdownto 0) := XARG(RESULT'LEFTdownto 0);
else -- Need to pad the result with some '0's.
RESULT(RESULT'LEFTdownto XARG'LEFT + 1) := (others => '0');
RESULT(XARG'LEFTdownto 0) := XARG;
end if; return RESULT;
end RESIZE;

```

```

function "+" (L, R : UNSIGNED) return UNSIGNED is -- Overloaded '+'.
constant SIZE : NATURAL := MAX(L'LENGTH, R'LENGTH);
begin -- If length of L or R < 1 return a null array.
if ((L'LENGTH < 1) or (R'LENGTH < 1)) then return NAU; end if;
return ADD_UNSIGNED(RESIZE(L, SIZE), RESIZE(R, SIZE), '0') end "+";
end Part_NUMERIC_BIT;

```

```

function TO_INTEGER (ARG : UNSIGNED) return NATURAL;
function TO_INTEGER (ARG : SIGNED) return INTEGER;
function TO_UNSIGNED (ARG, SIZE : NATURAL) return UNSIGNED;
function TO_SIGNED (ARG : INTEGER; SIZE : NATURAL) return SIGNED;
function RESIZE (ARG : SIGNED; NEW_SIZE : NATURAL) return SIGNED;
function RESIZE (ARG : UNSIGNED; NEW_SIZE : NATURAL) return UNSIGNED;
-- set XMAP to convert unknown values, default is 'X'-'>'0'
function TO_01(S : UNSIGNED; XMAP : STD_LOGIC := '0') return
UNSIGNED;
function TO_01(S : SIGNED; XMAP : STD_LOGIC := '0') return SIGNED;

```

```

library IEEE; use IEEE.STD_LOGIC_1164 all;
package Part_NUMERIC_STD is
type UNSIGNED is array (NATURAL range <>) of STD_LOGIC;
type SIGNED is array (NATURAL range <>) of STD_LOGIC;
end Part_NUMERIC_STD;

```

```

-- function STD_MATCH (L, R: T) return BOOLEAN;
-- T = STD_ULOGIC UNSIGNED SIGNED STD_LOGIC_VECTOR STD_ULOGIC_VECTOR

```

```

type BOOLEAN_TABLE is array(STD_ULOGIC, STD_ULOGIC) of BOOLEAN;
constant MATCH_TABLE : BOOLEAN_TABLE := (

```

```

-----
-- U      X      0      1      Z      W      L      H      -
-----
( FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE ), -- | U |
( FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE ), -- | X |
( FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, TRUE, FALSE, TRUE ), -- | 0 |
( FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, TRUE, TRUE ), -- | 1 |
( FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE ), -- | Z |
( FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE ), -- | W |
( FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, TRUE, FALSE, TRUE ), -- | L |
( FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, TRUE, TRUE ), -- | H |
( TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE ); -- | - |

```

```

IM_TRUE = STD_MATCH(STD_LOGIC_VECTOR ("10HLXWZ-"),
                    STD_LOGIC_VECTOR ("HL10----")) -- is TRUE

```

```

entity Counter_1 is end; --1
library STD; use STD.TEXTIO.all; --2

```

```
library IEEE; use IEEE.STD_LOGIC_1164 all;           --3
use work.NUMERIC_STD all;                             --4
architecture Behave_2 of Counter_1 is                --5
    signal Clock : STD_LOGIC := '0';                 --6
    signal Count : UNSIGNED (2 downto 0) := "000";   --7
begin                                                --8
    process begin                                    --9
        wait for 10 ns; Clock <= not Clock;         --10
        if (now > 340 ns) then wait;                --11
        end if;                                      --12
    end process;                                     --13
    process begin                                    --14
        wait until (Clock = '0');                    --15
        if (Count = 7)                               --16
            then Count <= "000";                     --17
            else Count <= Count + 1;                 --18
        end if;                                      --19
    end process;                                     --20
    process (Count) variable L: LINE; begin write(L, now); --21
    write(L, STRING(" Count=")); write(L, TO_INTEGER(Count)); --22
    writeline(output, L);                            --23
    end process;                                     --24
end;                                                 --25
```

10.13 Concurrent Statements

```
concurrent_statement ::=
  block_statement
  | process_statement
  | [ label : ] [ postponed ] procedure_call ;
  | [ label : ] [ postponed ] assertion ;
  | [ label : ] [ postponed ] conditional_signal_assignment
  | [ label : ] [ postponed ] selected_signal_assignment
  | component_instantiation_statement
  | generate_statement
```

10.13.1 Block Statement

Key terms and concepts: **guard expression** • GUARD • **guarded signals (register and bus)** • **driver** • **disconnected** • **disconnect** statement

```
block_statement ::=
  block_label: block [(guard_expression)] [is]
    [generic (generic_interface_list);
    [generic map (generic_association_list);]]
    [port (port_interface_list);
    [port map (port_association_list);]]
    {block_declarative_item}
    begin
    {concurrent_statement}
  end block [block_label] ;
```

```
library ieee; use ieee.std_logic_1164 all;
entity bus_drivers is end;
```

```
architecture Structure_1 of bus_drivers is
signal TSTATE: STD_LOGIC bus; signal A, B, OEA, OEB : STD_LOGIC :=
'0';
begin
process begin OEA <= '1' after 100 ns, '0' after 200 ns;
OEB <= '1' after 300 ns; wait; end process;
B1 : block (OEA = '1')
disconnect all : STD_LOGIC after 5 ns; -- Only needed for float time.
```

```

begin TSTATE <= guarded not A after 3 ns; end block;
B2 : block (OEB = '1')
disconnect all : STD_LOGIC after 5 ns; -- Float time = 5 ns.
begin TSTATE <= guarded not B after 3 ns; end block;
end;

```

```

architecture Structure_2 of bus_drivers is
signal TSTATE : STD_LOGIC; signal A, B, OEA, OEB : STD_LOGIC := '0';
begin
process begin
OEA <= '1' after 100 ns, '0' after 200 ns; OEB <= '1' after 300 ns;
wait; end process;
process(OEA, OEB, A, B) begin
if (OEA = '1') then TSTATE <= not A after 3 ns;
elsif (OEB = '1') then TSTATE <= not B after 3 ns;
else TSTATE <= 'Z' after 5 ns;
end if;
end process;
end;

```

10.13.2 Process Statement

Key terms and concepts: process **sensitivity set** • process execution occurs during a **simulation cycle**—made up of **delta cycles**

```

process_statement ::=
[process_label:]
[postponed] process [(signal_name {, signal_name})]
[is] {subprogram_declaration | subprogram_body
| type_declaration | subtype_declaration
| constant_declaration | variable_declaration
| file_declaration | alias_declaration
| attribute_declaration | attribute_specification
| use_clause
| group_declaration | group_template_declaration}
begin
{sequential_statement}
end [postponed] process [process_label];

```

```
entity Mux_1 is port (i0, i1, sel : in BIT := '0'; y : out BIT); end;
architecture Behave of Mux_1 is
  begin process (i0, i1, sel) begin -- i0, i1, sel = sensitivity set
    case sel is when '0' => y <= i0; when '1' => y <= i1; end case;
  end process; end;
```

```
entity And_1 is port (a, b : in BIT := '0'; y : out BIT); end;
architecture Behave of And_1 is
  begin process (a, b) begin y <= a and b; end process; end;
```

```
entity FF_1 is port (clk, d: in BIT := '0'; q : out BIT); end;
architecture Behave of FF_1 is
  begin process (clk) begin
    if clk'EVENT and clk = '1' then q <= d; end if;
  end process; end;
```

```
entity FF_2 is port (clk, d: in BIT := '0'; q : out BIT); end;
architecture Behave of FF_2 is
  begin process begin -- The equivalent process has a wait at the end:
    if clk'event and clk = '1' then q <= d; end if; wait on clk;
  end process; end;
```

```
entity FF_3 is port (clk, d: in BIT := '0'; q : out BIT); end;
architecture Behave of FF_3 is
  begin process begin -- No sensitivity set with a wait statement.
    wait until clk = '1'; q <= d;
  end process; end;
```

10.13.3 Concurrent Procedure Call

```
package And_Pkg is procedure V_And(a,b:BIT; signal c:out BIT); end;
```

```
package body And_Pkg is procedure V_And(a,b:BIT; signal c:out BIT) is
  begin c <= a and b; end; end And_Pkg;
```

```
use work.And_Pkg.all; entity Proc_Call_2 is end;
architecture Behave of Proc_Call_2 is signal A, B, Y : BIT := '0';
```

```

begin V_And (A, B, Y); -- Concurrent procedure call.
process begin wait; end process; -- Extra process to stop.
end;

```

10.13.4 Concurrent Signal Assignment

Key terms and concepts:

There are two forms of **concurrent signal assignment statement**:

A **selected signal assignment statement** is equivalent to a case statement inside a process statement [VHDL LRM9.5.2].

A **conditional signal assignment statement** is, in its most general form, equivalent to an if statement inside a process statement [VHDL LRM9.5.1].

```

selected_signal_assignment ::=
  with expression select
    name|aggregate <= guarded]
      [transport|reject time_expression] inertial]
        waveform when choice { | choice }
        { , waveform when choice { | choice } } ;

```

```

entity Selected_1 is end; architecture Behave of Selected_1 is
signal y,i1,i2 : INTEGER; signal sel : INTEGER range 0 to 1;
begin with sel select y <= i1 when 0, i2 when 1; end;

```

```

entity Selected_2 is end; architecture Behave of Selected_2 is
signal i1,i2,y : INTEGER; signal sel : INTEGER range 0 to 1;
begin process begin
  case sel is when 0 => y <= i1; when 1 => y <= i2; end case;
  wait on i1, i2;
end process; end;

```

```

conditional_signal_assignment ::=
  name|aggregate <= guarded]
  [transport|reject time_expression] inertial]
    {waveform when boolean_expression else}
    waveform [when boolean_expression];

```

```
entity Conditional_1 is end; architecture Behave of Conditional_1 is
signal y,i,j : INTEGER; signal clk : BIT;
begin y <= i when clk = '1' else j; -- conditional signal assignment
end;
```

```
entity Conditional_2 is end; architecture Behave of Conditional_2 is
signal y,i : INTEGER; signal clk : BIT;
begin process begin
  if clk = '1' then y <= i; else y <= y; end if; wait on clk;
end process; end;
```

A concurrent signal assignment statement can look like a sequential signal assignment statement:

```
entity Assign_1 is end; architecture Behave of Assign_1 is
signal Target, Source : INTEGER;
  begin Target <= Source after 1 ns; -- looks like signal assignment
end;
```

Here is the equivalent process:

```
entity Assign_2 is end; architecture Behave of Assign_2 is
signal Target, Source : INTEGER;
begin process begin
  Target <= Source after 1 ns; wait on Source;
end process; end;
```

```
entity Assign_3 is end; architecture Behave of Assign_3 is
signal Target, Source : INTEGER; begin process begin
  wait on Source; Target <= Source after 1 ns;
end process; end;
```

10.13.5 Concurrent Assertion Statement

A **concurrent assertion statement** is equivalent to a passive process statement (without a sensitivity list) that contains an assertion statement followed by a wait statement.


```
concurrent_assertion_statement
 ::= [ label : ] [ postponed ] assertion ;
```

If the assertion condition contains a signal, then the equivalent `process` statement will include a final `wait` statement with a sensitivity clause.

A concurrent assertion statement with a condition that is static expression is equivalent to a `process` statement that ends in a `wait` statement that has no sensitivity clause.

The equivalent process will execute once, at the beginning of simulation, and then wait indefinitely.

10.13.6 Component Instantiation

```
component_instantiation_statement ::=
 instantiation_label:
  [component] component_name
  [entity entity_name [(architecture_identifier)]]
  [configuration configuration_name
   [generic map (generic_association_list)
    port map (port_association_list)] ;
```

```
entity And_2 is port (i1, i2 : in BIT; y : out BIT); end;
architecture Behave of And_2 is begin y <= i1 and i2; end;
entity Xor_2 is port (i1, i2 : in BIT; y : out BIT); end;
architecture Behave of Xor_2 is begin y <= i1 xor i2; end;
```

```
entity Half_Adder_2 is port (a,b : BIT := '0'; sum, cry :out BIT);
end;
architecture Netlist_2 of Half_Adder_2 is
use work.all; -- need this to see the entities Xor_2 and And_2
begin
  X1 : entity Xor_2(Behave) port map (a, b, sum); -- VHDL-93 only
  A1 : entity And_2(Behave) port map (a, b, cry); -- VHDL-93 only
end;
```

10.13.7 Generate Statement

```

generate_statement ::=
generate_label: for generate_parameter_specification
                | if boolean_expression
generate [{block_declarative_item} begin]
  {concurrent_statement}
end generate [generate_label] ;

```

```

entity Full_Adder is port (X, Y, Cin : BIT; Cout, Sum: out BIT); end;
architecture Behave of Full_Adder is begin Sum <= X xor Y xor Cin;
Cout <= (X and Y) or (X and Cin) or (Y and Cin); end;

```

```

entity Adder_1 is
  port (A, B : in BIT_VECTOR (7 downto 0) := (others => '0'));
  Cin : in BIT := '0'; Sum : out BIT_VECTOR (7 downto 0);
  Cout : out BIT);
end;

```

```

architecture Structure of Adder_1 is use work.all;

```

```

component Full_Adder port (X, Y, Cin: BIT; Cout, Sum: out BIT);
end component;
signal C : BIT_VECTOR(7 downto 0);
begin AllBits : for i in 7 downto 0 generate
  LowBit : if i = 0 generate
    FA : Full_Adder port map (A(0), B(0), Cin, C(0), Sum(0));
  end generate;
  OtherBits : if i /= 0 generate
    FA : Full_Adder port map (A(i), B(i), C(i-1), C(i), Sum(i));
  end generate;
end generate;
Cout <= C(7);
end;

```

For i=6, FA' INSTANCE_NAME is

```

:adder_1(structure):allbits(6):otherbits:fa:

```

10.14 Execution

Key terms and concepts: **sequential execution • concurrent execution • difference between update for signals and variables**

Variables and signals in VHDL	
Variables	Signals
<pre>entity Execute_1 is end; architecture Behave of Execute_1 is begin process variable v1 : INTEGER := 1; variable v2 : INTEGER := 2; begin v1 := v2; -- before: v1 = 1, v2 = 2 v2 := v1; -- after: v1 = 2, v2 = 2 wait; end process; end;</pre>	<pre>entity Execute_2 is end; architecture Behave of Execute_2 is signal s1 : INTEGER := 1; signal s2 : INTEGER := 2; begin process begin s1 <= s2; -- before: s1 = 1, s2 = 2 s2 <= s1; -- after: s1 = 2, s2 = 1 wait; end process; end;</pre>

Concurrent and sequential statements in VHDL		
Concurrent [VHDL LRM9]	Sequential [VHDL LRM8]	
block	wait	case
process	assertion	loop
concurrent_procedure_call	signal_assignment	next
concurrent_assertion	variable_assignment	exit
concurrent_signal_assignment	procedure_call	return
component_instantiation	if	null
generate		

```
entity Sequential_1 is end; architecture Behave of Sequential_1 is
  signal s1, s2 : INTEGER := 0;
begin
  process begin
    s1 <= 1;          -- sequential signal assignment 1
    s2 <= s1 + 1;    -- sequential signal assignment 2
    wait on s1, s2 ;
```

```
    end process;  
end;
```

```
entity Concurrent_1 is end; architecture Behave of Concurrent_1 is  
signal s1, s2 : INTEGER := 0; begin  
    L1 : s1 <= 1;          -- concurrent signal assignment 1  
    L2 : s2 <= s1 + 1; -- concurrent signal assignment 2  
end;
```

```
entity Concurrent_2 is end; architecture Behave of Concurrent_2 is  
signal s1, s2 : INTEGER := 0; begin  
    P1 : process begin s1 <= 1;          wait on s2 ; end process;  
    P2 : process begin s2 <= s1 + 1; wait on s1 ; end process;  
end;
```

10.15 Configurations and Specifications

Key terms and concepts:

A **configuration declaration** defines a configuration—it is a library unit and is one of the basic units of VHDL code.

A **block configuration** defines the configuration of a **block statement** or a design entity. A block configuration appears inside a configuration declaration, a component configuration, or nested in another block configuration.

A **configuration specification** may appear in the declarative region of a generate statement, block statement, or architecture body.

A **component declaration** may appear in the declarative region of a generate statement, block statement, architecture body, or package.

A **component configuration** defines the configuration of a component and appears in a block

configuration.

VHDL binding examples

```
entity AD2 is port (A1, A2: in BIT; Y: out BIT); end;
architecture B of AD2 is begin Y <= A1 and A2; end;
entity XR2 is port (X1, X2: in BIT; Y: out BIT); end;
architecture B of XR2 is begin Y <= X1 xor X2; end;
```

component

declaration

configuration

specification

```
entity Half_Adder is port (X, Y: BIT; Sum, Cout: out BIT);
end;
architecture Netlist of Half_Adder is use work.all;
component MX port (A, B: BIT; Z :out BIT);end component;
component MA port (A, B: BIT; Z :out BIT);end component;
for G1:MX use entity XR2(B) port map(X1 => A,X2 => B,Y =>
Z);
begin
  G1:MX port map(X, Y, Sum); G2:MA port map(X, Y, Cout);
end;
```

configuration

declaration

block

configuration

component

configuration

```
configuration C1 of Half_Adder is
use work.all;
  for Netlist
    for G2:MA
      use entity AD2(B) port map(A1 => A,A2 => B,Y => Z);
    end for;
  end for;
end;
```

VHDL binding	
configuration declaration_	<pre> configuration identifier of <i>entity_name</i> is {<i>use_clause</i> <i>attribute_specification</i> <u><i>group_declaration</i></u>} <i>block_configuration</i> end [configuration] [<i>configuration_identifier</i>]; </pre>
block configuration	<pre> for <i>architecture_name</i> <i>block_statement_label</i> <i>generate_statement_label</i> [(<i>index_specification</i>)] {use <i>selected_name</i> {, <i>selected_name</i>};} {<i>block_configuration</i> <i>component_configuration</i>} end for ; </pre>
configuration specification	<pre> for <i>instantiation_label</i>{, <i>instantiation_label</i>}:<i>component_name</i> others:<i>component_name</i> all:<i>component_name</i> [use entity <i>entity_name</i> [(<i>architecture_identifier</i>)] configuration <i>configuration_name</i> open] [generic map (<i>generic_association_list</i>)] [port map (<i>port_association_list</i>)]; </pre>
component declaration	<pre> component identifier [is] [generic (<i>local_generic_interface_list</i>);] [port (<i>local_port_interface_list</i>);] end component [<u><i>component_identifier</i></u>]; </pre>
component configuration	<pre> for <i>instantiation_label</i> {, <i>instantiation_label</i>}:<i>component_name</i> others:<i>component_name</i> all:<i>component_name</i> [use entity <i>entity_name</i> [(<i>architecture_identifier</i>)] configuration <i>configuration_name</i> open] [generic map (<i>generic_association_list</i>)] [port map (<i>port_association_list</i>)];] [<i>block_configuration</i>] end for; </pre>

10.16 An Engine Controller

A temperature converter

```

library IEEE;
use IEEE.STD_LOGIC_1164.all; -- type STD_LOGIC,
rising_edge
use IEEE.NUMERIC_STD.all ; -- type UNSIGNED, "+", "/"
entity tconv is generic TPD : TIME:= 1 ns;
  port (T_in : in UNSIGNED(11 downto 0);
        clk, rst : in STD_LOGIC; T_out : out UNSIGNED(11
downto 0));
end;
architecture rtl of tconv is
signal T : UNSIGNED(7 downto 0);
constant T2  : UNSIGNED(1 downto 0) := "10" ;
constant T4  : UNSIGNED(2 downto 0) := "100" ;
constant T32 : UNSIGNED(5 downto 0) := "100000" ;
begin
  process(T) begin T_out <= T + T/T2 + T/T4 + T32 after
TPD;
  end process;
end rtl;

```

T_in = temperature in °C

T_out = temperature in °F

The conversion formula from Centigrade to Fahrenheit is:

$$T(^{\circ}\text{F}) = (9/5) \times T(^{\circ}\text{C}) + 32$$

This converter uses the approximation:

$$9/5 \quad 1.75 = 1 + 0.5 + 0.25$$

A digital filter

```

library IEEE;
use IEEE.STD_LOGIC_1164.all; -- STD_LOGIC type,
rising_edge
use IEEE.NUMERIC_STD.all; -- UNSIGNED type, "+" and "/"
entity filter is
  generic TPD : TIME := 1 ns;
  port (T_in : in UNSIGNED(11 downto 0);
        rst, clk : in STD_LOGIC;
        T_out: out UNSIGNED(11 downto 0));
end;
architecture rtl of filter is
type arr is array (0 to 3) of UNSIGNED(11 downto 0);
signal i : arr ;
constant T4 : UNSIGNED(2 downto 0) := "100";
begin
  process(rst, clk) begin
    if (rst = '1') then
      for n in 0 to 3 loop i(n) <= (others =>'0') after
TPD;
      end loop;
    else
      if(rising_edge(clk)) then
        i(0) <= T_in after TPD;i(1) <= i(0) after TPD;
        i(2) <= i(1) after TPD;i(3) <= i(2) after TPD;
        end if;
      end if;
    end process;
    process(i) begin
      T_out <= ( i(0) + i(1) + i(2) + i(3) )/T4 after
TPD;
    end process;
end rtl;

```

The filter computes a moving average over four successive samples in time.

Notice

$i(0)$ $i(1)$ $i(2)$ $i(3)$
are each 12 bits wide.

Then the sum

$i(0) + i(1) + i(2) + i(3)$
is 14 bits wide, and the
average

$(i(0) + i(1) + i(2) + i(3))/T4$

is 12 bits wide.

All delays are generic TPD.

The input register

```

library IEEE;
use IEEE.STD_LOGIC_1164.all; -- type STD_LOGIC, rising_edge
use IEEE.NUMERIC_STD.all ; -- type UNSIGNED
entity register_in is
generic ( TPD : TIME := 1 ns);
port (T_in : in UNSIGNED(11 downto 0));
clk, rst : in STD_LOGIC; T_out : out UNSIGNED(11 downto 0));
end;
architecture rtl of register_in is
begin
  process(clk, rst) begin
    if (rst = '1') then T_out <= (others => '0') after TPD;
    else
      if (rising_edge(clk)) then T_out <= T_in after TPD; end
if;
    end if;
  end process;
end rtl ;

```

12-bit-wide register for the temperature input signals.

If the input is asynchronous (from an A/D converter with a separate clock, for example), we would need to worry about metastability.

All delays are generic TPD.

A first-in, first-out stack (FIFO)

```

library IEEE; use IEEE.NUMERIC_STD.all ; -- UNSIGNED type
use ieee.std_logic_1164.all; -- STD_LOGIC type, rising_edge
entity fifo is
  generic (width : INTEGER := 12; depth : INTEGER := 16);
  port (clk, rst, push, pop : STD_LOGIC;
    Di : in UNSIGNED (width-1 downto 0);
    Do : out UNSIGNED (width-1 downto 0);
    empty, full : out STD_LOGIC);
end fifo;
architecture rtl of fifo is
  subtype ptype is INTEGER range 0 to (depth-1);
  signal diff, Ai, Ao : ptype; signal f, e : STD_LOGIC;
  type a is array (ptype) of UNSIGNED(width-1 downto 0);
  signal mem : a ;
  function bump(signal ptr : INTEGER range 0 to (depth-1))
  return INTEGER is begin
    if (ptr = (depth-1)) then return 0;
    else return (ptr + 1);
    end if;
end;
begin
  process(f,e) begin full <= f ; empty <= e; end process;
  process(diff) begin
    if (diff = depth -1) then f <= '1'; else f <= '0'; end if;
    if (diff = 0) then e <= '1'; else e <= '0'; end if;
    end process;
  process(clk, Ai, Ao, Di, mem, push, pop, e, f) begin
    if(rising_edge(clk)) then
      if(push='0')and(pop='1')and(e = '0') then Do <= mem(Ao);
    end if;
      if(push='1')and(pop='0')and(f = '0') then mem(Ai) <= Di;
    end if;
    end if ;
    end process;
  process(rst, clk) begin
    if(rst = '1') then Ai <= 0; Ao <= 0; diff <= 0;
    else if(rising_edge(clk)) then
      if (push = '1') and (f = '0') and (pop = '0') then
        Ai <= bump(Ai); diff <= diff + 1;
      elsif (pop = '1') and (e = '0') and (push = '0') then
        Ao <= bump(Ao); diff <= diff - 1;
      end if;
    end if;
    end if;
    end process;
end;

```

FIFO (first-in, first-out) register

Reads (pop = 1) and writes (push = 1) are synchronous to the rising edge of the clock.

Read and write should not occur at the same time. The width (number of bits in each word) and depth (number of words) are generics.

External signals:

clk, clock

rst, reset active-high

push, write to FIFO

pop, read from FIFO

Di, data in

Do, data out

empty, FIFO flag

full, FIFO flag

Internal signals:

diff, difference pointer

Ai, input address

Ao, output address

f, full flag

e, empty flag

No delays in this model.

A FIFO controller

```

library IEEE; use IEEE.STD_LOGIC_1164.all; use
IEEE.NUMERIC_STD.all;
entity fifo_control is generic TPD : TIME := 1 ns;
  port(D_1, D_2 : in UNSIGNED(11 downto 0));
  sel : in UNSIGNED(1 downto 0) ;
  read , f1, f2, e1, e2 : in STD_LOGIC;
  r1, r2, w12 : out STD_LOGIC; D : out UNSIGNED(11 downto
0)) ;
end;
architecture rtl of fifo_control is
  begin process
    (read, sel, D_1, D_2, f1, f2, e1, e2)
  begin
    r1 <= '0' after TPD; r2 <= '0' after TPD;
    if (read = '1') then
      w12 <= '0' after TPD;
      case sel is
        when "01" => D <= D_1 after TPD; r1 <= '1' after TPD;
        when "10" => D <= D_2 after TPD; r2 <= '1' after TPD;
        when "00" => D(3) <= f1 after TPD; D(2) <= f2 after
TPD;
          D(1) <= e1 after TPD; D(0) <= e2 after
TPD;
        when others => D <= "ZZZZZZZZZZZZ" after TPD;
      end case;
    elsif (read = '0') then
      D <= "ZZZZZZZZZZZZ" after TPD; w12 <= '1' after TPD;
    else D <= "ZZZZZZZZZZZZ" after TPD;
    end if;
  end process;
end rtl;

```

This handles the reading and writing to the FIFOs under control of the processor (mpu). The mpu can ask for data from either FIFO or for status flags to be placed on the bus.

Inputs:

D_1

data in from FIFO1

D_2

data in from FIFO2

sel

FIFO select from mpu

read

FIFO read from mpu

f1, f2, e1, e2

flags from FIFOs

Outputs:

r1, r2

read enables for FIFOs

w12

write enable for FIFOs

D

data out to mpu bus

Top level of temperature controller

```

library IEEE; use IEEE.STD_LOGIC_1164.all; use IEEE.NUMERIC_STD.all;
entity T_Control is port (T_in1, T_in2 : in UNSIGNED (11 downto 0));
    sensor: in UNSIGNED(1 downto 0);
    clk, RD, rst : in STD_LOGIC; D : out UNSIGNED(11 downto 0));
end;
architecture structure of T_Control is use work.TC_Components.all;
signal F, E : UNSIGNED (2 downto 1);
signal T_out1, T_out2, R_out1, R_out2, F1, F2, FIFO1, FIFO2 : UNSIGNED(11 downto
0);
signal RD1, RD2, WR: STD_LOGIC ;
begin
RG1 : register_in generic map (1ns) port map (T_in1,clk,rst,R_out1);
RG2 : register_in generic map (1ns) port map (T_in2,clk,rst,R_out2);
TC1 : tconv generic map (1ns) port map (R_out1, T_out1);
TC2 : tconv generic map (1ns) port map (R_out2, T_out2);
TF1 : filter generic map (1ns) port map (T_out1, rst, clk, F1);
TF2 : filter generic map (1ns) port map (T_out2, rst, clk, F2);
FI1 : fifo generic map (12,16) port map (clk, rst, WR, RD1, F1, FIFO1, E(1),
F(1));
FI2 : fifo generic map (12,16) port map (clk, rst, WR, RD2, F2, FIFO2, E(2),
F(2));
FC1 : fifo_control port map
(FIFO1, FIFO2, sensor, RD, F(1), F(2), E(1), E(2), RD1, RD2, WR, D);
end structure;

```

```

package TC_Components is

```

```

component register_in generic (TPD : TIME := 1 ns);
port (T_in : in UNSIGNED(11 downto 0);
clk, rst : in STD_LOGIC; T_out : out UNSIGNED(11 downto 0));
end component;

```

```

component tconv generic (TPD : TIME := 1 ns);
port (T_in : in UNSIGNED (7 downto 0);
    clk, rst : in STD_LOGIC; T_out : out UNSIGNED(7 downto 0));
end component;

```

```

component filter generic (TPD : TIME := 1 ns);
port (T_in : in UNSIGNED (7 downto 0);
    rst, clk : in STD_LOGIC; T_out : out UNSIGNED(7 downto 0));
end component;

```

```

component fifo generic (width:INTEGER := 12; depth : INTEGER := 16);
  port (clk, rst, push, pop : STD_LOGIC;
    Di : UNSIGNED (width-1 downto 0);
    Do : out UNSIGNED (width-1 downto 0);
    empty, full : out STD_LOGIC);
end component;

```

```

component fifo_control generic (TPD:TIME := 1 ns);
  port (D_1, D_2 : in UNSIGNED(7 downto 0);
    select : in UNSIGNED(1 downto 0); read, f1, f2, e1, e2 : in
STD_LOGIC;
    r1, r2, w12 : out STD_LOGIC; D : out UNSIGNED(7 downto 0)) ;
end component;
end;

```

```

library IEEE;
use IEEE.std_logic_1164 all; -- type STD_LOGIC
use IEEE.numeric_std all; -- type UNSIGNED
entity test_TC is end;

```

```

architecture testbench of test_TC is
component T_Control port (T_1, T_2 : in UNSIGNED(11 downto 0);
  clk : in STD_LOGIC; sensor: in UNSIGNED( 1 downto 0) ;
  read : in STD_LOGIC; rst : in STD_LOGIC;
  D : out UNSIGNED(7 downto 0)); end component;
signal T_1, T_2 : UNSIGNED(11 downto 0);
signal clk, read, rst : STD_LOGIC;
signal sensor : UNSIGNED(1 downto 0);
signal D : UNSIGNED(7 downto 0);
begin TT1 : T_Control port map (T_1, T_2, clk, sensor, read, rst, D);
process begin
rst <= '0'; clk <= '0';
wait for 5 ns; rst <= '1'; wait for 5 ns; rst <= '0';
T_in1 <= "000000000011"; T_in2 <= "000000000111"; read <= '0';
  for i in 0 to 15 loop -- fill the FIFOs
    clk <= '0'; wait for 5ns; clk <= '1'; wait for 5 ns;
  end loop;
  assert (false) report "FIFOs full" severity NOTE;
  clk <= '0'; wait for 5ns; clk <= '1'; wait for 5 ns;
read <= '1'; sensor <= "01";
  for i in 0 to 15 loop -- empty the FIFOs
    clk <= '0'; wait for 5ns; clk <= '1'; wait for 5 ns;

```

```
end loop;  
  assert (false) report "FIFOs empty" severity NOTE;  
  clk <= '0'; wait for 5ns; clk <= '1'; wait;  
end process;  
end;
```

10.17 Summary

Key terms and concepts:

The use of an `entity` and an `architecture`

The use of a `configuration` to bind entities and their architectures

The compile, elaboration, initialization, and simulation steps

Types, subtypes, and their use in expressions

The logic systems based on `BIT` and `Std_Logic_1164` types

The use of the IEEE synthesis packages for `BIT` arithmetic

Ports and port modes

Initial values and the difference between simulation and hardware

The difference between a `signal` and a `variable`

The different assignment statements and the timing of updates

The `process` and `wait` statements

VHDL summary			
VHDL feature	Example		93LRM
Comments	<code>-- this is a comment</code>		13.8
Literals (fixed-value items)	<code>12 1.0E6 '1' "110" 'Z'</code> <code>2#1111_1111# "Hello world"</code> <code>STRING'("110")</code>		13.4
Identifiers (case-insensitive, start with letter)	<code>a_good_name Same same</code> <code>2_Bad bad_ _bad very__bad</code>		13.3
Several basic units of code	entity architecture configuration		1.1–1.3
Connections made through ports	port (signal in <code>i : BIT</code> ; out <code>o : BIT</code>);		4.3
Default expression	port (<code>i : BIT := '1'</code>); <code>-- i='1' if left open</code>		4.3
No built-in logic-value system. BIT and BIT_VECTOR (STD).	type <code>BIT is ('0', '1');</code> <code>-- predefined</code> signal <code>myArray: BIT_VECTOR (7 downto 0);</code>		14.2
Arrays	<code>myArray(1 downto 0) <= ('0', '1');</code>		3.2.1
Two basic types of logic signals	a signal corresponds to a real wire a variable is a memory location in RAM		4.3.1.2 4.3.1.3
Types and explicit initial/default value	signal <code>ONE : BIT := '1' ;</code>		4.3.2
Implicit initial/default value	<code>BIT'LEFT = '0'</code>		4.3.2
Predefined attributes	<code>clk'EVENT, clk'STABLE</code>		14.1
Sequential statements inside processes model things that happen one after another and repeat	process begin <code> wait until alarm = ring;</code> <code> eat; work; sleep;</code> end process;		8
Timing with wait statement	wait for <code>1 ns;</code> <code>-- not wait 1 ns</code> wait on <code>light until light = green;</code>		8.1
Update to signals occurs at the end of a simulation cycle	<code>signal <= 1;</code> <code>-- delta time delay</code> <code>signal <= variable1 after 2 ns;</code>		8.3
Update to variables is immediate	<code>variable := 1;</code> <code>-- immediate update</code>		8.4
Processes and concurrent statements model things that happen at the same time	process begin <code>rain;</code> end process; process begin <code>sing;</code> end process; process begin <code>dance;</code> end process;		9.2
IEEE Std_Logic_1164 (defines logic operators on 1164 types)	<code>STD_ULOGIC, STD_LOGIC,</code> <code>STD_ULOGIC_VECTOR, STD_LOGIC_VECTOR</code> type <code>STD_ULOGIC is</code> <code>('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-');</code>		—
IEEE Numeric_Bit and Numeric_Std (defines arithmetic operators on BIT and 1164 types)	<code>UNSIGNED and SIGNED</code> <code>X <= "10" * "01"</code> <code>-- OK with numeric pkgs.</code>		—

VERILOG HDL

11

Key terms and concepts: syntax and semantics • operators • hierarchy • procedures and assignments • timing controls and delay • tasks and functions • control statements • logic-gate modeling • modeling delay • altering parameters • other Verilog features: PLI

History: Gateway Design Automation developed Verilog as a simulation language • Cadence purchased Gateway in 1989 • Open Verilog International (OVI) was created to develop the Verilog language as an IEEE standard • Verilog LRM, IEEE Std 1364-1995 • problems with a normative LRM

11.1 A Counter

Key terms and concepts: Verilog **keywords** • simulation language • compilation • interpreted, compiled, and native code simulators

```

`timescale 1ns/1ns // Set the units of time to be nanoseconds. //1
module counter; //2
    reg clock; // Declare a reg data type for the clock. //3
    integer count; // Declare an integer data type for the count. //4
initial // Initialize things; this executes once at t=0. //5
    begin //6
        clock = 0; count = 0; // Initialize signals. //7
        #340 $finish; // Finish after 340 time ticks. //8
    end //9
/* An always statement to generate the clock; only one statement
follows the always so we don't need a begin and an end. */ //10
always //11
    #10 clock = ~ clock; // Delay (10ns) is set to half the clock
cycle. //12
/* An always statement to do the counting; this executes at the same
time (concurrently) as the preceding always statement. */ //13
always //14
    begin //15

```

```

// Wait here until the clock goes from 1 to 0. //16
@ (negedge clock); //17
// Now handle the counting. //18
if (count == 7) //19
    count = 0; //20
else //21
    count = count + 1; //22
    $display("time = ", $time, " count = ", count); //23
end //24
endmodule //25

```

11.2 Basics of the Verilog Language

Key terms and concepts: **identifier** • Verilog is case-sensitive • system tasks and functions begin with a dollar sign '\$'

```
identifier ::= simple_identifier | escaped_identifier
```

```
simple_identifier ::= [a-zA-Z][a-zA-Z_]
```

```
escaped_identifier ::=
```

```
    \ {Any_ASCII_character_except_white_space} white_space
```

```
white_space ::= space | tab | newline
```

```

module identifiers; //1
/* Multiline comments in Verilog //2
   look like C comments and // is OK in here. */ //3
// Single-line comment in Verilog. //4
reg legal_identifier,two__underscores; //5
reg _OK,OK_,OK_$,OK_123,CASE_SENSITIVE, case_sensitive; //6
reg \/clock ,\a*b ; // Add white_space after escaped identifier. //7
//reg $_BAD,123_BAD; // Bad names even if we declare them! //8
initial begin //9
legal_identifier = 0; // Embedded underscores are OK, //10
two__underscores = 0; // even two underscores in a row. //11
_OK = 0; // Identifiers can start with underscore //12
OK_ = 0; // and end with underscore. //13
OK$ = 0; // $ sign is OK, but beware foreign keyboards. //14
OK_123 =0; // Embedded digits are OK. //15
CASE_SENSITIVE = 0; // Verilog is case-sensitive (unlike VHDL). //16

```

```

case_sensitive = 1; //17
\/clock = 0; // An escaped identifier with \ breaks rules, //18
\a*b = 0; // but be careful to watch the spaces! //19
$display("Variable CASE_SENSITIVE= %d",CASE_SENSITIVE); //20
$display("Variable case_sensitive= %d",case_sensitive); //21
$display("Variable \/clock = %d",\/clock ); //22
$display("Variable \\a*b = %d",\a*b ); //23
end //24
endmodule //25

```

11.2.1 Verilog Logic Values

Key terms and concepts: predefined **logic-value system or value set** • four logic values: '0', '1', 'x', and 'z' (lowercase) • uninitialized or an unknown logic value (either '1', '0', 'z', or in a state of change) • high-impedance value (usually treated as an 'x' value) • internal logic-value system resolves conflicts between drivers on the same node

11.2.2 Verilog Data Types

Key terms and concepts: **data types** • **nets** • **wire** and **tri** (identical) • **supply1** and **supply0** (positive and negative power) • default initial value for a **wire** is 'z' • **integer**, **time**, **event**, and **real** data types • **register** data type (keyword **reg**) • default initial value for a **reg** is 'x' • a **reg** is not always equivalent to a register, flip-flop, or latch • **scalar** • **vector** • **range** • access (or **expand**) bits in a vector using a **bit-select**, or as a contiguous subgroup of bits using a **part-select** • no multidimensional arrays • **memory** data type is an array of registers • integer arrays • time arrays • no real arrays

```

module declarations_1; //1
wire pwr_good, pwr_on, pwr_stable; // Explicitly declare wires. //2
integer i; // 32-bit, signed (2's complement). //3
time t; // 64-bit, unsigned, behaves like a 64-bit reg. //4
event e; // Declare an event data type. //5
real r; // Real data type of implementation defined size. //6
// An assign statement continuously drives a wire: //7
assign pwr_stable = 1'b1; assign pwr_on = 1; // 1 or 1'b1 //8
assign pwr_good = pwr_on & pwr_stable; //9
initial begin //10
i = 123.456; // There must be a digit on either side //11
r = 123456e-3; // of the decimal point if it is present. //12
t = 123456e-3; // Time is rounded to 1 second by default. //13

```

```

$display("i=%0g",i," t=%6.2f",t," r=%f",r); //14
#2 $display("TIME=%0d",$time," ON=",pwr_on, //15
  " STABLE=",pwr_stable," GOOD=",pwr_good); //16
$finish; end //17
endmodule //18

module declarations_2; //1
reg Q, Clk; wire D; //2
// Drive the wire (D): //3
assign D = 1; //4
// At a +ve clock edge assign the value of wire D to the reg Q: //5
always @(posedge Clk) Q = D; //6
initial Clk = 0; always #10 Clk = ~ Clk; //7
initial begin #50; $finish; end //8
always begin //9
$display("T=%2g", $time," D=",D," Clk=",Clk," Q=",Q); #10 end //10
endmodule //11

module declarations_3; //1
reg a,b,c,d,e; //2
initial begin //3
  #10; a = 0;b = 0;c = 0;d = 0; #10; a = 0;b = 1;c = 1;d = 0; //4
  #10; a = 0;b = 0;c = 1;d = 1; #10; $stop; //5
end //6
always begin //7
  @(a or b or c or d) e = (a|b)&(c|d); //8
  $display("T=%0g",$time," e=",e); //9
end //10
endmodule //11

module declarations_4; //1
wire Data; // A scalar net of type wire. //2
wire [31:0] ABus, DBus; // Two 32-bit-wide vector wires: //3
// DBus[31] = leftmost = most-significant bit = msb //4
// DBus[0] = rightmost = least-significant bit = lsb //5
// Notice the size declaration precedes the names. //6
// wire [31:0] TheBus, [15:0] BigBus; // This is illegal. //7
reg [3:0] vector; // A 4-bit vector register. //8
reg [4:7] nibble; // msb index < lsb index is OK. //9
integer i; //10
initial begin //11
i = 1; //12
vector = 'b1010; // Vector without an index. //13
nibble = vector; // This is OK too. //14

```

```

#1; $display("T=%0g",$time," vector=", vector," nibble=", nibble);//15
#2; $display("T=%0g",$time," Bus=%b",DBus[15:0]); //16
end //17
assign DBus [1] = 1; // This is a bit-select. //18
assign DBus [3:0] = 'b1111; // This is a part-select. //19
// assign DBus [0:3] = 'b1111; // Illegal : wrong direction. //20
endmodule //21

module declarations_5; //1
reg [31:0] VideoRam [7:0]; // An 8-word by 32-bit wide memory. //2
initial begin //3
VideoRam[1] = 'bxz; // We must specify an index for a memory. //4
VideoRam[2] = 1; //5
VideoRam[7] = VideoRam[VideoRam[2]]; // Need 2 clock cycles for this //6
VideoRam[8] = 1; // Careful! the compiler won't complain about this //7
// Verify what we entered: //8
$display("VideoRam[0] is %b",VideoRam[0]); //9
$display("VideoRam[1] is %b",VideoRam[1]); //10
$display("VideoRam[2] is %b",VideoRam[2]); //11
$display("VideoRam[7] is %b",VideoRam[7]); //12
end //13
endmodule //14

module declarations_6; //1
integer Number [1:100]; // Notice that size follows name //2
time Time_Log [1:1000]; // - as in an array of reg. //3
// real Illegal [1:10]; // Illegal. There are no real arrays. //4
endmodule //5

```

11.2.3 Other Wire Types

Key terms and concepts: wand, wor, triand, and trior model wired logic • ECL or EPROM, • one area in which the logic values 'z' and 'x' are treated differently • tri0 and tri1 model resistive connections to VSS or VDD • trireg is like a wire but associates some capacitance with the net and models charge storage • scalared and vectored are properties of vectors • small, medium, and large model the charge strength of trireg

11.2.4 Numbers

Key terms and concepts: **constant numbers** are integer or real constants • **integer constants** are written as width'radix value • **radix** (or base): **decimal** (d or D), **hex** (h or H), **octal** (o or O), or **binary** (b or B) • **sized** or **unsized** (implementation dependent) • 1'bx and 1'bz for 'x'

and 'z' • **parameter** (local scope) • **real constants** 100.0 or 1e2 (IEEE Std 754-1985) • **reals** round to the nearest integer, ties away from zero

```

module constants; //1
parameter H12_UNSIZED = 'h 12; // Unsized hex 12 = decimal 18. //2
parameter H12_SIZED = 6'h 12; // Sized hex 12 = decimal 18. //3
// Note: a space between base and value is OK. //4
// Note: `` (single apostrophes) are not the same as the ' character //5
parameter D42 = 8'B0010_1010; // bin 101010 = dec 42 //6
// OK to use underscores to increase readability. //7
parameter D123 = 123; // Unsized decimal (the default). //8
parameter D63 = 8'o 77; // Sized octal, decimal 63. //9
// parameter ILLEGAL = 1'o9; // No 9's in octal numbers! //10
// A = 'hx and B = 'ox assume a 32 bit width. //11
parameter A = 'h x, B = 'o x, C = 8'b x, D = 'h z, E = 16'h ?????; //12
// Note the use of ? instead of z, 16'h ????? is the same as 16'h
zzzz. //13
// Also note the automatic extension to a width of 16 bits. //14
reg [3:0] B0011,Bxxx1,Bzzz1; real R1,R2,R3; integer I1,I3,I_3; //15
parameter BXZ = 8'b1x0x1z0z; //16
initial begin //17
B0011 = 4'b11; Bxxx1 = 4'bx1; Bzzz1 = 4'bz1; // Left padded. //18
R1 = 0.1e1; R2 = 2.0; R3 = 30E-01; // Real numbers. //19
I1 = 1.1; I3 = 2.5; I_3 = -2.5; // IEEE rounds away from 0. //20
end //21
initial begin #1; //22
$display //23
("H12_UNSIZED, H12_SIZED (hex) = %h, %h",H12_UNSIZED, H12_SIZED); //24
$display("D42 (bin) = %b",D42," (dec) = %d",D42); //25
$display("D123 (hex) = %h",D123," (dec) = %d",D123); //26
$display("D63 (oct) = %o",D63); //27
$display("A (hex) = %h",A," B (hex) = %h",B); //28
$display("C (hex) = %h",C," D (hex) = %h",D," E (hex) = %h",E); //29
$display("BXZ (bin) = %b",BXZ," (hex) = %h",BXZ); //30
$display("B0011, Bxxx1, Bzzz1 (bin) = %b, %b, %b",B0011,Bxxx1,Bzzz1); //31
$display("R1, R2, R3 (e, f, g) = %e, %f, %g", R1, R2, R3); //32
$display("I1, I3, I_3 (d) = %d, %d, %d", I1, I3, I_3); //33
end //34
endmodule //35

```


11.2.5 Negative Numbers

Key terms and concepts: Integers are **signed** (two's complement) or **unsigned** • Verilog only “keeps track” of the sign of a negative constant if it is (1) assigned to an `integer` or (2) assigned to a `parameter` without using a base (essentially the same thing) • in other cases a negative constant is treated as an unsigned number • once Verilog “loses” a sign, keeping track of signed numbers is your responsibility

```

module negative_numbers;                                     //1
parameter PA = -12, PB = -'d12, PC = -32'd12, PD = -4'd12; //2
integer IA , IB , IC , ID ; reg [31:0] RA , RB , RC , RD ; //3
initial begin #1;                                          //4
IA = -12; IB = -'d12; IC = -32'd12; ID = -4'd12;          //5
RA = -12; RB = -'d12; RC = -32'd12; RD = -4'd12; #1;      //6
$display("          parameter    integer    reg[31:0]"); //7
$display ("-12          =", PA, IA, , , RA);                //8
$displayh("          ", , , , PA, , , , IA, , , , , RA); //9
$display ("- 'd12     =", , , PB, IB, , , , RB);          //10
$displayh("          ", , , , , PB, , , , , IB, , , , , RB); //11
$display ("-32'd12   =", , , PC, IC, , , , RC);           //12
$displayh("          ", , , , , PC, , , , , IC, , , , , RC); //13
$display ("-4'd12   =", , , , , , , , , , PD, ID, , , , RD); //14
$displayh("          ", , , , , , , , , , , PD, , , , , ID, , , , , RD); //15
end                                                         //16
endmodule                                                 //17

```

	parameter	integer	reg[31:0]
-12	= -12	-12	4294967284
	ffffffff4	ffffffff4	ffffffff4
-'d12	= 4294967284	-12	4294967284
	ffffffff4	ffffffff4	ffffffff4
-32'd12	= 4294967284	-12	4294967284
	ffffffff4	ffffffff4	ffffffff4
-4'd12	= 4	-12	4294967284
	4	ffffffff4	ffffffff4

11.2.6 Strings

Key terms and concepts: ISO/ANSI defines characters, but not their appearance • problem characters are quotes and accents • **string constants** • **define** directive is a compiler directive (global scope)

```

module characters; /* //1
" is ASCII 34 (hex 22), double quote. //2
' is ASCII 39 (hex 27), tick or apostrophe. //3
/ is ASCII 47 (hex 2F), forward slash. //4
\ is ASCII 92 (hex 5C), back slash. //5
` is ASCII 96 (hex 60), accent grave. //6
| is ASCII 124 (hex 7C), vertical bar. //7
There are no standards for the graphic symbols for codes above 128.//8
^ is 171 (hex AB), accent acute in almost all fonts. //9
" is 210 (hex D2), open double quote, like 66 (in some fonts). //10
" is 211 (hex D3), close double quote, like 99 (in some fonts). //11
` is 212 (hex D4), open single quote, like 6 (in some fonts). //12
' is 213 (hex D5), close single quote, like 9 (in some fonts). //13
*/ endmodule //14

module text; //1
parameter A_String = "abc"; // string constant, must be on one line//2
parameter Say = "Say \"Hey!\""; //3
// use escape quote \" for an embedded quote //4
parameter Tab = "\t"; // tab character //5
parameter NewLine = "\n"; // newline character //6
parameter BackSlash = "\\"; // back slash //7
parameter Tick = "\047"; // ASCII code for tick in octal //8
// parameter Illegal = "\500"; // illegal - no such ASCII code //9
initial begin //10
$display("A_String(str) = %s ",A_String," (hex) = %h ",A_String); //11
$display("Say = %s ",Say," Say \"Hey!\""); //12
$display("NewLine(str) = %s ",NewLine," (hex) = %h ",NewLine); //13
$display("\\(str) = %s ",BackSlash," (hex) = %h ",BackSlash); //14
$display("Tab(str) = %s ",Tab," (hex) = %h ",Tab,"1 newline..."); //15
$display("\n"); //16
$display("Tick(str) = %s ",Tick," (hex) = %h ",Tick); //17
#1.23; $display("Time is %t", $time); //18

```

```

end //19
endmodule //20

module define; //1
`define G_BUSWIDTH 32 // Bus width parameter (G_ for global). //2
/* Note: there is no semicolon at end of a compiler directive. The
character ` is ASCII 96 (hex 60), accent grave, it slopes down from
left to right. It is not the tick or apostrophe character ' (ASCII 39
or hex 27)*/ //3
wire [`G_BUSWIDTH:0]MyBus; // A 32-bit bus. //4
endmodule //5

```

11.3 Operators

Key terms and concepts: three types of operators: unary, binary, or a single ternary operator • similar to C programming language (but no ++ or --)

Verilog unary operators

Operator	Name	Examples
!	logical negation	!123 is 'b0 [0, 1, or x for ambiguous; legal for real]
~	bitwise unary negation	~1'b10xz is 1'b01xx
&	unary reduction and	& 4'b1111 is 1'b1, & 2'bx1 is 1'bx, & 2'bz1 is 1'bx
~&	unary reduction nand	~& 4'b1111 is 1'b0, ~& 2'bx1 is 1'bx
	unary reduction or	Note:
~	unary reduction nor	Reduction is performed left (first bit) to right
^	unary reduction xor	Beware of the non-associative reduction operators
~^ ^~	unary reduction xnor	z is treated as x for all unary operators
+	unary plus	+2'bxz is +2'bxz [+m is the same as m; legal for real]
-	unary minus	-2'bxz is x [-m is unary minus m; legal for real]

```

module operators; //1
parameter A10xz = {1'b1,1'b0,1'bx,1'bz}; // Concatenation and //2
parameter A01010101 = {4{2'b01}}; // replication, illegal for real. //3
// Arithmetic operators: +, -, *, /, and modulus % //4
parameter A1 = (3+2) %2; // The sign of a % b is the same as sign of
a. //5
// Logical shift operators: << (left), >> (right) //6
parameter A2 = 4 >> 1; parameter A4 = 1 << 2; // Note: zero fill. //7

```

Verilog operators (in increasing order of precedence)

? : (conditional) [legal for real; associates right to left (others associate left to right)]
 || (logical or) [A smaller operand is zero-filled from its msb (0-fill); legal for real]
 && (logical and) [0-fill, legal for real]
 | (bitwise or) ~| (bitwise nor) [0-fill]
 ^ (bitwise xor) ^~ ~^ (bitwise xnor, equivalence) [0-fill]
 & (bitwise and) ~& (bitwise nand) [0-fill]
 == (logical) != (logical) === (case) !== (case) [0-fill, logical versions are legal for real]
 < (lt) <= (lt or equal) > (gt) >= (gt or equal) [0-fill, all are legal for real]
 << (shift left) >> (shift right) [zero fill; no -ve shifts; shift by x or z results in unknown]
 + (addition) - (subtraction) [if any bit is x or z for + - * / % then entire result is unknown]
 * (multiply) / (divide) % (modulus) [integer divide truncates fraction; + - * / legal for real]
 Unary operators: ! ~ & ~& | ~| ^ ~^ ^~ + -

```
// Relational operators: <, <=, >, >= //8
initial if (1 > 2) $stop; //9
// Logical operators: ! (negation), && (and), || (or) //10
parameter B0 = !12; parameter B1 = 1 && 2; //11
reg [2:0] A00x; initial begin A00x = 'b111; A00x = !2'bx1; end //12
parameter C1 = 1 || (1/0); /* This may or may not cause an //13
error: the short-circuit behavior of && and || is undefined. An //14
evaluation including && or || may stop when an expression is known //15
to be true or false. */ //16
// == (logical equality), != (logical inequality) //17
parameter Ax = (1==1'bx); parameter Bx = (1'bx!=1'bz); //18
parameter D0 = (1==0); parameter D1 = (1==1); //19
// === case equality, !== (case inequality) //20
// The case operators only return true (1) or false (0). //21
parameter E0 = (1===1'bx); parameter E1 = 4'b01xz === 4'b01xz; //22
parameter F1 = (4'bxxxx === 4'bxxxx); //23
// Bitwise logical operators: //24
// ~ (negation), & (and), | (inclusive or), //25
// ^ (exclusive or), ~^ or ^~ (equivalence) //26
parameter A00 = 2'b01 & 2'b10; //27
// Unary logical reduction operators: //28
// & (and), ~& (nand), | (or), ~| (nor), //29
// ^ (xor), ~^ or ^~ (xnor) //30
parameter G1 = & 4'b1111; //31
// Conditional expression f = a ? b : c [if (a) then f=b else f=c] //32
```

```

// if a=(x or z), then (bitwise) f=0 if b=c=0, f=1 if b=c=1, else f≠x33
reg H0, a, b, c; initial begin a=1; b=0; c=1; H0=a?b:c; end //34
reg[2:0] J01x, Jxxx, J01z, J011; //35
initial begin Jxxx = 3'bxxx; J01z = 3'b01z; J011 = 3'b011; //36
J01x = Jxxx ? J01z : J011; end // A bitwise result. //37
initial begin #1; //38
$display("A10xz=%b",A10xz," A01010101=%b",A01010101); //39
$display("A1=%0d",A1," A2=%0d",A2," A4=%0d",A4); //40
$display("B1=%b",B1," B0=%b",B0," A00x=%b",A00x); //41
$display("C1=%b",C1," Ax=%b",Ax," Bx=%b",Bx); //42
$display("D0=%b",D0," D1=%b",D1); //43
$display("E0=%b",E0," E1=%b",E1," F1=%b",F1); //44
$display("A00=%b",A00," G1=%b",G1," H0=%b",H0); //45
$display("J01x=%b",J01x); end //46
endmodule //47

```

11.3.1 Arithmetic

Key terms and concepts: arithmetic on n -bit objects is performed modulo 2^n • arithmetic on vectors (reg or wire) are predefined • once Verilog “loses” a sign, it cannot get it back

```

module modulo; reg [2:0] Seven; //1
initial begin //2
#1 Seven = 7; #1 $display("Before=", Seven); //3
#1 Seven = Seven + 1; #1 $display("After =", Seven); //4
end //5
endmodule //6

module LRM_arithmetic; //1
integer IA, IB, IC, ID, IE; reg [15:0] RA, RB, RC; //2
initial begin //3
IA = -4'd12; RA = IA / 3; // reg is treated as unsigned. //4
RB = -4'd12; IB = RB / 3; // //5
IC = -4'd12 / 3; RC = -12 / 3; // real is treated as signed //6
ID = -12 / 3; IE = IA / 3; // (two's complement). //7
end //8
initial begin #1; //9
$display(" hex default"); //10
$display("IA = -4'd12 = %h%d",IA,IA); //11
$display("RA = IA / 3 = %h %d",RA,RA); //12
$display("RB = -4'd12 = %h %d",RB,RB); //13
$display("IB = RB / 3 = %h%d",IB,IB); //14
$display("IC = -4'd12 / 3 = %h%d",IC,IC); //15

```

```

$display("RC = -12 / 3    =    %h    %d",RC,RC);           //16
$display("ID = -12 / 3    = %h%d",ID,ID);               //17
$display("IE = IA / 3     = %h%d",IE,IE);               //18
end                                                    //19
endmodule                                           //20

                hex    default
IA = -4'd12     = ffffffff4    -12
RA = IA / 3     =    fffc      65532
RB = -4'd12     =    fff4      65524
IB = RB / 3     = 00005551     21841
IC = -4'd12 / 3 = 55555551 1431655761
RC = -12 / 3    =    fffc      65532
ID = -12 / 3    = ffffffff      -4
IE = IA / 3     = ffffffff      -4

```

11.4 Hierarchy

Key terms and concepts: **module** • the **module interface** interconnects two Verilog modules using **ports** • ports must be explicitly declared as **input**, **output**, or **inout** • a `reg` cannot be input or inout port (to connection of a `reg` to another `reg`) • **instantiation** • ports are linked using **named association** or **positional association** • **hierarchical name** (`m1.weekend`) • The compiler will first search downward (or inward) then upward (outward)

Verilog ports.

Verilog port	input	output	inout
Characteristics	wire (or other net)	reg or wire (or other net) We <i>can</i> read an output port inside a module	wire (or other net)

```

module holiday_1(sat, sun, weekend);           //1
    input sat, sun; output weekend;           //2
    assign weekend = sat | sun;               //3
endmodule                                   //4

`timescale 100s/1s // Units are 100 seconds with precision of 1s. //1
module life; wire [3:0] n; integer days;    //2
    wire wake_7am, wake_8am; // Wake at 7 on weekdays else at 8. //3
    assign n = 1 + (days % 7); // n is day of the week (1-7) //4
always@(wake_8am or wake_7am)              //5

```

```

$display("Day=",n," hours=%0d ",($time/36)%24," 8am = ", //6
wake_8am," 7am = ",wake_7am," m2.weekday = ", m2.weekday); //7
initial days = 0; //8
initial begin #(24*36*10);$finish;end // Run for 10 days. //9
always #(24*36) days = days + 1; // Bump day every 24hrs. //10
rest m1(n, wake_8am); // Module instantiation. //11
// Creates a copy of module rest with instance name m1, //12
// ports are linked using positional notation. //13
work m2(.weekday(wake_7am), .day(n)); //14
// Creates a copy of module work with instance name m2, //15
// Ports are linked using named association. //16
endmodule //17

module rest(day, weekend); // Module definition. //1
// Notice the port names are different from the parent. //2
input [3:0] day; output weekend; reg weekend; //3
always begin #36 weekend = day > 5; end // Need a delay here. //4
endmodule //5

module work(day, weekday); //1
input [3:0] day; output weekday; reg weekday; //2
always begin #36 weekday = day < 6; end // Need a delay here. //3
endmodule //4

```

11.5 Procedures and Assignments

Key terms and concepts: a **procedure** is an always or initial statement, a task, or a function) • statements within a **sequential block** (between a begin and an end) that is part of a procedure execute sequentially, but the procedure executes concurrently with other procedures • **continuous assignments** appear outside procedures • **procedural assignments** appear inside procedures

```

module holiday_1(sat, sun, weekend); //1
input sat, sun; output weekend; //2
assign weekend = sat | sun; // Assignment outside a procedure. //3
endmodule //4

module holiday_2(sat, sun, weekend); //1
input sat, sun; output weekend; reg weekend; //2

```

```

    always #1 weekend = sat | sun; // Assignment inside a procedure. //3
endmodule //4

module assignments //1
//... Continuous assignments go here. //2
always // beginning of a procedure //3
    begin // beginning of sequential block //4
        //... Procedural assignments go here. //5
    end //6
endmodule //7

```

11.5.1 Continuous Assignment Statement

Key terms and concepts: a **continuous assignment statement** assigns to a wire like a real logic gate drives a real wire,

```

module assignment_1(); //1
wire pwr_good, pwr_on, pwr_stable; reg Ok, Fire; //2
assign pwr_stable = Ok & (!Fire); //3
assign pwr_on = 1; //4
assign pwr_good = pwr_on & pwr_stable; //5
initial begin Ok = 0; Fire = 0; #1 Ok = 1; #5 Fire = 1; end //6
initial begin $monitor("TIME=%0d", $time, " ON=", pwr_on, " STABLE=", //7
    pwr_stable, " OK=", Ok, " FIRE=", Fire, " GOOD=", pwr_good); //8
    #10 $finish; end //9
endmodule //10

module assignment_2; reg Enable; wire [31:0] Data; //1
/* The following single statement is equivalent to a declaration and //2
continuous assignment. */ //2
wire [31:0] DataBus = Enable ? Data : 32'bz; //3
assign Data = 32'b10101101101011101111000010100001; //4
    initial begin //5
        $monitor("Enable=%b DataBus=%b ", Enable, DataBus); //6
        Enable = 0; #1; Enable = 1; #1; end //7
endmodule //8

```

11.5.2 Sequential Block

Key terms and concepts: a **sequential block** is a group of statements between a **begin** and an **end** • to declare new variables within a sequential block we must name the block • a sequential block is a statement, so that we may nest sequential blocks • a sequential block in an **always**

statement executes repeatedly • an **initial statement** executes only once, so a sequential block in an initial statement only executes once at the beginning of a simulation

```

module always_1; reg Y, Clk; //1
always // Statements in an always statement execute repeatedly: //2
begin: my_block // Start of sequential block. //3
    @(posedge Clk) #5 Y = 1; // At +ve edge set Y=1, //4
    @(posedge Clk) #5 Y = 0; // at the NEXT +ve edge set Y=0. //5
end // End of sequential block. //6
always #10 Clk = ~ Clk; // We need a clock. //7
initial Y = 0; // These initial statements execute //8
initial Clk = 0; // only once, but first. //9
initial $monitor("T=%2g", $time, " Clk=", Clk, " Y=", Y); //10
initial #70 $finish; //11
endmodule //12

```

11.5.3 Procedural Assignments

Key terms and concepts: the value of an expression on the RHS of an assignment within a procedure (a **procedural assignment**) updates a **reg** (or memory element) immediately • a **reg** holds its value until changed by another procedural assignment • a **blocking assignment** is one type of procedural assignment

blocking_assignment ::= reg-lvalue = [delay_or_event_control]
expression

```

module procedural_assign; reg Y, A; //1
always @(A) //2
    Y = A; // Procedural assignment. //3
initial begin A=0; #5; A=1; #5; A=0; #5; $finish;end //4
initial $monitor("T=%2g", $time, "A=", A, "Y=", Y); //5
endmodule //6

```

T= 0 A=0 Y=0

T= 5 A=1 Y=1

T=10 A=0 Y=0

11.6 Timing Controls and Delay

Key terms and concepts: statements in a sequential block are executed, in the absence of any delay, at the same simulation time—the current **time step** • delays are modeled using a **timing control**

11.6.1 Timing Control

Key terms and concepts: a **timing control** is a delay control or an event control • a **delay control** delays an assignment by a specified amount of time • **timescale compiler directive** is used to specify the units of time and precision • ``timescale 1ns/10ps` (s, ns, ps, or fs and the multiplier must be 1, 10, or 100) • intra-assignment delay • delayed assignment • an **event control** delays an assignment until a specified event occurs • **posedge** is a transition from '0' to '1' or 'x', or a transition from 'x' to '1' (transitions to or from 'z' don't count) • events can be declared (as **named events**), triggered, and detected

```
x = #1 y; // intra-assignment delay
#1 x = y; // delayed assignment
```

```
begin // Equivalent to intra-assignment delay.
  hold = y; // Sample and hold y immediately.
  #1; // Delay.
  x = hold; // Assignment to x. Overall same as x = #1 y.
end
```

```
begin // Equivalent to delayed assignment.
  #1; // Delay.
  x = y; // Assign y to x. Overall same as #1 x = y.
end
```

```
event_control ::= @ event_identifier | @ (event_expression)
```

```

event_expression ::= expression | event_identifier
  | posedge expression | negedge expression
  | event_expression or event_expression

module delay_controls; reg X, Y, Clk, Dummy; //1
always #1 Dummy=!Dummy; // Dummy clock, just for graphics. //2
// Examples of delay controls: //3
always begin #25 X=1;#10 X=0;#5; end //4
// An event control: //5
always @(posedge Clk) Y=X; // Wait for +ve clock edge. //6
always #10 Clk = !Clk; // The real clock. //7
initial begin Clk = 0; //8
  $display("T Clk X Y"); //9
  $monitor("%2g", $time, Clk, X, Y); //10
  $dumpvars;#100 $finish; end //11
endmodule //12

module show_event; //1
reg clock; //2
event event_1, event_2; // Declare two named events. //3
always @(posedge clock) -> event_1; // Trigger event_1. //4
always @ event_1 //5
begin $display("Strike 1!!"); -> event_2;end // Trigger event_2. //6
always @ event_2 begin $display("Strike 2!!"); //7
$finish; end // Stop on detection of event_2. //8
always #10 clock = ~ clock; // We need a clock. //9
initial clock = 0; //10
endmodule //11

```

11.6.2 Data Slip

```

module data_slip_1 (); reg Clk, D, Q1, Q2; //1
/***** bad sequential logic below *****/ //2
always @(posedge Clk) Q1 = D; //3
always @(posedge Clk) Q2 = Q1; // Data slips here! //4
/***** bad sequential logic above *****/ //5
initial begin Clk = 0; D = 1; end always #50 Clk = ~Clk; //6
initial begin $display("t Clk D Q1 Q2"); //7
$monitor("%3g", $time, Clk, D, Q1, Q2); end //8
initial #400 $finish; // Run for 8 cycles. //9

```

```

initial $dumpvars; //10
endmodule //11

always @(posedge Clk) Q1 = #1 D; // The delays in the assignments //1
always @(posedge Clk) Q2 = #1 Q1; // fix the data slip. //2

```

11.6.3 Wait Statement

Key terms and concepts: **wait statement** suspends a procedure until a condition is true • beware “infinite hold” • level-sensitive

```

wait (Done) $stop; // Wait until Done = 1 then stop.

module test_dff_wait; //1
reg D, Clock, Reset; dff_wait u1(D, Q, Clock, Reset); //2
initial begin D=1; Clock=0;Reset=1'b1; #15 Reset=1'b0; #20 D=0;end //3
always #10 Clock = !Clock; //4
initial begin $display("T Clk D Q Reset"); //5
    $monitor("%2g", $time, ,Clock, , , ,D, ,Q, ,Reset); #50 $finishend //6
endmodule //7

module dff_wait(D, Q, Clock, Reset); //1
output Q; input D, Clock, Reset; reg Q; wire D; //2
always @(posedge Clock) if (Reset !== 1) Q = D; //3
always begin wait (Reset == 1) Q = 0; wait (Reset !== 1); end //4
endmodule //5

module dff_wait(D,Q,Clock,Reset); //1
output Q; input D,Clock,Reset; reg Q; wire D; //2
always @(posedge Clock) if (Reset !== 1) Q = D; //3
// We need another wait statement here or we shall spin forever. //4
always begin wait (Reset == 1) Q = 0; end //5
endmodule //6

```

11.6.4 Blocking and Nonblocking Assignments

Key terms and concepts: a procedural assignment (**blocking procedural assignment statement**) with a timing control delays or **blocks** execution • **nonblocking procedural assignment statement** allows execution to continue • registers are updated at end of current time step • synthesis tools don't allow blocking and nonblocking procedural assignments to the same `reg` within a sequential block

```

module delay; //1
reg a,b,c,d,e,f,g,bds,bsd; //2
initial begin //3

```

```

a = 1; b = 0; // No delay control. //4
#1 b = 1; // Delayed assignment. //5
c = #1 1; // Intra-assignment delay. //6
#1; // Delay control. //7
d = 1; // //8
e <= #1 1; // Intra-assignment delay, nonblocking assignment //9
#1 f <= 1; // Delayed nonblocking assignment. //10
g <= 1; // Nonblocking assignment. //11
end //12
initial begin #1 bds = b; end // Delay then sample (ds). //13
initial begin bsd = #1 b; end // Sample then delay (sd). //14
initial begin $display("t a b c d e f g bds bsd"); //15
$monitor("%g", $time, ,a,,b,,c,,d,,e,,f,,g,,bds,,,,bsd); end //16
endmodule //17

```

```

t a b c d e f g bds bsd
0 1 0 x x x x x x x
1 1 1 x x x x x 1 0
2 1 1 1 x x x x 1 0
3 1 1 1 1 x x x 1 0
4 1 1 1 1 1 1 1 1 0

```

11.6.5 Procedural Continuous Assignment

Key terms and concepts: **procedural continuous assignment statement** (or quasicontinuous assignment statement) is a special form `assign` within a sequential block

Verilog assignment statements.

Type of Verilog assignment	Continuous assignment statement	Procedural assignment statement	Nonblocking procedural assignment statement	Procedural continuous assignment statement
Where it can occur	outside an <code>always</code> or <code>initial</code> statement, task, or function	inside an <code>always</code> or <code>initial</code> statement, task, or function	inside an <code>always</code> or <code>initial</code> statement, task, or function	<code>always</code> or <code>initial</code> statement, task, or function
Example	<pre>wire [31:0] DataBus; assign DataBus = Enable ? Data : 32'bz</pre>	<pre>reg Y; always @(posedge Enable ? Data : clock) Y = 1;</pre>	<pre>reg Y; always Y <= 1;</pre>	<pre>always @(Enable) if(Enable) assign Q = D; else deassign Q;</pre>
Valid LHS of assignment	net	register or memory element	register or memory element	net
Valid RHS of assignment	<expression> net, reg or memory element	<expression> net, reg or memory element	<expression> net, reg or memory element	<expression> net, reg or memory element
Book	11.5.1	11.5.3	11.6.4	11.6.5
Verilog LRM	6.1	9.2	9.2.2	9.3

```

module dff_procedural_assign; //1
reg d,clr_,pre_,clk; wire q; dff_clr_pre dff_1(q,d,clr_,pre_,clk); //2
always #10 clk = ~clk; //3
initial begin clk = 0; clr_ = 1; pre_ = 1; d = 1; //4
  #20; d = 0; #20; pre_ = 0; #20; pre_ = 1; #20; clr_ = 0; //5
  #20; clr_ = 1; #20; d = 1; #20; $finish;end //6
initial begin //7
  $display("T CLK PRE_ CLR_ D Q"); //8
  $monitor("%3g", $time,,,clk,,,,pre_,,,clr_,,,d,,q) end //9
endmodule //10

module dff_clr_pre(q,d,clear_,preset_,clock); //1
output q; input d,clear_,preset_,clock; reg q; //2
always @(clear_ or preset_) //3

```

```

    if (!clear_) assign q = 0; // active-low clear //4
    else if(!preset_) assign q = 1; // active-low preset //5
    else deassign q; //6
always @(posedge clock) q = d; //7
endmodule //8

module all_assignments //1
//... continuous assignments. //2
always // beginning of procedure //3
    begin // beginning of sequential block //4
//... blocking procedural assignments. //5
//... nonblocking procedural assignments. //6
//... procedural continuous assignments. //7
    end //8
endmodule //9

```

11.7 Tasks and Functions

Key terms and concepts: a **task** is a procedure called from another procedure • a task may call other tasks and functions • a **function** is a procedure used in an expression • a function may not call a task • tasks may contain timing controls but functions may not

```

Call_A_Task_And_Wait (Input1, Input2, Output);
Result_Immediate = Call_A_Function (All_Inputs);

```

```

module F_subset_decode; reg [2:0]A, B, C, D, E, F; //1
initial begin A = 1; B = 0; D = 2; E = 3; //2
    C = subset_decode(A, B); F = subset_decode(D,E); //3
    $display("A B C D E F"); $display(A,,B,,C,,D,,E,,F) end //4
function [2:0] subset_decode; input [2:0] a, b; //5
    begin if (a <= b) subset_decode = a; else subset_decode = b; end //6
endfunction //7
endmodule //8

```

11.8 Control Statements

Key terms and concepts: if, case, loop, disable, fork, and join statements control execution

11.8.1 Case and If Statement

Key terms and concepts: an **if statement** represents a two-way branch • a **case statement** represents a multiway branch • a **controlling expression** is matched with **case expressions** in each of the **case items** (or arms) to determine a match • the `case` statement must be inside a sequential block (inside an `always` statement) and needs some delay • a **casex statement** handles both 'z' and 'x' as don't care • the **casez statement** handles only 'z' bits as don't care • bits in case expressions may be set to '?' representing don't care values

```

if(switch) Y = 1; else Y = 0;

module test_mux; reg a, b, select; wire out; //1
mux mux_1(a, b, out, select); //2
initial begin #2; select = 0; a = 0; b = 1; //3
    #2; select = 1'bx; #2; select = 1'bz; #2; select = 1'end //4
initial $monitor("T=%2g", $time, " Select=", select, " Out=", out); //5
initial #10 $finish; //6
endmodule //7

module mux(a, b, mux_output, mux_select); input a, b, mux_select; //1
output mux_output; reg mux_output; //2
always begin //3
case(mux_select) //4
    0: mux_output = a; //5
    1: mux_output = b; //6
    default mux_output = 1'bx; // If select = x or z set output to
x. //7
endcase //8
#1; // Need some delay, otherwise we'll spin forever. //9
end //10
endmodule //11

casex (instruction_register[31:29])
    3b'??1 : add;
    3b'?1? : subtract;
    3b'1?? : branch;
endcase

```


11.8.2 Loop Statement

Key terms and concepts: A **loop statement** is a **for**, **while**, **repeat**, or **forever** statement •

```

module loop_1;                                     //1
integer i; reg [31:0] DataBus; initial DataBus = 0; //2
initial begin                                     //3
  /***** Insert loop code after here. *****/
  /* for(Execute this assignment once before starting loop; exit loop
  if this expression is false; execute this assignment at end of loop
  before the check for end of loop.) */
  for(i = 0; i <= 15; i = i+1) DataBus[i] = 1;      //4
  /***** Insert loop code before here. *****/
end                                               //5
initial begin                                     //6
  $display("DataBus = %b",DataBus);                //7
  #2; $display("DataBus = %b",DataBus); $finish;    //8
end                                               //9
endmodule                                         //10

```

```

i = 0;
/* while(Execute next statement while this expression is true.) */
while(i <= 15) begin DataBus[i] = 1; i = i+1; end

```

```

i = 0;
/* repeat(Execute next statement the number of times corresponding to
the evaluation of this expression at the beginning of the loop.) */
repeat(16) begin DataBus[i] = 1; i = i+1; end

```

```

i = 0;
/* A forever statement loops continuously. */
forever begin : my_loop
  DataBus[i] = 1;
  if (i == 15) #1 disable my_loop; // Need to let time advance to
  exit.
  i = i+1;
end

```

11.8.3 Disable

Key terms and concepts: The **disable statement** stops the execution of a labeled sequential block and skips to the end of the block • difficult to implement in hardware

```
forever
begin: microprocessor_block // Labeled sequential block.
    @(posedge clock)
    if (reset) disable microprocessor_block; // Skip to end of block.
    else Execute_code;
end
```

11.8.4 Fork and Join

Key terms and concepts: The **fork statement** and **join statement** allows the execution of two or more parallel threads in a **parallel block** • difficult to implement in hardware

```
module fork_1 //1
event eat_breakfast, read_paper; //2
initial begin //3
    fork //4
        @eat_breakfast; @read_paper; //5
    join //6
end //7
endmodule //8
```

11.9 Logic-Gate Modeling

Key terms and concepts: Verilog has a set of built-in logic models and you may also define your own models.

11.9.1 Built-in Logic Models

Key terms and concepts: **primitives:** and, nand, nor, or, xor, xnor • strong drive strength is the default • the first port of a primitive gate is always the output port • remaining ports are the input ports

Definition of the Verilog primitive 'and' gate

'and'	0	1	x	z
0	0	0	0	0
1	0	1	x	x
x	0	x	x	x
z	0	x	x	x

```

module primitive; //1
nand (strong0, strong1) #2.2 //2
    Nand_1(n001, n004, n005), //3
    Nand_2(n003, n001, n005, n002); //4
nand (n006, n005, n002); //5
endmodule //6

```

11.9.2 User-Defined Primitives

Key terms and concepts: a **user-defined primitive (UDP)** uses a truth-table specification • the first port of a UDP must be an output port (no vector or inout ports) • inputs in a **UDP truth table** are '0', '1', and 'x' • any 'z' input is treated as an 'x' • default output is 'x' • any next state goes between an input and an output in UDP table • shorthand notation for levels • (ab) represents a change from a to b • (01) represents a rising edge • shorthand notations for edges

```

primitive Adder(Sum, InA, InB); //1
output Sum; input Ina, InB; //2
table //3
// inputs : output //4
00 : 0; //5
01 : 1; //6
10 : 1; //7
11 : 0; //8

```

```

endtable //9
endprimitive //10
primitive DLatch(Q, Clock, Data); //1
output Q; reg Q; input Clock, Data; //2
table //3
//inputs : present state : output (next state) //4
1 0 : ? : 0; // ? represents 0,1, or x (input or present state). //5
1 1 : b : 1; // b represents 0 or 1 (input or present state). //6
1 1 : x : 1; // Could have combined this with previous line. //7
0 ? : ? : -; // - represents no change in an output. //8
endtable //9
endprimitive //10
primitive DFlipFlop(Q, Clock, Data); //1
output Q; reg Q; input Clock, Data; //2
table //3
//inputs : present state : output (next state) //4
r 0 : ? : 0 ; // rising edge, next state = output = 0 //5
r 1 : ? : 1 ; // rising edge, next state = output = 1 //6
(0x) 0 : 0 : 0 ; // rising edge, next state = output = 0 //7
(0x) 1 : 1 : 1 ; // rising edge, next state = output = 1 //8
(?0) ? : ? : - ; // falling edge, no change in output //9
? (??) : ? : - ; // no clock edge, no change in output //10
endtable //11
endprimitive //12

```

11.10 Modeling Delay

Key terms and concepts: built-in delays • ASIC cell library models include logic delays as a function of fanout and estimated wiring loads • after layout, we can back-annotate • delay calculator calculates the net delays in **Standard Delay Format, SDF** • sign-off quality ASIC cell libraries

11.10.1 Net and Gate Delay

Key terms and concepts: minimum, typical, and maximum delays • first triplet specifies the min/typ/max rising delay ('0' or 'x' or 'z' to '1') and the second triplet specifies the falling

delay (to '0') • for a high-impedance output, we specify a triplet for rising, falling, and the delay to transition to 'z' (from '0' or '1'), the delay for a three-state driver to turn off or float

```
 #(1.1:1.3:1.7) assign delay_a = a; // min:typ:max
```

```
 wire #(1.1:1.3:1.7) a_delay; // min:typ:max
```

```
 wire #(1.1:1.3:1.7) a_delay = a; // min:typ:max
```

```
 nand #3.0 nd01(c, a, b);
```

```
 nand #(2.6:3.0:3.4) nd02(d, a, b); // min:typ:max
```

```
 nand #(2.8:3.2:3.4, 2.6:2.8:2.9) nd03(e, a, b);
```

```
 // #(rising, falling) delay
```

```
 wire #(0.5,0.6,0.7) a_z = a; // rise/fall/float delays
```

11.10.2 Pin-to-Pin Delay

Key terms and concepts: A **specify block** allows **pin-to-pin delays** across a module • $x \Rightarrow y$ specifies a **parallel connection** (or parallel path) • x and y must have the same number of bits • $x * \Rightarrow y$ specifies a **full connection** (or full path) • every bit in x is connected to y • x and y may be different sizes • **state-dependent path delay**

```
 module DFF_Spec; reg D, clk; //1
```

```
 DFF_Part DFF1 (Q, clk, D, pre, clr); //2
```

```
 initial begin D = 0; clk = 0; #1; clk = 1; end //3
```

```
 initial $monitor("T=%2g", $time, " clk=", clk, " Q=", Q); //4
```

```
 endmodule //5
```

```
 module DFF_Part(Q, clk, D, pre, clr); //1
```

```
 input clk, D, pre, clr; output Q; //2
```

```
 DFlop(Q, clk, D); // No preset or clear in this UDP. //3
```

```
 specify //4
```

```
 specparam //5
```

```
 tPLH_clk_Q = 3, tPHL_clk_Q = 2.9, //6
```

```
 tPLH_set_Q = 1.2, tPHL_set_Q = 1.1; //7
```

```
 (clk => Q) = (tPLH_clk_Q, tPHL_clk_Q); //8
```

```
 (pre, clr *> Q) = (tPLH_set_Q, tPHL_set_Q); //9
```

```
 endspecify //10
```

```
 endmodule //11
```

```
 `timescale 1 ns / 100 fs //1
```

```
 module M_Spec; reg A1, A2, B; M M1 (Z, A1, A2, B); //2
```

```

initial begin A1=0;A2=1;B=1;#5;B=0;#5;A1=1;A2=0;B=1;#5;B=0;end //3
initial //4
    $monitor("T=%4g",$realtime," A1=",A1," A2=",A2," B=",B," Z=",Z); //5
endmodule //6

`timescale 100 ps / 10 fs //1
module M(Z, A1, A2, B); input A1, A2, B; output Z; //2
or (Z1, A1, A2); nand (Z, Z1, B); // OAI21 //3
/*A1 A2 B Z Delay=10*100 ps unless indicated in the table below. //4
  0 0 0 1 //5
  0 0 1 1 //6
  0 1 0 1 B:0->1 Z:1->0 delay=t2 //7
  0 1 1 0 B:1->0 Z:0->1 delay=t1 //8
  1 0 0 1 B:0->1 Z:1->0 delay=t4 //9
  1 0 1 0 B:1->0 Z:0->1 delay=t3 //10
  1 1 0 1 //11
  1 1 1 0 */ //12
specify specparam t1 = 11, t2 = 12; specparam t3 = 13, t4 = 14; //13
  (A1 => Z) = 10; (A2 => Z) = 10; //14
  if (~A1) (B => Z) = (t1, t2); if (A1) (B => Z) = (t3, t4); //15
endspecify //16
endmodule //17

```

11.11 Altering Parameters

Key terms and concepts: parameter override in instantiated module • parameters have local scope • **defparam** statement and hierarchical name

```

module Vector_And(Z, A, B); //1
  parameter CARDINALITY = 1; //2
  input [CARDINALITY-1:0] A, B; //3
  output [CARDINALITY-1:0] Z; //4
  wire [CARDINALITY-1:0] Z = A & B; //5
endmodule //6

module Four_And_Gates(OutBus, InBusA, InBusB); //1
  input [3:0] InBusA, InBusB; output [3:0] OutBus; //2
  Vector_And #(4) My_AND(OutBus, InBusA, InBusB); // 4 AND gates //3
endmodule //4

module And_Gates(OutBus, InBusA, InBusB); //1
  parameter WIDTH = 1; //2
  input [WIDTH-1:0] InBusA, InBusB; output [WIDTH-1:0] OutBus; //3

```

```

    Vector_And #(WIDTH) My_And(OutBus, InBusA, InBusB);           //4
endmodule                                                       //5
module Super_Size; defparam And_Gates.WIDTH = 4; endmodule    //1

```

11.12 A Viterbi Decoder

11.12.1 Viterbi Encoder

```

/*****
/* module viterbi_encode                                     */
/*****
/* This is the encoder. X2N (msb) and X1N form the 2-bit input
message, XN. Example: if X2N=1, X1N=0, then XN=2. Y2N (msb), Y1N, and
Y0N form the 3-bit encoded signal, YN (for a total constellation of 8
PSK signals that will be transmitted). The encoder uses a state
machine with four states to generate the 3-bit output, YN, from the
2-bit input, XN. Example: the repeated input sequence XN = (X2N, X1N)
= 0, 1, 2, 3 produces the repeated output sequence YN = (Y2N, Y1N,
Y0N) = 1, 0, 5, 4. */
module viterbi_encode(X2N,X1N,Y2N,Y1N,Y0N,clk,res);
input X2N,X1N,clk,res; output Y2N,Y1N,Y0N;
wire X1N_1,X1N_2,Y2N,Y1N,Y0N;
dff dff_1(X1N,X1N_1,clk,res); dff dff_2(X1N_1,X1N_2,clk,res);
assign Y2N=X2N; assign Y1N=X1N ^ X1N_2; assign Y0N=X1N_1;
endmodule

```

11.12.2 The Received Signal

```

/*****
/* module viterbi_distances                                 */
/*****
/* This module simulates the front end of a receiver. Normally the
received analog signal (with noise) is converted into a series of
distance measures from the known eight possible transmitted PSK
signals: s0,...,s7. We are not simulating the analog part or noise in
this version, so we just take the digitally encoded 3-bit signal, Y,
from the encoder and convert it directly to the distance measures.
d[N] is the distance from signal = N to signal = 0
d[N] = (2*sin(N*PI/8))**2 in 3-bit binary (on the scale 2=100)
Example: d[3] = 1.85**2 = 3.41 = 110
inN is the distance from signal = N to encoder signal.

```

Example: in3 is the distance from signal = 3 to encoder signal.
 d[N] is the distance from signal = N to encoder signal = 0.
 If encoder signal = J, shift the distances by 8-J positions.
 Example: if signal = 2, in0 is d[6], in1 is D[7], in2 is D[0], etc.
 */

```

module viterbi_distances
  (Y2N,Y1N,Y0N,clk,res,in0,in1,in2,in3,in4,in5,in6,in7);
input clk,res,Y2N,Y1N,Y0N; output in0,in1,in2,in3,in4,in5,in6,in7;
reg [2:0] J,in0,in1,in2,in3,in4,in5,in6,in7;reg [2:0] d [7:0];
initial begin d[0]='b000;d[1]='b001;d[2]='b100;d[3]='b110;
d[4]='b111;d[5]='b110;d[6]='b100;d[7]='b001;end
always @(Y2N or Y1N or Y0N) begin
J[0]=Y0N;J[1]=Y1N;J[2]=Y2N;
J=8-J;in0=d[J];J=J+1;in1=d[J];J=J+1;in2=d[J];J=J+1;in3=d[J];
J=J+1;in4=d[J];J=J+1;in5=d[J];J=J+1;in6=d[J];J=J+1;in7=d[J];
end endmodule

```

11.12.3 Testing the System

```

/*****
/* module viterbi_test_CDD
/*****
/* This is the top-level module, viterbi_test_CDD, that models the
communications link. It contains three modules: viterbi_encode,
viterbi_distances, and viterbi. There is no analog and no noise in
this version. The 2-bit message, X, is encoded to a 3-bit signal, Y.
In this module the message X is generated using a simple counter.
The digital 3-bit signal Y is transmitted, received with noise as an
analog signal (not modeled here), and converted to a set of eight
3-bit distance measures, in0, ..., in7. The distance measures form
the input to the Viterbi decoder that reconstructs the transmitted
signal Y, with an error signal if the measures are inconsistent.
CDD = counter input, digital transmission, digital reception */
module viterbi_test_CDD;
wire Error; // decoder out
wire [2:0] Y, Out; // encoder out, decoder out
reg [1:0] X; // encoder inputs
reg Clk, Res; // clock and reset
wire [2:0] in0,in1,in2,in3,in4,in5,in6,in7;
always #500 $display("t   Clk X Y Out Error");
initial $monitor("%4g", $time, , Clk, , , X, , Y, , Out, , , Error);
initial $dumpvars; initial #3000 $finish;
always #50 Clk = ~Clk; initial begin Clk = 0;

```



```

X = 3; // No special reason to start at 3.
#60 Res = 1;#10 Res = 0 end // Hit reset after inputs are stable.
always @(posedge Clk) #1 X = X + 1; // Drive the input with a
counter.
viterbi_encode v_1
  (X[1],X[0],Y[2],Y[1],Y[0],Clk,Res);
viterbi_distances v_2
  (Y[2],Y[1],Y[0],Clk,Res,in0,in1,in2,in3,in4,in5,in6,in7);
viterbi v_3
  (in0,in1,in2,in3,in4,in5,in6,in7,Out,Clk,Res,Error);
endmodule

```

11.12.4 Verilog Decoder Model

```

/*****
/*      module dff                                */
/*****
/* A D flip-flop module. */

```

```

module dff(D,Q,Clock,Reset); // N.B. reset is active-low.
output Q; input D,Clock,Reset;
parameter CARDINALITY = 1; reg [CARDINALITY-1:0] Q;
wire [CARDINALITY-1:0] D;
always @(posedge Clock) if (Reset != 0) #1 Q = D;
always begin wait (Reset == 0); Q = 0; wait (Reset == 1); end
endmodule

```

/* Verilog code for a Viterbi decoder. The decoder assumes a rate 2/3 encoder, 8 PSK modulation, and trellis coding. The viterbi module contains eight submodules: subset_decode, metric, compute_metric, compare_select, reduce, pathin, path_memory, and output_decision.

The decoder accepts eight 3-bit measures of $||r-s_i||^2$ and, after an initial delay of thirteen clock cycles, the output is the best estimate of the signal transmitted. The distance measures are the Euclidean distances between the received signal r (with noise) and each of the (in this case eight) possible transmitted signals s_0 to s_7 .

Original by Christeen Gray, University of Hawaii. Heavily modified by MJSS; any errors are mine. Use freely. */

```

/*****
/*      module viterbi                                */

```

```

/*****
/* This is the top level of the Viterbi decoder. The eight input
signals {in0,...,in7} represent the distance measures,  $||r-s_i||^2$ .
The other input signals are clk and reset. The output signals are
out and error. */

```

```

module viterbi
    (in0,in1,in2,in3,in4,in5,in6,in7,
     out,clk,reset,error);
input [2:0] in0,in1,in2,in3,in4,in5,in6,in7;
output [2:0] out; input clk,reset; output error;
wire sout0,sout1,sout2,sout3;
wire [2:0] s0,s1,s2,s3;
wire [4:0] m_in0,m_in1,m_in2,m_in3;
wire [4:0] m_out0,m_out1,m_out2,m_out3;
wire [4:0] p0_0,p2_0,p0_1,p2_1,p1_2,p3_2,p1_3,p3_3;
wire ACS0,ACS1,ACS2,ACS3;
wire [4:0] out0,out1,out2,out3;
wire [1:0] control;
wire [2:0] p0,p1,p2,p3;
wire [11:0] path0;

    subset_decode u1(in0,in1,in2,in3,in4,in5,in6,in7,
                    s0,s1,s2,s3,sout0,sout1,sout2,sout3,clk,reset);
    metric u2(m_in0,m_in1,m_in2,m_in3,m_out0,
             m_out1,m_out2,m_out3,clk,reset);
    compute_metric u3(m_out0,m_out1,m_out2,m_out3,s0,s1,s2,s3,
                    p0_0,p2_0,p0_1,p2_1,p1_2,p3_2,p1_3,p3_3,error);
    compare_select u4(p0_0,p2_0,p0_1,p2_1,p1_2,p3_2,p1_3,p3_3,
                    out0,out1,out2,out3,ACS0,ACS1,ACS2,ACS3);
    reduce u5(out0,out1,out2,out3,
             m_in0,m_in1,m_in2,m_in3,control);
    pathin u6(sout0,sout1,sout2,sout3,
             ACS0,ACS1,ACS2,ACS3,path0,clk,reset);
    path_memory u7(p0,p1,p2,p3,path0,clk,reset,
                 ACS0,ACS1,ACS2,ACS3);
    output_decision u8(p0,p1,p2,p3,control,out);
endmodule

```

```

/*****
/* module subset_decode */
/*****

```

/* This module chooses the signal corresponding to the smallest of each set $\{||r-s0||**2, ||r-s4||**2\}$, $\{||r-s1||**2, ||r-s5||**2\}$, $\{||r-s2||**2, ||r-s6||**2\}$, $\{||r-s3||**2, ||r-s7||**2\}$. Therefore there are eight input signals and four output signals for the distance measures. The signals `sout0`, ..., `sout3` are used to control the path memory. The statement `dff #(3)` instantiates a vector array of 3 D flip-flops. */

```

module subset_decode
    (in0,in1,in2,in3,in4,in5,in6,in7,
     s0,s1,s2,s3,
     sout0,sout1,sout2,sout3,
     clk,reset);
input [2:0] in0,in1,in2,in3,in4,in5,in6,in7;
output [2:0] s0,s1,s2,s3;
output sout0,sout1,sout2,sout3;
input clk,reset;
wire [2:0] sub0,sub1,sub2,sub3,sub4,sub5,sub6,sub7;

    dff #(3) subout0(in0, sub0, clk, reset);
    dff #(3) subout1(in1, sub1, clk, reset);
    dff #(3) subout2(in2, sub2, clk, reset);
    dff #(3) subout3(in3, sub3, clk, reset);
    dff #(3) subout4(in4, sub4, clk, reset);
    dff #(3) subout5(in5, sub5, clk, reset);
    dff #(3) subout6(in6, sub6, clk, reset);
    dff #(3) subout7(in7, sub7, clk, reset);

    function [2:0] subset_decode; input [2:0] a,b;
        begin
            subset_decode = 0;
            if (a<=b) subset_decode = a; else subset_decode = b;
        end
    endfunction

    function set_control; input [2:0] a,b;
        begin
            if (a<=b) set_control = 0; else set_control = 1;
        end
    endfunction

assign s0 = subset_decode (sub0,sub4);

```

```

assign s1 = subset_decode (sub1,sub5);
assign s2 = subset_decode (sub2,sub6);
assign s3 = subset_decode (sub3,sub7);
assign sout0 = set_control(sub0,sub4);
assign sout1 = set_control(sub1,sub5);
assign sout2 = set_control(sub2,sub6);
assign sout3 = set_control(sub3,sub7);
endmodule

```

```

/*****
/*   module compute_metric                               */
/*****
/* This module computes the sum of path memory and the distance for
each path entering a state of the trellis. For the four states,
there are two paths entering it; therefore eight sums are computed
in this module. The path metrics and output sums are 5 bits wide.
The output sum is bounded and should never be greater than 5 bits
for a valid input signal. The overflow from the sum is the error
output and indicates an invalid input signal.*/

```

```

module compute_metric
    (m_out0,m_out1,m_out2,m_out3,
     s0,s1,s2,s3,p0_0,p2_0,
     p0_1,p2_1,p1_2,p3_2,p1_3,p3_3,
     error);
input [4:0] m_out0,m_out1,m_out2,m_out3;
input [2:0] s0,s1,s2,s3;
output [4:0] p0_0,p2_0,p0_1,p2_1,p1_2,p3_2,p1_3,p3_3;
output error;

```

```

assign
    p0_0 = m_out0 + s0,
    p2_0 = m_out2 + s2,
    p0_1 = m_out0 + s2,
    p2_1 = m_out2 + s0,
    p1_2 = m_out1 + s1,
    p3_2 = m_out3 + s3,
    p1_3 = m_out1 + s3,
    p3_3 = m_out3 + s1;

```

```

function is_error; input x1,x2,x3,x4,x5,x6,x7,x8;
begin
    if (x1||x2||x3||x4||x5||x6||x7||x8) is_error = 1;

```

```

    else is_error = 0;
end
endfunction

assign error = is_error(p0_0[4],p2_0[4],p0_1[4],p2_1[4],
    p1_2[4],p3_2[4],p1_3[4],p3_3[4]);
endmodule

/*****
/*  module compare_select
/*****
/* This module compares the summations from the compute_metric
module and selects the metric and path with the lowest value. The
output of this module is saved as the new path metric for each
state. The ACS output signals are used to control the path memory of
the decoder. */
module compare_select
    (p0_0,p2_0,p0_1,p2_1,p1_2,p3_2,p1_3,p3_3,
    out0,out1,out2,out3,
    ACS0,ACS1,ACS2,ACS3);
input [4:0] p0_0,p2_0,p0_1,p2_1,p1_2,p3_2,p1_3,p3_3;
output [4:0] out0,out1,out2,out3;
output ACS0,ACS1,ACS2,ACS3;

function [4:0] find_min_metric; input [4:0] a,b;
begin
    if (a <= b) find_min_metric = a; else find_min_metric = b;
end
endfunction

function set_control; input [4:0] a,b;
begin
    if (a <= b) set_control = 0; else set_control = 1;
end
endfunction

assign out0 = find_min_metric(p0_0,p2_0);
assign out1 = find_min_metric(p0_1,p2_1);
assign out2 = find_min_metric(p1_2,p3_2);
assign out3 = find_min_metric(p1_3,p3_3);

```

```

assign ACS0 = set_control (p0_0,p2_0);
assign ACS1 = set_control (p0_1,p2_1);
assign ACS2 = set_control (p1_2,p3_2);
assign ACS3 = set_control (p1_3,p3_3);
endmodule

/*****
/*  module path
/*****
/* This is the basic unit for the path memory of the Viterbi
decoder. It consists of four 3-bit D flip-flops in parallel. There
is a 2:1 mux at each D flip-flop input. The statement dff #(12)
instantiates a vector array of 12 flip-flops. */
module path(in,out,clk,reset,ACS0,ACS1,ACS2,ACS3);
input [11:0] in; output [11:0] out;
input clk,reset,ACS0,ACS1,ACS2,ACS3;wire [11:0] p_in;
dff #(12) path0(p_in,out,clk,reset);

    function [2:0] shift_path; input [2:0] a,b; input control;
        begin
            if (control == 0) shift_path = a;else shift_path = b;
        end
    endfunction

assign p_in[11:9] = shift_path(in[11:9],in[5:3],ACS0);
assign p_in[ 8:6] = shift_path(in[11:9],in[5:3],ACS1);
assign p_in[ 5:3] = shift_path(in[8: 6],in[2:0],ACS2);
assign p_in[ 2:0] = shift_path(in[8: 6],in[2:0],ACS3);
endmodule

/*****
/*  module path_memory
/*****
/* This module consists of an array of memory elements (D
flip-flops) that store and shift the path memory as new signals are
added to the four paths (or four most likely sequences of signals).
This module instantiates 11 instances of the path module. */
module path_memory
    (p0,p1,p2,p3,
     path0,clk,reset,
     ACS0,ACS1,ACS2,ACS3);
output [2:0] p0,p1,p2,p3; input [11:0] path0;

```

```

input clk,reset,ACS0,ACS1,ACS2,ACS3;
wire [11:0]out1,out2,out3,out4,out5,out6,out7,out8,out9,out10,out11;
    path x1 (path0,out1 ,clk,reset,ACS0,ACS1,ACS2,ACS3),
        x2 (out1, out2 ,clk,reset,ACS0,ACS1,ACS2,ACS3),
        x3 (out2, out3 ,clk,reset,ACS0,ACS1,ACS2,ACS3),
        x4 (out3, out4 ,clk,reset,ACS0,ACS1,ACS2,ACS3),
        x5 (out4, out5 ,clk,reset,ACS0,ACS1,ACS2,ACS3),
        x6 (out5, out6 ,clk,reset,ACS0,ACS1,ACS2,ACS3),
        x7 (out6, out7 ,clk,reset,ACS0,ACS1,ACS2,ACS3),
        x8 (out7, out8 ,clk,reset,ACS0,ACS1,ACS2,ACS3),
        x9 (out8, out9 ,clk,reset,ACS0,ACS1,ACS2,ACS3),
        x10(out9, out10,clk,reset,ACS0,ACS1,ACS2,ACS3),
        x11(out10,out11,clk,reset,ACS0,ACS1,ACS2,ACS3);
assign p0 = out11[11:9];
assign p1 = out11[ 8:6];
assign p2 = out11[ 5:3];
assign p3 = out11[ 2:0];
endmodule

```

```

/*****
/*  module pathin                                     */
/*****
/* This module determines the input signal to the path for each of
the four paths. Control signals from the subset decoder and compare
select modules are used to store the correct signal. The statement
dff #(12) instantiates a vector array of 12 flip-flops. */
module pathin
    (sout0,sout1,sout2,sout3,
    ACS0,ACS1,ACS2,ACS3,
    path0,clk,reset);
input sout0,sout1,sout2,sout3,ACS0,ACS1,ACS2,ACS3;
input clk,reset; output [11:0] path0;
wire [2:0] sig0,sig1,sig2,sig3;wire [11:0] path_in;

dff #(12) firstpath(path_in,path0,clk,reset);

function [2:0] subset0; input sout0;
    begin
        if(sout0 == 0) subset0 = 0;else subset0 = 4;
    end
endfunction

```

```

function [2:0] subset1; input sout1;
  begin
    if(sout1 == 0) subset1 = 1; else subset1 = 5;
  end
endfunction

```

```

function [2:0] subset2; input sout2;
  begin
    if(sout2 == 0) subset2 = 2; else subset2 = 6;
  end
endfunction

```

```

function [2:0] subset3; input sout3;
  begin
    if(sout3 == 0) subset3 = 3; else subset3 = 7;
  end
endfunction

```

```

function [2:0] find_path; input [2:0] a,b; input control;
  begin
    if(control==0) find_path = a; else find_path = b;
  end
endfunction

```

```

assign sig0 = subset0(sout0);
assign sig1 = subset1(sout1);
assign sig2 = subset2(sout2);
assign sig3 = subset3(sout3);
assign path_in[11:9] = find_path(sig0,sig2,ACS0);
assign path_in[ 8:6] = find_path(sig2,sig0,ACS1);
assign path_in[ 5:3] = find_path(sig1,sig3,ACS2);
assign path_in[ 2:0] = find_path(sig3,sig1,ACS3);
endmodule

```

```

/*****
/*   module metric                               */
/*****
/* The registers created in this module (using D flip-flops) store
the four path metrics. Each register is 5 bits wide. The statement
dff #(5) instantiates a vector array of 5 flip-flops. */

```



```

module metric
    (m_in0,m_in1,m_in2,m_in3,
     m_out0,m_out1,m_out2,m_out3,
     clk,reset);
input [4:0] m_in0,m_in1,m_in2,m_in3;
output [4:0] m_out0,m_out1,m_out2,m_out3;
input clk,reset;
    dff #(5) metric3(m_in3, m_out3, clk, reset);
    dff #(5) metric2(m_in2, m_out2, clk, reset);
    dff #(5) metric1(m_in1, m_out1, clk, reset);
    dff #(5) metric0(m_in0, m_out0, clk, reset);
endmodule

/*****
/*   module output_decision                               */
/*****
/* This module decides the output signal based on the path that
corresponds to the smallest metric. The control signal comes from
the reduce module. */

module output_decision(p0,p1,p2,p3,control,out);
    input [2:0] p0,p1,p2,p3; input [1:0] control; output [2:0] out;
    function [2:0] decide;
    input [2:0] p0,p1,p2,p3; input [1:0] control;
    begin
        if(control == 0) decide = p0;
        else if(control == 1) decide = p1;
        else if(control == 2) decide = p2;
        else decide = p3;
    end
    endfunction

assign out = decide(p0,p1,p2,p3,control);
endmodule

/*****
/*   module reduce                                       */
/*****
/* This module reduces the metrics after the addition and compare
operations. This algorithm selects the smallest metric and subtracts
it from all the other metrics. */

```

```
module reduce
    (in0,in1,in2,in3,
     m_in0,m_in1,m_in2,m_in3,
     control);
    input [4:0] in0,in1,in2,in3;
    output [4:0] m_in0,m_in1,m_in2,m_in3;
    output [1:0] control; wire [4:0] smallest;

    function [4:0] find_smallest;
        input [4:0] in0,in1,in2,in3; reg [4:0] a,b;
        begin
            if(in0 <= in1) a = in0; else a = in1;
            if(in2 <= in3) b = in2; else b = in3;
            if(a <= b) find_smallest = a;
            else find_smallest = b;
        end
    endfunction

    function [1:0] smallest_no;
        input [4:0] in0,in1,in2,in3,smallest;
        begin
            if(smallest == in0) smallest_no = 0;
            else if (smallest == in1) smallest_no = 1;
            else if (smallest == in2) smallest_no = 2;
            else smallest_no = 3;
        end
    endfunction

    assign smallest = find_smallest(in0,in1,in2,in3);
    assign m_in0 = in0 - smallest;
    assign m_in1 = in1 - smallest;
    assign m_in2 = in2 - smallest;
    assign m_in3 = in3 - smallest;
    assign control = smallest_no(in0,in1,in2,in3,smallest);
endmodule
```

11.13 Other Verilog Features

Key terms and concepts: **system tasks** and functions are part of the IEEE standard

11.13.1 Display Tasks

Key terms and concepts: **display system tasks** • \$display (format works like C) • \$write • \$strobe

```

module test_display; // display system tasks:
initial begin $display ("string, variables, or expression");
/* format specifications work like printf in C:
    %d=decimal %b=binary %s=string %h=hex %o=octal
    %c=character %m=hierarchical name %v=length %t=time format
    %e=scientific %f=decimal %g=shortest
examples: %d uses default width %0d uses minimum width
    %7.3g uses 7 spaces with 3 digits after decimal */
// $displayb, $displayh, $displayo print in b, h, o formats
// $write, $strobe, $monitor also have b, h, o versions

$write("write"); // as $display, but without newline at end of line

$strobe("strobe"); // as $display, values at end of simulation cycle

$monitor(v); // disp. @change of v (except v= $time,$stime,$realtime)
$monitoron; $monitoroff; // toggle monitor mode on/off

end endmodule

```

11.13.2 File I/O Tasks

Key terms and concepts: **file I/O system tasks** • \$fdisplay • \$fopen • \$fclose • **multichannel descriptor** • 32 flags • channel 0 is the standard output (screen) and is always open • \$readmemb and \$readmemh read a text file into a memory • file may contain only spaces, new lines, tabs, form feeds, comments, addresses, and binary (\$readmemb) or hex (\$readmemh)

```

module file_1; integer f1, ch; initial begin f1 = $fopen("f1.out");
if(f1==0) $stop(2); if(f1==2)$display("f1 open");
ch = f1|1; $fdisplay(ch,"Hello"); $fclose(f1);end endmodule

```

```

> vlog file_1.v
> vsim -c file_1
# Loading work.file_1
VSIM 1> run 10
# f1 open
# Hello
VSIM 2> q
> more f1.out
Hello
>

```

```

mem.dat
@2 1010_1111 @4 0101_1111 1010_1111 // @address in hex
x1x1_zzzz 1111_0000 /* x or z is OK */

```

```

module load; reg [7:0] mem[0:7]; integer i; initial begin
$readmemb("mem.dat", mem, 1, 6); // start_address=1, end_address=6
for (i= 0; i<8; i=i+1) $display("mem[%0d] %b", i, mem[i]);
end endmodule

```

```

> vsim -c load
# Loading work.load
VSIM 1> run 10
# ** Warning: $readmem (memory mem) file mem.dat line 2:
#   More patterns than index range (hex 1:6)
#   Time: 0 ns Iteration: 0 Instance:/
# mem[0] xxxxxxxx
# mem[1] xxxxxxxx
# mem[2] 10101111
# mem[3] xxxxxxxx
# mem[4] 01011111
# mem[5] 10101111
# mem[6] x1x1zzzz
# mem[7] xxxxxxxx
VSIM 2> q
>

```

11.13.3 Timescale, Simulation, and Timing-Check Tasks

Key terms and concepts: **timescale tasks:** \$prinntimescale and \$timeformat • **simulation control tasks:** \$stop and \$finish • **timing-check tasks** • **edge specifiers** •

'**edge** [01, 0x, x1] clock' is equivalent to '**posedge** clock' • edge transitions with 'z' are treated the same as transitions with 'x' • **notifier register** (changed when a timing-check task detects a violation)

Timing-check system task parameters

Timing task argument	Description of argument	Type of argument
reference_event	to establish reference time	module input or inout (scalar or vector net)
data_event	signal to check against reference_event	module input or inout (scalar or vector net)
limit	time limit to detect timing violation on data_event	constant expression or specparam
threshold	largest pulse width ignored by timing check \$width	constant expression or specparam
notifier	flags a timing violation (before -> after): x->0, 0->1, 1->0, z->z	register

edge_control_specifier ::= **edge** [edge_descriptor {, edge_descriptor}]

edge_descriptor ::= **01** | **0x** | **10** | **1x** | **x0** | **x1**

// timescale tasks:

```
module a; initial $printtimescale(b.c1); endmodule
```

```
module b; c c1 (); endmodule
```

```
`timescale 10 ns / 1 fs
```

```
module c_dat; endmodule
```

```
`timescale 1 ms / 1 ns
```

```
module Ttime; initial $timeformat(-9, 5, " ns", 10); endmodule
```

```
/* $timeformat [ ( n, p, suffix , min_field_width ) ] ;
```

```
units = 1 second ** (-n), n = 0->15, e.g. for n = 9, units = ns
```

```
p = digits after decimal point for %t e.g. p = 5 gives 0.00000
```

```
suffix for %t (despite timescale directive)
```

```
min_field_width is number of character positions for %t */
```

```

module test_simulation_control; // simulation control system tasks:
initial begin $stop; // enter interactive mode (default parameter 1)
$finish(2); // graceful exit with optional parameter as follows:
// 0 = nothing 1 = time and location 2 = time, location, and
statistics
end endmodule

module timing_checks (data, clock, clock_1,clock_2); //1
input data,clock,clock_1,clock_2;reg tSU,tH,tHIGH,tP,tSK,tR; //2
specify // timing check system tasks: //3
/* $setup (data_event, reference_event, limit [, notifier]); //4
violation = (T_reference_event)-(T_data_event) < limit */ //5
$setup(data, posedge clock, tSU); //6
/* $hold (reference_event, data_event, limit [, notifier]); //7
violation = //8
    (time_of_data_event)-(time_of_reference_event) < limit */ //9
$hold(posedge clock, data, tH); //10
/* $setuphold (reference_event, data_event, setup_limit, //11
    hold_limit [, notifier]); //12
parameter_restriction = setup_limit + hold_limit > 0 */ //13
$setuphold(posedge clock, data, tSU, tH); //14
/* $width (reference_event, limit, threshold [, notifier]); //15
violation = //16
    threshold < (T_data_event) - (T_reference_event) < limit //17
reference_event = edge //18
data_event = opposite_edge_of_reference_event */ //19
$width(posedge clock, tHIGH); //20
/* $period (reference_event, limit [, notifier]); //21
violation = (T_data_event) - (T_reference_event) < limit //22
reference_event = edge //23
data_event = same_edge_of_reference_event */ //24
$period(posedge clock, tP); //25
/* $skew (reference_event, data_event, limit [, notifier]); //26
violation = (T_data_event) - (T_reference_event) > limit */ //27
$skew(posedge clock_1, posedge clock_2, tSK); //28
/* $recovery (reference_event, data_event, limit, [, notifier]); //29
violation = (T_data_event) - (T_reference_event) < limit */ //30
$recovery(posedge clock, posedge clock_2, tR); //31
/* $nochange (reference_event, data_event, start_edge_offset, //32
    end_edge_offset [, notifier]); //33
reference_event = posedge | negedge //34

```

```
violation = change while reference high (posedge)/low (negedge) //35
+ve start_edge_offset moves start of window later //36
+ve end_edge_offset moves end of window later */ //37
$nochange (posedge clock, data, 0, 0); //38
endspecify endmodule //39
```

```
primitive dff_udp(q, clock, data, notifier);
output q; reg q; input clock, data, notifier;
table // clock data notifier:state: q
    r    0    ?    : ? : 0 ;
    r    1    ?    : ? : 1 ;
    n    ?    ?    : ? : - ;
    ?    *    ?    : ? : - ;
    ?    ?    *    : ? : x ; endtable // notifier
endprimitive
```

```
`timescale 100 fs / 1 fs
module dff(q, clock, data); output q; input clock, data; reg
notifier;
dff_udp(q1, clock, data, notifier); buf(q, q1);
specify
    specparam tSU = 5, tH = 1, tPW = 20, tPLH = 4:5:6, tPHL = 4:5:6;
    (clock *> q) = (tPLH, tPHL);
    $setup(data, posedge clock, tSU, notifier); // setup: data to clock
    $hold(posedge clock, data, tH, notifier); // hold: clock to data
    $period(posedge clock, tPW, notifier); // clock: period
endspecify
endmodule
```

11.13.4 PLA Tasks

Key terms and concepts: The **PLA modeling tasks** model two-level logic • eqntott logic equations • **array format** ('1' or '0' in **personality array**) • espresso input plane format • **plane format** allows '1', '0', '?' or 'z' (either may be used for don't care) in personality array

```
b1 = a1 & a2; b2 = a3 & a4 & a5 ; b3 = a5 & a6 & a7;
```

```
array.dat
1100000
```

```
0011100
0000111
```

```
module pla_1 (a1,a2,a3,a4,a5,a6,a7,b1,b2,b3);
input a1, a2, a3, a4, a5, a6, a7 ; output b1, b2, b3;
reg [1:7] mem[1:3]; reg b1, b2, b3;
initial begin
    $readmemb("array.dat", mem);
    #1; b1=1; b2=1; b3=1;
    $async$and$array(mem,{a1,a2,a3,a4,a5,a6,a7},{b1,b2,b3});
end
initial $monitor("%4g", $time, ,b1, ,b2, ,b3);
endmodule
```

```
b1 = a1 & !a2; b2 = a3; b3 = !a1 & !a3; b4 = 1;
```

```
module pla_2; reg [1:3] a, mem[1:4]; reg [1:4] b;
initial begin
    $async$and$plane(mem,{a[1],a[2],a[3]},{b[1],b[2],b[3],b[4]});
    mem[1] = 3'b10?; mem[2] = 3'b??1; mem[3] = 3'b0?0; mem[4] = 3'b???;
    #10 a = 3'b111; #10 $displayb(a, " -> ", b);
    #10 a = 3'b000; #10 $displayb(a, " -> ", b);
    #10 a = 3'bxxx; #10 $displayb(a, " -> ", b);
    #10 a = 3'b101; #10 $displayb(a, " -> ", b);
end endmodule
```

```
111 -> 0101
000 -> 0011
xxx -> xxx1
101 -> 1101
```

11.13.5 Stochastic Analysis Tasks

Key terms and concepts: The **stochastic analysis tasks** model queues

```
module stochastic; initial begin // stochastic analysis system tasks:

/* $q_initialize (q_id, q_type, max_length, status) ;
q_id is an integer that uniquely identifies the queue
```


Status values for the stochastic analysis tasks.

Status value	Meaning
0	OK
1	queue full, cannot add
2	undefined q_id
3	queue empty, cannot remove
4	unsupported q_type, cannot create queue
5	max_length <= 0, cannot create queue
6	duplicate q_id, cannot create queue
7	not enough memory, cannot create queue

q_type 1=FIFO 2=LIFO

max_length is an integer defining the maximum number of entries */

```
$q_initialize (q_id, q_type, max_length, status) ;
```

```
/* $q_add (q_id, job_id, inform_id, status) ;
```

```
job_id = integer input
```

```
inform_id = user-defined integer input for queue entry */
```

```
$q_add (q_id, job_id, inform_id, status) ;
```

```
/* $q_remove (q_id, job_id, inform_id, status) ; */
```

```
$q_remove (q_id, job_id, inform_id, status) ;
```

```
/* $q_full (q_id, status) ;
```

```
status = 0 = queue is not full, status = 1 = queue full */
```

```
$q_full (q_id, status) ;
```

```
/* $q_exam (q_id, q_stat_code, q_stat_value, status) ;
```

```
q_stat_code is input request as follows:
```

```
1=current queue length 2=mean inter-arrival time 3=max. queue length
```

```
4=shortest wait time ever
```

```
5=longest wait time for jobs still in queue 6=ave. wait time in queue
```

```
q_stat_value is output containing requested value */
$q_exam (q_id, q_stat_code, q_stat_value, status) ;
```

```
end endmodule
```

11.13.6 Simulation Time Functions

Key terms and concepts: The **simulation time functions** return the time

```
module test_time; initial begin // simulation time system functions:
$time ;
// returns 64-bit integer scaled to timescale unit of invoking module

$stime ;
// returns 32-bit integer scaled to timescale unit of invoking module

$realtime ;
// returns real scaled to timescale unit of invoking module

end endmodule
```

11.13.7 Conversion Functions

Key terms and concepts: The **conversion functions** for reals handle real numbers:

```
module test_convert; // conversion functions for reals:
integer i; real r; reg [63:0] bits;
initial begin #1 r=256;#1 i = $rtoi(r);
#1; r = $itor(2 * i) ; #1 bits = $realtobits(2.0 * r) ;
#1; r = $bitstoreal(bits) ;end
initial $monitor("%3f", $time, ,i, ,r, ,bits); /*
$rtoi converts reals to integers w/truncation e.g. 123.45 -> 123
$itor converts integers to reals e.g. 123 -> 123.0
$realtobits converts reals to 64-bit vector
$bitstoreal converts bit pattern to real
Real numbers in these functions conform to IEEE Std 754. Conversion
rounds to the nearest valid number. */
endmodule
```

```

module test_real; wire [63:0]a; driver d (a); receiver r (a);
initial $monitor("%3g", $time, ,a, ,d.r1, ,r.r2); endmodule

```

```

module driver (real_net);
output real_net; real r1; wire [64:1] real_net = $realtobits(r1);
initial #1 r1 = 123.456; endmodule

```

```

module receiver (real_net);
input real_net; wire [64:1] real_net; real r2;
initial assign r2 = $bitstoreal(real_net);
endmodule

```

11.13.8 Probability Distribution Functions

Key terms and concepts: probability distribution functions • \$random • uniform • normal • exponential • poisson • chi_square • t • erlang

```

module probability; // probability distribution functions: //1
/* $random [(seed)] returns random 32-bit signed integer //2
seed = register, integer, or time */ //3
reg [23:0] r1,r2; integer r3,r4,r5,r6,r7,r8,r9; //4
integer seed, start, \end , mean, standard_deviation; //5
integer degree_of_freedom, k_stage; //6
initial begin seed=1; start=0; \end =6; mean=5; //7
standard_deviation=2; degree_of_freedom=2; k_stage=1; #1; //8
r1 = $random % 60; // random -59 to 59 //9
r2 = {$random} % 60; // positive value 0-59 //10
r3=$dist_uniform (seed, start, \end ) ; //11
r4=$dist_normal (seed, mean, standard_deviation) ; //12
r5=$dist_exponential (seed, mean) ; //13
r6=$dist_poisson (seed, mean) ; //14
r7=$dist_chi_square (seed, degree_of_freedom) ; //15
r8=$dist_t (seed, degree_of_freedom) ; //16
r9=$dist_erlang (seed, k_stage, mean) ; end //17
initial #2 $display ("%3f", $time, ,r1, ,r2, ,r3, ,r4, ,r5); //18
initial begin #3; $display ("%3f", $time, ,r6, ,r7, ,r8, ,r9); end //19
/* All parameters are integer values. //20
Each function returns a pseudo-random number //21
e.g. $dist_uniform returns uniformly distributed random numbers //22
mean, degree_of_freedom, k_stage //23

```

```
(exponential, poisson, chi-square, t, erlang) > 0. //24
seed = inout integer initialized by user, updated by function //25
start, end ($dist_uniform) = integer bounding return values */ //26
endmodule //27
```

```
2.000000      8      57      0      4      9
3.000000      7      3      0      2
```

11.13.9 Programming Language Interface

Key terms and concepts: The C language **Programming Language Interface (PLI)** allows you to access the internal Verilog data structure • three generations of PLI routines • task/function (TF) routines (or utility routines) • access (ACC) routines access delay and logic values • Verilog Procedural Interface (VPI) routines are a superset of the TF and ACC routines

11.14 Summary

Key terms and concepts: concurrent processes and sequential execution • difference between a reg and a wire • scalars and vectors • arithmetic operations on reg and wire • data slip • delays and events

Verilog on one page

Verilog feature	Example
Comments	<pre>a = 0; // comment ends with newline /* This is a multiline or block comment */</pre>
Constants: string and numeric	<pre>parameter BW = 32 // local, BW `define G_BUS 32 // global, `G_BUS 4'b2 1'bx</pre>
Names (case-sensitive, start with letter or '_')	<pre>_l2name A_name \$BAD NotSame notsame</pre>
Two basic types of logic signals: wire and reg	<pre>wire myWire; reg myReg;</pre>
Use continuous assignment statement with wire	<pre>assign myWire = 1;</pre>
Use procedural assignment statement with reg	<pre>always myReg = myWire;</pre>
Buses and vectors use square brackets	<pre>reg [31:0] DBus; DBus[12] = 1'bx;</pre>
We can perform arithmetic on bit vectors	<pre>reg [31:0] DBus; DBus = DBus + 2;</pre>
Arithmetic is performed modulo 2^n	<pre>reg [2:0] R; R = 7 + 1; // now R = 0</pre>
Operators: as in C (but not ++ or --)	
Fixed logic-value system	1, 0, x (unknown), z (high-impedance)
Basic unit of code is the module	<pre>module bake (chips, dough, cookies); input chips, dough; output cookies; assign cookies = chips & dough; endmodule</pre>
Ports	input or input/output ports are wire output ports are wire or reg
Procedures happen at the same time and may be sensitive to an edge, posedge , negedge , or to a level.	<pre>always @rain sing; always @rain dance; always @(posedge clock) D = Q; // flop always @(a or b) c = a & b; // and gate</pre>
Sequential blocks model repeating things: always: executes forever initial: executes once only at start of simulation	<pre>initial born; always @alarm_clock begin : a_day metro=commute; bulot=work; dodo=sleep; end</pre>
Functions and tasks	<pre>function ... endfunction task ... endtask</pre>
Output	<pre>\$display("a=%f",a); \$dumpvars; \$monitor (a)</pre>
Control simulation	<pre>\$stop; \$finish // sudden/gentle halt</pre>
Compiler directives	<pre>`timescale 1ns/1ps // units/resolution</pre>
Delay	<pre>#1 a = b; // delay then sample b a = #1 b; // sample b then delay</pre>

LOGIC SYNTHESIS

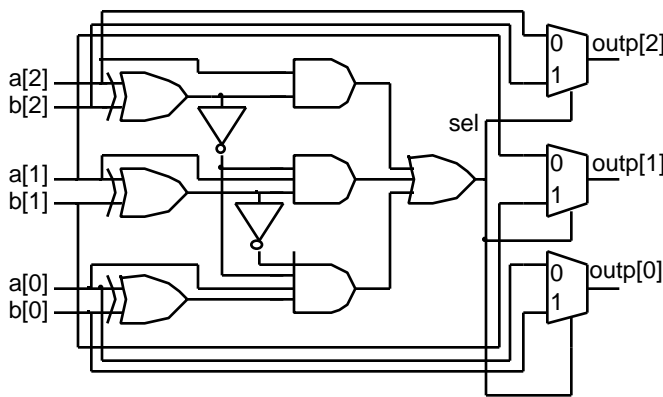
12

Key terms and concepts: **logic synthesis** converts an HDL **behavioral model** (Verilog or VHDL) to a netlist (**structural model**) the same way a C compiler converts C code to machine language • a cell library is called the **target library**

12.1 A Logic-Synthesis Example

A comparison of hand design with synthesis (using a 1.0 μm VLSI Technology cell library)

	Path delay/ ns	No. of standard cells	No. of transistors	Chip area/ mils ²
Hand design	41.6	1,359	16,545	21,877
Synthesized design	36.3	1,493	11,946	18,322



```
// comp_mux.v
module comp_mux(a, b, outp);
  input [2:0] a, b;
  output [2:0] outp;
  function [2:0] compare;
    input [2:0] ina, inb;
  begin
    if (ina <= inb) compare = ina;
    else compare = inb;
  end
endfunction
assign outp = compare(a, b);
endmodule
```

Comparison of the comparator/MUX designs using a 1.0 μm standard-cell library

	Delay /ns	No. of standard cells	No. of transistors	Area /mils ²
Hand design	4.3	12	116	68.68
Synthesized	2.9	15	66	46.43

12.2 A Comparator/MUX

Key terms and concepts: `synopsys_dc.setup` • **script** • **derived schematic** • **analysis** • **elaboration** • **logic optimization** • **logic-mapping** • **timing-analysis (timing engine)**

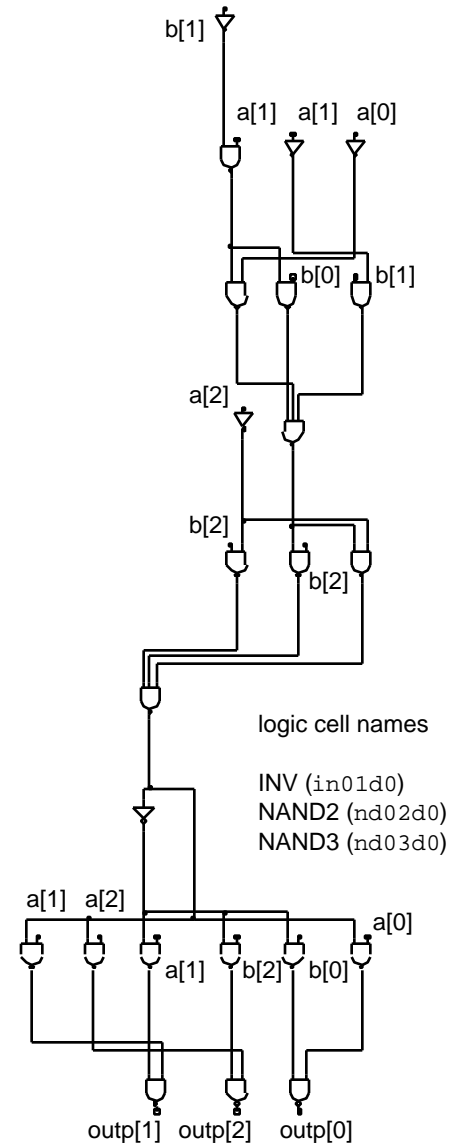
```

`timescale 1ns / 10ps
module comp_mux_u (a, b, outp);
input  [2:0] a; input  [2:0] b;
output [2:0] outp;
supply1 VDD; supply0 VSS;

in01d0 u2 (.I(b[1]), .ZN(u2_ZN));
nd02d0 u3 (.A1(a[1]), .A2(u2_ZN), .ZN(u3_ZN));
in01d0 u4 (.I(a[1]), .ZN(u4_ZN));
nd02d0 u5 (.A1(u4_ZN), .A2(b[1]), .ZN(u5_ZN));
in01d0 u6 (.I(a[0]), .ZN(u6_ZN));
nd02d0 u7 (.A1(u6_ZN), .A2(u3_ZN), .ZN(u7_ZN));
nd02d0 u8 (.A1(b[0]), .A2(u3_ZN), .ZN(u8_ZN));
nd03d0 u9 (.A1(u5_ZN), .A2(u7_ZN), .A3(u8_ZN),
.ZN(u9_ZN));
in01d0 u10 (.I(a[2]), .ZN(u10_ZN));
nd02d0 u11 (.A1(u10_ZN), .A2(u9_ZN),
.ZN(u11_ZN));
nd02d0 u12 (.A1(b[2]), .A2(u9_ZN), .ZN(u12_ZN));
nd02d0 u13 (.A1(u10_ZN), .A2(b[2]), .ZN(u13_ZN));
nd03d0 u14 (.A1(u11_ZN), .A2(u12_ZN),
.A3(u13_ZN), .ZN(u14_ZN));
nd02d0 u15 (.A1(a[2]), .A2(u14_ZN), .ZN(u15_ZN));
in01d0 u16 (.I(u14_ZN), .ZN(u16_ZN));
nd02d0 u17 (.A1(b[2]), .A2(u16_ZN), .ZN(u17_ZN));
nd02d0 u18 (.A1(u15_ZN), .A2(u17_ZN),
.ZN(outp[2]));
nd02d0 u19 (.A1(a[1]), .A2(u14_ZN), .ZN(u19_ZN));
nd02d0 u20 (.A1(b[1]), .A2(u16_ZN), .ZN(u20_ZN));
nd02d0 u21 (.A1(u19_ZN), .A2(u20_ZN),
.ZN(outp[1]));
nd02d0 u22 (.A1(a[0]), .A2(u14_ZN), .ZN(u22_ZN));
nd02d0 u23 (.A1(b[0]), .A2(u16_ZN), .ZN(u23_ZN));
nd02d0 u24 (.A1(u22_ZN), .A2(u23_ZN),
.ZN(outp[0]));

endmodule

```



The comparator/MUX after logic synthesis, but before logic optimization

The structural netlist, `comp_mux_u.v`, and its derived schematic

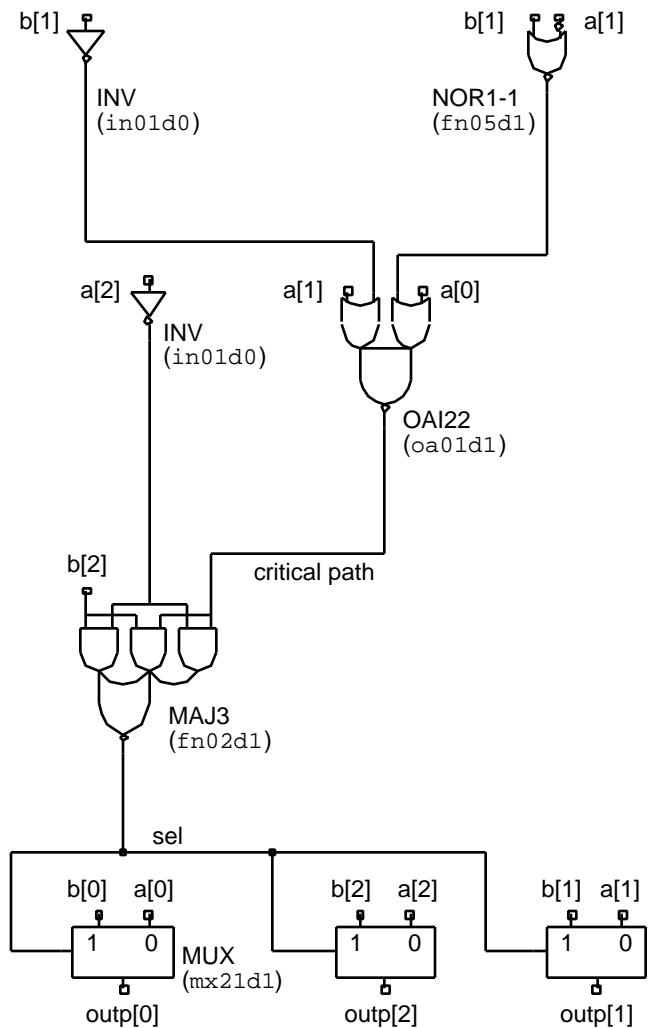

```

`timescale 1ns / 10ps
module comp_mux_o (a, b, outp);
input [2:0] a; input [2:0] b;
output [2:0] outp;
supply1 VDD; supply0 VSS;

in01d0 B1_i1 (.I(a[2]),
.ZN(B1_i1_ZN));
in01d0 B1_i2 (.I(b[1]),
.ZN(B1_i2_ZN));
oa01d1 B1_i3 (.A1(a[0]),
.A2(B1_i4_ZN), .B1(B1_i2_ZN),
.B2(a[1]), .ZN(B1_i3_Z);
fn05d1 B1_i4 (.A1(a[1]), .B1(b[1]),
.ZN(B1_i4_ZN));
fn02d1 B1_i5 (.A(B1_i3_ZN),
.B(B1_i1_ZN), .C(b[2]),
.ZN(B1_i5_ZN));
mx21d1 B1_i6 (.I0(a[0]), .I1(b[0]),
.S(B1_i5_ZN), .Z(outp[0]));
mx21d1 B1_i7 (.I0(a[1]), .I1(b[1]),
.S(B1_i5_ZN), .Z(outp[1]));
mx21d1 B1_i8 (.I0(a[2]), .I1(b[2]),
.S(B1_i5_ZN), .Z(outp[2]));

endmodule

```



The comparator/MUX after logic synthesis and logic optimization with the default settings

The structural netlist, `comp_mux_o.v`, and its derived schematic

12.2.1 An Actel Version of the Comparator/MUX

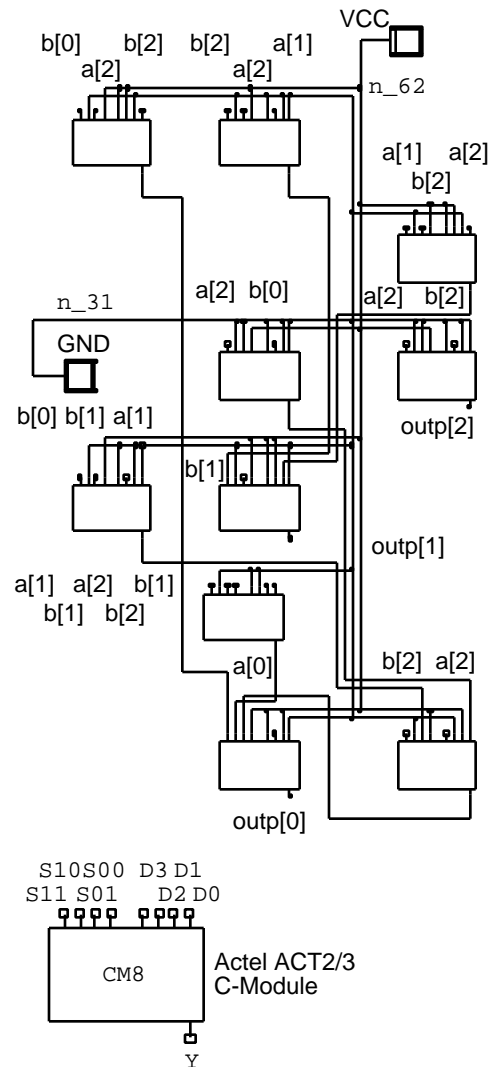
Key terms and concepts: Actel ACT 2/3 FPGA architecture • the symbols represent the eight-input ACT 2/3 C-Module • the logic synthesizer, in the technology-mapping step, decides the connections to the inputs to the logic macro, CM8

```

`timescale 1 ns/100 ps
module comp_mux_actel_o (a, b, outp);
input [2:0] a, b; output [2:0] outp;
wire n_13, n_17, n_19, n_21, n_23, n_27, n_29,
n_31, n_62;

CM8 I_5_CM8(.D0(n_31), .D1(n_62), .D2(a[0]),
.D3(n_62), .S00(n_62), .S01(n_13), .S10(n_23),
.S11(n_21), .Y(outp[0]));
CM8 I_2_CM8(.D0(n_31), .D1(n_19), .D2(n_62),
.D3(n_62), .S00(n_62), .S01(b[1]), .S10(n_31),
.S11(n_17), .Y(outp[1]));
CM8 I_1_CM8(.D0(n_31), .D1(n_31), .D2(b[2]),
.D3(n_31), .S00(n_62), .S01(n_31), .S10(n_31),
.S11(a[2]), .Y(outp[2]));
VCC VCC_I(.Y(n_62));
CM8 I_4_CM8(.D0(a[2]), .D1(n_31), .D2(n_62),
.D3(n_62), .S00(n_62), .S01(b[2]), .S10(n_31),
.S11(a[1]), .Y(n_19));
CM8 I_7_CM8(.D0(b[1]), .D1(b[2]), .D2(n_31),
.D3(n_31), .S00(a[2]), .S01(b[1]), .S10(n_31),
.S11(a[1]), .Y(n_23));
CM8 I_9_CM8(.D0(n_31), .D1(n_31), .D2(a[1]),
.D3(n_31), .S00(n_62), .S01(b[1]), .S10(n_31),
.S11(b[0]), .Y(n_27));
CM8 I_8_CM8(.D0(n_29), .D1(n_62), .D2(n_31),
.D3(a[2]), .S00(n_62), .S01(n_27), .S10(n_31),
.S11(b[2]), .Y(n_13));
CM8 I_3_CM8(.D0(n_31), .D1(n_31), .D2(a[1]),
.D3(n_31), .S00(n_62), .S01(a[2]), .S10(n_31),
.S11(b[2]), .Y(n_17));
CM8 I_6_CM8(.D0(b[2]), .D1(n_31), .D2(n_62),
.D3(n_62), .S00(n_62), .S01(a[2]), .S10(n_31),
.S11(b[0]), .Y(n_21));
CM8 I_10_CM8(.D0(n_31), .D1(n_31), .D2(b[0]),
.D3(n_31), .S00(n_62), .S01(n_31), .S10(n_31),
.S11(a[2]), .Y(n_29));
GND GND_I(.Y(n_31));
endmodule

```



The Actel version of the comparator/MUX after logic optimization

The structural netlist, `comp_mux_actel_o_ad1_e.y` and its derived schematic

12.3 Inside a Logic Synthesizer

Key terms and concepts: The logic synthesizer parses the Verilog and builds an internal data structure (CDFG) • **logic minimization** finds a minimum **cover** • **synthesized network** • **logic optimization** uses factoring, substitution, and elimination • **technology-decomposition** builds a generic network • **technology-mapping (logic-mapping)** matches pieces of the network with the logic cells • we **imply** A • the logic synthesizer has to **infer** B • we must write HDL code so A=B

12.4 Synthesis of the Viterbi Decoder

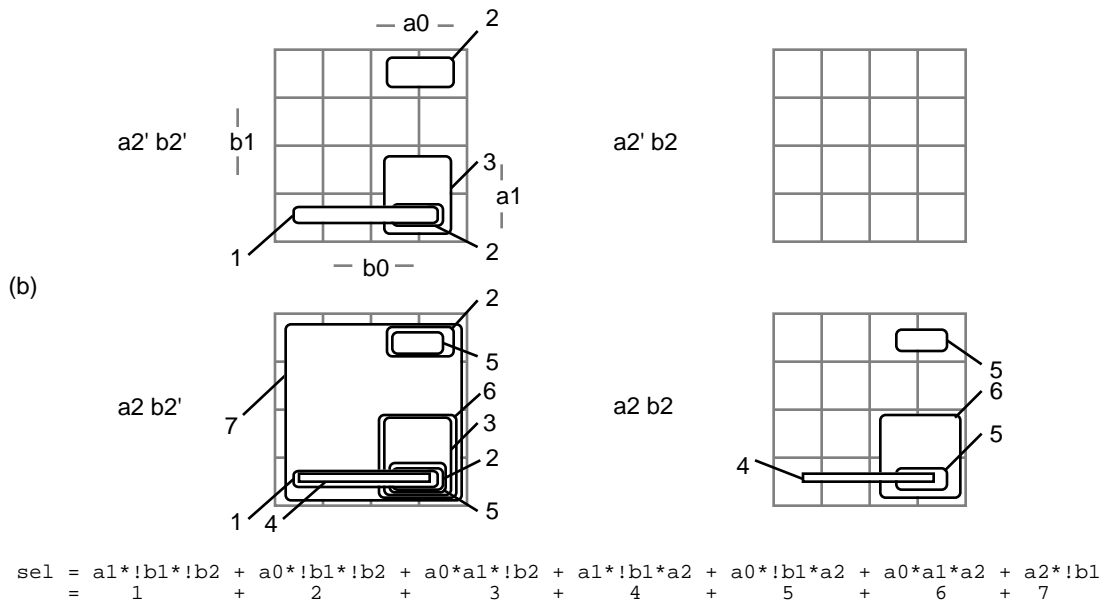
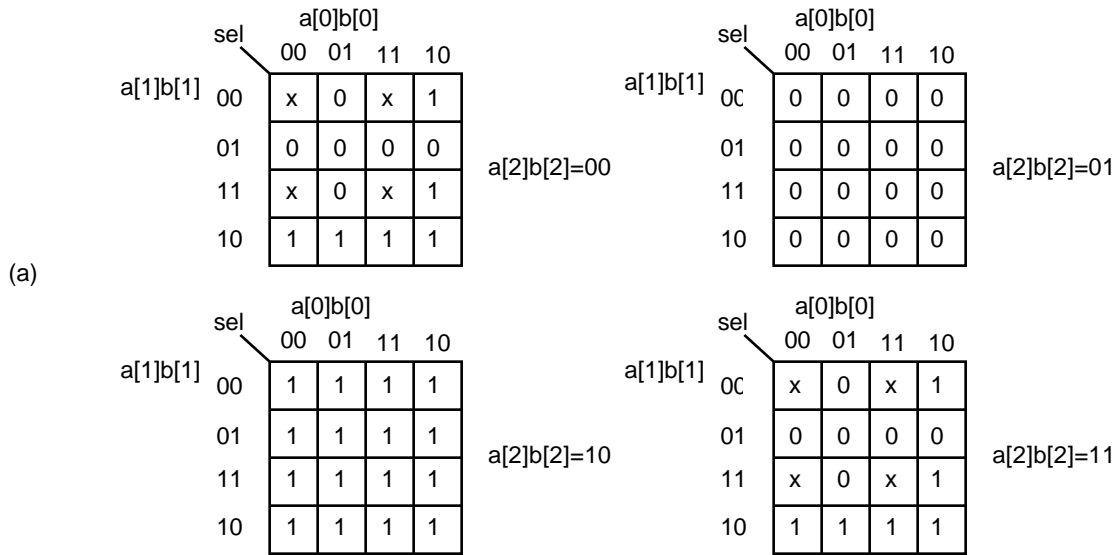
12.4.1 ASIC I/O

Key terms and concepts: inference of I/O cells • directives for special pads (clock buffers) • pull-up resistor, slew rate • no standards • no accepted way to set these parameters from an HDL • generic technology-independent I/O models • instantiate I/O cells directly from a library

```
// asPadBidir #(W, N, S, L, P) I (Pad, toCore, frCore, OEN) //1
// W = width, integer (default=1) //2
// N = pin number string, e.g. "1:3,5:8" //3
// S = strength = {2, 4, 8, 16} in mA drive //4
// L = level = {cmos, ttl, schmitt} (default = cmos) //5
// P = pull-up resistor = {down, float, none, up} //6
// Vxx = {Vss, Vdd} //7

module PadTri (Pad, I, Oen); // active-low output enable //1
parameter width = 1, pinNumbers = "", \strength = 1, //2
    level = "CMOS", externalVdd = 5; //3
output [width-1:0] Pad; input [width-1:0] I; input Oen; //4
assign #1 Pad = (Oen ? {width{1'bz}} : I); //5
endmodule //6

module PadBidir (C, Pad, I, Oen); // active-low output enable //1
parameter width = 1, pinNumbers = "", \strength = 1, //2
    level = "CMOS", pull = "none", externalVdd = 5; //3
output [width-1:0] C; inout [width-1:0] Pad; //4
input [width-1:0] I; input Oen; //5
assign #1 Pad = Oen ? {width{1'bz}} : I; assign #1 C = Pad; //6
endmodule //7
```



Logic maps for the comparator/MUX

(a) If the input b is less than a, then sel is '1'. If a=b, then sel = 'x' (don't care)

(b) A cover for sel.

12.4.2 Flip-Flops

Key terms and concepts: synthesis tools cannot handle two wait statements

```

module dff(D,Q,Clock,Reset); // N.B. reset is active-low //1
output Q; input D,Clock,Reset; //2
parameter CARDINALITY = 1; reg [CARDINALITY-1:0] Q; //3
wire [CARDINALITY-1:0] D; //4
always @(posedge Clock) if (Reset!=0) #1 Q=D; //5
always begin wait (Reset==0); Q=0; wait (Reset==1); end //6
endmodule //7

module dff(D, Q, Clk, Rst); // new flip-flop for Viterbi decoder //1
  parameter width = 1, reset_value = 0; input [width - 1 : 0] D; //2
  output [width - 1 : 0] Q; reg [width - 1 : 0] Q; input Clk, Rst; //3
  initial Q <= {width{1'bx}}; //4
  always @ ( posedge Clk or negedge Rst ) //5
    if ( Rst == 0 ) Q <= #1 reset_value; else Q <= #1 D; //6
endmodule //7

```

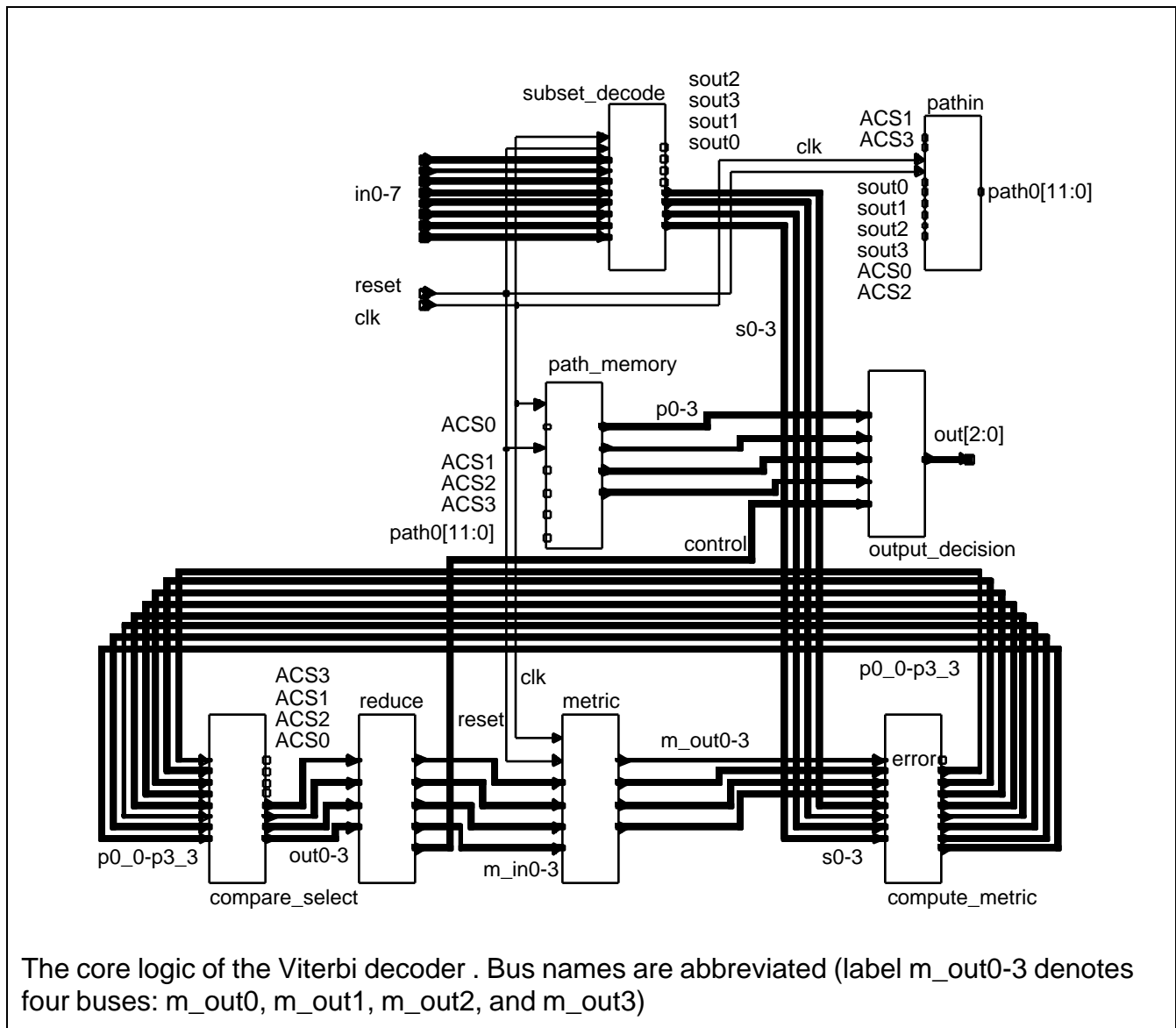
12.4.3 The Top-Level Model

Key terms and concepts: top-level Viterbi decoder • generic input, output, power, and clock I/O cells from the standard-component library

```

/* This is the top-level module, viterbi_ASIC.v */ //1
module viterbi_ASIC //2
(padin0, padin1, padin2, padin3, padin4, padin5, padin6, padin7, //3
padOut, padClk, padRes, padError); //4
input [2:0] padin0, padin1, padin2, padin3, //5
      padin4, padin5, padin6, padin7; //6
input padRes, padClk; output padError; output [2:0] padOut; //7
wire Error, Clk, Res; wire [2:0] Out; // core //8
wire padError, padClk, padRes; wire [2:0] padOut; //9
wire [2:0] in0,in1,in2,in3,in4,in5,in6,in7; // core //10
wire [2:0] //11
  padin0, padin1,padin2,padin3,padin4,padin5,padin6,padin7; //12
// Do not let the software mess with the pads. //13
//compass dontTouch u* //14
  asPadIn #(3,"1,2,3") u0 (in0, padin0); //15
  asPadIn #(3,"4,5,6") u1 (in1, padin1); //16
  asPadIn #(3,"7,8,9") u2 (in2, padin2); //17
  asPadIn #(3,"10,11,12") u3 (in3, padin3); //18
  asPadIn #(3,"13,14,15") u4 (in4, padin4); //19

```



```

asPadIn #(3,"16,17,18") u5 (in5, padin5); //20
asPadIn #(3,"19,20,21") u6 (in6, padin6); //21
asPadIn #(3,"22,23,24") u7 (in7, padin7); //22
asPadVdd #("25","both") u25 (vddb); //23
asPadVss #("26","both") u26 (vssb); //24
asPadClk #("27") u27 (Clk, padClk); //25
asPadOut #(1,"28") u28 (padError, Error); //26
asPadin #(1,"29") u29 (Res, padRes); //27
asPadOut #(3,"30,31,32") u30 (padOut, Out); //28
// Here is the core module: //29
viterbi v_1 //30

```

```
(in0,in1,in2,in3,in4,in5,in6,in7,Out,Clk,Res,Error); //31  
endmodule //32
```

12.5 Verilog and Logic Synthesis

Key terms and concepts: top-down design approach • **stubs** contain a minimum of code

```

module MyChip_ASIC()
  // behavioral "always", etc. ...
  SecondLevelStub1 port mapping
  SecondLevelStub2 port mapping
  ... endmodule
module SecondLevelStub1() ... assign Output1 = ~Input1; endmodule
module SecondLevelStub2() ... assign Output2 = ~Input2;
endmodule

```

12.5.1 Verilog Modeling

Key terms and concepts: **synthesizable** • **synthesis policy** • **modeling style** • **functionally identical**, or **functionally equivalent**

12.5.2 Delays in Verilog

Key terms and concepts: Synthesis tools ignore delay values

```

module Step_Time(clk, phase); //1
  input clk; output [2:0] phase; reg [2:0] phase; //2
  always @(posedge clk) begin //3
    phase <= 4'b0000; //4
    phase <= #1 4'b0001; phase <= #2 4'b0010; //5
    phase <= #3 4'b0011; phase <= #4 4'b0100; //6
  end //7
endmodule //8

module Step_Count (clk_5x, phase); //1
  input clk_5x; output [2:0] phase; reg [2:0] phase; //2
  always@(posedge clk_5x) //3
  case (phase) //4
    0:phase = #1 1; 1:phase = #1 2; 2:phase = #1 3; 3:phase = #1 4; //5
    default: phase = #1 0; //6
  endcase //7
endmodule //8

```

12.5.3 Blocking and Nonblocking Assignments

Key terms and concepts: **race condition** (or a **race**)


```

module race(clk, q0); input clk, q0; reg q1, q2;
always @(posedge clk) q1 = #1 q0; always @(posedge clk) q2 = #1 q1;
endmodule

```

```

module no_race_1(clk, q0, q2); input clk, q0; output q2; reg q1, q2;
always @(posedge clk) begin q2 = q1; q1 = q0; end
endmodule

```

```

module no_race_2(clk, q0, q2); input clk, q0; output q2; reg q1, q2;
always @(posedge clk) q1 <= #1 q0; always @(posedge clk) q2 <= #1 q1;
endmodule

```

12.5.4 Combinational Logic in Verilog

Key terms and concepts: level-sensitive sensitivity list • continuous assignment statements also imply combinational logic

```

module And_Always(x, y, z); input x,y; output z; reg z;
  always @(x or y) z <= x & y; // combinational logic method 1
endmodule

```

```

module And_Assign(x, y, z); input x,y; output z; wire z;
  assign z <= x & y; // combinational logic method 2 = method 1
endmodule

```

```

module And_Or (a,b,c,z); input a,b,c; output z; reg [1:0]z;
  always @(a or b or c) begin z[1]<= &{a,b,c}; z[2]<= |{a,b,c};end
endmodule

```

```

module Parity (BusIn, outp); input[7:0] BusIn; output outp; reg outp;
  always @(BusIn) if (^BusIn == 0) outp = 1; else outp = 0;
endmodule

```

```

module And_Bad(a, b, c); input a, b; output c; reg c;
always@(a) c <= a & b; // b is missing from this sensitivity list
endmodule

```

```

module CL_good(a, b, c); input a, b; output c; reg c;
always@(a or b)
begin c = a + b; d = a & b; e = c + d;end // c, d: LHS before RHS
endmodule

```

```

module CL_bad(a, b, c); input a, b; output c; reg c;
always@(a or b)
begin e = c + d; c = a + b; d = a & b;end // c, d: RHS before LHS
endmodule

```

```

// The complement of this function is too big for synthesis.
module Achilles (out, in); output out; input [30:1] in;
assign out = in[30]&in[29]&in[28] | in[27]&in[26]&in[25]
           | in[24]&in[23]&in[22] | in[21]&in[20]&in[19]
           | in[18]&in[17]&in[16] | in[15]&in[14]&in[13]
           | in[12]&in[11]&in[10] | in[9]& in[8]&in[7]
           | in[6] & in[5]&in[4] | in[3] & in[2]&in[1];
endmodule

```

12.5.5 Multiplexers In Verilog

Key terms and concepts: We imply a MUX using a case or if statement • metalogical values or **simbits** (such as 'x') are not “real” • avoid using casex and casez statements • if you need to “remember” a value, this implies sequential logic

```

module Mux_21a(sel, a, b, z); input sel, a , b; output z; reg z;
always @(a or b or sel)
begin case(sel) 1'b0: z <= a; 1'b1: z <= b;end
endmodule

```

```

module Mux_x(sel, a, b, z); input sel, a, b; output z; reg z;
always @(a or b or sel)
begin case(sel) 1'b0: z <= 0; 1'b1: z <= 1; 1'bx: z <= 'x';end
endmodule

```

```

module Mux_21b(sel, a, b, z); input sel, a, b; output z; reg z;
always @(a or b or sel) begin if (sel) z <= a else z <= b; end
endmodule

```

```

module Mux_Latch(sel, a, b, z); input sel, a, b; output z; reg z;
always @(a or sel) begin if (sel) z <= a; end
endmodule

```

```

module Mux_81(InBus, sel, OE, OutBit); //1
input [7:0] InBus; input [2:0] Sel; //2
input OE; output OutBit; reg OutBit; //3
always @(OE or sel or InBus) //4
  begin //5
    if (OE == 1) OutBit = InBus[sel]; else OutBit = 1'bz; //6
  end //7
endmodule //8

```

12.5.6 The Verilog Case Statement

Key terms and concepts: **exhaustive** • **compiler directive** • **synthesis directive** • **pseudocomment** • an 'x' (**synthesis don't care value**) gives the synthesizer flexibility in optimization • **priority encoder**

```

module case8_oneHot(oneHot, a, b, c, z); //1
input a, b, c; input [2:0] oneHot; output z; reg z; //2
always @(oneHot or a or b or c) //3
begin case(oneHot) //synopsys full_case //4
  3'b001: z <= a; 3'b010: z <= b; 3'b100: z <= c; //5
  default: z <= 1'bx; endcase //6
end //7
endmodule //8

```

```

module case8_priority(oneHot, a, b, c, z); //1
input a, b, c; input [2:0] oneHot; output z; reg z; //2
always @(oneHot or a or b or c) begin //3
case(1'b1) //synopsys parallel_case //4
  oneHot[0]: z <= a; //5
  oneHot[1]: z <= b; //6
  oneHot[2]: z <= c; //7
  default: z <= 1'bx; endcase //8
end //9
endmodule //10

```

12.5.7 Decoders In Verilog

Key terms and concepts: the synthesizer infers a three-state buffer from an assignment of 'z'

```

module Decoder_4To16(enable, In_4, Out_16); // 4-to-16 decoder //1
input enable; input [3:0] In_4; output [15:0] Out_16; //2
reg [15:0] Out_16; //3
always @(enable or In_4) //4
    begin Out_16 = 16'hzzzz; //5
        if (enable == 1) //6
            begin Out_16 = 16'h0000; Out_16[In_4] = 1; end //7
        end //8
    endmodule //9

if (enable === 1) // can't make logic to check for enable = x or z

```

12.5.8 Priority Encoder in Verilog

Key terms and concepts: The logic synthesizer must be able to unroll a loop in a for statement.

```

module Pri_Encoder32 (InBus, Clk, OE, OutBus); //1
input [31:0]InBus; input OE, Clk; output [4:0]OutBus; //2
reg j; reg [4:0]OutBus; //3
always@(posedge Clk) //4
    begin //5
        if (OE == 0) OutBus = 5'bz ; //6
        else //7
            begin OutBus = 0; //8
                for (j = 31; j >= 0; j = j - 1) //9
                    begin if (InBus[j] == 1) OutBus = j; end //10
                end //11
            end //12
        endmodule //13

```

12.5.9 Arithmetic in Verilog

Key terms and concepts: make room for the carry bit when you add two numbers in Verilog •

resource allocation • resource sharing • multiplication assumes nets are unsigned

```

module Adder_8 (A, B, Z, Cin, Cout); //1
input [7:0] A, B; input Cin; output [7:0] Z; output Cout; //2
assign {Cout, Z} = A + B + Cin; //3
endmodule //4

module Adder_16 (A, B, Sum, Cout); //1
input [15:0] A, B; output [15:0] Sum; output Cout; //2

```

```

reg [15:0] Sum; reg Cout; //3
always @(A or B) {Cout, Sum} = A + B + 1; // One adder not two! //4
endmodule //5

module Add_A (sel, a, b, c, d, y); //1
input a, b, c, d, sel; output y; reg y; //2
always@(sel or a or b or c or d) // One or two adders? //3
begin if (sel == 0) y <= a + b; else y <= c + d; end //4
endmodule //5

module Add_B (sel, a, b, c, d, y); //1
input a, b, c, d, sel; output y; reg t1, t2, y; //2
always@(sel or a or b or c or d) begin // One adder not two! //3
if (sel == 0) begin t1 = a; t2 = b; end // Temporary //4
else begin t1 = c; t2 = d; end // variables. //5
y = t1 + t2; end //6
endmodule //7

module Multiply_unsigned (A, B, Z); //1
input [1:0] A, B; output [3:0] Z; //2
assign Z <= A * B; //3
endmodule //4

module Multiply_signed (A, B, Z); //1
input [1:0] A, B; output [3:0] Z; //2
// 00 -> 00_00 01 -> 00_01 10 -> 11_10 11 -> 11_11 //3
assign Z = { { 2{A[1]} }, A} * { { 2{B[1]} }, B}; //4
endmodule //5

```

12.5.10 Sequential Logic in Verilog

Key terms and concepts: edges (**posedge** or **negedge**) in the sensitivity list of an **always** statement imply a clocked storage element • however, an **always** statement does not have to be edge-sensitive to imply sequential logic • all sequential logic cells must be initialized • **template** • **synthesis style guide**

```

always@(posedge clock) Q_flipflop = D; // A flip-flop.
always@(clock or D) if (clock) Q_latch = D; // A latch.
always@(posedge clock or negedge reset) // names mean nothing,
always@(posedge day or negedge year) // which is the reset?

```

```

always@(posedge clk or negedge reset) begin // Template for reset:
    if (reset == 0) Q = 0; // initialize,
    else Q = D;           // normal clocking
end

```

```

module Counter_With_Reset (count, clock, reset); //1
input clock, reset; output count; reg [7:0] count; //2
always @ (posedge clock or negedge reset) //3
    if (reset == 0) count = 0; else count = count + 1; //4
endmodule //5

```

```

module DFF_MasterSlave (D, clock, reset, Q); // D type flip-flop //1
input D, clock, reset; output Q; reg Q, latch; //2
always @(posedge clock or posedge reset) //3
    if (reset == 1) latch = 0; else latch = D; // the master. //4
always @(latch) Q = latch; // the slave. //5
endmodule //6

```

12.5.11 Component Instantiation in Verilog

Key terms and concepts: HDL description is technology-independent (CMOS, FPGA, TTL, GaAs) • the only way to use a particular cell is to use structural Verilog and **hand instantiation**

- dont_touch • **soft models** or **standard components** • **DesignWare**

```

//Compass dontTouch my_inv_8x or // synopsys dont_touch
INVD8 my_inv_8x(.I(a), .ZN(b) );

```

```

module Count4(clk, reset, Q0, Q1, Q2, Q3); //1
input clk, reset; output Q0, Q1, Q2, Q3; wire Q0, Q1, Q2, Q3; //2
//           Q , D , clk, reset //3
asDff dff0( Q0, ~Q0, clk, reset); // The asDff is a //4
asDff dff1( Q1, ~Q1, Q0, reset); // standard component, //5
asDff dff2( Q2, ~Q2, Q1, reset); // unique to one set of tools. //6
asDff dff3( Q3, ~Q3, Q2, reset); //7
endmodule //8

```

```

module asDff (D, Q, Clk, Rst); //1
parameter width = 1, reset_value = 0; //2
input [width-1:0] D; output [width-1:0] Q; reg [width-1:0] Q; //3
input Clk, Rst; initial Q = {width{1'bx}}; //4
    always @ ( posedge Clk or negedge Rst ) //5

```

```

    if ( Rst==0 ) Q <= #1 reset_value;else Q <= #1 D;           //6
endmodule                                                       //7

```

12.5.12 Datapath Synthesis in Verilog

Key terms and concepts: Datapath synthesis • Synopsys VHDL DesignWare • compiler directives • X-BLOX • LPM (library of parameterized modules) • RPM (relationally placed modules) • thinking like the hardware

```

module DP_csum(A1,B1,Z1); input [3:0] A1,B1; output Z1; reg [3:0] Z1;
always@(A1 or B1) Z1 <= A1 + B1;//Compass adder_arch cond_sum_add
endmodule

```

```

module DP_ripp(A2,B2,Z2); input [3:0] A2,B2; output Z2; reg [3:0] Z2;
always@(A2 or B2) Z2 <= A2 + B2;//Compass adder_arch ripple_add
endmodule

```

```

module DP_sub_A(A,B,OutBus,CarryIn); //1
input [3:0] A, B ; input CarryIn ; //2
output OutBus ; reg [3:0] OutBus ; //3
always@(A or B or CarryIn) OutBus <= A - B - CarryIn ; //4
endmodule //5

```

```

module DP_sub_B (A, B, CarryIn, Z) ; //1
input [3:0] A, B, CarryIn ; output [3:0] Z; reg [3:0] Z; //2
always@(A or B or CarryIn) begin //3
    case (CarryIn) //4
        1'b1 : Z <= A - B - 1'b1; //5
        default : Z <= A - B - 1'b0; endcase //6
    end //7
endmodule //8

```

12.6 VHDL and Logic Synthesis

Key terms and concepts: IEEE VHDL nine-value system • You can use '1', 'H', '0', and 'L' in any manner • Some synthesis tools do not accept 'U' • You can use logic states 'Z', 'X', 'W', and '-' in assignments in any manner • 'Z' is synthesized to three-state logic • 'X', 'W',

and ' - ' are treated as unknown or don't care values • The IEEE synthesis packages provide the `STD_MATCH` function for comparisons

12.6.1 Initialization and Reset

Key terms and concepts: a VHDL `process` with a sensitivity list synthesizes to clocked logic with a reset

```
process (signal_1, signal_2) begin
  if (signal_2'EVENT and signal_2 = '0')
    then -- Insert initialization and reset statements.
    elsif (signal_1'EVENT and signal_1 = '1')
    then -- Insert clocking statements.
  end if;
end process;
```

12.6.2 Combinational Logic Synthesis in VHDL

Key terms and concepts: a **level-sensitive process** has a sensitivity list with signals that are not tested for event attributes ('EVENT' or 'STABLE', for example) • combinational logic uses a level-sensitive `process` or a concurrent assignment statement • some synthesizers do not allow a signal inside a level-sensitive `process` unless the signal is in the sensitivity list

```
entity And_Bad is port (a, b: in BIT; c: out BIT); end And_Bad;
```

```
architecture Synthesis_Bad of And_Bad is
  begin process (a) -- this should be process (a, b)
    begin c <= a and b;
  end process;
end Synthesis_Bad;
```

12.6.3 Multiplexers in VHDL

Key terms and concepts: multiplexers can be synthesized using an (exhaustive) `case` statement (avoid the reserved word 'select') • a concurrent signal assignment is equivalent


```

entity Mux4 is port
(i: BIT_VECTOR(3 downto 0); sel: BIT_VECTOR(1 downto 0); s: out BIT);
end Mux4;

```

```

architecture Synthesis_1 of Mux4 is
  begin process(sel, i) begin
    case sel is
      when "00" => s <= i(0); when "01" => s <= i(1);
      when "10" => s <= i(2); when "11" => s <= i(3);
    end case;
  end process;
end Synthesis_1;

```

```

architecture Synthesis_2 of Mux4 is
  begin with sel select s <=
    i(0) when "00", i(1) when "01", i(2) when "10", i(3) when "11";
end Synthesis_2;

```

```

library IEEE; use ieee.std_logic_1164 all;
entity Mux8 is port
(InBus : in STD_LOGIC_VECTOR(7 downto 0);
Sel : in INTEGER range 0 to 7;
OutBit : out STD_LOGIC);
end Mux8;

```

```

architecture Synthesis_1 of Mux8 is
  begin process(InBus, Sel)
    begin OutBit <= InBus(Sel);
  end process;
end Synthesis_1;

```

12.6.4 Decoders in VHDL

```

library IEEE; --1
use IEEE.STD_LOGIC_1164 all; use IEEE.NUMERIC_STD all; --2
entity Decoder is port (enable : in BIT; --3
  Din: STD_LOGIC_VECTOR (2 downto 0); --4
  Dout: out STD_LOGIC_VECTOR (7 downto 0)); --5
end Decoder; --6

```

```

architecture Synthesis_1 of Decoder is                                --7
  begin                                                                --8
  with enable select Dout <=                                         --9
  STD_LOGIC_VECTOR                                                    --10
  (UNSIGNED'                                                           --11
    (shift_left                                                       --12
      ("00000001", TO_INTEGER (UNSIGNED(Din))                         --13
    )                                                                    --14
  )                                                                    --15
)                                                                        --16
  when '1',                                                            --17
  "11111111" when '0', "00000000" when others;                    --18
end Synthesis_1;                                                    --19

library IEEE;                                                        --1
use IEEE.NUMERIC_STD all; use IEEE.STD_LOGIC_1164 all;         --2

entity Concurrent_Decoder is port (                                  --3
  enable : in BIT;                                                    --4
  Din : in STD_LOGIC_VECTOR (2 downto 0);                            --5
  Dout : out STD_LOGIC_VECTOR (7 downto 0));                          --6
end Concurrent_Decoder;                                             --7

architecture Synthesis_1 of Concurrent_Decoder is                --8
begin process (Din, enable)                                          --9
  variable T : STD_LOGIC_VECTOR(7 downto 0);                        --10
  begin                                                                --11
  if (enable = '1') then                                            --12
    T := "00000000"; T( TO_INTEGER (UNSIGNED(Din))) := '1';         --13
    Dout <= T ;                                                       --14
  else Dout <= (others => 'Z');                                     --15
  end if;                                                            --16
end process;                                                        --17
end Synthesis_1;                                                    --18

```

12.6.5 Adders in VHDL

Key terms and concepts: To add two n -bit numbers and keep the overflow bit, we need to assign to a signal with more bits

```

library IEEE;                                                        --1
use IEEE.NUMERIC_STD all; use IEEE.STD_LOGIC_1164 all;         --2

entity Adder_1 is                                                  --3
port (A, B: in UNSIGNED(3 downto 0); C: out UNSIGNED(4 downto 0)); --4
end Adder_1;                                                        --5

architecture Synthesis_1 of Adder_1 is                            --6

```

```

    begin C <= ('0' & A) + ('0' & B);           --7
end Synthesis_1;                             --8

```

12.6.6 Sequential Logic in VHDL

Key terms and concepts: Sensitivity to an edge implies sequential logic in VHDL • Either: (1) no sensitivity list with a **wait until** statement (2) a sensitivity list and test for 'EVENT plus a specific level • any signal assigned in an edge-sensitive **process** statement should be reset—but be careful to distinguish between asynchronous and synchronous resets

```

library IEEE; use IEEE.STD_LOGIC_1164 all; entity DFF_With_Reset is
    port(D, Clk, Reset : in STD_LOGIC; Q : out STD_LOGIC);
end DFF_With_Reset;

```

```

architecture Synthesis_1 of DFF_With_Reset is
    begin process(Clk, Reset) begin
        if (Reset = '0') then Q <= '0'; -- asynchronous reset
            elsif rising_edge(Clk) then Q <= D;
        end if;
    end process;
end Synthesis_1;

```

```

architecture Synthesis_2 of DFF_With_Reset is
    begin process begin
        wait until rising_edge(Clk);
-- This reset is gated with the clock and is synchronous:
        if (Reset = '0') then Q <= '0'; else Q <= D; end if;
    end process;
end Synthesis_2;

```

Key terms and concepts: sequential logic results when we have to “remember” something between successive executions of a **process** statement. This occurs when a **process** statement contains one or more of the following situations (1) A signal is read but is not in the

sensitivity list of a **process** statement (2) A signal or variable is read before it is updated (3) A signal is not always updated (4) There are multiple **wait** statements

Not all of the models that we could write using the above constructs will be synthesizable. Any models that do use one or more of these constructs and that are synthesizable will result in sequential logic.

12.6.7 Instantiation in VHDL

Key terms and concepts: to help hand instantiate a component generate a structural netlist

```

`timescale 1ns/1ns //1
module halfgate (myInput, myOutput); //2
input myInput; output myOutput; wire myOutput; //3
    assign myOutput = ~myInput; //4
endmodule //5

library IEEE; use IEEE.STD_LOGIC_1164 all; --1
library COMPASS_LIB; use COMPASS_LIB.COMPASS all; --2
--compass compile_off -- synopsys etc. --3
use COMPASS_LIB.COMPASS_ETC all; --4
--compass compile_on -- synopsys etc. --5
entity halfgate_u is --6
--compass compile_off -- synopsys etc. --7
generic ( --8
    myOutput_cap : Real := 0.01; --9
    INSTANCE_NAME : string := "halfgate_u" ); --10
--compass compile_on -- synopsys etc. --11
port ( myInput : in Std_Logic := 'U'; --12
myOutput : out Std_Logic := 'U' ); --13
end halfgate_u; --14

architecture halfgate_u of halfgate_u is --15
component in01d0 --16
port ( I : in Std_Logic; ZN : out Std_Logic ); end component; --17
begin --18
u2: in01d0 port map ( I => myInput, ZN => myOutput ); --19
end halfgate_u; --20

--compass compile_off -- synopsys etc. --21
library cb60hd230d; --22
configuration halfgate_u_CON of halfgate_u is --23
    for halfgate_u --24
        for u2 : in01d0 use configuration cb60hd230d.in01d0_CON --25
            generic map ( --26

```

```

        ZN_cap => 0.0100 + myOutput_cap,           --27
        INSTANCE_NAME => INSTANCE_NAME&"/u2" )    --28
    port map ( I => I, ZN => ZN);                 --29
    end for;                                     --30
end for;                                        --31
end halfgate_u_CON;                             --32
--compass compile_on -- synopsys etc.          --33

component ASDFF
    generic (WIDTH : POSITIVE := 1;
        RESET_VALUE : STD_LOGIC_VECTOR := "0" );
    port      (Q      : out STD_LOGIC_VECTOR (WIDTH-1 downto 0);
        D      : in  STD_LOGIC_VECTOR (WIDTH-1 downto 0);
        CLK    : in  STD_LOGIC;
        RST    : in  STD_LOGIC );
end component;

library IEEE, COMPASS_LIB;                      --1
use IEEE.STD_LOGIC_1164 all; use COMPASS_LIB.STDCOMP all; --2
entity Ripple_4 is                              --3
    port (Trig, Reset: STD_LOGIC; QN0_5x:out STD_LOGIC; --4
        Q : inout STD_LOGIC_VECTOR(0 to 3)); --5
end Ripple_4;                                   --6
architecture structure of Ripple_4 is          --7
    signal QN : STD_LOGIC_VECTOR(0 to 3); --8
    component in01d1 --9
    port ( I : in Std_Logic; ZN : out Std_Logic );end component; --10
    component in01d5 --11
    port ( I : in Std_Logic; ZN : out Std_Logic );end component; --12
begin --13
--compass dontTouch inv5x -- synopsys dont_touch etc. --14
-- Named association for hand-instantiated library cells: --15
    inv5x: IN01D5 port map( I=>Q(0), ZN=>QN0_5x ); --16
    inv0 : IN01D1 port map( I=>Q(0), ZN=>QN(0) ); --17
    inv1 : IN01D1 port map( I=>Q(1), ZN=>QN(1) ); --18
    inv2 : IN01D1 port map( I=>Q(2), ZN=>QN(2) ); --19
    inv3 : IN01D1 port map( I=>Q(3), ZN=>QN(3) ); --20
-- Positional association for standard components: --21
--
--           Q           D           Clk   Rst
d0: asDFF port map(Q (0 to 0), QN(0 to 0), Trig, Reset); --23
d1: asDFF port map(Q (1 to 1), QN(1 to 1), Q(0), Reset); --24
d2: asDFF port map(Q (2 to 2), QN(2 to 2), Q(1), Reset); --25

```

```

    d3: asDFF port map(Q (3 to 3), QN(3 to 3), Q(2), Reset);      --26
end structure;                                                --27

`timescale 1ns / 10ps                                         //1
module ripple_4_u (trig, reset, qn0_5x, q);                   //2
input trig; input reset; output qn0_5x; inout [3:0] q;       //3
wire [3:0] qn; supply1 VDD; supply0 VSS;                     //4
in01d5 inv5x (.I(q[0]),.ZN(qn0_5x));                           //5
in01d1 inv0 (.I(q[0]),.ZN(qn[0]));                             //6
in01d1 inv1 (.I(q[1]),.ZN(qn[1]));                             //7
in01d1 inv2 (.I(q[2]),.ZN(qn[2]));                             //8
in01d1 inv3 (.I(q[3]),.ZN(qn[3]));                             //9
dfctnb d0(.D(qn[0]),.CP(trig),.CDN(reset),.Q(q[0]),.QN(\d0.QN )); //10
dfctnb d1(.D(qn[1]),.CP(q[0]),.CDN(reset),.Q(q[1]),.QN(\d1.QN )); //11
dfctnb d2(.D(qn[2]),.CP(q[1]),.CDN(reset),.Q(q[2]),.QN(\d2.QN )); //12
dfctnb d3(.D(qn[3]),.CP(q[2]),.CDN(reset),.Q(q[3]),.QN(\d3.QN )); //13
endmodule                                                       //14

```

12.6.8 Shift Registers and Clocking in VHDL

```

library IEEE;                                                  --1
use IEEE.STD_LOGIC_1164 all; use IEEE.NUMERIC_STD all;       --2

entity SIPO_1 is port (                                       --3
    Clk : in STD_LOGIC;                                       --4
    SI : in STD_LOGIC; -- serial in                            --5
    PO : buffer STD_LOGIC_VECTOR(3 downto 0)); -- parallel out --6
end SIPO_1;                                                    --7

architecture Synthesis_1 of SIPO_1 is                          --8
    begin process (Clk) begin                                  --9
        if (Clk = '1') then PO <= SI & PO(3 downto 1); end if; --10
    end process;                                               --11
end Synthesis_1;                                              --12

module sipo_1_u (clk, si, po);                                  //1
input clk; input si; output [3:0] po;                          //2
supply1 VDD; supply0 VSS;                                     //3
dfntnb po_ff_b0 (.D(po[1]),.CP(clk),.Q(po[0]),.QN(\po_ff_b0.QN)); //4
dfntnb po_ff_b1 (.D(po[2]),.CP(clk),.Q(po[1]),.QN(\po_ff_b1.QN)); //5
dfntnb po_ff_b2 (.D(po[3]),.CP(clk),.Q(po[2]),.QN(\po_ff_b2.QN)); //6
dfntnb po_ff_b3 (.D(si),.CP(clk),.Q(po[3]),.QN(\po_ff_b3.QN )); //7
endmodule                                                       //8

library IEEE;                                                  --1
use IEEE.STD_LOGIC_1164 all; use IEEE.NUMERIC_STD all;       --2

```

```

entity SIPO_R is port (                                     --3
  clk : in STD_LOGIC ; res : in STD_LOGIC ;                 --4
  SI : in STD_LOGIC ; PO : out STD_LOGIC_VECTOR(3 downto 0)); --5
end;                                                         --6

architecture Synthesis_1 of SIPO_R is                       --7
  signal PO_t : STD_LOGIC_VECTOR(3 downto 0);                 --8
  begin                                                         --9
  process (PO_t) begin PO <= PO_t; end process;               --10
  process (clk, res) begin                                     --11
    if (res = '0') then PO_t <= (others => '0');               --12
    elsif (rising_edge(clk)) then PO_t <= SI & PO_t(3 downto 1); --13
    end if;                                                   --14
  end process;                                               --15
end Synthesis_1;                                             --16

```

12.6.9 Adders and Arithmetic Functions

Key terms and concepts: to perform BIT_VECTOR or STD_LOGIC_VECTOR arithmetic you have three choices: (1) Use a vendor-supplied package (2) Convert to SIGNED (or UNSIGNED) and use the IEEE standard synthesis packages (3) Use overloaded functions in packages or functions that you define yourself

```

library IEEE;                                               --1
use IEEE.STD_LOGIC_1164 all; use IEEE.NUMERIC_STD all;   --2

entity Adder4 is port (                                     --3
  in1, in2 : in BIT_VECTOR(3 downto 0) ;                   --4
  mySum : out BIT_VECTOR(3 downto 0) ) ;                   --5
end Adder4;                                                 --6

architecture Behave_A of Adder4 is                         --7
  function DIY(L,R: BIT_VECTOR(3 downto 0)) return BIT_VECTOR is --8
  variable sum:BIT_VECTOR(3 downto 0);variable lt,rt,st,cry: BIT; --9
  begin cry := '0';                                          --10
  for i in L'REVERSE_RANGE loop                               --11
    lt := L(i); rt := R(i); st := ltxor rt;                  --12
    sum(i):= st xor cry; cry:= (lt and rt) or (st and cry); --13
  end loop;                                                 --14
  return sum;                                               --15
  end;                                                         --16
  begin mySum <= DIY (in1, in2); -- do it yourself (DIY) add --17
end Behave_A;                                             --18

library IEEE;                                               --1
use IEEE.STD_LOGIC_1164 all; use IEEE.NUMERIC_STD all;   --2

```

```

entity Adder4 is port (                                     --3
  in1, in2 : in UNSIGNED(3 downto 0) ;                    --4
  mySum : out UNSIGNED(3 downto 0) ) ;                    --5
end Adder4;                                               --6

architecture Behave_B of Adder4 is                       --7
  begin mySum <= in1 + in2; -- This uses an overloaded '+'. --8
end Behave_B;                                           --9

```

12.6.10 Adder/Subtractor and Don't Cares

Key terms and concepts: whether to use simple code or more complex code that more accurately describes the hardware?

```

library IEEE;                                           --1
use IEEE.STD_LOGIC_1164 all; use IEEE.NUMERIC_STD all; --2
entity Adder_Subtractor is port (                       --3
  xin : in UNSIGNED(15 downto 0);                       --4
  clk, addsub, clr: in STD_LOGIC;                       --5
  result : out UNSIGNED(15 downto 0));                 --6
end Adder_Subtractor;                                    --7

architecture Behave_A of Adder_Subtractor is           --8
  signal addout, result_t: UNSIGNED(15downto 0);        --9
  begin                                                  --10
    result <= result_t;                                  --11
    with addsub select                                  --12
    addout <= (xin + result_t) when '1',                --13
              (xin - result_t) when '0',                --14
              (others => '-') when others;             --15
    process (clr, clk) begin                             --16
      if (clr = '0') then result_t <= (others => '0'); --17
      elsif rising_edge(clk) then result_t <= addout; --18
      end if;                                           --19
    end process;                                        --20
  end Behave_A;                                         --21

architecture Behave_B of Adder_Subtractor is         --1
  signal result_t: UNSIGNED(15 downto 0);              --2
  begin                                                 --3
    result <= result_t;                                  --4
    process (clr, clk) begin                             --5
      if (clr = '0') then result_t <= (others => '0'); --6
      elsif rising_edge(clk) then                       --7
        case addsub is                                   --8
          when '1' => result_t <= (xin + result_t);      --9
          when '0' => result_t <= (xin - result_t);     --10

```



```

end //19
always @(curSt or yOut) // Assign the next state: //20
begin:Comb //21
    case (curSt) //22
        `resSt:nextSt = `S3; `S1:nextSt = `S2; //23
        `S2:nextSt = `S1; `S3:nextSt = `S1; //24
        default:nextSt = `resSt; //25
    endcase //26
end //27
endmodule //28

module StateMachine_2 (reset, clk, yOutReg); //1
    input reset, clk; output yOutReg; reg yOutReg, yOut; //2
    parameter [1:0] //synopsys enum states //3
        resSt = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11; //4
    reg [1:0] /* synopsys enum states */ curSt, nextSt; //5
//synopsys state_vector curSt //6
always @(posedge clk or posedge reset) begin //7
    if (reset == 1) //8
        begin yOut = 0; yOutReg = yOut; curSt = resSt;end //9
    else begin //10
        case (curSt) resSt:yOut = 0;S1:yOut = 1;S2:yOut = 1;S3:yOut = 1//11
            default:yOut = 0; endcase //12
        yOutReg = yOut; curSt = nextSt;end //13
    end //14
always @(curSt or yOut) begin //15
    case (curSt) //16
        resSt:nextSt = S3; S1:nextSt = S2; S2:nextSt = S1; S3:nextSt = S1//17
        default:nextSt = S1; endcase //18
    end //19
endmodule //20

parameter [3:0] //synopsys enum states
    resSt = 4'b0000, S1 = 4'b0010, S2 = 4'b0100, S3 = 4'b1000;

```

12.7.2 FSM Synthesis in VHDL

Key terms and concepts: Moore state machine • Mealy state machine • An FSM compiler extracts a state machine

```

library IEEE; use IEEE.STD_LOGIC_1164 all;                                --1
entity SM1 is                                                                --2
  port (aIn, clk : in Std_logic; yOut: out Std_logic);                       --3
end SM1;                                                                       --4

architecture Moore of SM1 is                                               --5
  type state is (s1, s2, s3, s4);                                           --6
  signal pS, nS : state;                                                     --7
  begin                                                                        --8
  process (aIn, pS) begin                                                    --9
    case pS is                                                                --10
    when s1 => yOut <= '0'; nS <= s4;                                         --11
    when s2 => yOut <= '1'; nS <= s3;                                         --12
    when s3 => yOut <= '1'; nS <= s1;                                         --13
    when s4 => yOut <= '1'; nS <= s2;                                         --14
    end case;                                                                  --15
  end process;                                                                --16
  process begin                                                                --17
    -- synopsys etc.                                                           --18
    --compass Statemachine adj pS                                             --19
    wait until clk = '1'; pS <= nS;                                           --20
  end process;                                                                --21
end Moore;                                                                     --22

library IEEE; use IEEE.STD_LOGIC_1164 all;                                --1
entity SM2 is                                                                --2
  port (aIn, clk : in Std_logic; yOut: out Std_logic);                       --3
end SM2;                                                                       --4

architecture Mealy of SM2 is                                               --1
  type state is (s1, s2, s3, s4);                                           --2
  signal pS, nS : state;                                                     --3
  begin                                                                        --4
  process(aIn, pS) begin                                                    --5
  case pS is                                                                --6
  when s1 => if (aIn = '1')                                                  --7
    then yOut <= '0'; nS <= s4;                                               --8
    else yOut <= '1'; nS <= s3;                                               --9
    end if;                                                                    --10
  when s2 => yOut <= '1'; nS <= s3;                                           --11
  when s3 => yOut <= '1'; nS <= s1;                                           --12

```

```

when s4 => if (aIn = '1')                                --13
    then yOut <= '1'; nS <= s2;                          --14
    else yOut <= '0'; nS <= s1;                          --15
    end if;                                              --16
end case;                                              --17
end process;                                          --18
process begin                                          --19
wait until clk = '1' ;                                --20
--Compass Statemachine oneHot pS                      --21
pS <= nS;                                           --22
end process;                                          --23
end Mealy;                                           --24

```

12.8 Memory Synthesis

Key terms and concepts: approaches to memory synthesis: (1) Random logic using flip-flops or latches (2) Register files in datapaths (3) RAM standard components (4) RAM compilers

12.8.1 Memory Synthesis in Verilog

Key terms and concepts: Verilog memory array • an array of latches or flip-flops

```

reg [31:0] MyMemory [3:0]; // a 4 x 32-bit register

module RAM_1(A, CEB, WEB, OEB, INN, OUTT);              //1
    input [6:0] A; input CEB,WEB,OEB; input [4:0]INN;  //2
    output [4:0] OUTT;                                 //3
    reg [4:0] OUTT; reg [4:0] int_bus; reg [4:0] memory [127:0]; //4
always@(negedge CEB) begin                            //5
    if (CEB == 0) begin                               //6
        if (WEB == 1) int_bus = memory[A];           //7
        else if (WEB == 0) begin memory[A] = INN; int_bus = INN;end //8
        else int_bus = 5'bxxxxx;                    //9
    end                                               //10
end                                                  //11
always@(OEB or int_bus) begin                        //12
    case (OEB) 0 : OUTT = int_bus;                   //13
        default : OUTT = 5'bzzzzz; endcase          //14

```

```

end //15
endmodule //16

memory[i + 1] = memory[i]; // needs two clock cycles
pointer = memory[memory[i]]; // needs two clock cycles
pc = memory[addr1]; memory[addr2] = pc + 1; // not on the same cycle

```

12.8.2 Memory Synthesis in VHDL

Key terms and concepts: VHDL multidimensional arrays • array of latches • standard-cell RAM

```

type memStor is array(3 downto 0) of integer; -- This is OK.

subtype MemReg is STD_LOGIC_VECTOR(15 downto 0); -- So is this.
type memStor is array(3 downto 0) of MemReg;
-- other code...
signal Mem1 : memStor;

library IEEE; --1
use IEEE.STD_LOGIC_1164 all; --2
package RAM_package is --3
constant numOut : INTEGER := 8; --4
constant wordDepth: INTEGER := 8; --5
constant numAddr : INTEGER := 3; --6
subtype MEMV is STD_LOGIC_VECTOR(numOut-1 downto 0); --7
type MEM is array (wordDepth-1 downto 0) of MEMV; --8
end RAM_package; --9

library IEEE; --10
use IEEE.STD_LOGIC_1164 all; use IEEE.NUMERIC_STD all; --11
use work.RAM_package all; --12
entity RAM_1 is --13
  port (signal A : in STD_LOGIC_VECTOR(numAddr-1 downto 0); --14
  signal CEB, WEB, OEB : in STD_LOGIC; --15
  signal INN : in MEMV; --16
  signal OUTT : out MEMV); --17
end RAM_1; --18

architecture Synthesis_1 of RAM_1 is --19
  signal i_bus : MEMV; -- RAM internal data latch --20
  signal mem : MEM; -- RAM data --21
  begin --22
  process begin --23
    wait until CEB = '0'; --24

```

```

    if WEB = '1' then i_bus <= mem(TO_INTEGER(UNSIGNED(A))); --25
    elsif WEB = '0' then --26
        mem(TO_INTEGER(UNSIGNED(A))) <= INN; --27
        i_bus <= INN; --28
    else i_bus <= (others => 'X'); --29
    end if; --30
end process; --31

process(OEB, int_bus) begin -- control output drivers: --32
    case (OEB) is --33
        when '0' => OUTT <= i_bus; --34
        when '1' => OUTT <= (others => 'Z'); --35
        when others => OUTT <= (others => 'X'); --36
    end case; --37
end process; --38
end Synthesis_1; --39

```

12.9 The Multiplier

Key terms and concepts: warnings and errors during elaboration

```
Sum <= X xor Y xor Cin after TS;
```

Warning: AFTER clause in a waveform element is not supported

```
port (A, B : in BIT_VECTOR (7 downto 0); Sel : in BIT := '0'; Y : out
BIT_VECTOR (7 downto 0));
```

Warning: Default values on interface signals are not supported

```
port (X:BIT_VECTOR; F:out BIT );
```

Error: An index range must be specified for this data type

```
begin assert (D'LENGTH <= Q'LENGTH)
    report "D wider than output Q" severity Failure;
```

Warning: Assertion statements are ignored

Error: Statements in entity declarations are not supported

```
if CLR = '1' then St := (others => '0'); Q <= St after TCQ;
```

Error: Illegal use of aggregate with the choice "others": the derived subtype of an array aggregate that has a choice "others" must be a constrained array subtype

```
signal SRA, SRB, ADDout, MUXout, REGout: BIT_VECTOR(7 downto 0);
```

Warning: Name is reserved word in VHDL-93: sra

```
signal Zero, Init, Shift, Add, Low: BIT := '0'; signal High: BIT := '1';
```

Warning: Initial values on signals are only for simulation and setting the value of undriven signals in synthesis. A synthesized circuit can not be guaranteed to be in any known state when the power is turned on.

12.9.1 Messages During Synthesis

Key terms and concepts: error and warning messages during synthesis

These unused instances are being removed: in full_adder_p_dup8: u5, u2, u3, u4

These unused instances are being removed: in dffclr_p_dup1: u2

```
architecture Behave of DFFClr is --1
  signal Qi : BIT; --2
  begin QB <= not Qi; Q <= Qi; --3
  process (CLR, CLK) begin --4
    if CLR = '1' then Qi <= '0' after TRQ; --5
    elsif CLK'EVENT and CLK = '1' then Qi <= D after TCQ; --6
    end if; --7
  end process; --8
end; --9
```

```
A1: Adder8 port map(A=>SRB, B=>REGout, Cin=>Low, Cout=>OFL, Sum=>ADDout);
```

```
Cout <= (X and Y) or (X and Cin) or (Y and Cin) after TC;
```

12.10 The Engine Controller

Key terms and concepts: warnings and errors during optimization • unassigned or uninitialized variables

Warning: Made latches to store values on: net d(4), d(5), d(6), d(7), d(8), d(9), d(10), d(11), in module fifo_control

```

case sel is
  when "01" => D <= D_1 after TPD; r1 <= '1' after TPD;
  when "10" => D <= D_2 after TPD; r2 <= '1' after TPD;
  when "00" => D(3) <= f1 after TPD; D(2) <= f2 after TPD;
                D(1) <= e1 after TPD; D(0) <= e2 after TPD; -- Bad!
  when others => D <= "ZZZZZZZZZZZZ" after TPD;
end case;

  when "00" => D(3) <= f1 after TPD; D(2) <= f2 after TPD; -- Write
                D(1) <= e1 after TPD; D(0) <= e2 after TPD; -- to
                D(11 downto 4) <= "ZZZZZZZZ" after TPD; -- all bits.

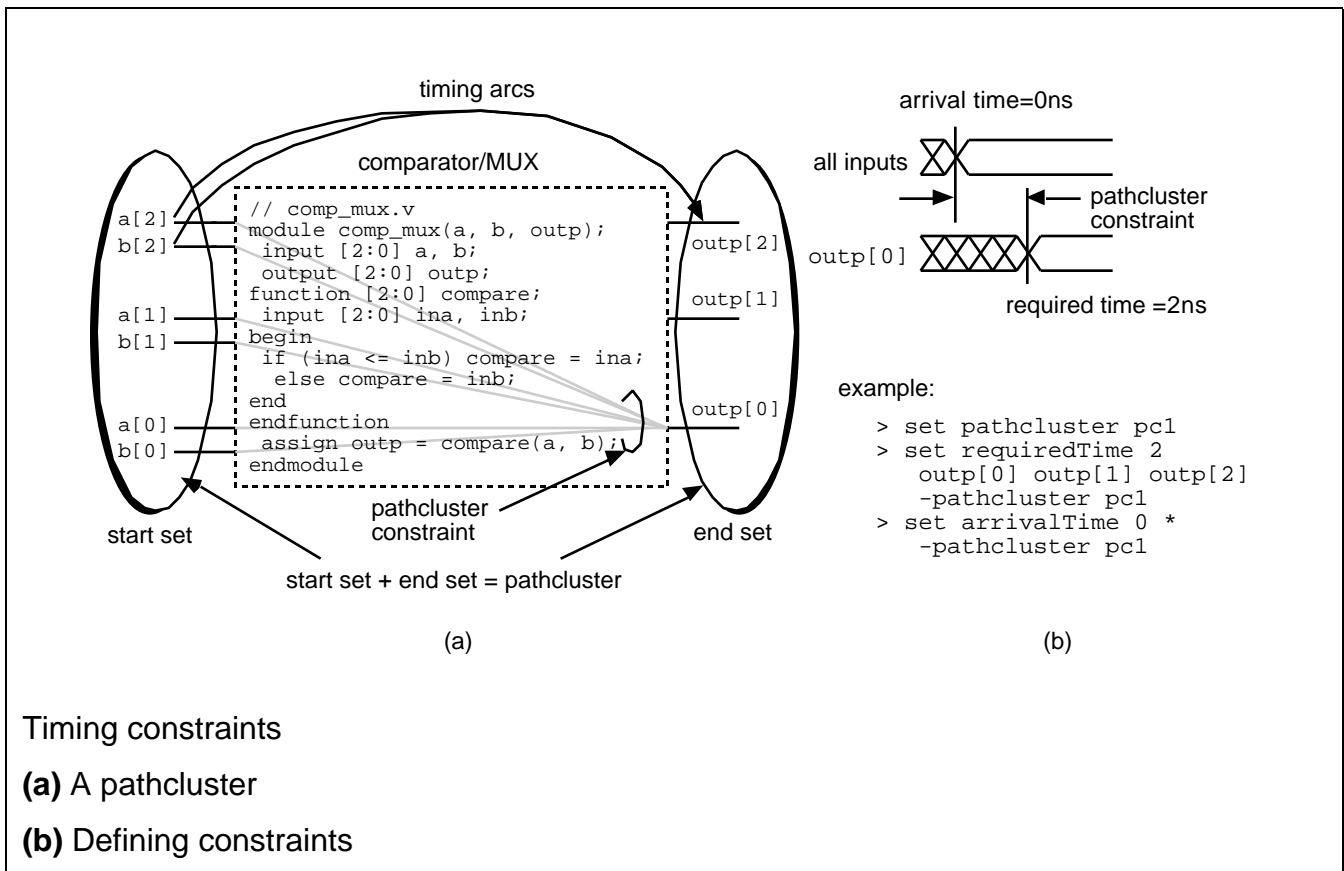
```

12.11 Performance-Driven Synthesis

Key terms and concepts: use of directives and pseudocomments • **timing arcs** (or timing paths)
 • a **pathcluster** (a group of circuit nodes) • **required time** for a signal to reach the output nodes (the **end set**) • **arrival time** of the signals at all the inputs • constrained delay • timing constraint
 • **slack** • the timing constraint is **met** or **violated**

12.12 Optimization of the Viterbi Decoder

Key terms and concepts: set the **environment** using worst-case conditions • die temperature of 25°C (fastest logic) to 120°C (slowest logic) • power supply voltage of $V_{DD}=5.5V$ (fastest logic) to $V_{DD}=4.5V$ (slowest logic) • worst process (slowest logic) to best process (fastest logic)



12.13 Summary

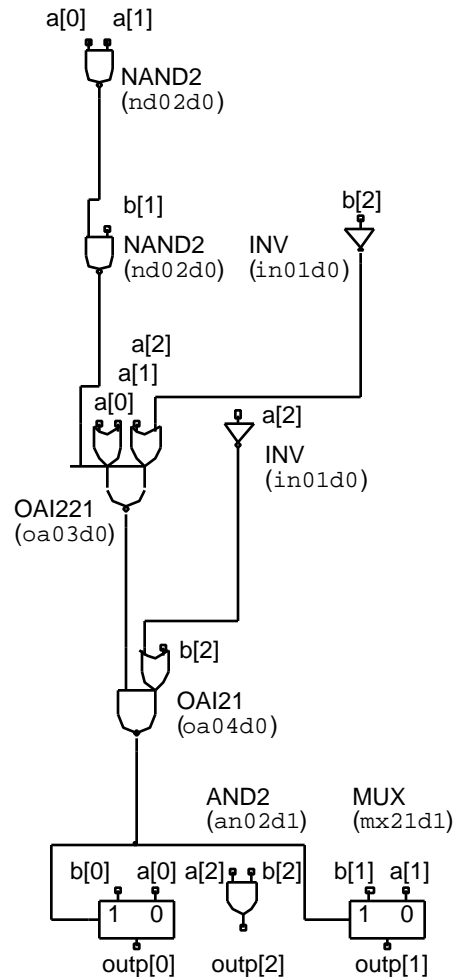
Key terms and concepts: A logic synthesizer may contain over 500,000 lines of code • danger of the “garbage in, garbage out” syndrome • “What do I expect to see at the output?” • “Does the output make sense?” • the worst thing you can do is write and simulate a huge amount of code, read it into the synthesis tool, and try and optimize it all at once with the default settings • interconnect delay is increasingly dominant • it is important to begin physical design as early as possible • ideally floorplanning and logic synthesis should be completed at the same time

```

`timescale 1ns / 10ps
module comp_mux_o (a, b, outp);
input [2:0] a; input [2:0] b;
output [2:0] outp;
supply1 VDD; supply0 VSS;

mx21d1 B1_i1 (.I0(a[0]), .I1(b[0]),
.S(B1_i6_ZN), .Z(outp[0]));
oa03d1 B1_i2 (.A1(B1_i9_ZN), .A2(a[2]),
.B1(a[0]), .B2(a[1]), .C(B1_i4_ZN),
.ZN(B1_i2_ZN));
nd02d0 B1_i3 (.A1(a[1]), .A2(a[0]),
.ZN(B1_i3_ZN));
nd02d0 B1_i4 (.A1(b[1]), .A2(B1_i3_ZN),
.ZN(B1_i4_ZN));
mx21d1 B1_i5 (.I0(a[1]), .I1(b[1]),
.S(B1_i6_ZN), .Z(outp[1]));
oa04d1 B1_i6 (.A1(b[2]), .A2(B1_i7_ZN),
.B(B1_i2_ZN), .ZN(B1_i6_ZN));
in01d0 B1_i7 (.I(a[2]), .ZN(B1_i7_ZN));
an02d1 B1_i8 (.A1(b[2]), .A2(a[2]),
.Z(outp[2]));
in01d0 B1_i9 (.I(b[2]), .ZN(B1_i9_ZN));
endmodule

```



The comparator/MUX example after logic optimization with timing constraints

The structural netlist, `comp_mux_o2.v`, and its derived schematic

SIMULATION

13

Key terms and concepts: Engineers used to prototype systems to check designs • Breadboarding is feasible for systems constructed from a few TTL parts • It is impractical for an ASIC • Instead engineers turn to **simulation**

13.1 Types of Simulation

Key terms and concepts: **simulation modes** (high-level to low-level simulation—high-level is more abstract, low-level more detailed): Behavioral simulation • Functional simulation • Static timing analysis • Gate-level simulation • Switch-level simulation • Transistor-level or circuit-level simulation

13.2 The Comparator/MUX Example

Key terms and concepts: using **input vectors** to test or **exercise** a behavioral model • simulation can only prove a design does not work; it cannot prove that hardware will work

```
// comp_mux.v //1
module comp_mux(a, b, outp); input [2:0] a, b; output [2:0] outp; //2
function [2:0] compare; input [2:0] ina, inb; //3
begin if (ina <= inb) compare = ina; else compare = inb; end //4
endfunction //5
assign outp = compare(a, b); //6
endmodule //7

// testbench.v //1
module comp_mux_testbench; //2
integer i, j; //3
reg [2:0] x, y, smaller; wire [2:0] z; //4
always @(x) $display("t x y actual calculated"); //5
initial $monitor("%4g", $time, ,x, ,y, ,z, , , , , ,smaller); //6
initial $dumpvars; initial #1000 $finish; //7
initial //8
```

```

begin                                                    //9
  for (i = 0; i <= 7; i = i + 1)                        //10
  begin                                                //11
    for (j = 0; j <= 7; j = j + 1)                    //12
    begin                                              //13
      x = i; y = j; smaller = (x <= y) ? x : y;        //14
      #1 if (z != smaller) $display("error");          //15
    end                                                //16
  end                                                  //17
end                                                    //18
comp_mux v_1 (x, y, z);                                //19
endmodule                                              //20

```

13.2.1 Structural Simulation

Key terms and concepts: logic synthesis produces a structural model from a behavioral model • reference model • derived model • **vector-based simulation** (or **dynamic simulation**)

```

`timescale 1ns / 10ps // comp_mux_o2.v                //1
module comp_mux_o (a, b, outp);                        //2
input  [2:0] a; input  [2:0] b;                       //3
output [2:0] outp;                                    //4
supply1 VDD; supply0 VSS;                            //5
mx21d1 b1_i1 (.i0(a[0]), .i1(b[0]), .s(b1_i6_zn), .z(outp[0])); //6
oa03d1 b1_i2 (.a1(b1_i9_zn), .a2(a[2]), .b1(a[0]), .b2(a[1]), //7
  .c(b1_i4_zn), .zn(b1_i2_zn));                          //8
nd02d0 b1_i3 (.a1(a[1]), .a2(a[0]), .zn(b1_i3_zn));      //9
nd02d0 b1_i4 (.a1(b[1]), .a2(b1_i3_zn), .zn(b1_i4_zn)); //10
mx21d1 b1_i5 (.i0(a[1]), .i1(b[1]), .s(b1_i6_zn), .z(outp[1])); //11
oa04d1 b1_i6 (.a1(b[2]), .a2(b1_i7_zn), .b(b1_i2_zn), //12
  .zn(b1_i6_zn));                                        //13
in01d0 b1_i7 (.i(a[2]), .zn(b1_i7_zn));                  //14
an02d1 b1_i8 (.a1(b[2]), .a2(a[2]), .z(outp[2]));      //15
in01d0 b1_i9 (.i(b[2]), .zn(b1_i9_zn));                //16
endmodule                                             //17

`timescale 1 ns / 10 ps                                //1
module mx21d1 (z, i0, i1, s); input i0, i1, s; output z; //2
  not G3(N3, s);                                        //3
  and G4(N4, i0, N3), G5(N5, s, i1), G6(N6, i0, i1);   //4
  or G7(z, N4, N5, N6);                                //5
specify                                              //6
  (i0*>z) = (0.279:0.504:0.900, 0.276:0.498:0.890); //7

```

```

    (i1*>z) = (0.248:0.448:0.800, 0.264:0.476:0.850); //8
    (s*>z) = (0.285:0.515:0.920, 0.298:0.538:0.960); //9
endspecify //10
endmodule //11

`timescale 1 ps / 1 ps // comp_mux_testbench2.v //1
module comp_mux_testbench2; //2
integer i, j; integer error; //3
reg [2:0] x, y, smaller; wire [2:0] z, ref; //4
always @(x) $display("t          x y derived reference"); //5
// initial $monitor("%8.2f", $time/1e3, ,x, ,y, ,z, , , , , , , ,ref); //6
initial $dumpvars; //7
initial begin //8
    error = 0; #1e6 $display("%4g", error, " errors"); //9
    $finish; //10
end //11
initial begin //12
    for (i = 0; i <= 7; i = i + 1) begin //13
        for (j = 0; j <= 7; j = j + 1) begin //14
            x = i; y = j; #10e3; //15
            $display("%8.2f", $time/1e3, ,x, ,y, ,z, , , , , , , ,ref); //16
            if (z != ref) //17
                begin $display("error"); error = error + 1; end //18
            end //19
        end //20
    end //21
comp_mux_o v_1 (x, y, z); // comp_mux_o2.v //22
reference v_2 (x, y, ref); //23
endmodule //24

// reference.v //1
module reference(a, b, outp); //2
input [2:0] a, b; output [2:0] outp; //3
    assign outp = (a <= b) ? a : b; // different from comp_mux //4
endmodule //5

```

13.2.2 Static Timing Analysis

Key terms and concepts: "What is the longest delay in my circuit?" • timing analysis finds the critical path and its delay • timing analysis does not find the input vectors that activate the critical path • **Boolean relations** • **false paths** • a timing-analyzer is more logic calculator than logic simulator

13.2.3 Gate-Level Simulation

Key terms and concepts: differences between functional simulation, timing analysis, and gate-level simulation

```
# The calibration was done at Vdd=4.65V, Vss=0.1V, T=70 degrees C
Time = 0:0 [0 ns]
        a = 'D6 [0] (input)(display)
        b = 'D7 [0] (input)(display)
    outp = 'Buuu ('Du) [0] (display)
    outp --> 'Bluu ('Du) [.47]
    outp --> 'B1lu ('Du) [.97]
    outp --> 'D6 [4.08]
        a --> 'D7 [10]
        b --> 'D6 [10]
    outp --> 'D7 [10.97]
    outp --> 'D6 [14.15]
Time = 0:0 +20ns [20 ns]
```

13.2.4 Net Capacitance

Key terms and concepts: **net capacitance** (interconnect capacitance or wire capacitance) • **wire-load model, wire-delay model, or interconnect model**

```
@nodes
a R10 W1; a[2] a[1] a[0]
b R10 W1; b[2] b[1] b[0]
outp R10 W1; outp[2] outp[1] outp[0]
@data
        .00          a -> 'D6
        .00          b -> 'D7
        .00          outp -> 'Du
        .53          outp -> 'Du
        .93          outp -> 'Du
        4.42         outp -> 'D6
        10.00         a -> 'D7
        10.00         b -> 'D6
        11.03         outp -> 'D7
        14.43         outp -> 'D6
### END OF SIMULATION TIME = 20 ns
@end
```

13.3 Logic Systems

Key terms and concepts: **Digital simulation** • **logic values** (or **logic states**) from a **logic system** • A **two-value logic system** (or two-state logic system) has logic value '0' (**logic level** 'zero') and a logic value '1' (logic level 'one') • logic value 'x' (unknown logic level) or **unknown** • an unknown can **propagate** through a circuit • to model a three-state bus, we need a **high-impedance state** (logic level of 'zero' or 'one') but it is not being driven • A **four-value logic system**

A four-value logic system

Logic state	Logic level	Logic value
0	zero	zero
1	one	one
X	zero or one	unknown
Z	zero, one, or neither	high impedance

13.3.1 Signal Resolution

Key terms and concepts: **signal-resolution function** • commutative and associative

A resolution function $R\{A, B\}$ that predicts the result of two drivers simultaneously attempting to drive signals with values A and B onto a bus

$R\{A, B\}$	B=0	B=1	B=X	B=Z
A=0	0	X	X	0
A=1	X	1	X	1
A=X	X	X	X	X
A=Z	0	1	X	Z

13.3.2 Logic Strength

Key terms and concepts: n-channel transistors produce a logic level 'zero' (with a forcing strength) • p-channel transistors force a logic level 'one' • An n-channel transistor provides a

weak logic level 'one', a **resistive 'one'**, with **resistive strength • high impedance • Verilog logic system • VHDL signal resolution** using **VHDL signal-resolution functions**

A 12-state logic system

Logic strength	Logic level		
	zero	unknown	one
strong	S0	SX	S1
weak	W0	WX	W1
high impedance	Z0	ZX	Z1
unknown	U0	UX	U1

Verilog logic strengths

Logic strength	Strength number	Models	Abbreviation	
supply drive	7	power supply	supply	Su
strong drive	6	default gate and assign output strength	strong	St
pull drive	5	gate and assign output strength	pull	Pu
large capacitor	4	size of trireg net capacitor	large	La
weak drive	3	gate and assign output strength	weak	We
medium capacitor	2	size of trireg net capacitor	medium	Me
small capacitor	1	size of trireg net capacitor	small	Sm
high impedance	0	not applicable	highz	Hi

The nine-value logic system, IEEE Std 1164-1993.

Logic state	Logic value	Logic state	Logic value
'0'	strong low	'X'	strong unknown
'1'	strong high	'W'	weak unknown
'L'	weak low	'Z'	high impedance
'H'	weak high	'-'	don't care
		'U'	uninitialized

```
function "and"(l,r : std_ulogic_vector) return std_ulogic_vector
is
  alias lv : std_ulogic_vector (1 to l'LENGTH) is l;
  alias rv : std_ulogic_vector (1 to r'LENGTH) is r;
variable result : std_ulogic_vector (1 to l'LENGTH);
```

```

constant and_table : stdlogic_table := (                                --5
-----                                                                    --6
--|  U    X    0    1    Z    W    L    H    -    |    |    --7
-----                                                                    --8
( 'U', 'U', '0', 'U', 'U', 'U', '0', 'U', 'U' ), -- | U |    --9
( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | X |    --10
( '0', '0', '0', '0', '0', '0', '0', 'U', '0' ), -- | 0 |    --11
( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- | 1 |    --12
( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | Z |    --13
( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | W |    --14
( '0', '0', '0', '0', '0', '0', '0', '0', '0' ), -- | L |    --15
( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- | H |    --16
( 'U', 'X', '0', 'X', 'X', 'X', '0', 'X', 'X' ), -- | - | ); --17
begin                                                                    --18
  if (l'LENGTH /= r'LENGTH) then assert false report                --19
"arguments of overloaded 'and' operator are not of the same          --20
length"                                                                --21
  severity failure;                                                    --22
else                                                                    --23
  for i in result'RANGE loop                                          --24
    result(i) := and_table ( lv(i), rv(i) );                            --25
  end loop;                                                            --26
end if;                                                                --27
  return result;                                                       --28
end "and";                                                            --29

```

13.4 How Logic Simulation Works

Key terms and concepts: event-driven simulator • event • event queue or event list • evaluation • **time step** • interpreted-code simulator • compiled-code simulator • native-code simulator • evaluation list • simulation cycle, or an event–evaluation cycle • time wheel

```

model nd01d1 (a, b, zn)
function (a, b) !(a & b); function end
model end

```

```
nand nd01d1(a2, b3, r7)
```

```

struct Event {
    event_ptr fwd_link, back_link; /* event list */
    event_ptr node_link; /* list of node events */
    node_ptr event_node; /* node for the event */
    node_ptr cause; /* node causing event */
    port_ptr port; /* port which caused this event */
    long event_time; /* event time, in units of delta */
    char new_value; /* new value: '1' '0' etc. */
};

```

13.4.1 VHDL Simulation Cycle

Key terms and concepts: **simulation cycle** • elaboration • a **delta cycle** takes **delta time** • **time step** • postponed processes

A VHDL simulation cycle consists of the following steps:

1. The current time, t_c is set equal to t_n .
2. Each active signal in the model is updated and events may occur as a result.
3. For each process P, if P is currently sensitive to a signal S, and an event has occurred on signal S in this simulation cycle, then process P resumes.
4. Each resumed process is executed until it suspends.
5. The time of the next simulation cycle, t_n , is set to the earliest of:
 - a. the next time at which a driver becomes active or
 - b. the next time at which a process resumes
6. If $t_n = t_c$, then the next simulation cycle is a delta cycle.
7. Simulation is complete when we run out of time ($t_n = \text{TIME 'HIGH}$) and there are no active drivers or process resumptions at t_n

13.4.2 Delay

Key terms and concepts: **delay mechanism** • **transport delay** is characteristic of wires and transmission lines • **Inertial delay** models the behavior of logic cells • a logic cell will not transmit a pulse that is shorter than the switching time of the circuit, the default **pulse-rejection limit**

```

Op <= Ip after 10 ns;                                --1
Op <= inertial Ip after 10 ns;                       --2
Op <= reject 10 ns inertial Ip after 10 ns;        --3

-- Assignments using transport delay:                --1
Op <= transport Ip after 10 ns;                     --2
Op <= transport Ip after 10 ns, not Ip after 20 ns; --3

```

```
-- Their equivalent assignments: --4
Op <= reject 0 ns inertial Ip after 10 ns; --5
Op <= reject 0 ns inertial Ip after 10 ns, not Ip after 10 ns; --6
```

13.5 Cell Models

Key terms and concepts: delay model • power model • timing model • primitive model

There are several different kinds of logic cell models:

- Primitive models, produced by the ASIC library company and describe the function and properties of logic cells using primitive functions.
- Verilog and VHDL models produced by an ASIC library company from the primitive models.
- Proprietary models produced by library companies that describe small logic cells or functions such as microprocessors.

13.5.1 Primitive Models

Key terms and concepts: **primitive model** • a designer does not normally see a primitive model; it may only be used by an ASIC library company to generate other models

```
Function
(timingModel = oneOf("ism","pr"); powerModel = oneOf("pin"); )
Rec
Logic = Function (A1; A2; )Rec ZN = not (A1 AND A2); End; End;
miscInfo = Rec Title = "2-Input NAND, 1X Drive"; freq_fact = 0.5;
tml = "nd02d1 nand 2 * zn a1 a2";
MaxParallel = 1; Transistors = 4; power = 0.179018;
Width = 4.2; Height = 12.6; productName = "stdcell35"; libraryName =
"cb35sc"; End;
Pin = Rec
A1 = Rec input; cap = 0.010; doc = "Data Input"; End;
A2 = Rec input; cap = 0.010; doc = "Data Input"; End;
ZN = Rec output; cap = 0.009; doc = "Data Output"; End; End;
Symbol = Select
timingModel
On pr Do Rec
tA1D_fr = |( Rec prop = 0.078; ramp = 2.749; End);
tA1D_rf = |( Rec prop = 0.047; ramp = 2.506; End);
tA2D_fr = |( Rec prop = 0.063; ramp = 2.750; End);
tA2D_rf = |( Rec prop = 0.052; ramp = 2.507; End); End
On ism Do Rec
```

```

tA1D_fr = |( Rec A0 = 0.0015; dA = 0.0789; D0 = -0.2828;
dD = 4.6642; B = 0.6879; Z = 0.5630; End );
tA1D_rf = |( Rec A0 = 0.0185; dA = 0.0477; D0 = -0.1380;
dD = 4.0678; B = 0.5329; Z = 0.3785; End );
tA2D_fr = |( Rec A0 = 0.0079; dA = 0.0462; D0 = -0.2819;
dD = 4.6646; B = 0.6856; Z = 0.5282; End );
tA2D_rf = |( Rec A0 = 0.0060; dA = 0.0464; D0 = -0.1408;
dD = 4.0731; B = 0.6152; Z = 0.4064; End ); End; End;
Delay = |( Rec from = pin.A1; to = pin.ZN;
edges = Rec fr = Symbol.tA1D_fr; rf = Symbol.tA1D_rf; End; End, Rec
from = pin.A2; to = pin.ZN; edges = Rec fr = Symbol.tA2D_fr; rf =
Symbol.tA2D_rf; End; End );
MaxRampTime = |( Rec check = pin.A1; riseTime = 3.000; fallTime =
3.000; End, Rec check = pin.A2; riseTime = 3.000; fallTime =
3.000; End, Rec check = pin.ZN; riseTime = 3.000; fallTime =
3.000; End );
DynamicPower = |( Rec rise = { ZN }; val = 0.003; End); End; End

```

13.5.2 Synopsys Models

Key terms and concepts: **vendor models** • each logic cell is part of a file that also contains wire-load models and other characterization information for the cell library • not all of the information from a primitive model is present in a vendor model

```

cell (nd02d1) {
/* title : 2-Input NAND, 1X Drive */
/* pmd checksum : 'HBA7EB26C */
area : 1;
  pin(a1) { direction : input; capacitance : 0.088;
    fanout_load : 0.088; }
  pin(a2) { direction : input; capacitance : 0.087;
    fanout_load : 0.087; }
  pin(zn) { direction : output; max_fanout : 1.786;
    max_transition : 3; function : "(a1 a2)";
  timing() {
    timing_sense : "negative_unate"
    intrinsic_rise : 0.24 intrinsic_fall : 0.17
    rise_resistance : 1.68 fall_resistance : 1.13
    related_pin : "a1" }
  timing() { timing_sense : "negative_unate"
    intrinsic_rise : 0.32 intrinsic_fall : 0.18
    rise_resistance : 1.68 fall_resistance : 1.13

```

```

    related_pin : "a2"
} } } /* end of cell */

```

13.5.3 Verilog Models

Key terms and concepts: Verilog timing models • SDF file contains back-annotation timing delays • delays are calculated by a **delay calculator** • `$sdf_annotate` performs back-annotation • **golden simulator**

```

`celldefine //1
`delay_mode_path //2
`suppress_faults //3
`enable_portfaults //4
`timescale 1 ns / 1 ps //5
module in01d1 (zn, i); input i; output zn; not G2(zn, i); //6
specify specparam //7
InCap$i = 0.060, OutCap$zn = 0.038, MaxLoad$zn = 1.538, //8
R_Ramp$i$zn = 0.542:0.980:1.750, F_Ramp$i$zn = 0.605:1.092:1.950; //9
specparam cell_count = 1.000000; specparam Transistors = 4 ; //10
specparam Power = 1.400000; specparam MaxLoadedRamp = 3 ; //11
    (i => zn) = (0.031:0.056:0.100, 0.028:0.050:0.090); //12
endspecify //13
endmodule //14
`nosuppress_faults //15
`disable_portfaults //16
`endcelldefine //17

`timescale 1 ns / 1 ps //1
module SDF_b; reg A; in01d1 i1 (B, A); //2
initial begin A = 0; #5; A = 1; #5; A = 0; end //3
initial $monitor("T=%6g", $realtime, " A=", A, " B=", B); //4
endmodule //5

```

```

T=      0 A=0 B=x
T= 0.056 A=0 B=1
T=      5 A=1 B=1
T=  5.05 A=1 B=0
T=     10 A=0 B=0
T=10.056 A=0 B=1

```

```

(DELAYFILE
  (SDFVERSION "3.0") (DESIGN "SDF.v") (DATE "Aug-13-96")
  (VENDOR "MJSS") (PROGRAM "MJSS") (VERSION "v0")
  (DIVIDER .) (TIMESCALE 1 ns)
  (CELL (CELLTYPE "in01d1")
    (INSTANCE SDF_b.i1)
    (DELAY (ABSOLUTE
      (IOPATH i zn (1.151:1.151:1.151) (1.363:1.363:1.363))
    ))
  )
)

`timescale 1 ns / 1 ps //1
module SDF_b; reg A; in01d1 i1 (B, A); //2
initial begin //3
  $sdf_annotate ( "SDF_b.sdf", SDF_b, , "sdf_b.log", "minimum", , ); //4
  A = 0; #5; A = 1; #5; A = 0; end //5
initial $monitor("T=%6g", $realtime, " A=", A, " B=", B); //6
endmodule //7

```

Here is the output (from MTI V-System/Plus) including back-annotated timing:

```

T=      0 A=0 B=x
T=  1.151 A=0 B=1
T=      5 A=1 B=1
T=  6.363 A=1 B=0
T=     10 A=0 B=0
T=11.151 A=0 B=1

```

13.5.4 VHDL Models

Key terms and concepts: VHDL alone does not offer a standard way to perform back-annotation.

- VITAL

```

library IEEE; use IEEE.STD_LOGIC_1164 all;
library COMPASS_LIB; use COMPASS_LIB.COMPASS_ETC all;
entity bknot is
  generic (derating : REAL := 1.0; Z1_cap : REAL := 0.000;
    INSTANCE_NAME : STRING := "bknot");
  port (Z2 : in Std_Logic; Z1 : out STD_LOGIC);
end bknot;

```

```

architecture bknot of bknot is
constant tplh_Z2_Z1 : TIME := (1.00 ns + (0.01 ns * Z1_Cap)) *
derating;
constant tphl_Z2_Z1 : TIME := (1.00 ns + (0.01 ns * Z1_Cap)) *
derating;
begin
  process(Z2)
    variable int_Z1 : Std_Logic := 'U';
    variable tplh_Z1, tphl_Z1, Z1_delay : time := 0 ns;
    variable CHANGED : BOOLEAN;
    begin
      int_Z1 := not (Z2);
      if Z2'EVENT then
        tplh_Z1 := tplh_Z2_Z1; tphl_Z1 := tphl_Z2_Z1;
      end if;
      Z1_delay := F_Delay(int_Z1, tplh_Z1, tphl_Z1);
      Z1 <= int_Z1 after Z1_delay;
    end process;
end bknot;
configuration bknot_CON of bknot is for bknot end for;
end bknot_CON;

```

13.5.5 VITAL Models

Key terms and concepts: VITAL • VHDL Initiative Toward ASIC Libraries, IEEE Std 1076.4 [1995] • **sign-off** quality ASIC libraries using an approved cell library and a golden simulator

```

library IEEE; use IEEE.STD_LOGIC_1164 all; --1
use IEEE.VITAL_timing all; use IEEE.VITAL_primitives all; --2
entity IN01D1 is --3
  generic ( --4
    tpd_I : VitalDelayType01 := (0 ns, 0 ns); --5
    tpd_I_ZN : VitalDelayType01 := (0 ns, 0 ns) ); --6
  port ( --7
    I : in STD_LOGIC := 'U'; --8
    ZN : out STD_LOGIC := 'U' ); --9
attribute VITAL_LEVEL0 of IN01D1 : entity is TRUE; --10
end IN01D1; --11
architecture IN01D1 of IN01D1 is --12
attribute VITAL_LEVEL1 of IN01D1 : architecture is TRUE; --13
signal I_ipd : STD_LOGIC := 'X'; --14
begin --15
WIREDelay:block --16
  begin VitalWireDelay(I_ipd, I, tpd_I);end block; --17

```



```

end process;                                --19
end SDF_testbench;                          --20

```

```

(DELAYFILE
  (SDFVERSION "3.0") (DESIGN "SDF.vhd") (DATE "Aug-13-96")
  (VENDOR "MJSS") (PROGRAM "MJSS") (VERSION "v0")
  (DIVIDER .) (TIMESCALE 1 ns)
  (CELL (CELLTYPE "in01d1")
    (INSTANCE i1)
    (DELAY (ABSOLUTE
      (IOPATH i zn (1.151:1.151:1.151) (1.363:1.363:1.363))
      (PORT i (0.021:0.021:0.021) (0.025:0.025:0.025))
    ))
  )
)

```

```
<msmith/MTI/vital> vsim -c -sdfmax /sdf_b=SDF_b.sdf sdf_testbench
```

```

...
#      0 ps A=0 B=0
#      0 ps A=0 B=0
#    1176 ps A=0 B=1
#     5000 ps A=1 B=1
#     6384 ps A=1 B=0
#    10000 ps A=0 B=0
#    11176 ps A=0 B=1

```

13.5.6 SDF in Simulation

Key terms and concepts: SDF is also used to describe forward-annotation of timing constraints from logic synthesis

```

(DELAYFILE
  (SDFVERSION "1.0")
  (DESIGN "halfgate_ASIC_u")
  (DATE "Aug-13-96")
  (VENDOR "Compass")
  (PROGRAM "HDL Asst")
  (VERSION "v9r1.2")
  (DIVIDER .)
  (TIMESCALE 1 ns)

```

```

(CELL (CELLTYPE "in01d0")
  (INSTANCE v_1.B1_i1)
  (DELAY (ABSOLUTE
    (IOPATH I ZN (1.151:1.151:1.151) (1.363:1.363:1.363))
  ))
)
(CELL (CELLTYPE "pc5o06")
  (INSTANCE u1_2)
  (DELAY (ABSOLUTE
    (IOPATH I PAD (1.216:1.216:1.216) (1.249:1.249:1.249))
  ))
)
(CELL (CELLTYPE "pc5d01r")
  (INSTANCE u0_2)
  (DELAY (ABSOLUTE
    (IOPATH PAD CIN (.169:.169:.169) (.199:.199:.199))
  ))
)
)
)

```

```

(DELAYFILE
  ...
  (PROCESS "FAST-FAST")
  (TEMPERATURE 0:55:100)
  (TIMESCALE 100ps)
(CELL (CELLTYPE "CHIP")
  (INSTANCE TOP)
  (DELAY (ABSOLUTE
    (INTERCONNECT A.INV8.OUT B.DFF1.Q (:0.6:) (:0.6:))
  )))
)
)

```

```

(INSTANCE B.DFF1)
(DELAY (ABSOLUTE
  (IOPATH (POSEDGE CLK) Q (12:14:15) (11:13:15))))
)

```

```

(DELAYFILE
(DSIGN "MYDESIGN")
(DATE "26 AUG 1996")
  (VENDOR "ASICS_INC")
  (PROGRAM "SDF_GEN")
(VERSION "3.0")
(DIVIDER .)
)

```

```

(VOLTAGE 3.6:3.3:3.0)
(PROCESS "-3.0:0.0:3.0")
(TEMPERATURE 0.0:25.0:115.0)
(TIMESCALE )
(CELL
  (CELLTYPE "AOI221")
  (INSTANCE X0)
  (DELAY (ABSOLUTE
    (IOPATH A1 Y (1.11:1.42:2.47) (1.39:1.78:3.19))
    (IOPATH A2 Y (0.97:1.30:2.34) (1.53:1.94:3.50))
    (IOPATH B1 Y (1.26:1.59:2.72) (1.52:2.01:3.79))
    (IOPATH B2 Y (1.10:1.45:2.56) (1.66:2.18:4.10))
    (IOPATH C1 Y (0.79:1.04:1.91) (1.36:1.62:2.61))
  )))

```

13.6 Delay Models

Key terms and concepts: timing model describes delays outside logic cells • delay model describes delays inside logic cells • **pin-to-pin delay** is a delay between an input pin and an output pin of a logic cell • **pin delay** is a delay lumped to a certain pin of a logic cell (usually an input) • **net delay** or **wire delay** is a delay outside a logic cell • **prop-ramp delay model**

```

specify specparam //1
InCap$i = 0.060, OutCap$zn = 0.038, MaxLoad$zn = 1.538, //2
R_Ramp$i$zn = 0.542:0.980:1.750, F_Ramp$i$zn = 0.605:1.092:1.950; //3
specparam cell_count = 1.000000; specparam Transistors = 4 ; //4
specparam Power = 1.400000; specparam MaxLoadedRamp = 3 ; //5
(i=>zn)=(0.031:0.056:0.100, 0.028:0.050:0.090); //6

```

13.6.1 Using a Library Data Book

Key terms and concepts: **area-optimized library** (small) • **performance-optimized library** (fast)

Input capacitances for an inverter family (pF)

Library	inv1	invh	invs	inv8	inv12
Area	0.034	0.067	0.133	0.265	0.397
Performance	0.145	0.292	0.584	1.169	1.753

Delay information for a 2:1 MUX

From input	To output	Propagation delay			
		Area		Performance	
		Extrinsic/ nspF ⁻¹	Intrinsic / ns	Extrinsic / ns	Intrinsic / ns
D0\	Z\	2.10	1.42	0.5	0.8
D0/	Z/	3.66	1.23	0.68	0.70
D1\	Z\	2.10	1.42	0.50	0.80
D1/	Z/	3.66	1.23	0.68	0.70
SD\	Z\	2.10	1.42	0.50	0.80
SD\	Z/	3.66	1.09	0.70	0.73
SD/	Z\	2.10	2.09	0.5	1.09
SD/	Z/	3.66	1.23	0.68	0.70

Process derating factors

Process	Derating factor
Slow	1.31
Nominal	1.0
Fast	0.75

Temperature and voltage derating factors

Temperature/°C	Supply voltage				
	4.5V	4.75V	5.00V	5.25V	5.50V
-40	0.77	0.73	0.68	0.64	0.61
0	1.00	0.93	0.87	0.82	0.78
25	1.14	1.07	1.00	0.94	0.90
85	1.50	1.40	1.33	1.26	1.20
100	1.60	1.49	1.41	1.34	1.28
125	1.76	1.65	1.56	1.47	1.41

13.6.2 Input-Slope Delay Model

Key terms and concepts: submicron technologies must account for the effects of the rise (and fall) time of the input waveforms to a logic cell • nonlinear delay model

The input-slope model predicts delay in the fast-ramp region, $D_{ISM}(50\%, FR)$, as follows (0.5 trip points):

$$\begin{aligned}
 D_{ISM}(50\%, FR) &= A_0 + D_0 C_L + 0.5 O_R = A_0 + D_0 C_L + d_A/2 + d_D C_L/2 \\
 &= 0.0015 + 0.5 \times 0.0789 + (-0.2828 + 0.5 \times 4.6642) C_L \\
 &= 0.041 + 2.05 C_L
 \end{aligned}$$

13.6.3 Limitations of Logic Simulation

Key terms and concepts: pin-to-pin delay model • timing information for most gate-level simulators is calculated once, before simulation • **state-dependent timing**

Switching characteristics of a two-input NAND gate

Symbol	Parameter	Fanout					K /nspF ⁻¹
		FO = 0 /ns	FO = 1 /ns	FO = 2 /ns	FO = 4 /ns	FO = 8 /ns	
t_{PLH}	Propagation delay, A to X	0.25	0.35	0.45	0.65	1.05	1.25
t_{PHL}	Propagation delay, B to X	0.17	0.24	0.30	0.42	0.68	0.79
t_r	Output rise time, X	1.01	1.28	1.56	2.10	3.19	3.40
t_f	Output fall time, X	0.54	0.69	0.84	1.13	1.71	1.83

Switching characteristics of a half adder

Symbol	Parameter	Fanout					K /nspF ⁻¹
		FO = 0 /ns	FO = 1 /ns	FO = 2 /ns	FO = 4 /ns	FO = 8 /ns	
t _{PLH}	Delay, A to S (B = '0')	0.58	0.68	0.78	0.98	1.38	1.25
t _{PHL}	Delay, A to S (B = '1')	0.93	0.97	1.00	1.08	1.24	0.48
t _{PLH}	Delay, B to S (B = '0')	0.89	0.99	1.09	1.29	1.69	1.25
t _{PHL}	Delay, B to S (B = '1')	1.00	1.04	1.08	1.15	1.31	0.48
t _{PLH}	Delay, A to CO	0.43	0.53	0.63	0.83	1.23	1.25
t _{PHL}	Delay, A to CO	0.59	0.63	0.67	0.75	0.90	0.48
t _r	Output rise time, X	1.01	1.28	1.56	2.10	3.19	3.40
t _f	Output fall time, X	0.54	0.69	0.84	1.13	1.71	1.83

13.7 Static Timing Analysis

Key terms and concepts: static timing analysis • pipelining • critical path

Instance name	in pin-->out pin	tr	total	incr	cell

END_OF_PATH					
outp_2_		R	27.26		
OUT1	: D--->PAD	R	27.26	7.55	OUTBUF
I_1_CM8	: S11--->Y	R	19.71	4.40	CM8
I_2_CM8	: S11--->Y	R	15.31	5.20	CM8
I_3_CM8	: S11--->Y	R	10.11	4.80	CM8
IN1	: PAD--->Y	R	5.32	5.32	INBUF
a_2_		R	0.00	0.00	

BEGIN_OF_PATH

```
// comp_mux_rrr.v //1
module comp_mux_rrr(a, b, clock, outp); //2
input [2:0] a, b; output [2:0] outp; input clock; //3
reg [2:0] a_r, a_rr, b_r, b_rr, outp; reg sel_r; //4
wire sel = ( a_r <= b_r ) ? 0 : 1; //5
always @ (posedge clock) begin a_r <= a; b_r <= b; end //6
always @ (posedge clock) begin a_rr <= a_r; b_rr <= b_r; end //7
always @ (posedge clock) outp <= sel_r ? b_rr : a_rr; //8
always @ (posedge clock) sel_r <= sel; //9
endmodule //10
```

-----INPAD to SETUP longest path-----

Rise delay, Worst case

Instance name	in pin-->out pin	tr	total	incr	cell

```

END_OF_PATH
D.a_r_ff_b2                R      4.52   0.00   DF1
INBUF_24                   : PAD--->Y    R      4.52   4.52   INBUF
a_2_                       R      0.00   0.00
BEGIN_OF_PATH

```

```

-----CLOCK to SETUP longest path-----
Rise delay, Worst case

```

Instance name	in pin-->out pin	tr	total	incr	cell

END_OF_PATH					
D.sel_r_ff		R	9.99	0.00	DF1
I_1_CM8	: S10--->Y	R	9.99	0.00	CM8
I_3_CM8	: S00--->Y	R	9.99	4.40	CM8
a_r_ff_b1	: CLK--->Q	R	5.60	5.60	DF1
BEGIN_OF_PATH					

```

-----CLOCK to OUTPAD longest path-----
Rise delay, Worst case

```

Instance name	in pin-->out pin	tr	total	incr	cell

END_OF_PATH					
outp_2_		R	11.95		
OUTBUF_31	: D--->PAD	R	11.95	7.55	OUTBUF
outp_ff_b2	: CLK--->Q	R	4.40	4.40	DF1
BEGIN_OF_PATH					

A timing analyzer examines the following types of paths:

1. An **entry path** (or input-to-D path) to a pipelined design. The longest **entry delay** (or input-to-setup delay) is 4.52 ns.
2. A **stage path** (register-to-register path or clock-to-D path) in a pipeline stage. The longest **stage delay** (clock-to-D delay) is 9.99 ns.
3. An **exit path** (clock-to-output path) from the pipeline. The longest **exit delay** (clock-to-output delay) is 11.95 ns.

13.7.1 Hold Time

Key terms and concepts: Hold-time problems occur if there is clock skew between adjacent flip-flops • To check for hold-time violations we find the clock skew for each clock-to-D path

```

timer> shortest
 1st shortest path to all endpoints
Rank Total Start pin      First Net      End Net      End pin
  0    4.0 b_rr_ff_b1:CLK b_rr_1_      DEF_NET_48    outp_ff_b1:D
  1    4.1 a_rr_ff_b2:CLK a_rr_2_      DEF_NET_46    outp_ff_b2:D
... 8 similar lines omitted ...

```


13.7.2 Entry Delay

Key terms and concepts: Before we can measure clock skew, we need to analyze the entry delays, including the clock tree

13.7.3 Exit Delay

Key terms and concepts: exit delays (the longest path between clock-pad input and an output) • critical path and operating frequency

13.7.4 External Setup Time

Key terms and concepts: external set-up time • internal set-up time • clock delay

Each of the six chip data inputs must satisfy the following set-up equation:

$$t_{SU}(\text{external}) > t_{SU}(\text{internal}) - (\text{clock delay}) + (\text{data delay})$$

13.8 Formal Verification

Key terms and concepts: logic synthesis converts a behavioral model to a structural model • How do we know that the two are the same? • **formal verification** can prove they are equivalent

13.8.1 An Example

Key terms and concepts: **reference model** • **derived model** • (1) the HDL is parsed • (2) a **finite-state machine compiler** extracts the states • (3) a **proof generator** automatically generates formulas to be proved • (4) the **theorem prover** attempts to prove the formulas

```

entity Alarm is                                     --1
  port(Clock, Key, Trip : in bit; Ring : out bit);  --2
end Alarm;                                         --3

architecture RTL of Alarm is                       --1
  type States is (Armed, Off, Ringing); signal State : States; --2
begin                                             --3
  process (Clock) begin                            --4
    if Clock = '1' and Clock'EVENT then          --5
      case State is                                --6
        when Off => if Key = '1' then State <= Armed; end if; --7
        when Armed => if Key = '0' then State <= Off; --8
                     elsif Trip = '1' then State <= Ringing; --9
                     end if;                      --10
      end if;
    end if;
  end process;
end architecture;

```

```

    when Ringing => if Key = '0' then State <= Off; end if;      --11
  end case;                                                    --12
end if;                                                        --13
end process;                                                  --14
Ring <= '1' when State = Ringing else '0';                    --15
end RTL;                                                       --16

library cells; use cells.all; // ...contains logic cell models --1
architecture Gates of Alarm is                                 --2
component Inverter port(i : in BIT;z : out BIT) ; end component; --3
component NAnd2 port(a,b : in BIT;z : out BIT) ; end component; --4
component NAnd3 port(a,b,c : in BIT;z : out BIT) ; end component; --5
component DFF port(d,c : in BIT; q,qn : out BIT) ; end component; --6
signal State, NextState : BIT_VECTOR(1downto 0);             --7
signal s0, s1, s2, s3 : BIT;                                  --8
begin                                                         --9
  g2: Inverter port map ( i => State(0), z => s1 );            --10
  g3: NAnd2 port map ( a => s1, b => State(1), z => s2 );      --11
  g4: Inverter port map ( i => s2, z => Ring );                --12
  g5: NAnd2 port map ( a => State(1), b => Key, z => s0 );     --13
  g6: NAnd3 port map ( a => Trip, b => s1, c => Key, z => s3 ); --14
  g7: NAnd2 port map ( a => s0, b => s3, z => NextState(1) ); --15
  g8: Inverter port map ( i => Key, z => NextState(0) );      --16
  state_ff_b0: DFF port map                                   --17
    ( d => NextState(0), c => Clock, q => State(0), qn =>open ); --18
  state_ff_b1: DFF port map                                   --19
    ( d => NextState(1), c => Clock, q => State(1), qn =>open ); --20
end Gates;                                                    --21

```

13.8.2 Understanding Formal Verification

Key terms and concepts: The **formulas** to be proved are generated as **proof statements** • An **axiom** is an explicit or implicit fact (signal of type BIT may only be '0' and '1') • An **assertion** is derived from a statement placed in the HDL code • **implication** • **equivalence**

```
assert Key /= '1' or Trip /= '1' or NextState = Ringing
  report "Alarm on and tripped but not ringing";
```

Implication and equivalence

A	B	A	B	A	B
F	F	T		T	
F	T	T		F	
T	F	F		F	
T	T	T		T	

13.8.3 Adding an Assertion

Key terms and concepts: “The axioms of the reference model do not imply that the assertions of the reference model imply the assertions of the derived model.” Translation: “These two architectures differ in some way.”

```
<E> Assertion may be violated
SEVERITY: ERROR
REPORT: Alarm on and tripped but not ringing
FILE: ../alarm-rtl3.vhdl
FSM: alarm-rtl3
STATEMENT or DECLARATION: line8
../alarm-rtl3.vhdl (line 8)
Context of the message is:
(key And trip And memoryofdriver__state(0))
```

```

case State is --1
  when Off => if Key = '1' then State <= Armed; end if; --2
  when Armed => if Key = '0' then State <= Off; --3
    elsif Trip = '1' then State <= Ringing; --4
    end if; --5
  when Ringing => if Key = '0' then State <= Off; end if; --6
end case; --7
```

```
Prove (Axiom_ref => (Assert_ref => Assert_der))
Formula is NOT VALID
But is VALID under Assert Context of alarm-rtl3
```

13.8.4 Completing a Proof

```

...
case State is
  when Off =>
    if Key = '1' then
      if Trip = '1' then NextState <= Ringing;
      else NextState <= Armed;
      end if;
    end if;
  when Armed => if Key = '0' then NextState <= Off;
    elsif Trip = '1' then NextState <= Ringing;
    end if;
  when Ringing => if Key = '0' then NextState <= Off; end if;
end case;
...

```

13.9 Switch-Level Simulation

Key terms and concepts: The **switch-level simulator** is a more detailed level of simulation than we have discussed so far • Example: a true single-phase flip-flop using true single-phase clocking (TSPC)

13.10 Transistor-Level Simulation

Key terms and concepts: **transistor-level simulation** or **circuit-level simulation** • **SPICE** (or **Spice, Simulation Program with Integrated Circuit Emphasis**) developed at UC Berkeley

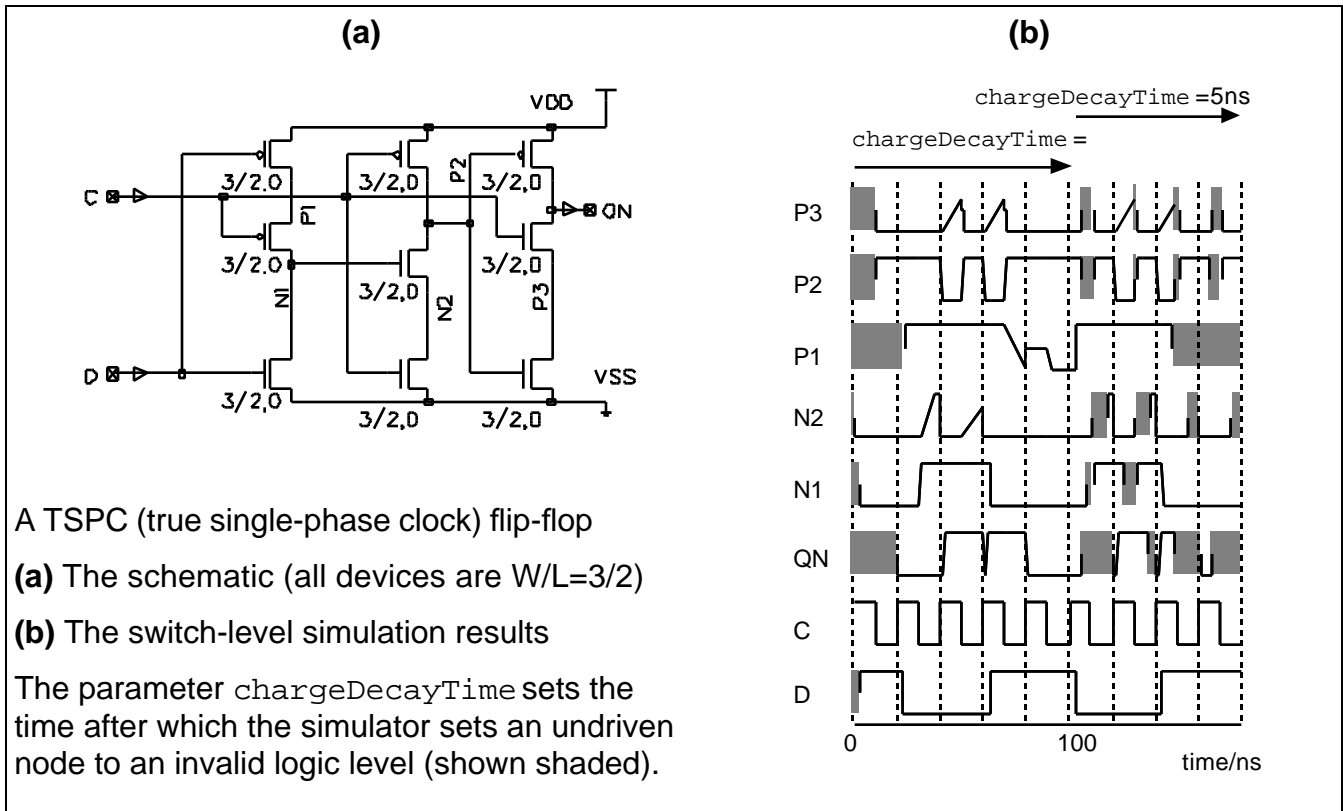
13.10.1 A PSpice Example

Key terms and concepts: **PSpice input deck**

```

OB September 5, 1996 17:27
.TRAN/OP 1ns 20ns
.PROBE
c1 output Ground 10pF
VIN input Ground PWL(0us 5V 10ns 5V 12ns 0V 20ns 0V)
VGround 0 Ground DC 0V
Vdd +5V 0 DC 5V
m1 output input Ground Ground NMOS W=100u L=2u

```



```

m2 output input +5V +5V PMOS W=200u L=2u
.model nmos nmos level=2 vto=0.78 tox=400e-10 nsub=8.0e15 xj=-0.15e-6
+ ld=0.20e-6 uo=650 ucrit=0.62e5 uexp=0.125 vmax=5.1e4 neff=4.0
+ delta=1.4 rsh=37 cgso=2.95e-10 cgdo=2.95e-10 cj=195e-6 cjsw=500e-12
+ mj=0.76 mjsw=0.30 pb=0.80
.model pmos pmos level=2 vto=-0.8 tox=400e-10 nsub=6.0e15 xj=-0.05e-6
+ ld=0.20e-6 uo=255 ucrit=0.86e5 uexp=0.29 vmax=3.0e4 neff=2.65
+ delta=1 rsh=125 cgso=2.65e-10 cgdo=2.65e-10 cj=250e-6 cjsw=350e-12
+ mj=0.535 mjsw=0.34 pb=0.80
.end

```

13.10.2 SPICE Models

Key terms and concepts: SPICE parameters • LEVEL=3 parameters

SPICE transistor model parameters (LEVEL=3)

parameter	n-ch. value	p-ch. value	Units	Explanation
CGBO	4.0E-10	3.8E-10	Fm ⁻¹	Gate–bulk overlap capacitance (CGBoh, not CGBzero)
CGDO	3.0E-10	2.4E-10	Fm ⁻¹	Gate–drain overlap capacitance (CGDoh, not CGDzero)
CGSO	3.0E-10	2.4E-10	Fm ⁻¹	Gate–source overlap capacitance (CGSoh, not CGSzero)
CJ	5.6E-4	9.3E-4	Fm ⁻²	Junction area capacitance
CJSW	5E-11	2.9E-10	Fm ⁻¹	Junction sidewall capacitance
DELTA	0.7	0.29	m	Narrow-width factor for adjusting threshold voltage
ETA	3.7E-2	2.45E-2	1	Static-feedback factor for adjusting threshold voltage
GAMMA	0.6	0.47	V ^{0.5}	Body-effect factor
KAPPA	2.9E-2	8	V ⁻¹	Saturation-field factor (channel-length modulation)
KP	2E-4	4.9E-5	AV ⁻²	Intrinsic transconductance (μC_{ox} , not $0.5\mu C_{ox}$)
LD	5E-8	3.5E-8	m	Lateral diffusion into channel
LEVEL	3		none	Empirical model
MJ	0.56	0.47	1	Junction area exponent
MJSW	0.52	0.50	1	Junction sidewall exponent
NFS	6E11	6.5E11	cm ⁻² V ⁻¹	Fast surface-state density
NSUB	1.4E17	8.5E16	cm ⁻³	Bulk surface doping
PB	1	1	V	Junction area contact potential
PHI	0.7		V	Surface inversion potential
RSH	2		/ square	Sheet resistance of source and drain
THETA	0.27	0.29	V ⁻¹	Mobility-degradation factor
TOX	1E-8		m	Gate-oxide thickness
TPG	1	-1	none	Type of polysilicon gate
U0	550	135	cm ² V ⁻¹ s ⁻¹	Low-field bulk carrier mobility (Uzero, not Uoh)
XJ	0.2E-6		m	Junction depth
VMAX	2E5	2.5E5	ms ⁻¹	Saturated carrier velocity
VTO	0.65	-0.92	V	Zero-bias threshold voltage (VTzero, not VToh)

PSpice parameters for process G5 (PSpice LEVEL=4)

```

.MODEL NM1 NMOS LEVEL=4
+ VFB=-0.7, LVFB=-4E-2, WVFB=5E-2
+ PHI=0.84, LPHI=0, WPHI=0
+ K1=0.78, LK1=-8E-4, WK1=-5E-2
+ K2=2.7E-2, LK2=5E-2, WK2=-3E-2
+ ETA=-2E-3, LETA=2E-02, WETA=-5E-3
+ MUZ=600, DL=0.2, DW=0.5
+ U0=0.33, LU0=0.1, WU0=-0.1
+ U1=3.3E-2, LU1=3E-2, WU1=-1E-2
+ X2MZ=9.7, LX2MZ=-6, WX2MZ=7
+ X2E=4.4E-4, LX2E=-3E-3, WX2E=9E-4
+ X3E=-5E-5, LX3E=-2E-3, WX3E=-1E-3
+ X2U0=-1E-2, LX2U0=-1E-3, WX2U0=5E-3
+ X2U1=-1E-3, LX2U1=1E-3, WX2U1=-7E-4
+ MUS=700, LMUS=-50, WMUS=7
+ X2MS=-6E-2, LX2MS=1, WX2MS=4
+ X3MS=9, LX3MS=2, WX3MS=-6
+ X3U1=9E-3, LX3U1=2E-4, WX3U1=-5E-3
+ TOX=1E-2, TEMP=25, VDD=5
+ CGDO=3E-10, CGSO=3E-10, CGBO=4E-10
+ XPART=1
+ N0=1, LN0=0, WN0=0
+ NB=0, LNB=0, WNB=0
+ ND=0, LND=0, WND=0
* n+ diffusion
+ RSH=2.1, CJ=3.5E-4, CJSW=2.9E-10
+ JS=1E-8, PB=0.8, PBSW=0.8
+ MJ=0.44, MJSW=0.26, WDF=0
*, DS=0

.MODEL PM1 PMOS LEVEL=4
+ VFB=-0.2, LVFB=4E-2, WVFB=-0.1
+ PHI=0.83, LPHI=0, WPHI=0
+ K1=0.35, LK1=-7E-02, WK1=0.2
+ K2=-4.5E-2, LK2=9E-3, WK2=4E-2
+ ETA=-1E-2, LETA=2E-2, WETA=-4E-4
+ MUZ=140, DL=0.2, DW=0.5
+ U0=0.2, LU0=6E-2, WU0=-6E-2
+ U1=1E-2, LU1=1E-2, WU1=7E-4
+ X2MZ=7, LX2MZ=-2, WX2MZ=1
+ X2E= 5E-5, LX2E=-1E-3, WX2E=-2E-4
+ X3E=8E-4, LX3E=-2E-4, WX3E=-1E-3
+ X2U0=9E-3, LX2U0=-2E-3, WX2U0=2E-3
+ X2U1=6E-4, LX2U1=5E-4, WX2U1=3E-4
+ MUS=150, LMUS=10, WMUS=4
+ X2MS=6, LX2MS=-0.7, WX2MS=2
+ X3MS=-1E-2, LX3MS=2, WX3MS=1
+ X3U1=-1E-3, LX3U1=-5E-4, WX3U1=1E-3
+ TOX=1E-2, TEMP=25, VDD=5
+ CGDO=2.4E-10, CGSO=2.4E-10, CGBO=3.8E-
10
+ XPART=1
+ N0=1, LN0=0, WN0=0
+ NB=0, LNB=0, WNB=0
+ ND=0, LND=0, WND=0
* p+ diffusion
+ RSH=2, CJ=9.5E-4, CJSW=2.5E-10
+ JS=1E-8, PB=0.85, PBSW=0.85
+ MJ=0.44, MJSW=0.24, WDF=0
*, DS=0

```

13.11 Summary

Key terms and concepts: Behavioral simulation can only tell you only if your design will not work

- Prelayout simulation estimates of performance
- Finding a critical path is difficult because you need to construct input vectors to exercise the model
- Static timing analysis is the most widely used form of simulation
- Formal verification compares two different representations. It cannot prove your design will work
- Switch-level simulation can check the behavior of circuits that may not always have nodes that are driven or that use logic that is not complementary
- Transistor-level simulation is used when you need to know the analog, rather than the digital, behavior of circuit voltages
- trade-off in accuracy against run time

TEST

Key terms and concepts: production test • wafer test or wafer sort • probe card • production tester • test program • test response • test vector • final test • goods-inward test • printed-circuit board (PCB or board) • failure analysis • field repair

14.1 The Importance of Test

Key terms and concepts: product quality • defect level • average quality level (AQL)

Defect levels in printed-circuit boards (PCB)		
ASIC defect level	Defective ASICs	Total PCB repair cost
5%	5000	\$1million
1%	1000	\$200,000
0.1%	100	\$20,000
0.01%	10	\$2,000

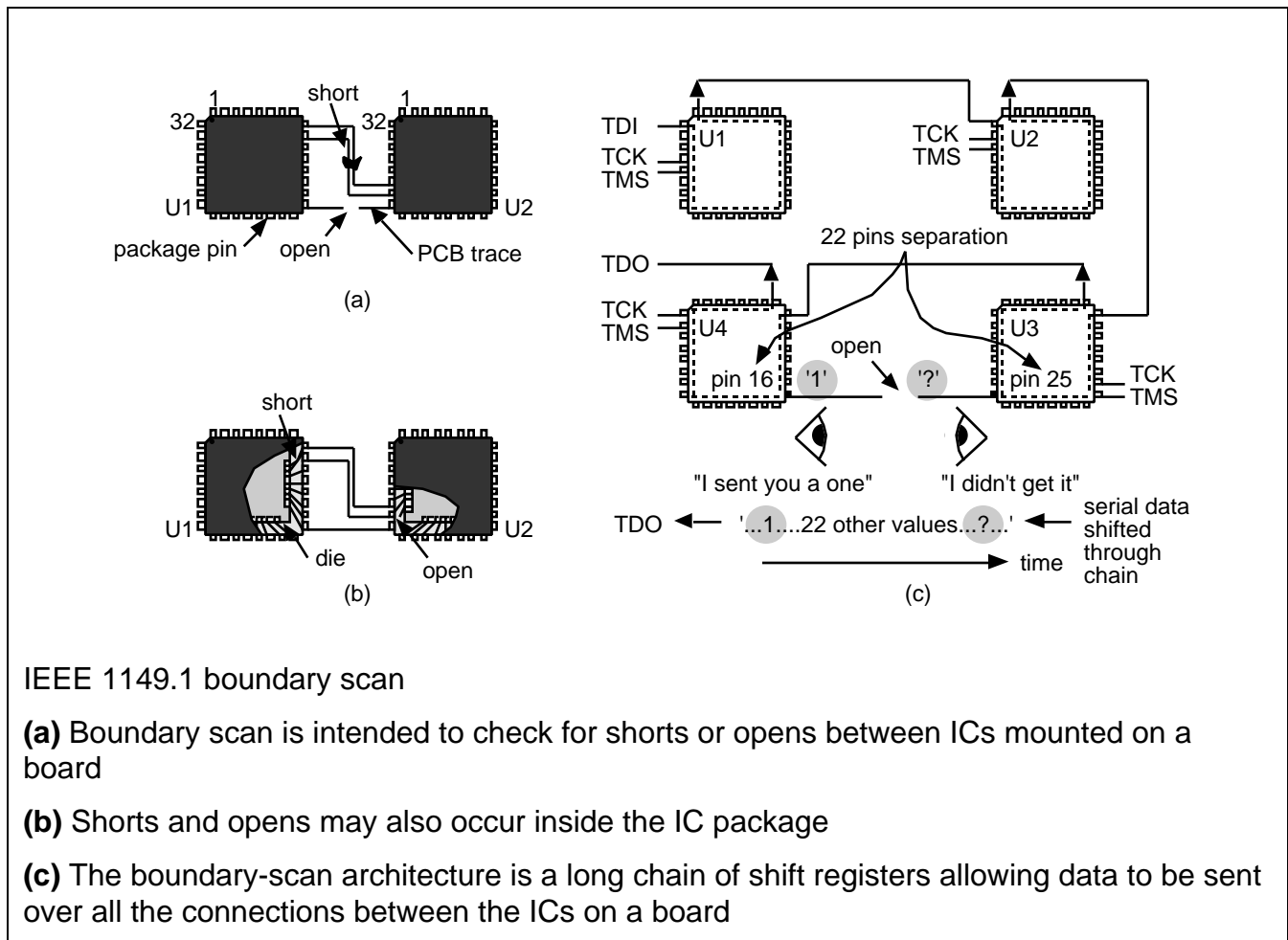
Defect levels in systems			
ASIC defect level	Defective ASICs	Defective boards	Total repair cost at system level
5%	5000	500	\$5million
1%	1000	100	\$1million
0.1%	100	10	\$100,000
0.01%	10	1	\$10,000

14.2 Boundary-Scan Test

Key terms and concepts: 4/5-wire interface for board-level test • Joint Test Action Group (JTAG)

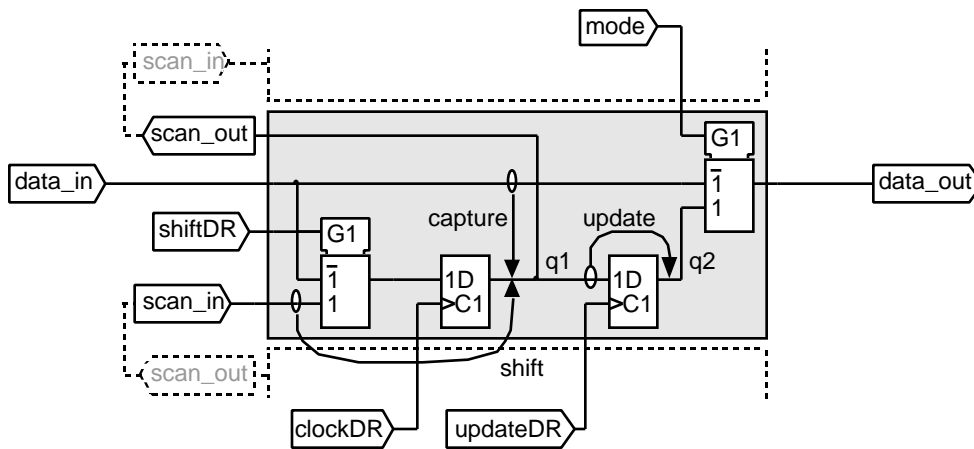
• **IEEE Standard 1149.1 Test Port and Boundary-Scan Architecture** • boundary-scan test (BST) • test-data output (TDO) • test-data registers (TDR) • test clock (TCK) • test-mode select (TMS) • test-reset input signal (TRST*) • test-access port (TAP)

Boundary-scan terminology		
Acronym	Meaning	Explanation
BR	Bypass register	A TDR, directly connects TDI and TDO, bypassing BSR
BSC	Boundary-scan cell	Each I/O pad has a BSC to monitor signals
BSR	Boundary-scan register	A TDR, a shift register formed from a chain of BSCs
BST	Boundary-scan test	Not to be confused with BIST (built-in self-test)
IDCODE	Device-identification register	Optional TDR, contains manufacturer and part number
IR	Instruction register	Holds a BST instruction, provides control signals
JTAG	Joint Test Action Group	The organization that developed boundary scan
TAP	Test-access port	Four- (or five-)wire test interface to an ASIC
TCK	Test clock	A TAP wire, the clock that controls BST operation
TDI	Test-data input	A TAP wire, the input to the IR and TDRs
TDO	Test-data output	A TAP wire, the output from the IR and TDRs
TDR	Test-data register	Group of BST registers: IDCODE, BR, BSR
TMS	Test-mode select	A TAP wire, together with TCK controls the BST state
TRST* or nTRST	Test-reset input signal	Optional TAP wire, resets the TAP controller (active-low)



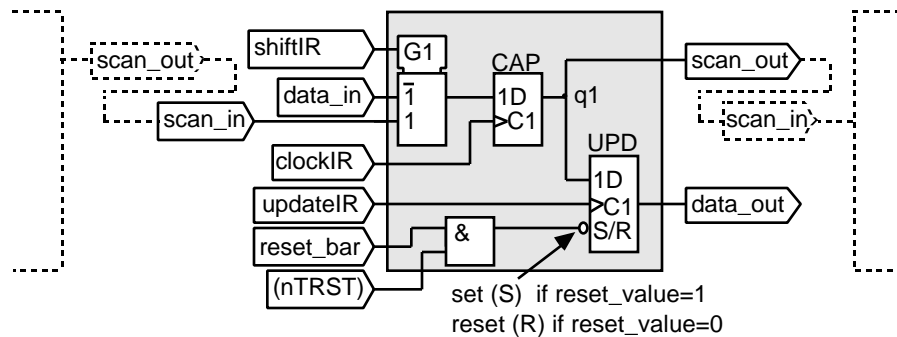
14.2.1 BST Cells

Key terms and concepts: **data-register cell (DR cell)** • **boundary-scan cell (BS cell, or BSC)** • capture flip-flop or capture register • update flip-flop, or update latch • scan in (serial in or SI) • data in (parallel in or PI) • mode (also called test/normal) • scan out (serial out or SO) • data out (parallel out or PO) • reversible • **bypass-register cell (BR cell)** • **instruction-register cell (IR cell)**



A DR (data register) cell

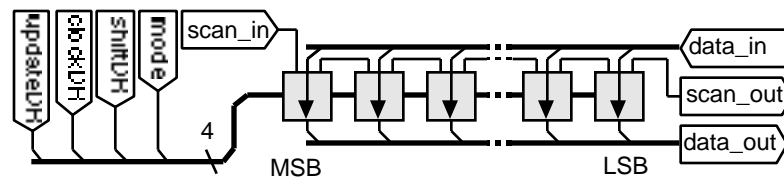
The most common use of this cell is as a boundary-scan cell (BSC)



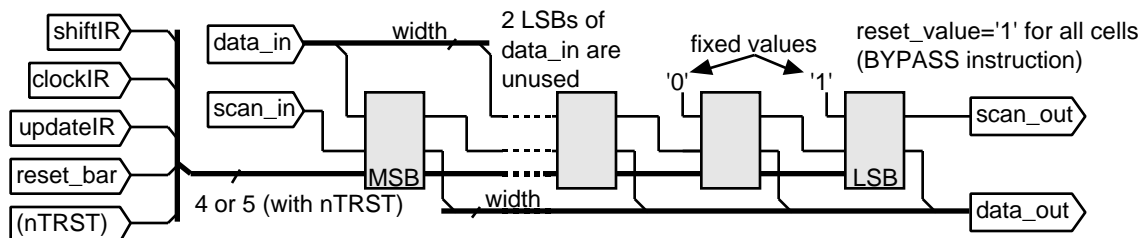
An IR (instruction register) cell

14.2.2 BST Registers

Key terms and concepts: **boundary-scan register (BSR)** • **instruction register (IR)**



A BSR (boundary-scan register)



An IR (instruction register)

14.2.3 Instruction Decoder

instruction decoder • device-identification register

An IR (instruction register) decoder

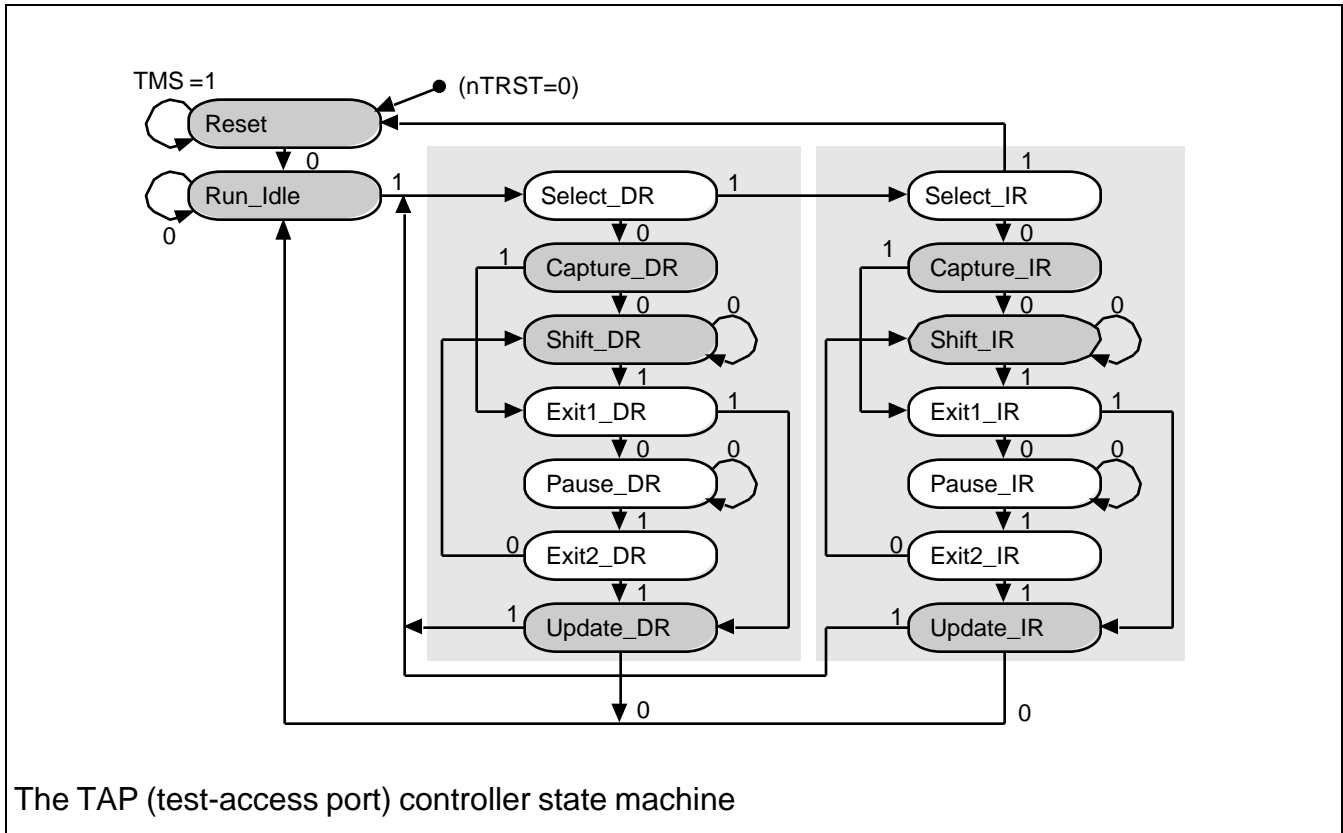
```

entity IR_decoder is generic (width : INTEGER := 4); port (
  shiftDR, clockDR, updateDR : BIT; IR_PO : BIT_VECTOR (width-1 downto 0) ;
  test_mode, selectBR, shiftBR, clockBR, shiftBSR, clockBSR, updateBSR : out BIT );
end IR_decoder;
architecture behave of IR_decoder is
  type INSTRUCTION is (EXTEST, SAMPLE_PRELOAD, IDCODE, BYPASS);
  signal I : INSTRUCTION;
begin process (IR_PO) begin case BIT_VECTOR'( IR_PO(1), IR_PO(0) ) is
  when "00" => I <= EXTEST; when "01" => I <= SAMPLE_PRELOAD;
  when "10" => I <= IDCODE; when "11" => I <= BYPASS;
end case; end process;
test_mode <= '1' when I = EXTEST else '0';
selectBR <= '1' when (I = BYPASS or I = IDCODE) else '0';
shiftBR <= shiftDR;
clockBR <= clockDR when (I = BYPASS or I = IDCODE) else '1';
shiftBSR <= shiftDR;
clockBSR <= clockDR when (I = EXTEST or I = SAMPLE_PRELOAD) else '1';
updateBSR <= updateDR when (I = EXTEST or I = SAMPLE_PRELOAD) else '0';
end behave;

```

14.2.4 TAP Controller

Key terms and concepts: JTAG “brain” • four-button digital watch • **clean** signal • dirty gated clocks



14.2.5 Boundary-Scan Controller

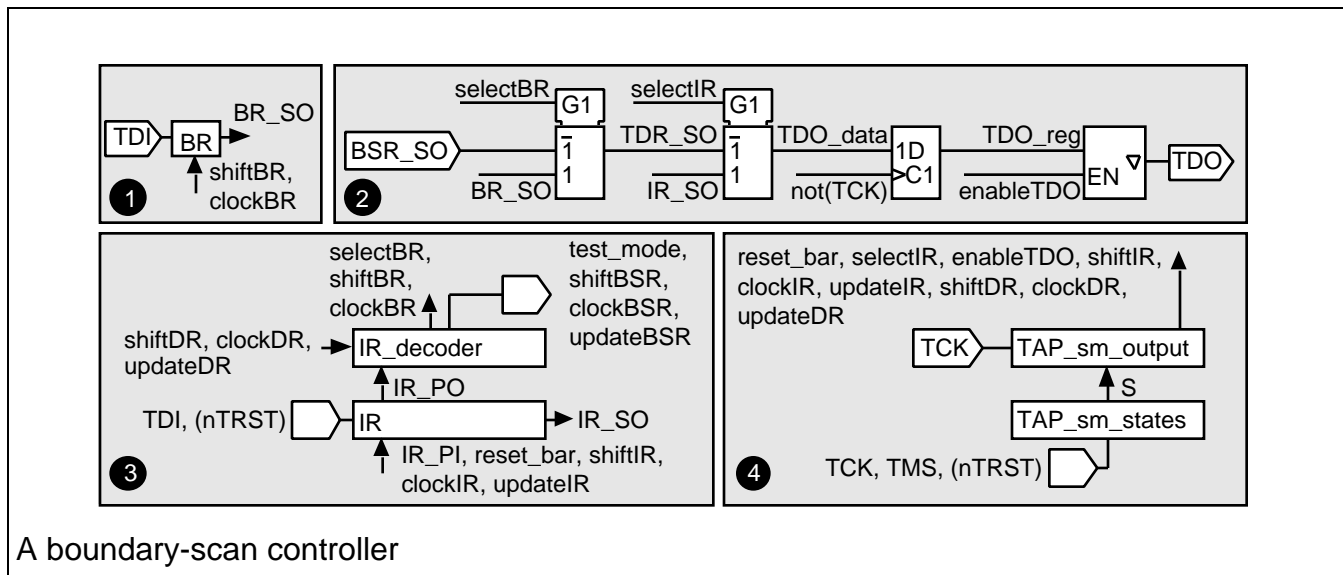
Key terms and concepts: bypass register • TDO output circuit. • instruction register and instruction decoder • TAP controller

14.2.6 A Simple Boundary-Scan Example

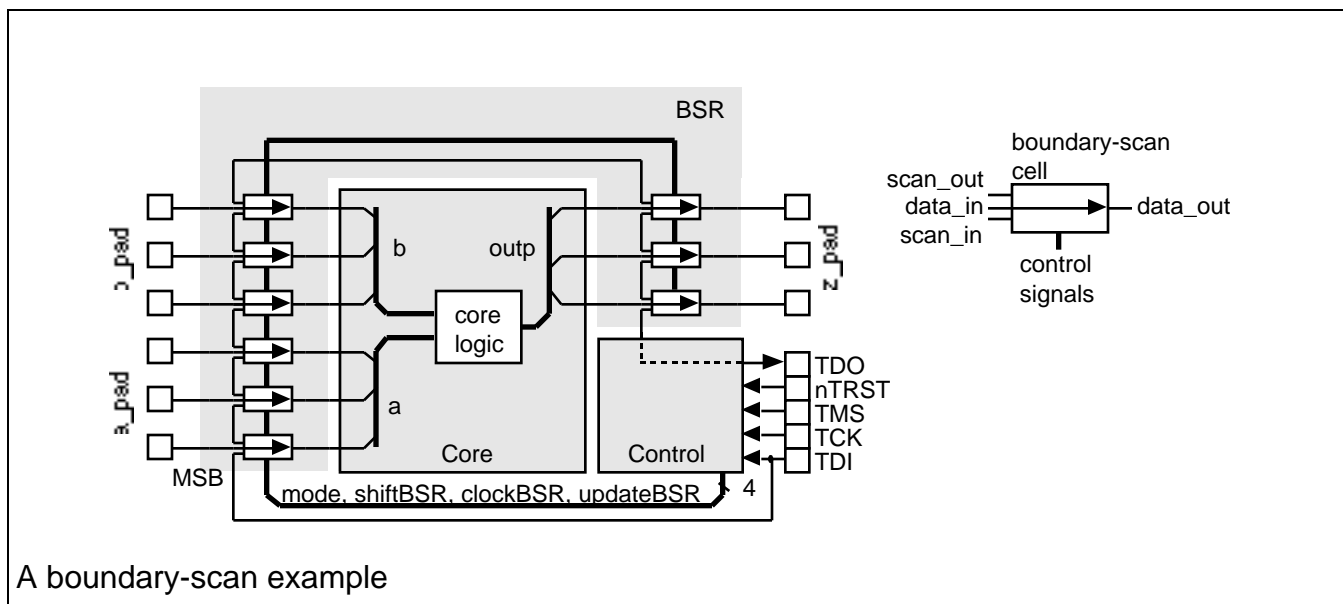
Key terms and concepts: Example: comparator/MUX containing boundary scan

14.2.7 BSDL

Key terms and concepts: **boundary-scan description language (BSDL)**



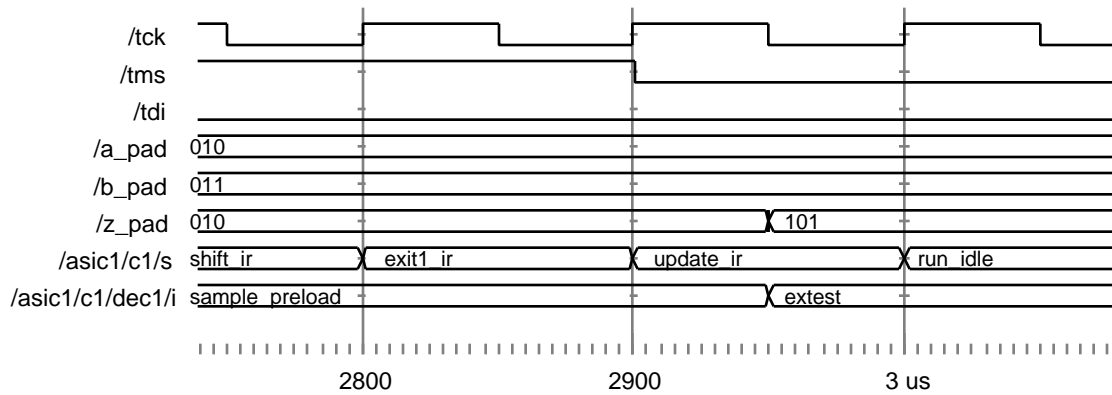
A boundary-scan controller



A boundary-scan example

14.3 Faults

Key terms and concepts: defect • **fault** • defect mechanisms • bridge or short circuit (shorts) • breaks or open circuits (opens) • rework



Results from the MTI simulator for the boundary-scan testbench

14.3.1 Reliability

Key terms and concepts: infant mortality • bathtub curve • wearout mechanisms • burn-in • $\exp(-E_a/kT)$ • Arrhenius equation • activation energy • reliability • mean time between failures (MTBF) • mean time to failure (MTTF) • failures in time (FITs)

14.3.2 Fault Models

Key terms and concepts: fault level • physical fault • fault model • logical fault • degradation fault • parametric fault • delay fault (timing fault) • open-circuit fault • short-circuit fault • bridging faults • metal coverage • feedback bridging faults and nonfeedback bridging faults

Mapping physical faults to logical faults		Logical fault		
		Degradation fault	Open-circuit fault	Short-circuit fault
Fault level	Physical fault			
Chip	Leakage or short between package leads	•		•
	Broken, misaligned, or poor wire bonding		•	
	Surface contamination, moisture	•		
	Metal migration, stress, peeling		•	•
	Metallization (open or short)		•	•
Gate	Contact opens		•	
	Gate to S/D junction short	•		•
	Field-oxide parasitic device	•		•
	Gate-oxide imperfection, spiking	•		•
	Mask misalignment	•		•

14.3.3 Physical Faults

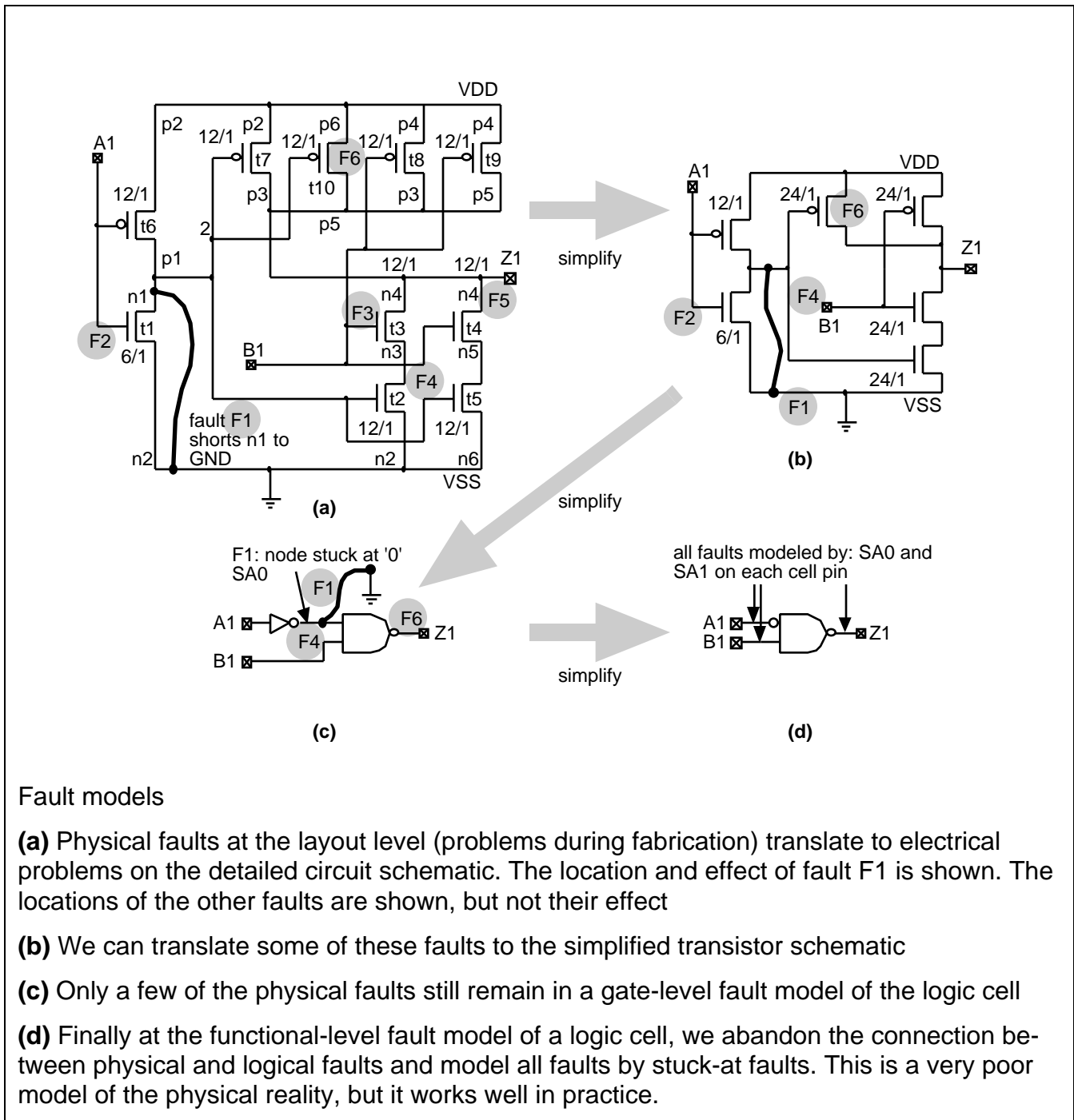
Key terms and concepts: **stuck-at fault model**

14.3.4 Stuck-at Fault Model

Key terms and concepts: single stuck-at fault (SSF) • multiple stuck-at fault model • stuck-on fault and stuck-open fault (or stuck-off fault) • stuck-at faults are: a stuck-at-1 fault (abbreviated to SA1 or s@1) and a stuck-at-0 fault (SA0 or s@0) • place faults (inject faults, seed faults, or apply faults) • fault origin • net fault • input fault • output fault • supply-strength fault (or rail-strength fault) • output-fault strength • node fault • pin-fault model • structural level, gate level, or cell level • transistor level or switch level • fault effect • fault propagation • structural fault propagation • behavioral fault propagation • mixed-level fault simulation

14.3.5 Logical Faults

Key terms and concepts: not all physical faults translate to logical faults—most do not



14.3.6 IDDQ Test

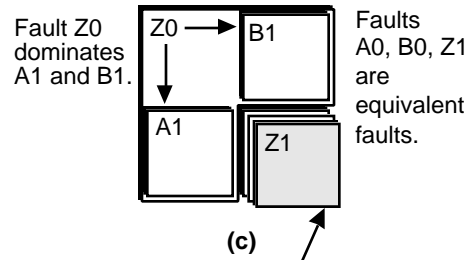
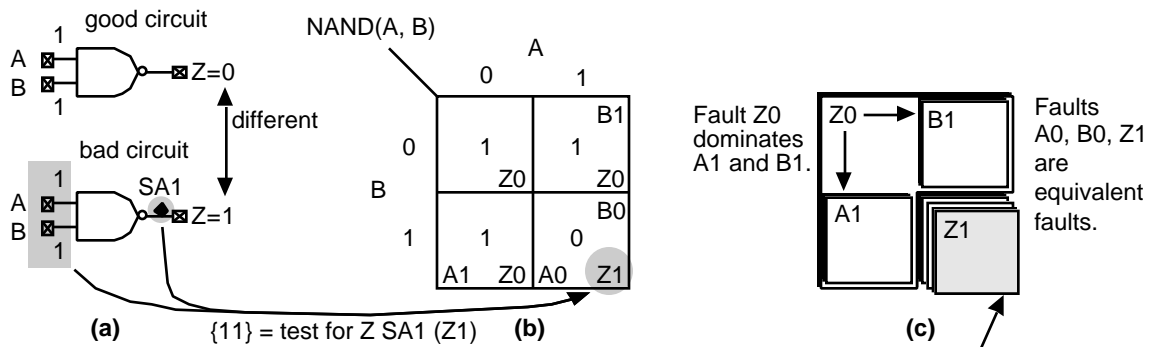
Key terms and concepts: **IDDQ** • high supply current can result from bridging faults

14.3.7 Fault Collapsing

Key terms and concepts: bad circuit (also called the faulty circuit or faulty machine) • fault collapsing • equivalent faults (or indistinguishable faults) • fault-equivalence class • prime fault or representative fault • dominant fault • dominant fault collapsing

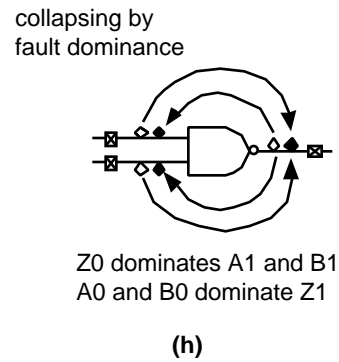
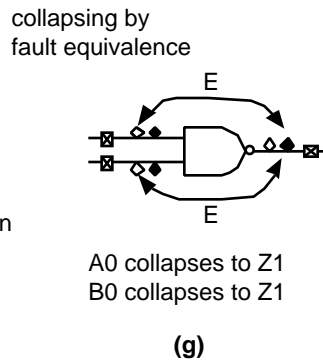
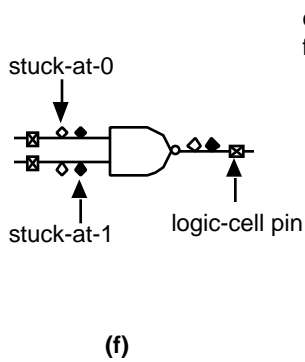
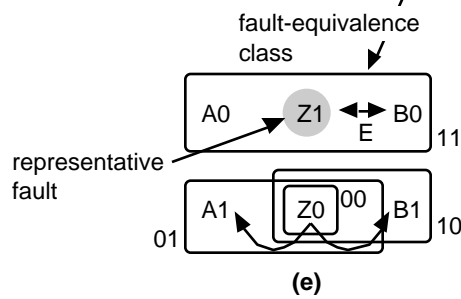
14.3.8 Fault-Collapsing Example

Key terms and concepts: gate collapsing • node collapsing



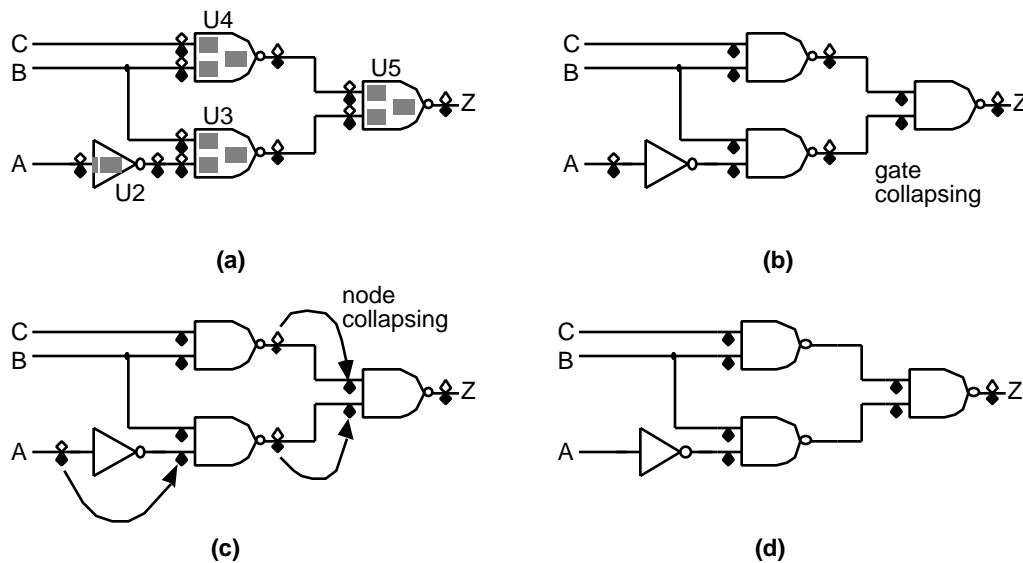
Test sets		
	SA0	SA1
Z	{00, 01, 10}	{11}
A	{11}	{01}
B	{11}	{10}

dominance, equivalence, E



Fault dominance and fault equivalence

- (a) A test for fault Z0 (Z stuck at 0) makes the bad circuit differ from the good circuit
- (b) Some test vectors provide tests for more than one fault
- (c) A test for A1 also tests for Z0, Z0 dominates A1. A0, B0, Z1 are the same (equivalent)
- (d) There are six sets of input vectors that test for the six stuck-at faults
- (e) We only need to choose a subset of all test vectors that test for all faults
- (f) The six stuck-at faults for a two-input NAND logic cell
- (g) Using fault equivalence we can collapse six faults to four
- (h) Using fault dominance we can collapse six faults to three.



Fault collapsing for $A'B + BC$

(a) A pin-fault model. Each pin has stuck-at-0 and stuck-at-1 faults

(b) Using fault equivalence the pin faults at the input pins and output pins of logic cells are collapsed. This is gate collapsing

(c) We can reduce the number of faults we need to consider further by collapsing equivalent faults on nodes and between logic cells. This is node collapsing

(d) The final circuit has eight stuck-at faults (reduced from the 22 original faults). If we wished to use fault dominance we could also eliminate the stuck-at-0 fault on Z. Notice that in a pin-fault model we cannot collapse the faults U4.A1.SA1 and U3.A2.SA1 even though they are on the same net.

14.4 Fault Simulation

Key terms and concepts: **fault simulation** • primary inputs (PIs) and primary outputs (POs) • stimulus • test vector • test program • test-cycle time • sense (or strobe) • detected fault • undetected fault • fault origins • fault coverage

Average quality level as a function of single stuck-at fault coverage		
Fault coverage	Average defect level	Average quality level (AQL)
50%	7%	93%
90%	3%	97%
95%	1%	99%
99%	0.1%	99.9%
99.9%	0.01%	99.99%

14.4.1 Serial Fault Simulation

Key terms and concepts: serial fault simulation • machines • good machine • faulty machine

14.4.2 Parallel Fault Simulation

Key terms and concepts: **parallel fault simulation** uses multiple bits per word • a bit is either a '1' or '0' for each node in the circuit • a 32-bit word can simulate 32 circuits at once

14.4.3 Concurrent Fault Simulation

Key terms and concepts: **concurrent fault simulation** takes advantage of the fact that a fault does not affect the whole circuit • diverged circuit • fault-activity signature • faults per pass

14.4.4 Nondeterministic Fault Simulation

Key terms and concepts: serial, parallel, and concurrent fault-simulation algorithms are forms of deterministic fault simulation • **probabilistic fault simulation** simulates a subset or sample of the faults and extrapolates coverage • statistical fault simulation performs a fault-free simulation and use the results to predict fault coverage • toggle test • vector quality • toggle coverage

14.4.5 Fault-Simulation Results

Key terms and concepts: fault categories • testable fault • controllable net • observable net • uncontrollable net and unobservable net • untested fault • **hard-detected fault** • undetected fault

- possibly detected fault
- soft-detected fault
- fault-drop threshold
- fault dropping
- redundant fault
- irredundant
- oscillatory fault
- hyperactive fault

(a) Detectable fault: A circuit with three primary inputs (PIs) A, B, and C, and one primary output (PO) D. A fault is shown on the line from input B to an AND gate. The circuit is controllable (inputs A, B, C are set to 1) and observable (output D is monitored).
 observe
 D = '1' (good circuit)
 D = '0' (bad circuit)

(b) Undetectable fault: A circuit with an uncontroltable net (input fixed to 0) and an undetectable fault.

(c) Undetectable fault: A circuit with an unobservable net (output QN) and an undetectable fault.

(d) Possible-detect fault: A circuit where a fault produces an unknown 'X' at output Z, which can be detected if it differs from the good circuit's output ('1' or '0').
 Z = '1' or '0' (good circuit)
 Z = 'X' (bad circuit)

(e) Redundant fault: A circuit where a fault on input B of an AND gate does not affect the output C because the gate is redundant ($AB + B' = A + B'$).

Fault categories

(a) A detectable fault requires the ability to control and observe the fault origin

(b) A net that is fixed in value is uncontrollable and therefore will produce one undetected fault

(c) Any net that is unconnected is unobservable and will produce undetected faults

(d) A net that produces an unknown 'X' in the faulty circuit and a '1' or a '0' in the good circuit may be detected (depending on whether the 'X' is in fact a '0' or '1'), but we cannot say for sure. At some point this type of fault is likely to produce a discrepancy between good and bad circuits and will eventually be detected

(e) A redundant fault does not affect the operation of the good circuit. In this case the AND gate is redundant since $AB + B' = A + B'$

14.4.6 Fault-Simulator Logic Systems

Key terms and concepts: fault grading • dead test cycles • fault list • faulty output vector • fault signature

		The VeriFault concurrent fault simulator logic system					
		Faulty circuit					
		0	1	Z	L	H	X
Good circuit	0	U	D	P	P	P	P
	1	D	U	P	P	P	P
	Z	U	U	U	U	U	U
	L	U	U	U	U	U	U
	H	U	U	U	U	U	U
	X	U	U	U	U	U	U

14.4.7 Hardware Acceleration

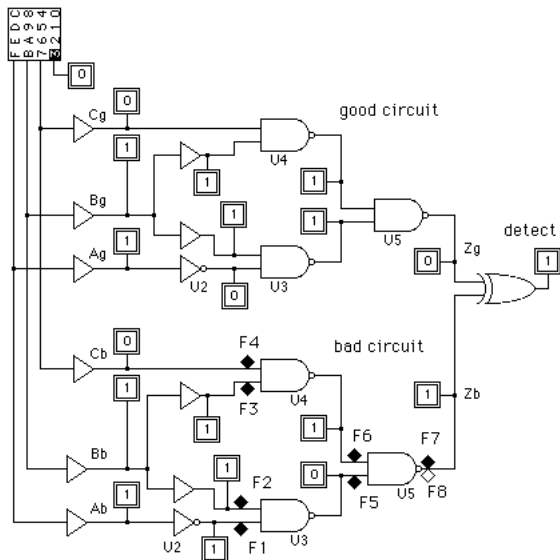
Key terms and concepts: simulation engines or hardware accelerators • distributed fault simulation

14.4.8 A Fault-Simulation Example

Key terms and concepts: test-vector compression or test-vector compaction • structurally equivalent

14.4.9 Fault Simulation in an ASIC Design Flow

Key terms and concepts: canned test vectors



Fault	Type	Vectors (hex)	Good output	Bad output
F1	SA1	3	0	1
F2	SA1	0, 4	0, 0	1, 1
F3	SA1	4, 5	0, 0	1, 1
F4	SA1	3	0	1
F5	SA1	2	1	0
F6	SA1	7	1	0
F7	SA1	0, 1, 3, 4, 5	0, 0, 0, 0, 0	1, 1, 1, 1, 1
F8	SA0	2, 6, 7	1, 1, 1	0, 0, 0

¹Test vector format:

3 = 011, so that CBA = 011: C = '0', B = '1', A = '1'

Fault simulation of A'B+BC

The simulation results for fault F1 (U2 output stuck at 1) with test vector value hex **3** (shown in bold in the table) are shown on the LogicWorks schematic

Notice that the output of U2 is 0 in the good circuit and stuck at 1 in the bad circuit.

14.5 Automatic Test-Pattern Generation

Key terms and concepts: PODEM, for **automatic test-pattern generation (ATPG)** or automatic test-vector generation (ATVG)

(a) **(b)**

(c)

AND	0	1	D	\bar{D}	X
0	0	0	0	0	0
1	0	1	0	\bar{D}	X
D	0	D	0	0	X
\bar{D}	0	\bar{D}	0	\bar{D}	X
X	0	X	X	X	X

OR	0	1	D	\bar{D}	X
0	0	0	1	D	\bar{D}
1	1	1	1	1	1
D	D	D	1	D	1
\bar{D}	\bar{D}	\bar{D}	1	1	\bar{D}
X	X	1	X	X	X

NAND	0	1	D	\bar{D}	X
0	1	1	1	1	1
1	1	0	1	D	X
D	1	\bar{D}	1	1	X
\bar{D}	1	D	1	D	X
X	1	X	X	X	X

NOR	0	1	D	\bar{D}	X
0	1	0	\bar{D}	D	X
1	0	0	0	0	0
D	\bar{D}	0	\bar{D}	0	X
\bar{D}	D	0	0	D	X
X	X	0	X	X	X

The D-calculus

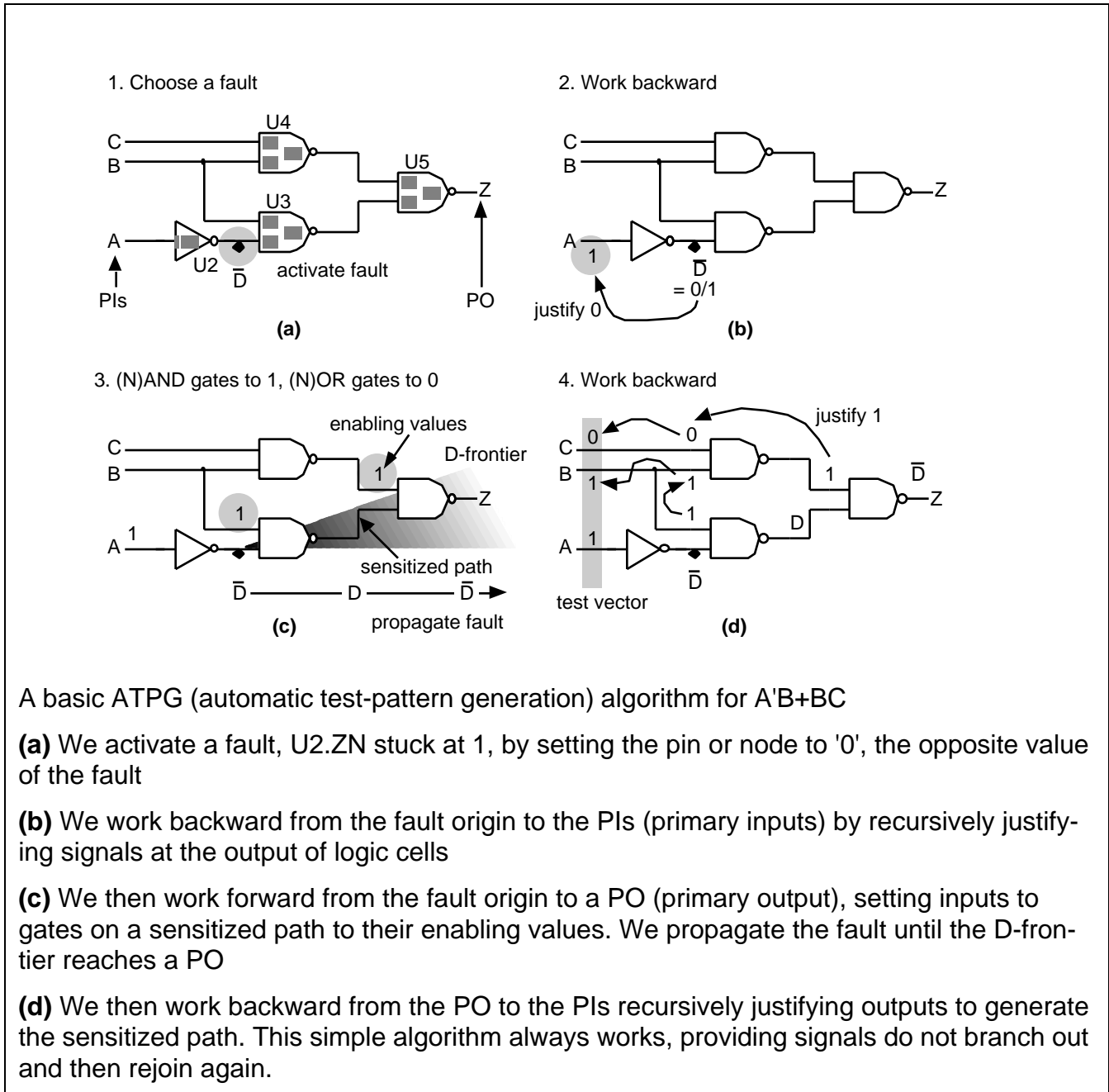
(a) We need a way to represent the behavior of the good circuit and the bad circuit at the same time

(b) The composite logic value D (for detect) represents a logic '1' in the good circuit and a logic '0' in the bad circuit. We can also write this as $D=1/0$

(c) The logic behavior of simple logic cells using the D-calculus. Composite logic values can propagate through simple logic gates if the other inputs are set to their enabling values.

14.5.1 The D-Calculus

Key terms and concepts: D-calculus • D-algorithm • D (for detect) • D=0/1 • g/b, a composite logic value • propagate • enabling value • controlling value • justifies



14.5.2 A Basic ATPG Algorithm

Key terms and concepts: activating (or exciting the fault) • sensitize • observed • D-frontier, • reconvergent fanout • multipath sensitization

(a) Signal B branches and then reconverges at logic gate U5, but the fault U4.A1 stuck at 1 can still be excited and a path sensitized using the basic algorithm

(b) Fault B stuck at 1 branches and then reconverges at gate U5. When we enable the inputs to both gates U3 and U4 we create two sensitized paths that prevent the fault from propagating to the PO (primary output). We can solve this problem by changing A to '0', but this breaks the rules of the algorithm. The PODEM algorithm solves this problem.

14.5.3 The PODEM Algorithm

Key terms and concepts: path-oriented decision making (PODEM) • objective • backtrack • implication • D-frontier • X-path check • backtrack • FAN (fanout-oriented test generation)

Iteration Objective Backtrace Implication D-frontier

1	U3.A2=0	J=1		
2	U3.A2=0	K=1	U7.ZN=1	
3	U3.A1=1	M=1	U3.ZN=D	U4, U6
4	U6.A2=1	N=1	U6.ZN= \bar{D}	U4, U8
5a	U8.A1=1	L=0	U8.ZN=1	U4, U8
5b	Retry	L=1	U8.ZN=D	A

The PODEM (path-oriented decision making) algorithm.

14.5.4 Controllability and Observability

Key terms and concepts: controllability (three l's) • observability • SCOAP (Sandia controllability/observability analysis program)• combinational controllability• sequential controllability• zero-controllability and one-controllability• combinational zero-controllability • logic distance • combinational one-controllability • **combinational observability**

(a)

(b)

(c)

Controllability measures

(a) Definition of combinational zero-controllability, CC0, and combinational one-controllability, CC1, for a two-input AND gate

(b) Examples of controllability calculations for simple gates, showing intermediate steps

(c) Controllability in a combinational circuit

(a) $O(X_1) = CC1(X_2) + O(Y) + 1$

(b) $O(X_1) = \min \{OC(X_2), OC(X_3)\}$

(c) $5 + 3 + 7 + 1 = 16$
 $5 + 1 + 7 + 1 = 14$
 $5 + 1 + 3 + 1 = 10$

(d) $3 + 1 + 1 = 5$
 $3 + 2 + 1 = 6$
 $5 + 1 = 6$
 $0 + 2 + 1 = 3$
 $0 + 2 + 1 = 3$

Observability measures

(a) The combinational observability, $OC(X_1)$, of an input, X_1 , to a two-input AND gate defined in terms of the controllability of the other input and the observability of the output

(b) The observability of a fanout node is equal to the observability of the most observable branch

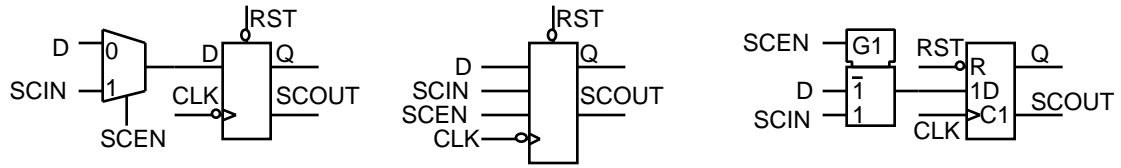
(c) Example of an observability calculation at a three-input NAND gate

(d) The observability of a combinational network can be calculated from the controllability measures, $CC0:CC1$. The observability of a PO (primary output) is defined to be zero.

14.6 ScanTest

Key terms and concepts: structured test • design for test • test compiler • scan insertion • pseudoprimary input • pseudoprimary output • partial scan • destructive scan • nondestructive scan • level-sensitive scan design (LSSD)

Scan flip-flop



14.7 Built-in Self-test

Key terms and concepts: **built-in self-test** (BIST) • circuit under test (CUT) or device under test (DUT)

14.7.1 LFSR

Key terms and concepts: **linear feedback shift register** (LFSR) • pseudorandom binary sequence (PRBS) • maximal-length sequence

LFSR example					
Clock tick, t=	$Q0_{t+1}=Q1_t$	$Q2_t$	$Q1_{t+1}=Q0_t$	$Q2_{t+1}=Q1_t$	Q0Q1Q2
1	1		1	1	7
2	0		1	1	3
3	0		0	1	1
4	1		0	0	4
5	0		1	0	2
6	1		0	1	5
7	1		1	0	6
8	1		1	1	7

A linear feedback shift register (LFSR).

A 3-bit maximal-length LFSR produces a repeating string of seven pseudorandom binary numbers: 7, 3, 1, 4, 2, 5, 6.

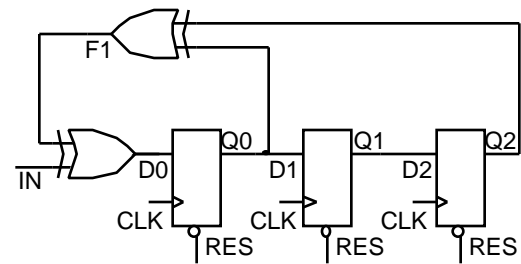
14.7.2 Signature Analysis

Key terms and concepts: data compaction • signature • serial-input signature register (SISR) • signature analysis • Hewlett-Packard

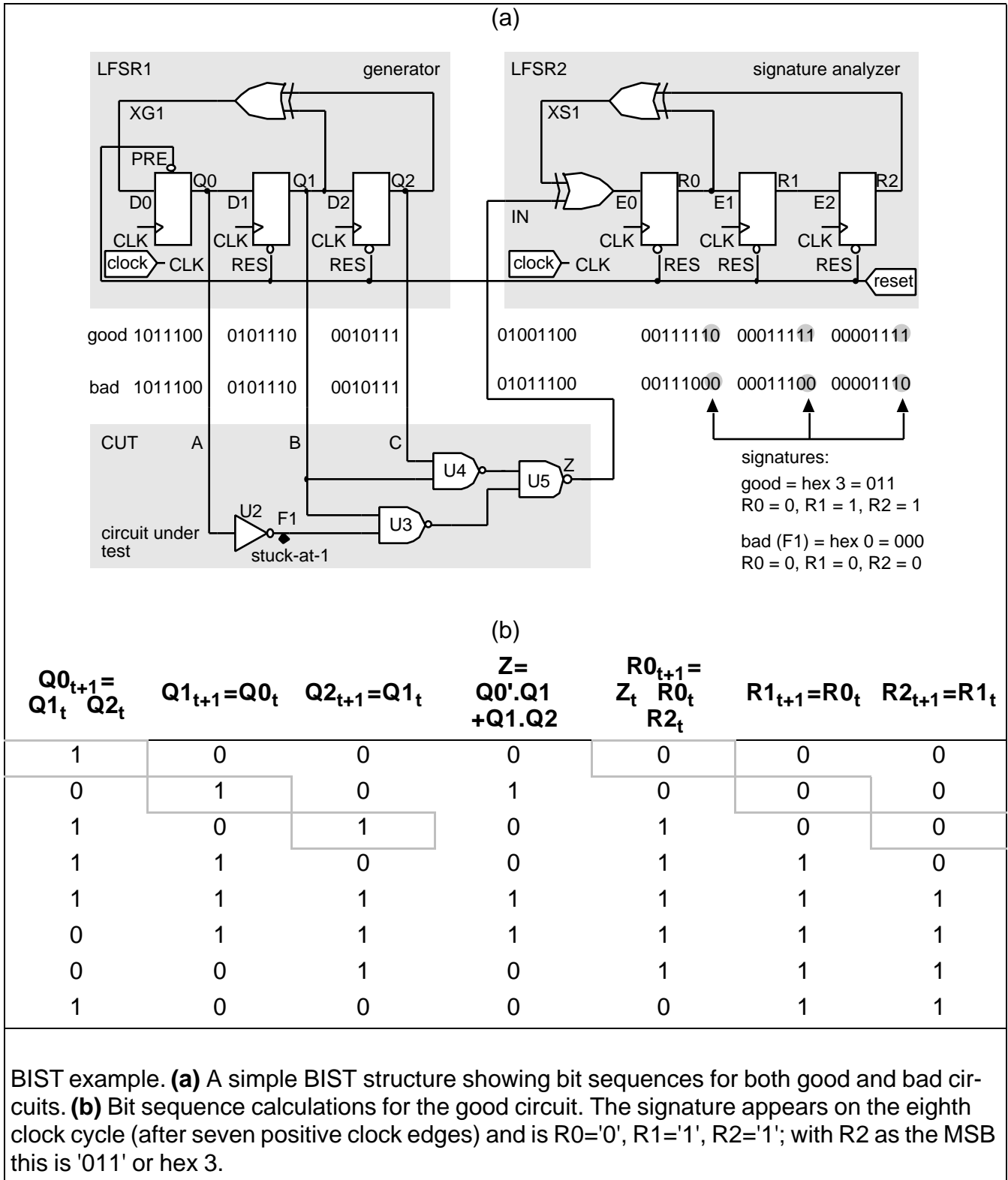
A 3-bit serial-input signature register (SISR) using an LFSR (linear feedback shift register)

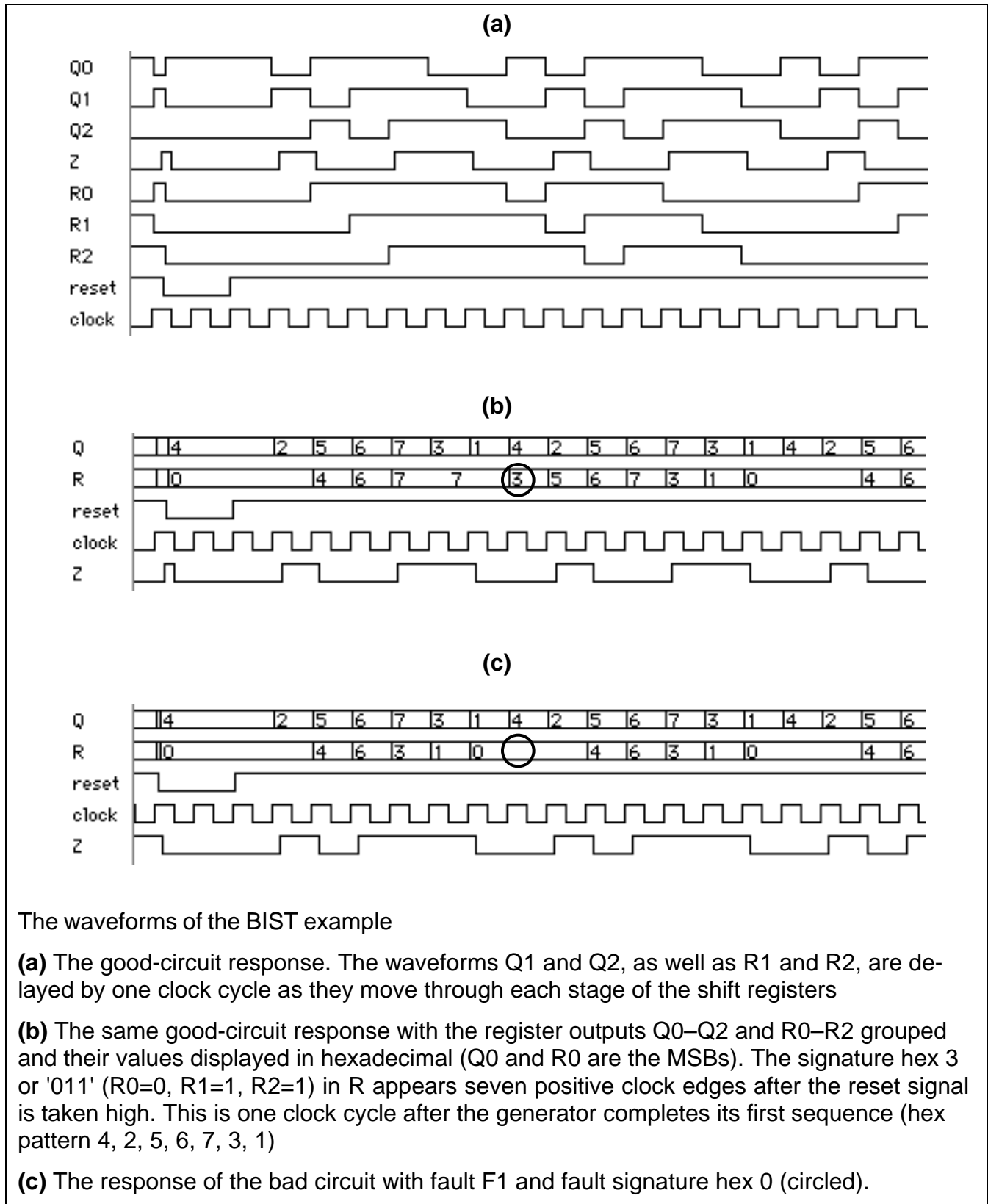
The LFSR is initialized to $Q_1Q_2Q_3='000'$ using the common RES (reset) signal

The signature, $Q_1Q_2Q_3$, is formed from shift-and-add operations on the sequence of input bits (IN)



14.7.3 A Simple BIST Example





14.7.4 Aliasing

Key terms and concepts: **aliasing** • error coverage

14.7.5 LFSR Theory

Key terms and concepts: polynomials and Galois-field theory • characteristic polynomial • primitive polynomials • external-XOR LFSR • type 1 LFSR • internal-XOR LFSR • type 2 LFSR

n	s	Octal	Binary
1	0, 1	3	11
2	0, 1, 2	7	111
3	0, 1, 3	13	1011
4	0, 1, 4	3	10011
5	0, 2, 5	45	100101
6	0, 1, 6	103	1000011
7	0, 1, 7	211	10001001
8	0, 1, 5, 6, 8	435	100011101
9	0, 4, 9	1021	1000010001
10	0, 3, 10	2011	10000001001

For $n=3$ and $s=0, 1, 3$: $c_0=1, c_1=1, c_2=0, c_3=1$

$P(x) = 1 \quad c_1x \quad \dots \quad c_{n-1}x^{n-1} \quad x^n$

or $P^*(x) = 1 \quad c_{n-1}x \quad \dots \quad c_1x^{n-1} \quad x^n$

Primitive polynomial coefficients for LFSRs (linear feedback shift registers) that generate a maximal-length PRBS (pseudorandom binary sequence)

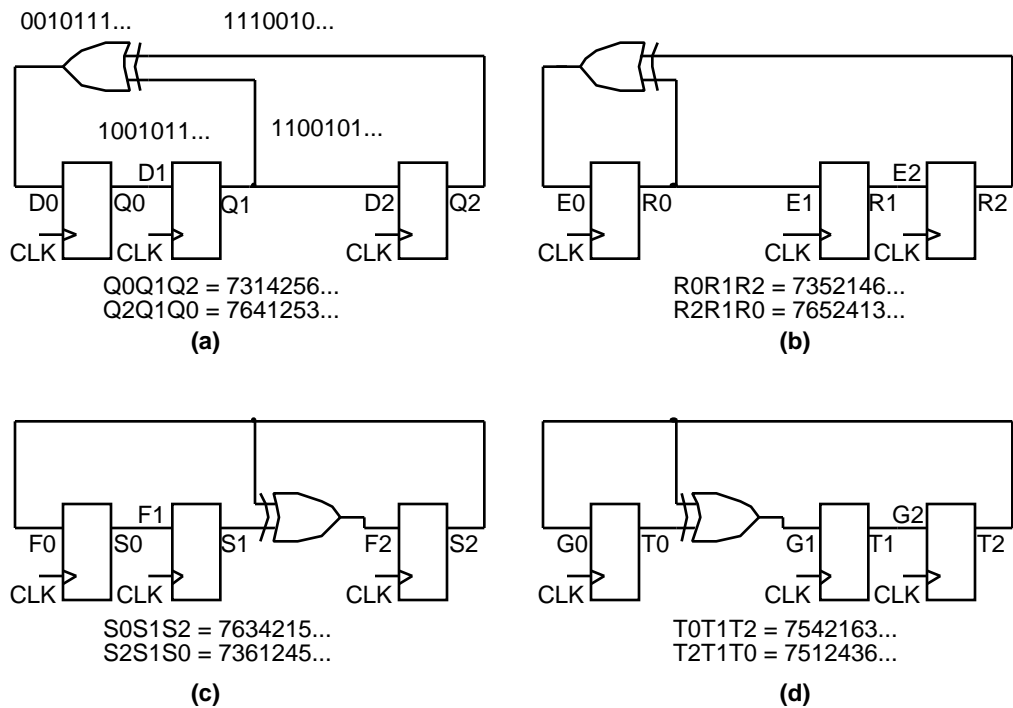
A schematic for a type 1 LFSR is shown.

14.7.6 LFSR Example

Key terms and concepts: automatic generation of LFSR and SISR structures

14.7.7 MISR

Key terms and concepts: multiple-input signature register (MISR) • built-in logic block observer (BILBO) • circular self-test path (CSTP) • complete LFSR • **scanBIST**



For every primitive polynomial there are four linear feedback shift registers (LFSRs).

There are two types of LFSR; one type uses external XOR gates (type 1) and the other type uses internal XOR gates (type 2).

For each type the feedback taps can be constructed either from the polynomial $P(x)$ or from its reciprocal, $P^*(x)$. The LFSRs in this figure correspond to $P(x)=1-x-x^3$ and $P^*(x)=1-x^2-x^3$.

Each LFSR produces a different pseudorandom sequence, as shown. The binary values of the LFSR seen as a register, with the bit labeled as zero being the MSB, are shown in hexadecimal.

The sequences shown are for each register initialized to '111', hex 7.

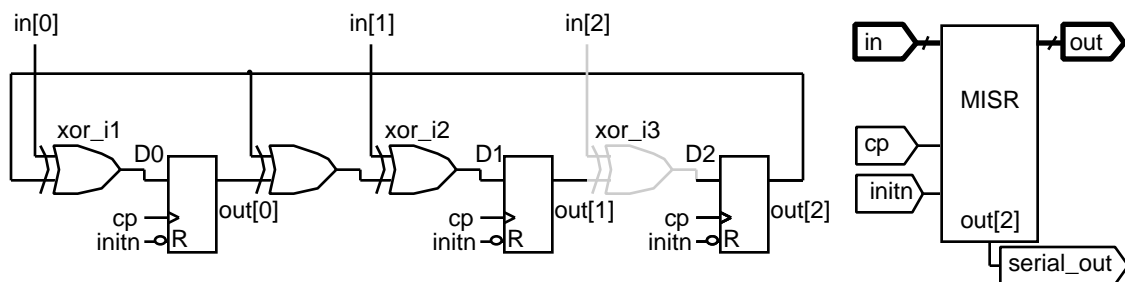
(a) Type 1, $P^*(x)$. **(b)** Type 1, $P(x)$. **(c)** Type 2, $P(x)$. **(d)** Type 1, $P^*(x)$.

Compiled LFSR generator, using $P^*(x)=1-x^2-x^3$

```

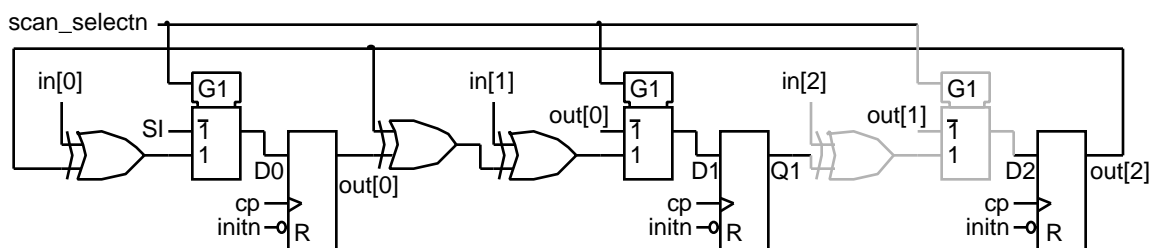
module lfsr_generator (OUT, SERIAL_OUT, INITN, CP);
output [2:0] OUT; output SERIAL_OUT; input INITN, CP;
  dfptnb FF2 (.D(FF0_Q), .CP(u4_Z), .SDN(u2_Z), .Q(FF2_Q), .QN(FF2_QN));
  dfctnb FF1 (.D(XOR0_Z), .CP(u4_Z), .CDN(u2_Z), .Q(FF1_Q), .QN(FF1_QN));
  dfctnb FF0 (.D(FF1_Q), .CP(u4_Z), .CDN(u2_Z), .Q(FF0_Q), .QN(FF0_QN));
  ni01d1 u2 (.I(u3_Z), .Z(u2_Z)); ni01d1 u3 (.I(INITN), .Z(u3_Z));
  ni01d1 u4 (.I(u5_Z), .Z(u4_Z)); ni01d1 u5 (.I(CP), .Z(u5_Z));
  xo02d1 XOR0 (.A1(FF2_Q), .A2(FF0_Q), .Z(XOR0_Z));
  in02d1 INV2X0 (.I(FF0_QN), .ZN(OUT[0]));
  in02d1 INV2X1 (.I(FF1_QN), .ZN(OUT[1]));
  in02d1 INV2X2 (.I(FF2_QN), .ZN(OUT[2]));
  in02d1 INV2X3 (.I(FF0_QN), .ZN(SERIAL_OUT));
endmodule

```



Multiple-input signature register (MISR).

This MISR is formed from the type 2 LFSR (with $P^*(x)=1-x^2-x^3$) by adding XOR gates *xor_i1*, *xor_i2*, and *xor_i3*. This 3-bit MISR can form a signature from logic with three outputs. If we only need to test two outputs then we do not need XOR gate, *xor_i3*, corresponding to input *in[2]*.

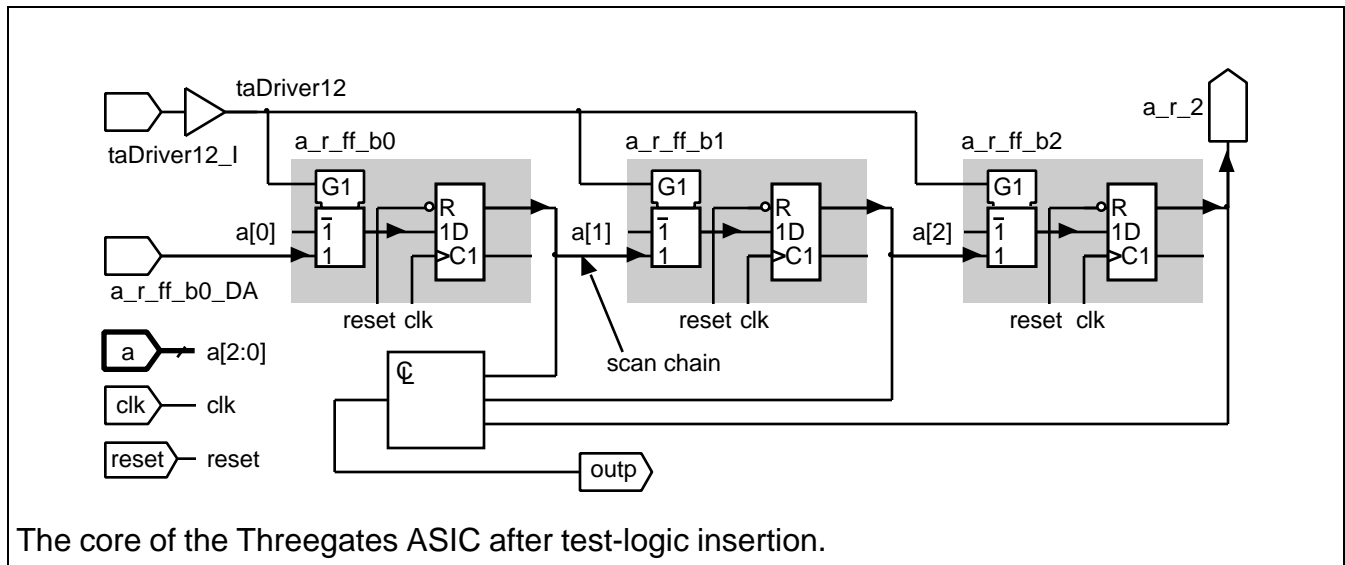


Multiple-input signature register (MISR) with scan

14.8 A Simple Test Example

14.8.1 Test-Logic Insertion

Key terms and concepts: $outp = a_r[0]' \cdot a_r[1] + a_r[1] \cdot a_r[2]$ • **test-logic insertion**

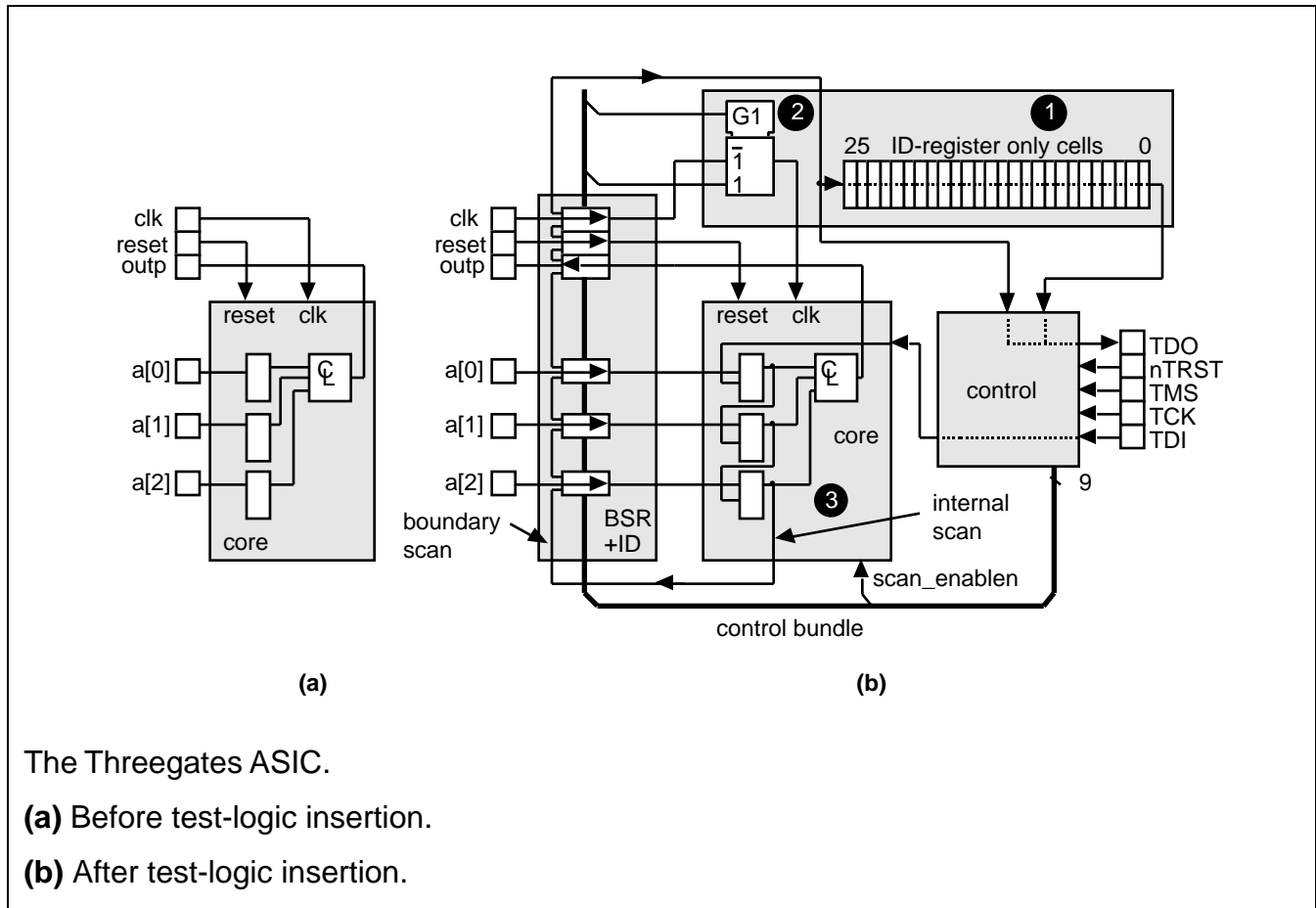


14.8.2 How the Test Software Works

Key terms and concepts: **polarity-hold flip-flop**

14.8.3 ATVG and Fault Simulation

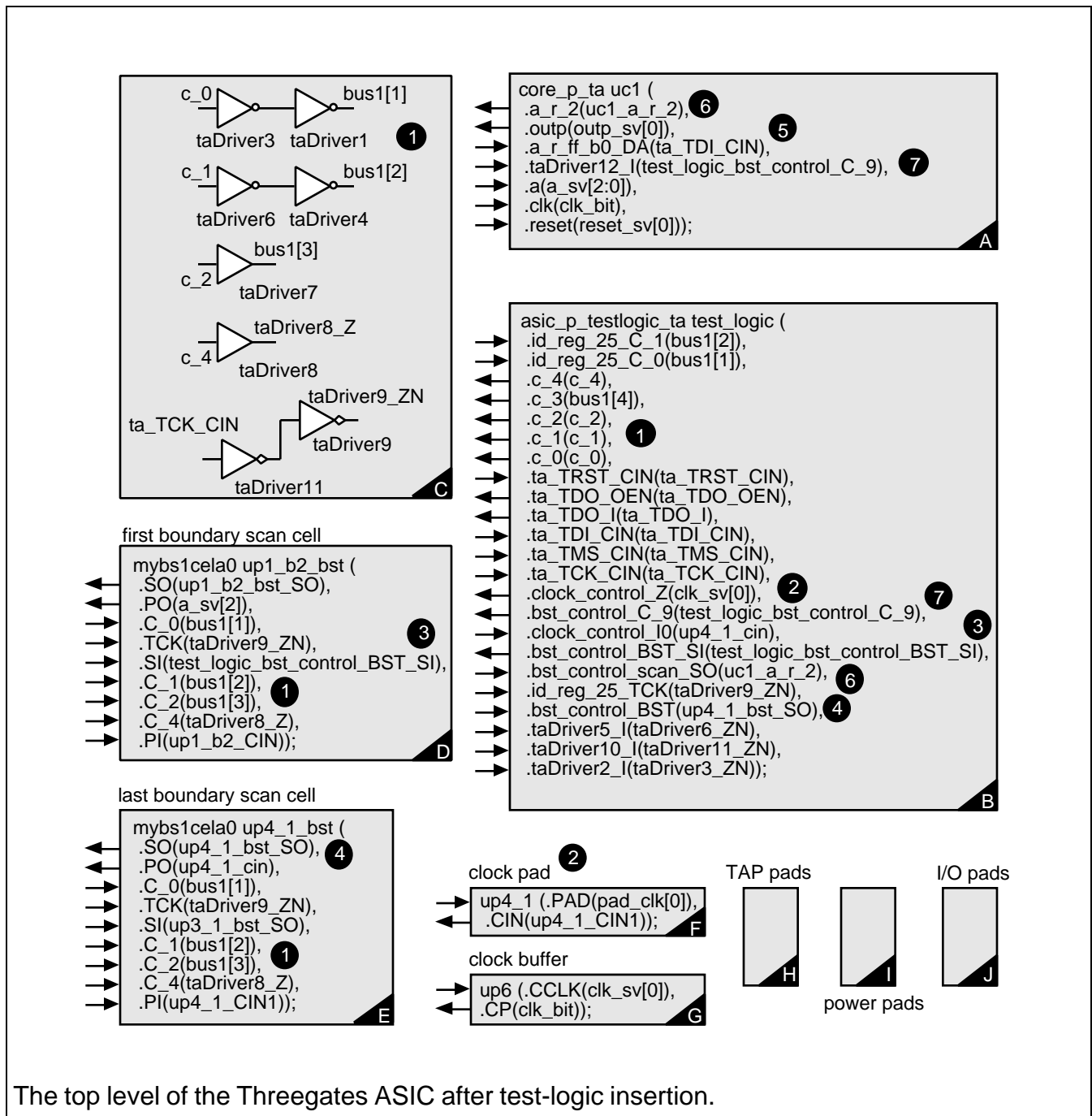
Key terms and concepts: **flush test**

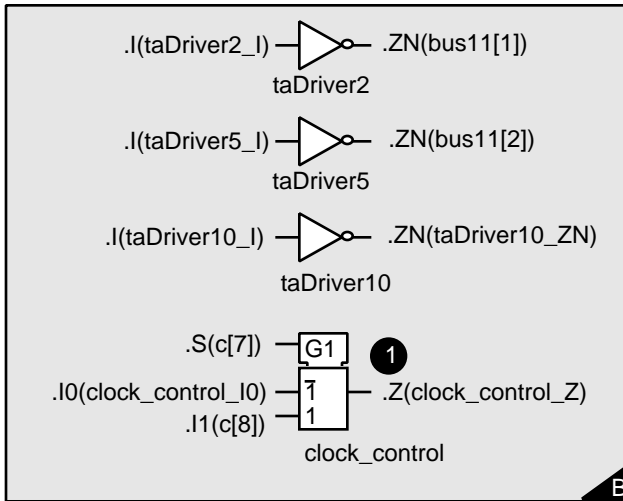


The Threegates ASIC.

(a) Before test-logic insertion.

(b) After test-logic insertion.





```

module asic_p_testlogic_ta (
  id_reg_25_C_1,
  id_reg_25_C_0,
  c_4, c_3, c_2, c_1, c_0,
  ta_TRST_CIN,
  ta_TDO_OEN,
  ta_TDO_I,
  ta_TDI_CIN,
  ta_TMS_CIN,
  ta_TCK_CIN,
  clock_control_Z,
  bst_control_C_9,
  clock_control_I0,
  bst_control_BST_SI,
  bst_control_scan_SO,
  id_reg_25_TCK,
  bst_control_BST,
  taDriver5_I,
  taDriver10_I,
  taDriver2_I);
  
```

```

bs1cong0 bst_control (
  .C({c_0, c_1, c_2, c_3, c_4, open_net1, open_net2, c[7], c[8], bst_control_C_9}),
  .OEN(ta_TDO_OEN),
  .TDO(ta_TDO_I),
  .BST(bst_control_BST),
  .DID(id_reg_0_SO),
  .TCK (ta_TCK_CIN),
  .TDI(ta_TDI_CIN),
  .TMS(ta_TMS_CIN),
  .TRST(ta_TRST_CIN),
  .scan_SO(bst_control_scan_SO),
  .BST_SI(bst_control_BST_SI));
  
```

first IDR cell

```

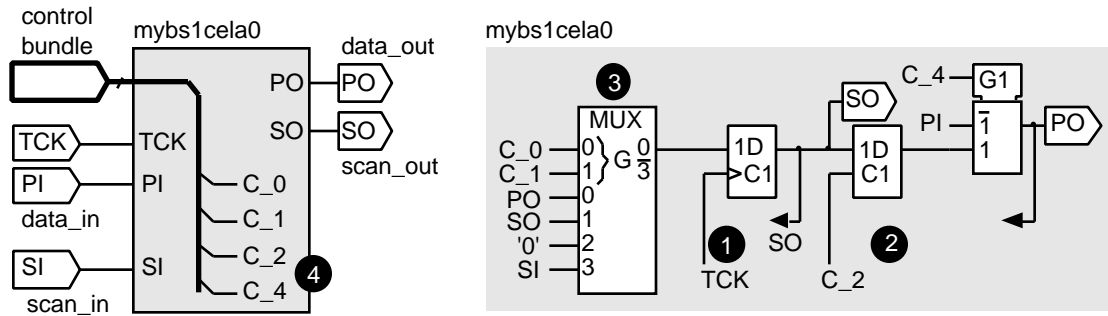
bs1celf0 id_reg_25 (
  .SO(id_reg_25_SO),
  .C({id_reg_25_C_0, id_reg_25_C_1}),
  .SI(bst_control_BST),
  .TCK(id_reg_25_TCK));
  
```

last IDR cell

```

bs1celf1 id_reg_0 (
  .SO(id_reg_0_SO),
  .C({bus11[1], bus11[2]}),
  .SI(id_reg_1_SO),
  .TCK(taDriver10_ZN));
  
```

Test logic inserted in the Threegate ASIC.



Input boundary-scan cell (BSC) for the Threegates ASIC.

Compare this to a generic data-register (DR) cell (used as a BSC).

ATVG (automatic test-vector generation) report for the Threegates ASIC

```

CREATE: Output vector database cell defaulted to [svf]asic_p_ta
CREATE: Backtrack limit defaulted to 30
CREATE: Minimal compression effort: 10 (default)
Fault list generation/collapsing
Total number of faults: 184
Number of faults in collapsed fault list: 80
Vector generation
#
# VECTORS   FAULTS   FAULT COVER
#           processed
#
#           5       184       60.54%
#
# Total number of backtracks: 0
# Highest backtrack           : 0
# Total number of vectors    : 5
#
# STAR RESULTS summary
#
# Fault counts:
#           Noncollapsed      Collapsed
# Aborted           0           0
# Detected          89          43
# Untested          58          20
#
#           -----
# Total of detectable 147          63
#
# Redundant         6           2
# Tied              31          15
#
# FAULT COVERAGE          60.54 %          68.25 %
#
# Fault coverage = nb of detected faults / nb of detectable faults
Vector/fault list database [svf]asic_p_ta created.
    
```

14.8.4 Test Vectors

Key terms and concepts: serial vectors • parallel vectors • broadside vectors

14.8.5 Production Tester Vector Formats

Key terms and concepts: Sentry tester file format

```
# Pin declaration: pin names are separated by semi-colons (all pins
# on a bus must be listed and separated by commas)
pre_; clr_; d; clk; q; q_;
# Pin declarations are separated from test vectors by $
$
# The first number on each line is the time since start in ns,
# followed by space or a tab.
# The symbols following the time are the test vectors
# (in the same order as the pin declaration)
# an "=" means don't do anything
# an "s" means sense the pin at the beginning of this time point
# (before the input changes at this time point have any effect)
#
# pcdcqq
# rlal _
# ertk
# __a
00 1010== # clear the flip-flop
10 1110ss # d=1, clock=0
20 1111ss # d=1, clock=1
30 1110ss # d=1, clock=0
40 1100ss # d=0, clock=0
50 1101ss # d=0, clock=1
60 1100ss # d=0, clock=0
70 =====ss
```

14.8.6 Test Flow

Key terms and concepts: test-vector generation and the production-test program generation is the last step in ASIC design after physical design is complete

Timing effects of test-logic insertion for the Viterbi decoder

Timing of critical paths before test-logic insertion

#	Slack(ns)	Num	Paths
#	-3.3826	1	*
#	-1.7536	18	*****
#	-.1245	4	**
#	1.5045	1	*
#	3.1336	0	*
#	4.7626	0	*
#	6.3916	134	*****
#	8.0207	6	***
#	9.6497	3	**
#	11.2787	0	*
#	12.9078	24	*****

#	instance name	incr	arrival	trs	rampDel	cap	cell
#	inPin --> outPin	(ns)	(ns)		(ns)	(pf)	
#	v_1.u100.u1.subout6.Q_ff_b0						
#	CP --> QN	1.73	1.73	R	.20	.10	dfctnb
#	...						
#	v_1.u100.u2.metric0.Q_ff_b4						
#	setup: D --> CP	.16	21.75	F	.00	.00	dfctnh

After test-logic insertion

#	-4.0034	1	*
#	-1.9835	18	*****
#	.0365	4	**
#	2.0565	1	*
#	4.0764	0	*
#	6.0964	138	*****
#	8.1164	2	*
#	10.1363	3	**
#	12.1563	24	*****
#	14.1763	0	*
#	16.1963	187	*****

#	v_1.u100.u1.subout7.Q_ff_b1						
#	CP --> Q	1.40	1.40	R	.28	.13	mfctnb
#	...						
#	v_1.u100.u2.metric0.Q_ff_b4						
#	setup: DB --> CP	.39	21.98	F	.00	.00	mfctnh

14.9 The Viterbi Decoder Example

Fault coverage for the Viterbi decoder

```

Fault list generation/collapsing
Total number of faults: 8846
Number of faults in collapsed fault list: 3869
Vector generation
#
# VECTORS  FAULTS  FAULT COVER
#          processed
#
#      20      7515      82.92%
#      40      8087      89.39%
#      60      8313      91.74%
#      80      8632      95.29%
#      87      8846      96.06%

# Total number of backtracks: 3000
# Highest backtrack          : 30
# Total number of vectors   : 87

# STAR RESULTS summary
#                               Noncollapsed      Collapsed
# Fault counts:
#   Aborted                    178                85
#   Detected                    8427               3680
#   Untested                    168                60
#                               -----
#   Total of detectable        8773               3825
#
#   Redundant                   10                 6
#   Tied                         63                38
#
# FAULT COVERAGE                96.06 %            96.21 %

```

14.10 Summary

Key terms and concepts: Consider test early during ASIC design otherwise it can become very expensive • Boundary scan • Single stuck-at fault model • Controllability and observability • ATPG using test vectors • BIST with no test vectors

ASIC CONSTRUCTION

15

Key terms and concepts:

- A microelectronic system (or system on a chip) is the town and ASICs (or system blocks) are the buildings
- **System partitioning** corresponds to town planning.
- **Floorplanning** is the architect's job.
- **Placement** is done by the builder.
- **Routing** is done by the electrician.

15.1 Physical Design

Key terms and concepts: Divide and conquer • system partitioning • floorplanning • chip planning
• placement • routing • global routing • detailed routing

15.2 CADTools

Key terms and concepts: **goals** and **objectives** for each physical design step

System partitioning:

- Goal. Partition a system into a number of ASICs.
- Objectives. Minimize the number of external connections between the ASICs. Keep each ASIC smaller than a maximum size.

Floorplanning:

- Goal. Calculate the sizes of all the blocks and assign them locations.
- Objective. Keep the highly connected blocks physically close to each other.

Placement:

- Goal. Assign the interconnect areas and the location of all the logic cells within the flexible blocks.
- Objectives. Minimize the ASIC area and the interconnect density.

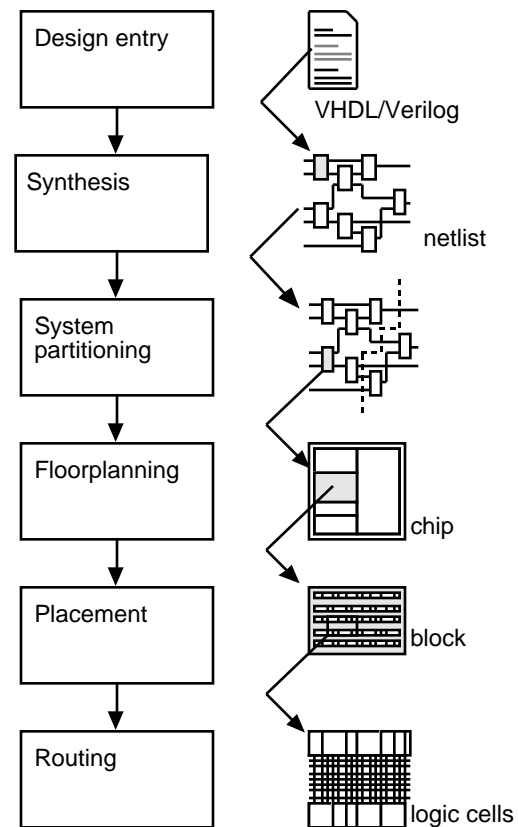
Part of an ASIC design flow showing the system partitioning, floorplanning, placement, and routing steps.

These steps may be performed in a slightly different order, iterated or omitted depending on the type and size of the system and its ASICs.

As the focus shifts from logic to interconnect, floorplanning assumes an increasingly important role.

Each of the steps shown in the figure must be performed and each depends on the previous step.

However, the trend is toward completing these steps in a parallel fashion and iterating, rather than in a sequential manner.



Global routing:

- Goal. Determine the location of all the interconnect.
- Objective. Minimize the total interconnect area used.

Detailed routing:

- Goal. Completely route all the interconnect on the chip.
- Objective. Minimize the total interconnect length used.

15.2.1 Methods and Algorithms

Key terms and concepts: **methods** or **algorithms** are exact or heuristic (algorithm is usually reserved for a method that always gives a solution) • The complexity $O(f(n))$ is important because n is very large • algorithms may be constant, logarithmic, linear, or quadratic in time • many VLSI problems are **NP-complete** • we need **metrics**: a **measurement function** or objective function, a **cost function** or gain function, and possibly **constraints**

15.3 System Partitioning

Key terms and concepts: **partitioning** • we can't do "What is the cheapest way to build my system?" • we can do "How do I split this circuit into pieces that will fit on a chip?"

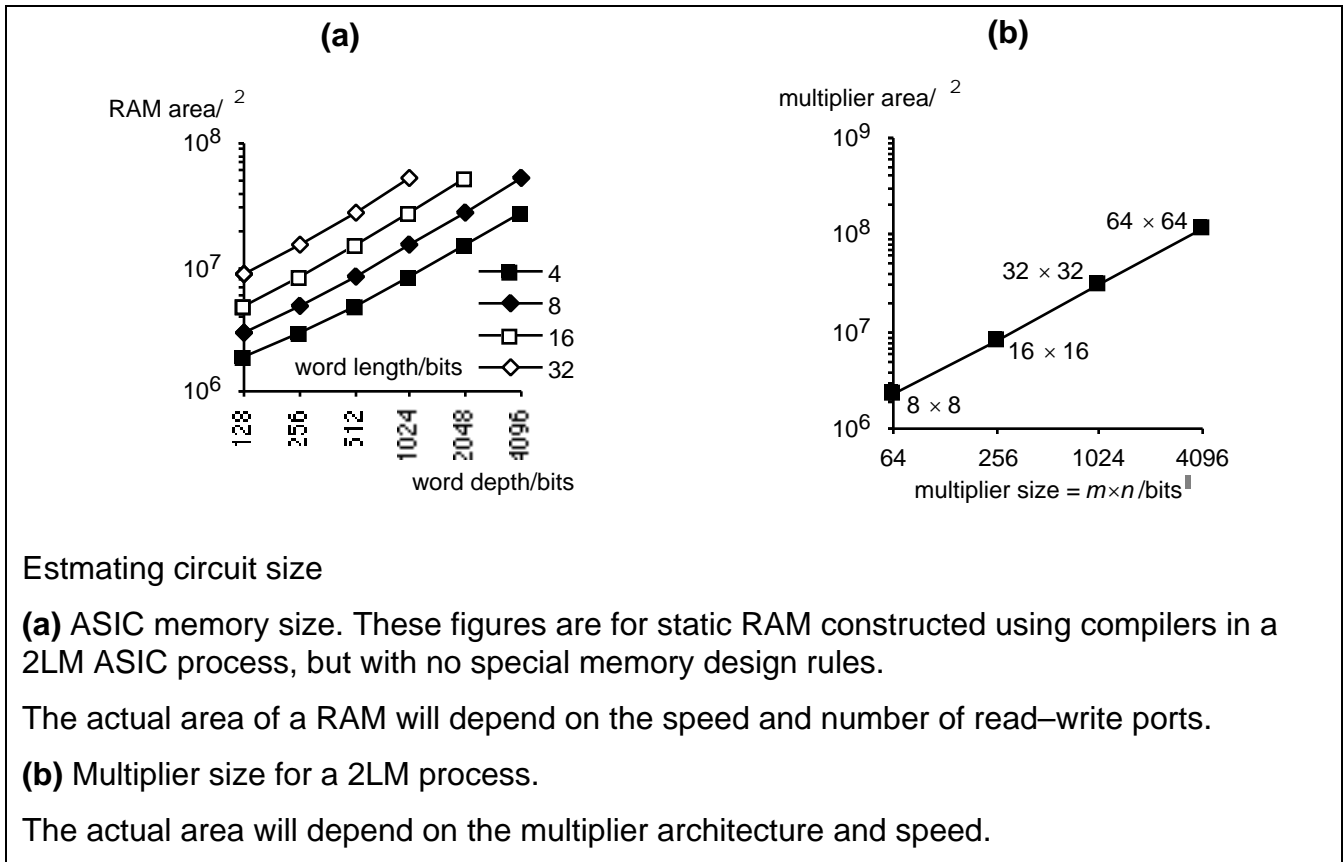
System partitioning for the Sun Microsystems SPARCstation 1					
	SPARCstation 1 ASIC	Gates /k-gate	Pins	Package	Type
1	SPARC IU (integer unit)	20	179	PGA	CBIC
2	SPARC FPU (floating-point unit)	50	144	PGA	FC
3	Cache controller	9	160	PQFP	GA
4	MMU (memory-management unit)	5	120	PQFP	GA
5	Data buffer	3	120	PQFP	GA
6	DMA (direct memory access) controller	9	120	PQFP	GA
7	Video controller/data buffer	4	120	PQFP	GA
8	RAM controller	1	100	PQFP	GA
9	Clock generator	1	44	PLCC	GA

15.4 Estimating ASIC Size

System partitioning for the Sun Microsystems SPARCstation 10

	SPARCstation 10 ASIC	Gates	Pins	Package	Type
1	SuperSPARC Superscalar SPARC	3M-transistors	293	PGA	FC
2	SuperCache cache controller	2M-transistors	369	PGA	FC
3	EMC memory control	40k-gate	299	PGA	GA
4	MSI MBus-SBus interface	40k-gate	223	PGA	GA
5	DMA2 Ethernet, SCSI, parallel port	30k-gate	160	PQFP	GA
6	SEC SBus to 8-bit bus	20k-gate	160	PQFP	GA
7	DBRI dual ISDN interface	72k-gate	132	PQFP	GA
8	MMCodec stereo codec	32k-gate	44	PLCC	FC

Some useful numbers for ASIC estimates, normalized to a 1 μm technology			
Parameter	Typical value	Comment	Scaling
Lambda,	0.5 μm =0.5 (minimum feature size)	In a 1 μm technology, 0.5 μm .	NA
Effective gate length	0.25 to 1.0 μm	Less than drawn gate length, usually by about 10 percent.	
I/O-pad width (pitch)	5 to 10mil =125 to 250 μm	For a 1 μm technology, 2LM (=0.5 μm). Scales less than linearly with .	
I/O-pad height	15 to 20mil =375 to 500 μm	For a 1 μm technology, 2LM (=0.5 μm). Scales approximately linearly with .	
Large die	1000 mil/side, 10 ⁶ mil ²	Approximately constant	1
Small die	100 mil/side, 10 ⁴ mil ²	Approximately constant	1
Standard-cell density	1.5 $\times 10^{-3}$ gate/ μm^2 =1.0 gate/mil ²	For 1 μm , 2LM, library = 4 $\times 10^{-4}$ gate/ ² (independent of scaling).	1/ ²
Standard-cell density	8 $\times 10^{-3}$ gate/ μm^2 = 5.0 gate/mil ²	For 0.5 μm , 3LM, library = 5 $\times 10^{-4}$ gate/ ² (independent of scaling).	1/ ²
Gate-array utilization	60 to 80% 80 to 90%	For 2LM, approximately constant For 3LM, approximately constant	1 1
Gate-array density	(0.8 to 0.9) \times standard cell density	For the same process as standard cells	1
Standard-cell routing factor=(cell area+route area)/cell area	1.5 to 2.5 (2LM) 1.0 to 2.0 (3LM)	Approximately constant	1
Package cost	\$0.01/pin, "penny per pin"	Varies widely, figure is for low-cost plastic package, approximately constant	1
Wafer cost	\$1k to \$5k average \$2k	Varies widely, figure is for a mature, 2LM CMOS process, approximately constant	1



15.5 Power Dissipation

Key terms and concepts: dynamic (switching current and short-circuit current) and static (leakage current and subthreshold current) power dissipation

15.5.1 Switching Current

Key terms and concepts: $I = C(dV/dt)$ • power dissipation = $0.5 CV_{DD}^2 = IV = CV(dV/dt)$ for one-half the period of the input, $t=1/(2f)$ • total power = $P_1 = fCV_{DD}^2$ • estimate power by counting nodes that toggle

15.5.2 Short-Circuit Current

Key terms and concepts: $P_2 = (1/12) f t_{rf}(V_{DD} - 2V_{tn})$ • short-circuit current is typically less than 20 percent of the switching current

15.5.3 Subthreshold and Leakage Current

Key terms and concepts: **subthreshold current** is normally less than $5\text{pA}\mu\text{m}^{-1}$ of gate width • subthreshold current for 10 million transistors (each $10\mu\text{m}$ wide) is 0.1mA • subthreshold current does not scale • it takes about 120mV to reduce subthreshold current by a factor of 10 • if $V_t = 0.36\text{V}$, at $V_{GS}=0\text{V}$ we can only reduce I_{DS} to 0.001 times its value at $V_{GS}=V_t$ • leakage current • field transistors • quiescent leakage current, I_{DDQ} • **IDDQ test**

15.6 FPGA Partitioning

15.6.1 ATM Simulator

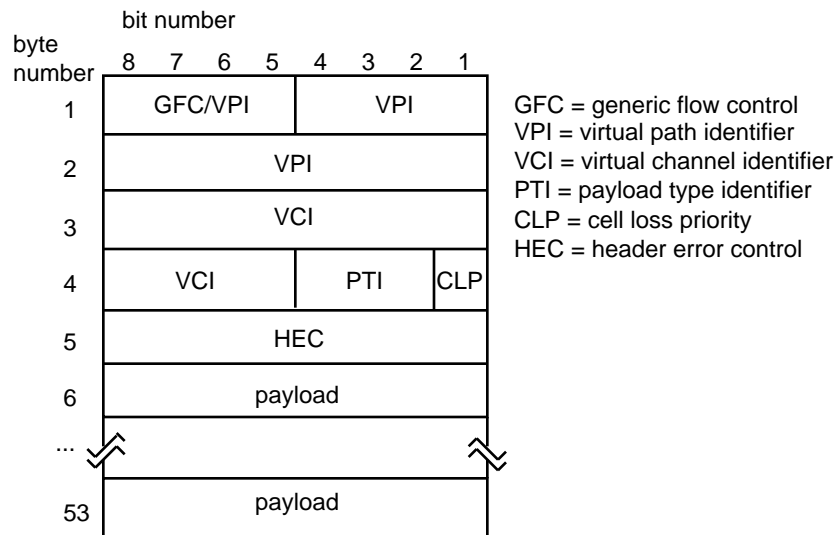
Partitioning of the ATM board using Lattice Logic ispLSI 1048 FPGAs. Each FPGA contains 48 generic logic blocks (GLBs)

Chip #	Size	Chip #	Size
1	42 GLBs	7	36 GLBs
2	64k-bit × 8 SRAM	8	22 GLBs
3	38 GLBs	9	256k-bit × 16 SRAM
4	38 GLBs	10	43 GLBs
5	42 GLBs	11	40 GLBs
6	64k-bit × 16 SRAM	12	30 GLBs

15.6.2 Automatic Partitioning with FPGAs

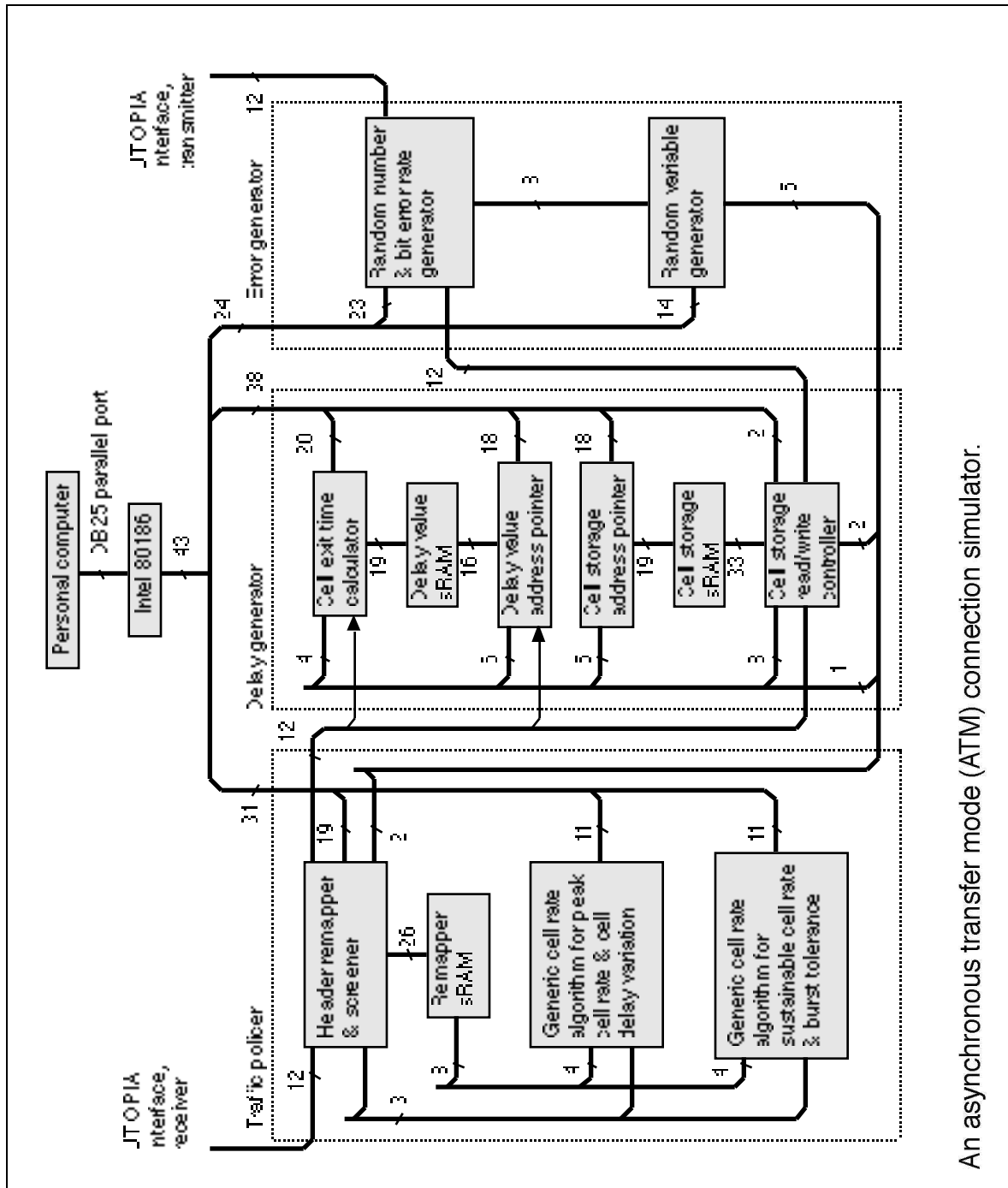
Key terms and concepts: In Altera AHDL you can direct the partitioner to automatically partition logic into chips within the same family, using the AUTO keyword:

```
DEVICE top_level IS AUTO; % let the partitioner assign logic
```



The asynchronous transfer mode (ATM) cell format.

The ATM protocol uses 53-byte cells or packets of information with a data payload and header information for routing and error control.



An asynchronous transfer mode (ATM) connection simulator.

15.7 Partitioning Methods

Key terms and concepts: Examples of goals: A maximum size for each ASIC • A maximum number of ASICs • A maximum number of connections for each ASIC • A maximum number of total connections between all ASICs

15.7.1 Measuring Connectivity

Key terms and concepts: a network has circuit modules (logic cells) and terminals (connectors or pins) • modelled by a graph with vertexes (logic cells) connected by edges (electrical connections, nets or signals) • cutset • net cutset • edge cutset (for the graph) • external connections • internal connections • net cuts • edge cuts

15.7.2 A Simple Partitioning Example

Key terms and concepts: two types of **network partitioning**: **constructive partitioning** and **iterative partitioning improvement**

15.7.3 Constructive Partitioning

Key terms and concepts: **seed growth** or **cluster growth** uses a **seed cell** and forms **clusters** or **cliques** • a useful starting point

15.7.4 Iterative Partitioning Improvement

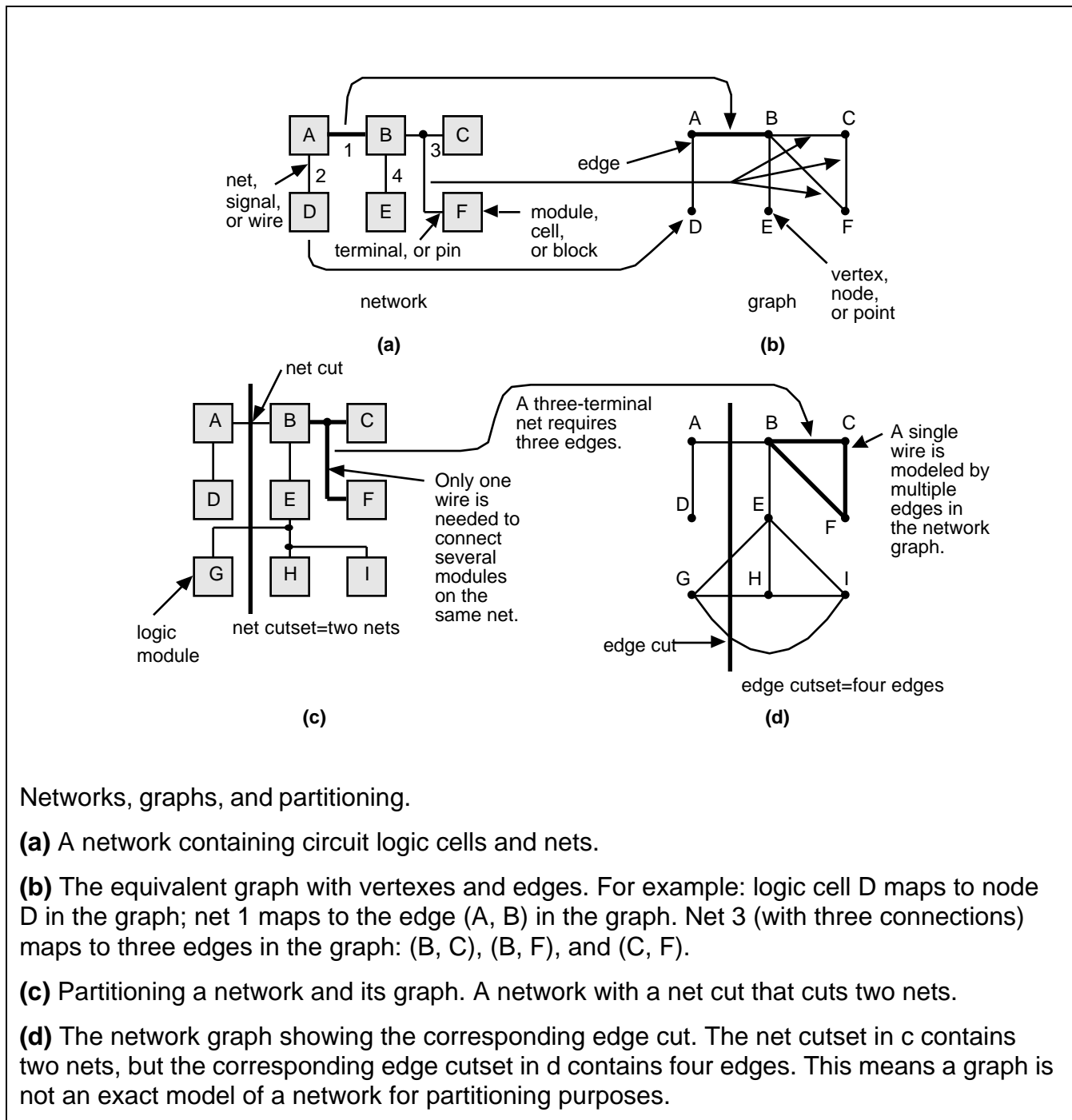
Key terms and concepts: interchange (swap two) and group (swap many) migration • greedy algorithms find a local minimum • group migration algorithms such as the Kernighan–Lin algorithm (basis of min-cut methods) can do better

15.7.5 The Kernighan–Lin Algorithm

Key terms and concepts: a cost matrix plus connectivity matrix models system • measure is the cut cost, or cut weight • careful to distinguish external edge cost and internal edge cost • net-cut partitioning and edge-cut partitioning • hypergraphs with stars, and hyperedges model connections better than edges • the Fiduccia–Mattheyses algorithm uses linked lists to reduce $O(K-L)$ algorithm) and is very widely used • base logic cell • balance • critical net

15.7.6 The Ratio-Cut Algorithm

Key terms and concepts: ratio-cut algorithm • ratio • set cardinality • ratio cut



Networks, graphs, and partitioning.

(a) A network containing circuit logic cells and nets.

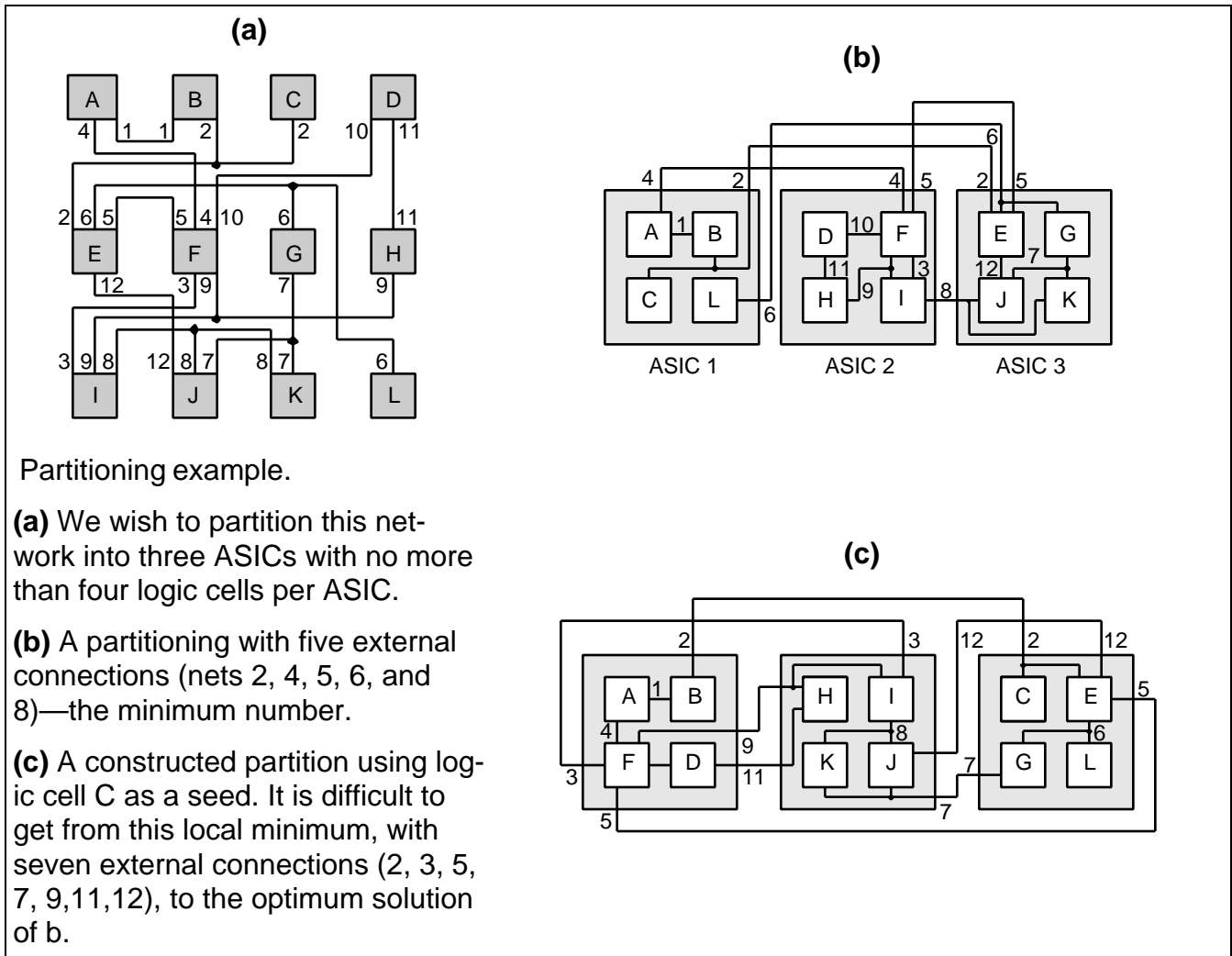
(b) The equivalent graph with vertexes and edges. For example: logic cell D maps to node D in the graph; net 1 maps to the edge (A, B) in the graph. Net 3 (with three connections) maps to three edges in the graph: (B, C), (B, F), and (C, F).

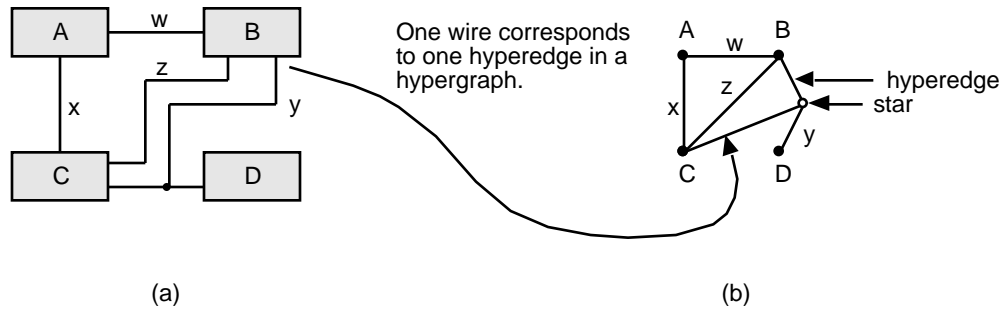
(c) Partitioning a network and its graph. A network with a net cut that cuts two nets.

(d) The network graph showing the corresponding edge cut. The net cutset in c contains two nets, but the corresponding edge cutset in d contains four edges. This means a graph is not an exact model of a network for partitioning purposes.

15.7.7 The Look-ahead Algorithm

Key terms and concepts: gain vector • look-ahead algorithm



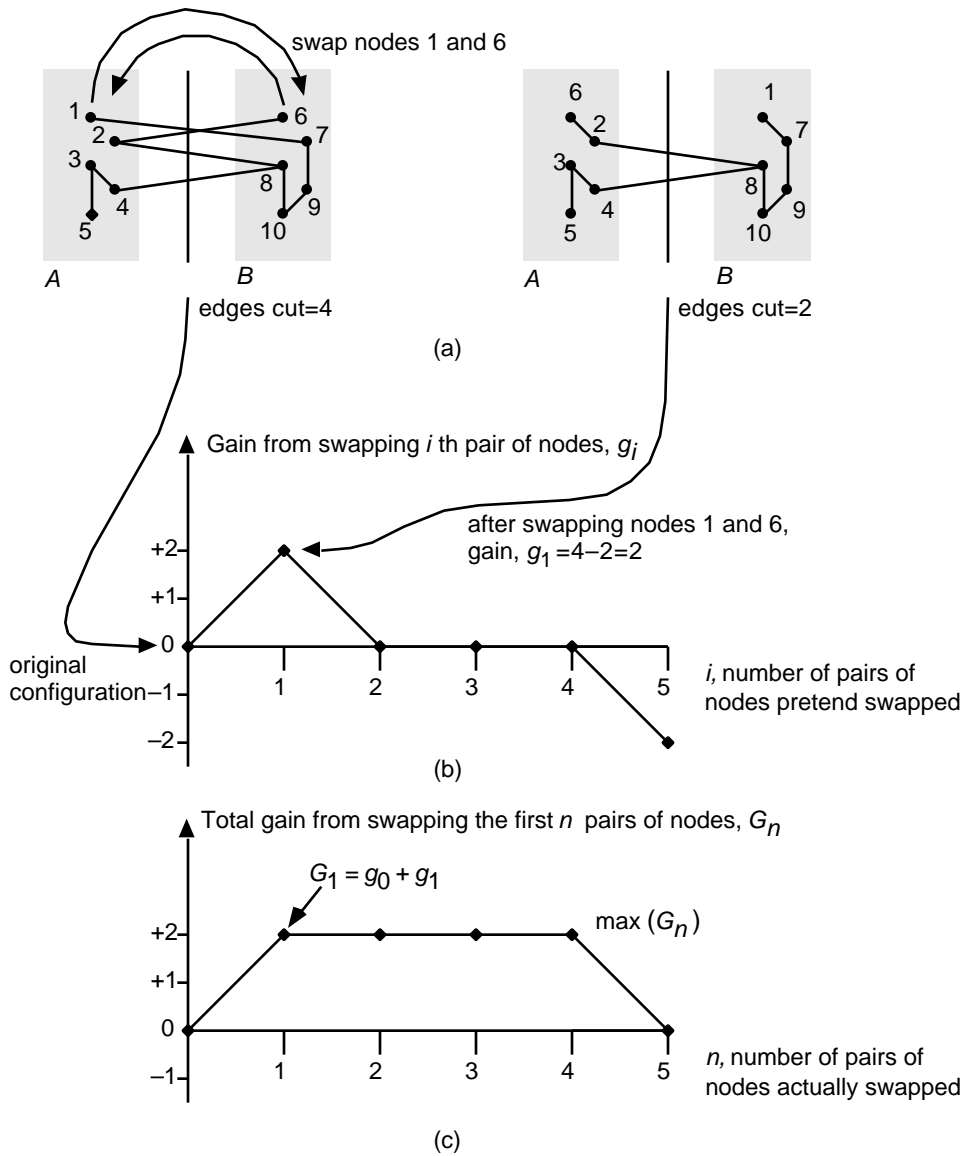


A hypergraph.

(a) The network contains a net y with three terminals.

(b) In the network hypergraph we can model net y by a single hyperedge (B, C, D) and a star node.

Now there is a direct correspondence between wires or nets in the network and hyperedges in the graph.

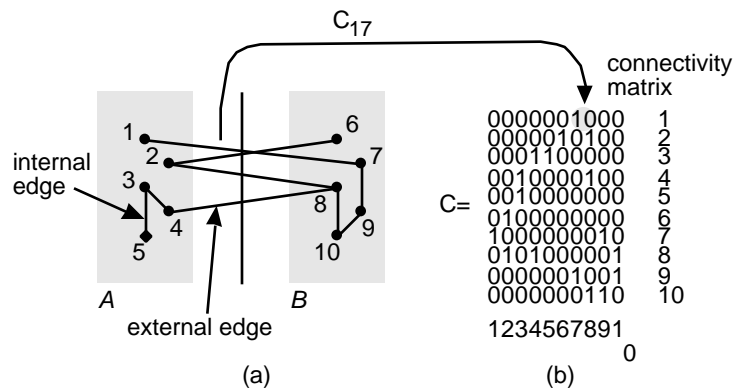


Partitioning a graph using the Kernighan–Lin algorithm.

(a) Shows how swapping node 1 of partition A with node 6 of partition B results in a gain of $g=1$.

(b) A graph of the gain resulting from swapping pairs of nodes.

(c) The total gain is equal to the sum of the gains obtained at each step.



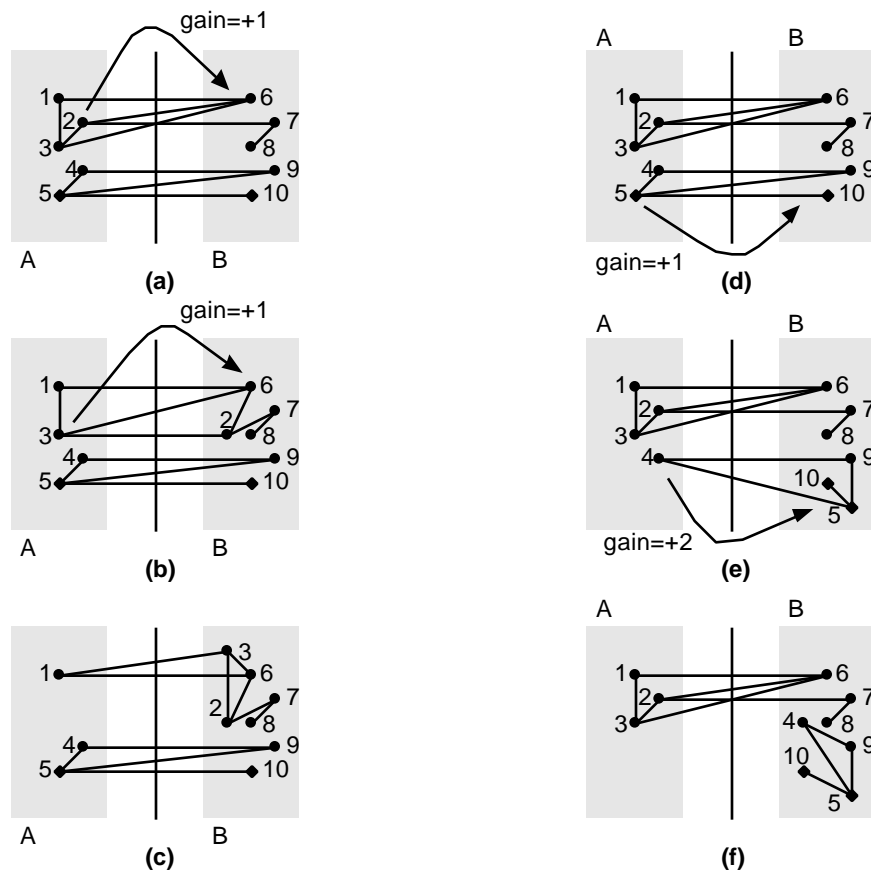
Terms used by the Kernighan–Lin partitioning algorithm.

(a) An example network graph.

(b) The connectivity matrix, C ; the column and rows are labeled to help you see how the matrix entries correspond to the node numbers in the graph.

For example, C_{17} (column 1, row 7) equals 1 because nodes 1 and 7 are connected.

In this example all edges have an equal weight of 1, but in general the edges may have different weights.



An example of network partitioning that shows the need to look ahead when selecting logic cells to be moved between partitions.

Partitionings **(a)**, **(b)**, and **(c)** show one sequence of moves, partitionings **(d)**, **(e)**, and **(f)** show a second sequence.

The partitioning in **(a)** can be improved by moving node 2 from A to B with a gain of 1.

The result of this move is shown in **(b)**.

This partitioning can be improved by moving node 3 to B, again with a gain of 1.

The partitioning shown in **(d)** is the same as **(a)**.

We can move node 5 to B with a gain of 1 as shown in **(e)**, but now we can move node 4 to B with a gain of 2.

15.7.8 Simulated Annealing

Key terms and concepts: simulated-annealing algorithm uses an energy function as a measure

- probability of accepting a move is $\exp(-E/T)$
- E is an increase in energy function
- T corresponds to temperature
- we hill climb to get out of a local minimum
- cooling schedule • $T_{i+1} = T_i$
- good results at the expense of long run times
- Xilinx used simulated annealing in one version of their tools

15.7.9 Other Partitioning Objectives

Key terms and concepts: timing, power, technology, cost and test constraints • many of these are hard to measure and not well handled by current tools

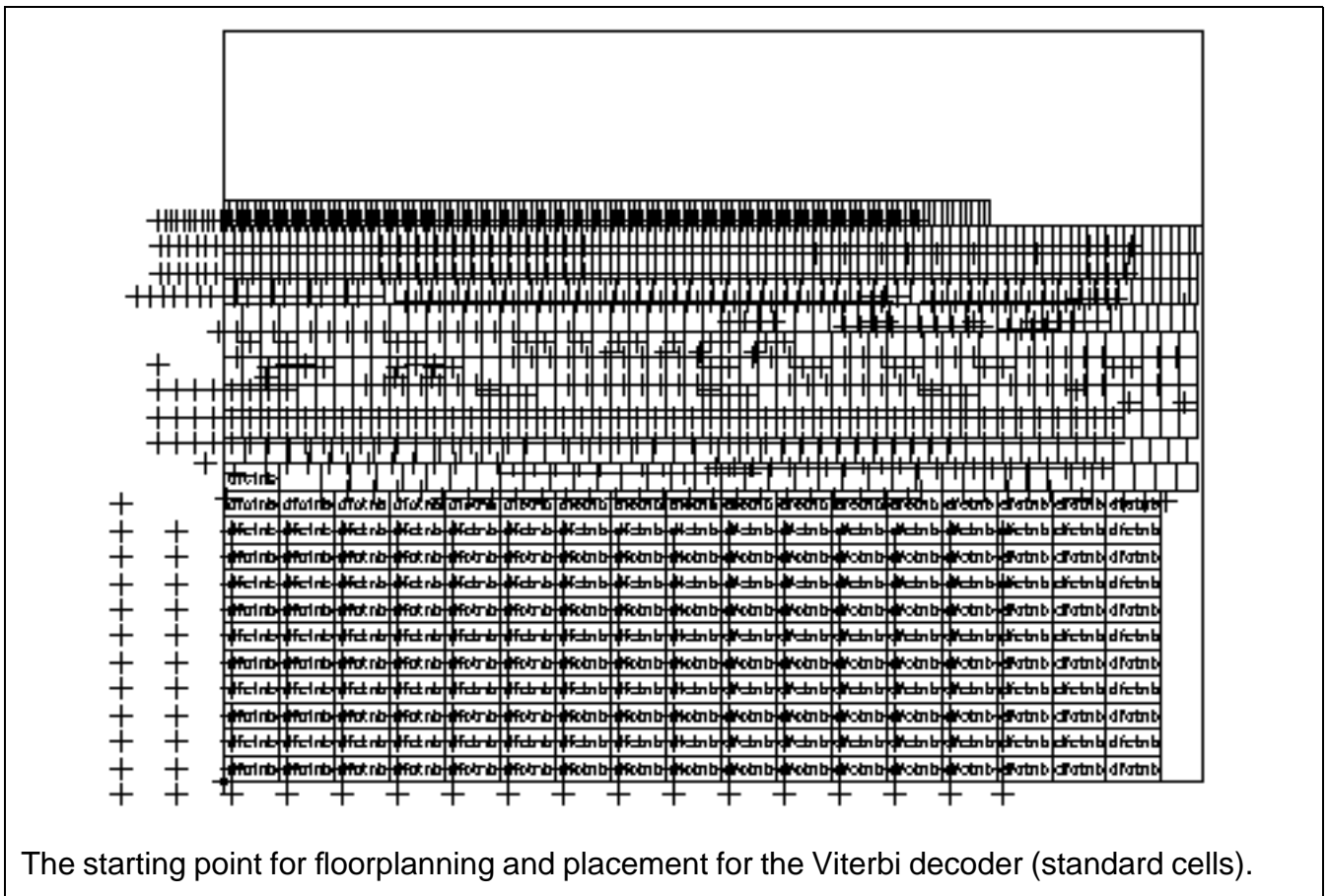
15.8 Summary

Key terms and concepts: The construction or physical design of a microelectronics system is a very large and complex problem. To solve the problem we divide it into several steps: **system partitioning, floorplanning, placement, and routing**. To solve each of these smaller problems we need **goals** and **objectives**, **measurement metrics**, as well as **algorithms** and **methods**

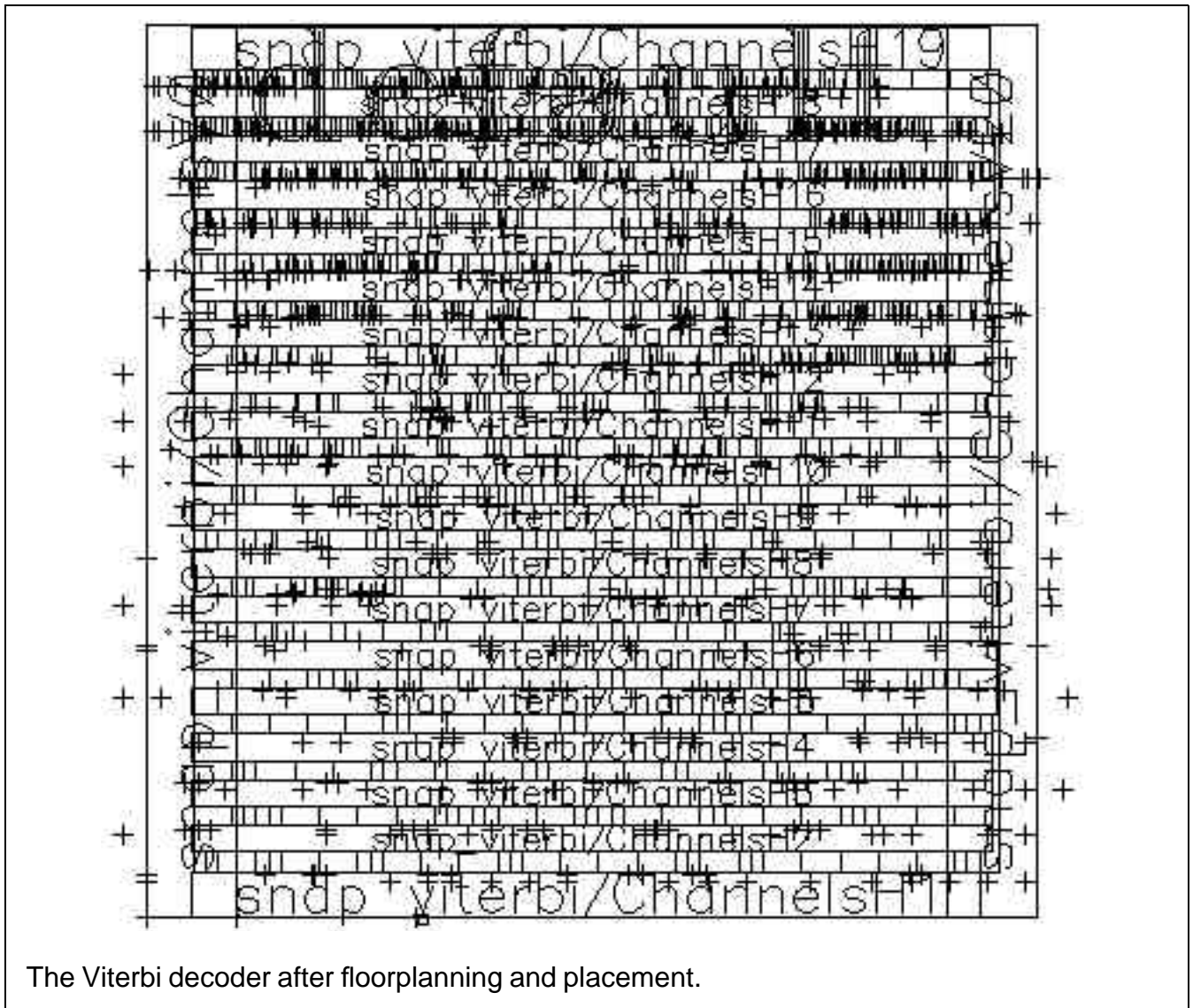
- The goals and objectives of partitioning
- Partitioning as an art not a science
- The simple nature of the algorithms necessary for VLSI-sized problems
- The random nature of the algorithms we use
- The controls for the algorithms used in ASIC design

FLOORPLANNING AND PLACEMENT

Key terms and concepts: The input to floorplanning is the output of system partitioning and design entry—a netlist. The output of the placement step is a set of directions for the routing tools.



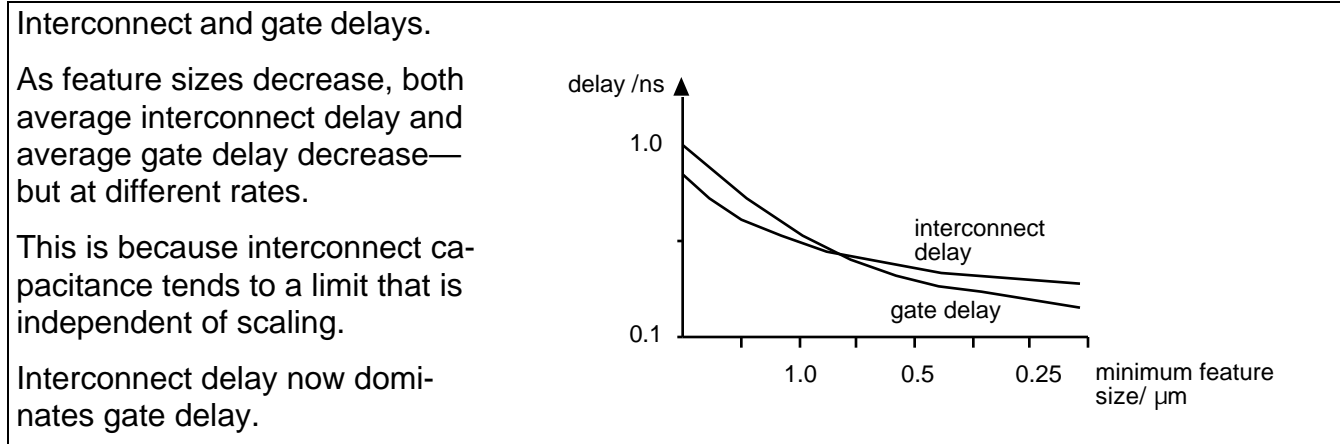
The starting point for floorplanning and placement for the Viterbi decoder (standard cells).



The Viterbi decoder after floorplanning and placement.

16.1 Floorplanning

Key terms and concepts: Interconnect and gate delay both decrease with feature size—but at different rates • Interconnect capacitance bottoms out at 2pFcm^{-1} for a minimum-width wire, but gate delay continues to decrease • Floorplanning predicts interconnect delay by estimating interconnect length



16.1.1 Floorplanning Goals and Objectives

Key terms and concepts: Floorplanning is a mapping between the **logical description** (the **netlist**) and the **physical description** (the **floorplan**).

Goals of floorplanning:

- arrange the blocks on a chip,
- decide the location of the I/O pads,
- decide the location and number of the power pads,
- decide the type of power distribution, and
- decide the location and type of clock distribution.

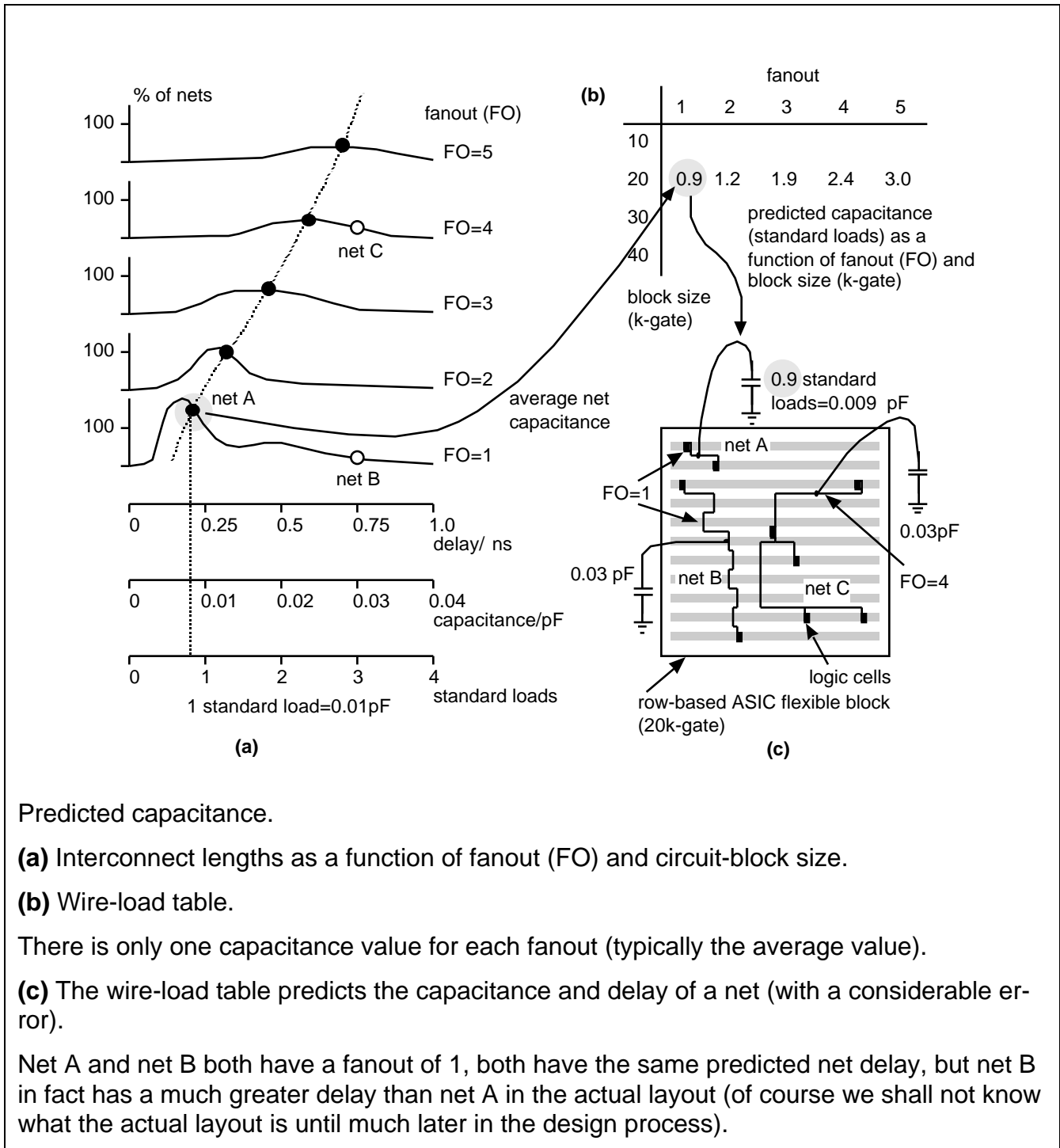
Objectives of floorplanning are:

- to minimize the chip area, and
- minimize delay.

16.1.2 Measurement of Delay in Floorplanning

Key terms and concepts: To predict performance before we complete routing we need to answer “How long does it take to get from Russia to China?” • In floorplanning we may even move Russia and China • We don’t yet know the **parasitics** of the **interconnect capacitance** • We

know only the **fanout (FO)** of a net and the size of the block • We estimate interconnect length from **predicted-capacitance tables (wire-load tables)**



Predicted capacitance.

(a) Interconnect lengths as a function of fanout (FO) and circuit-block size.

(b) Wire-load table.

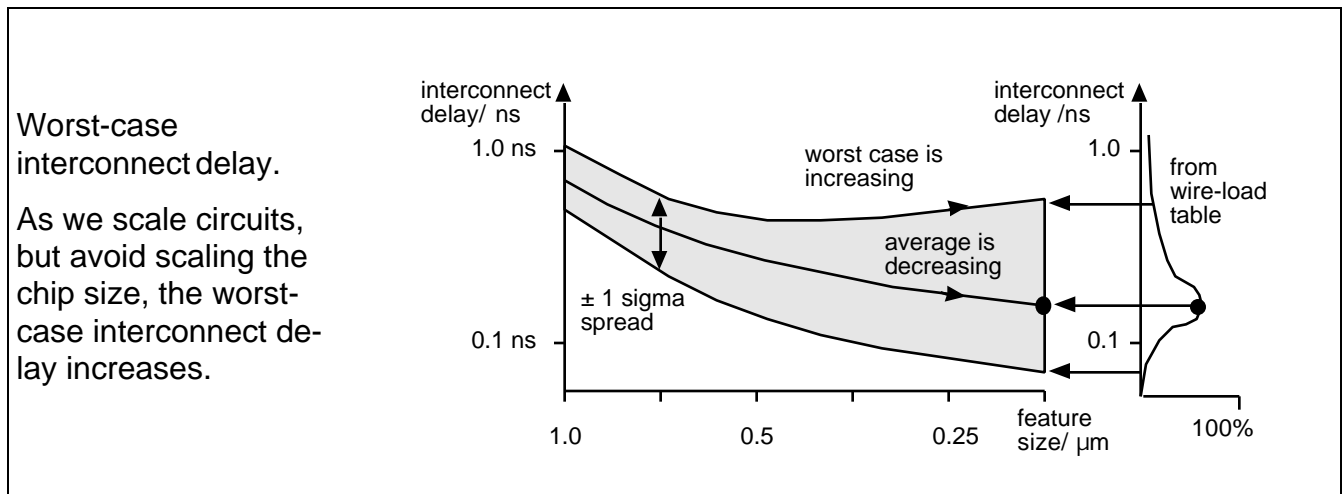
There is only one capacitance value for each fanout (typically the average value).

(c) The wire-load table predicts the capacitance and delay of a net (with a considerable error).

Net A and net B both have a fanout of 1, both have the same predicted net delay, but net B in fact has a much greater delay than net A in the actual layout (of course we shall not know what the actual layout is until much later in the design process).

A wire-load table showing average interconnect lengths (mm).

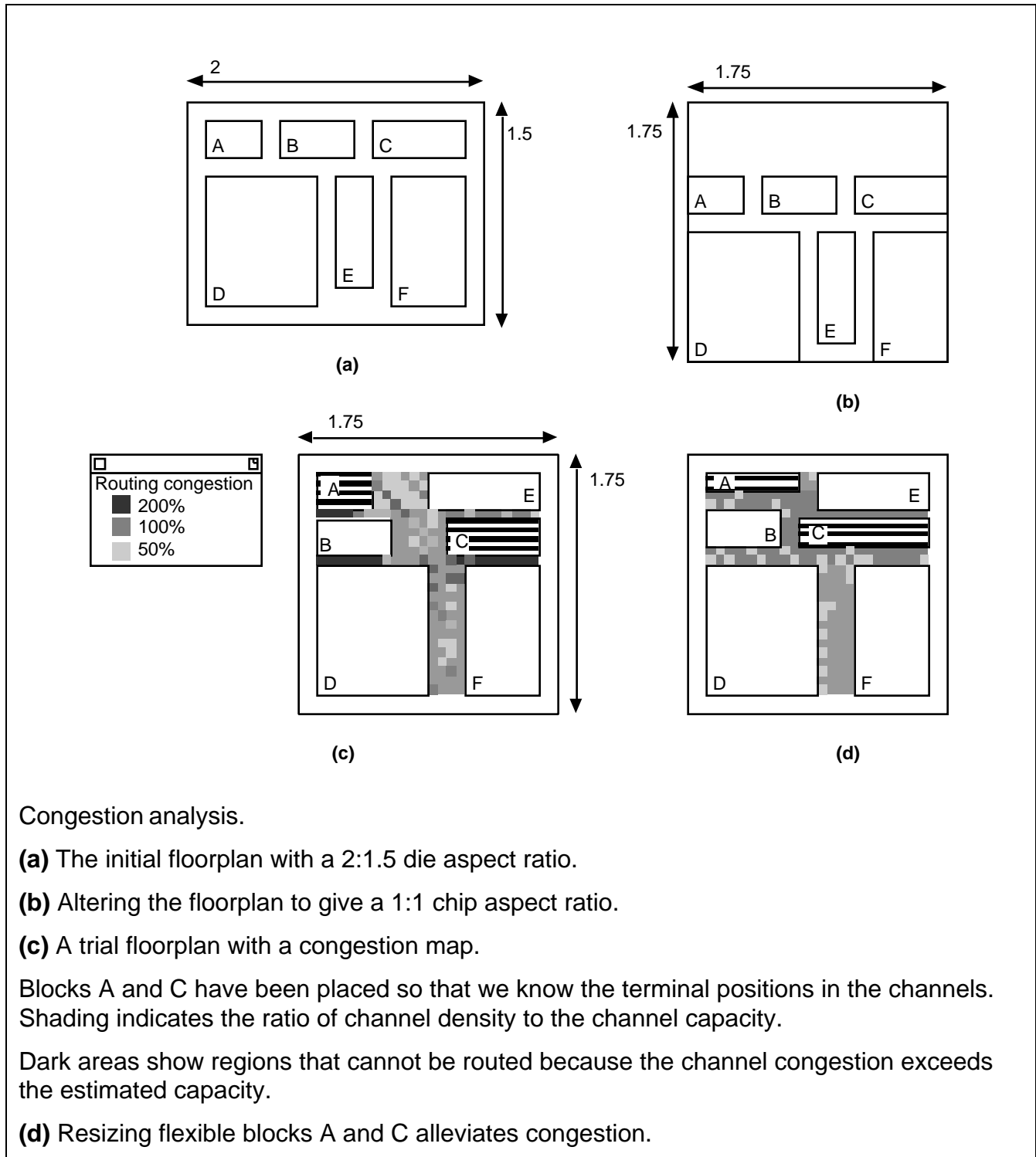
Array (available gates)	Chip size (mm)	Fanout		
		1	2	4
3k	3.45	0.56	0.85	1.46
11k	5.11	0.84	1.34	2.25
105k	12.50	1.75	2.70	4.92



16.1.3 Floorplanning Tools

Key terms and concepts: we start with a **random floorplan** generated by a floorplanning tool • **flexible blocks** and **fixed blocks** • **seeding** • **seed cells** • **wildcard symbol** • **hard seed** • **soft seed** • **seed connectors** • **rat's nest** • **bundles** • **flight lines** • **congestion** • **aspect ratio** • **die**

cavity • congestion map • routability • interconnect channels • channel capacity • channel density



Congestion analysis.

(a) The initial floorplan with a 2:1.5 die aspect ratio.

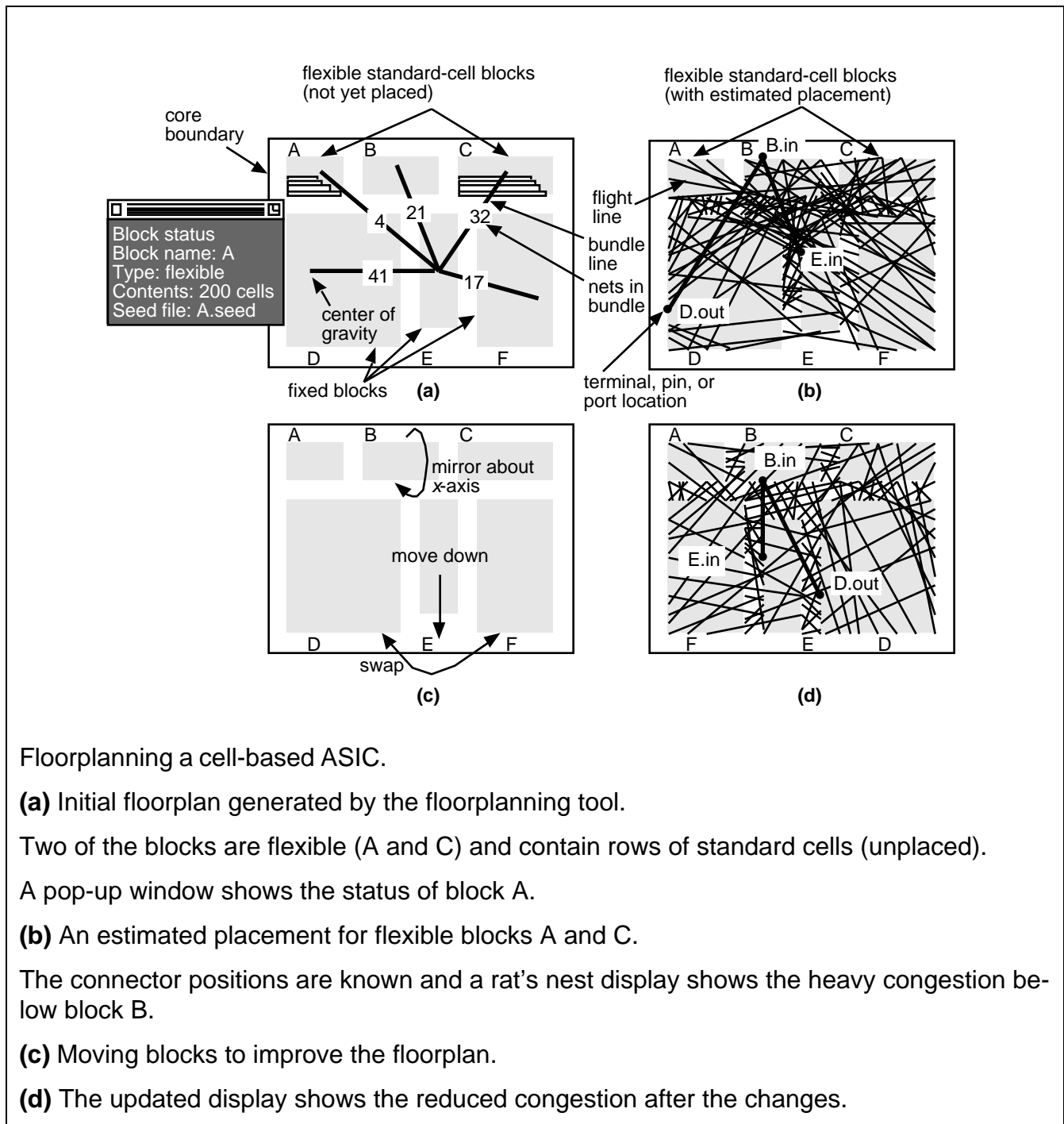
(b) Altering the floorplan to give a 1:1 chip aspect ratio.

(c) A trial floorplan with a congestion map.

Blocks A and C have been placed so that we know the terminal positions in the channels. Shading indicates the ratio of channel density to the channel capacity.

Dark areas show regions that cannot be routed because the channel congestion exceeds the estimated capacity.

(d) Resizing flexible blocks A and C alleviates congestion.



Floorplanning a cell-based ASIC.

(a) Initial floorplan generated by the floorplanning tool.

Two of the blocks are flexible (A and C) and contain rows of standard cells (unplaced).

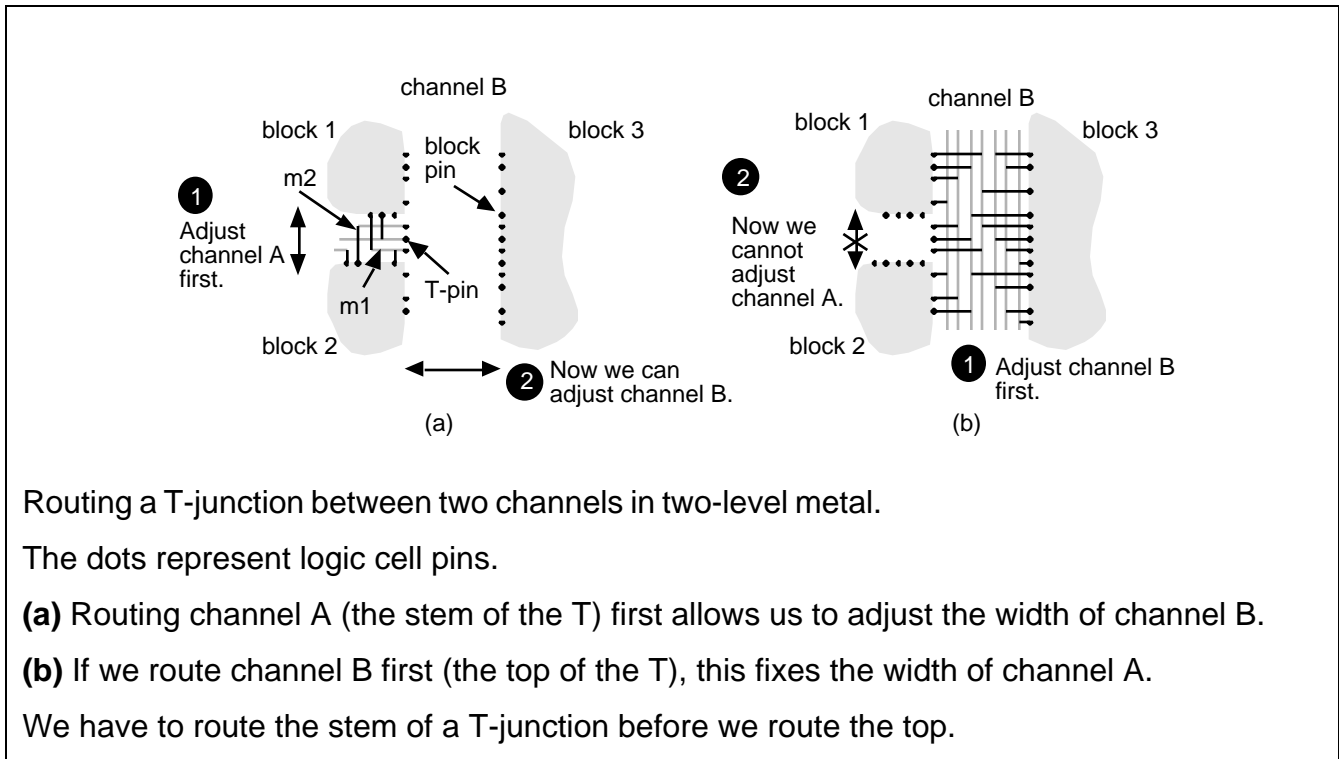
A pop-up window shows the status of block A.

(b) An estimated placement for flexible blocks A and C.

The connector positions are known and a rat's nest display shows the heavy congestion below block B.

(c) Moving blocks to improve the floorplan.

(d) The updated display shows the reduced congestion after the changes.



Routing a T-junction between two channels in two-level metal.

The dots represent logic cell pins.

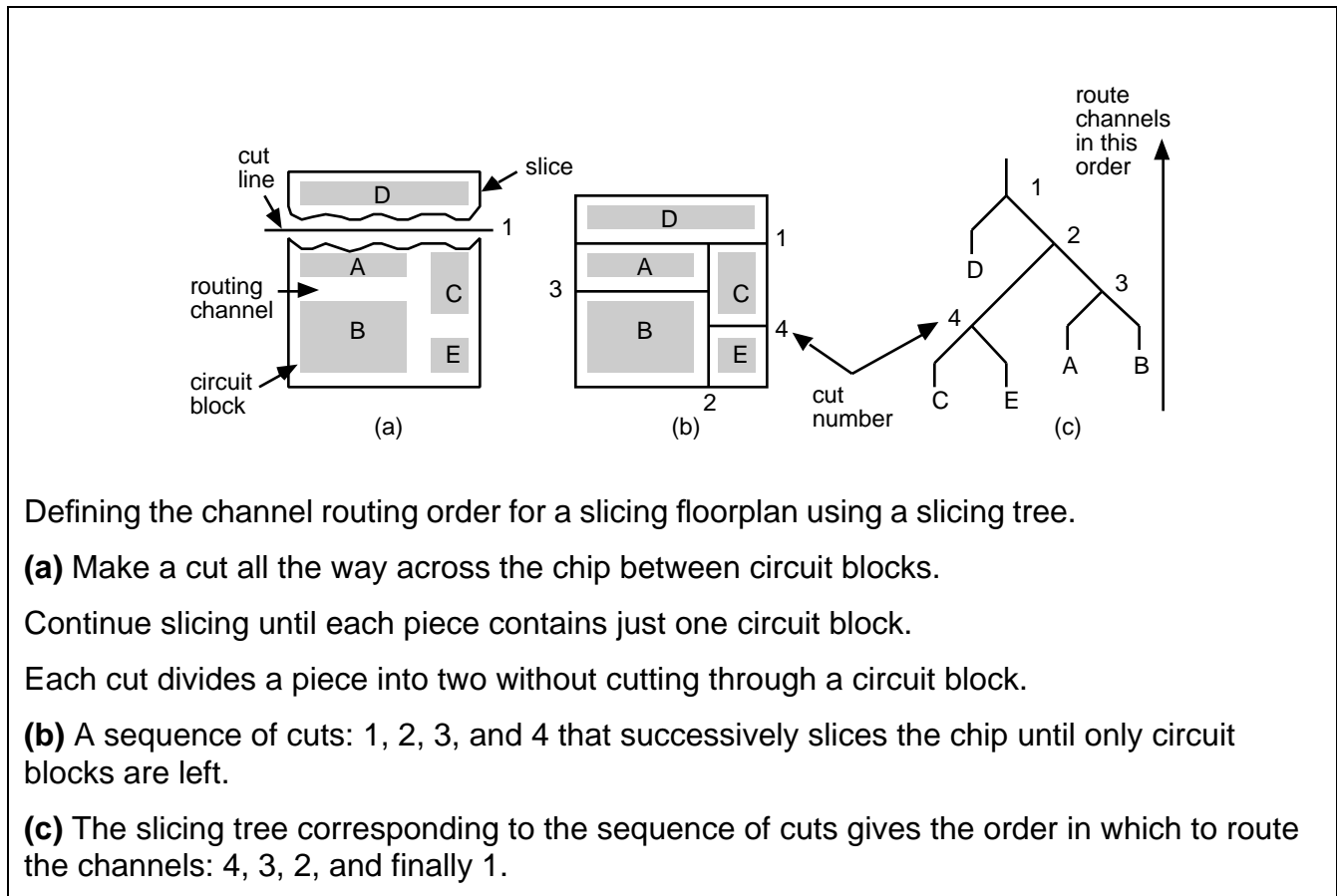
(a) Routing channel A (the stem of the T) first allows us to adjust the width of channel B.

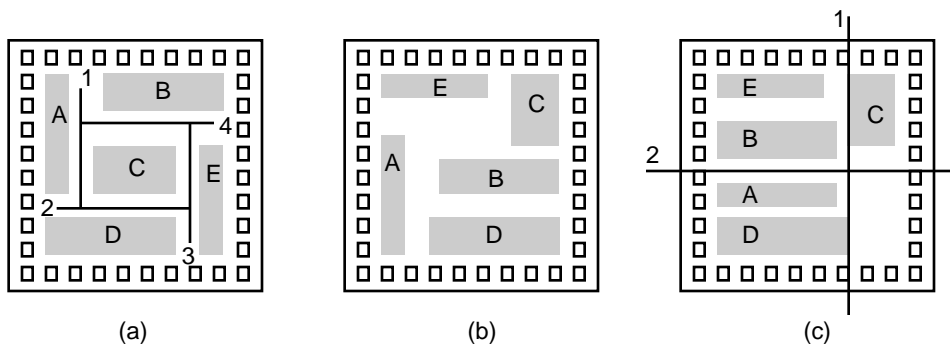
(b) If we route channel B first (the top of the T), this fixes the width of channel A.

We have to route the stem of a T-junction before we route the top.

16.1.4 Channel Definition

Key terms and concepts: **channel definition** or **channel allocation** • **channel ordering** • **slicing floorplan** • **cyclic constraint** • **switch box** • **merge** • **selective flattening** • **routing order**



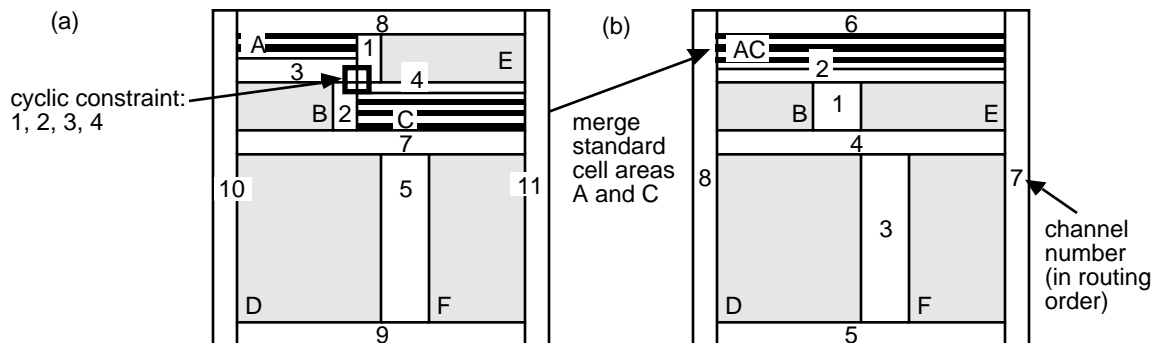


Cyclic constraints.

(a) A nonslicing floorplan with a cyclic constraint that prevents channel routing.

(b) In this case it is difficult to find a slicing floorplan without increasing the chip area.

(c) This floorplan may be sliced (with initial cuts 1 or 2) and has no cyclic constraints, but it is inefficient in area use and will be very difficult to route.



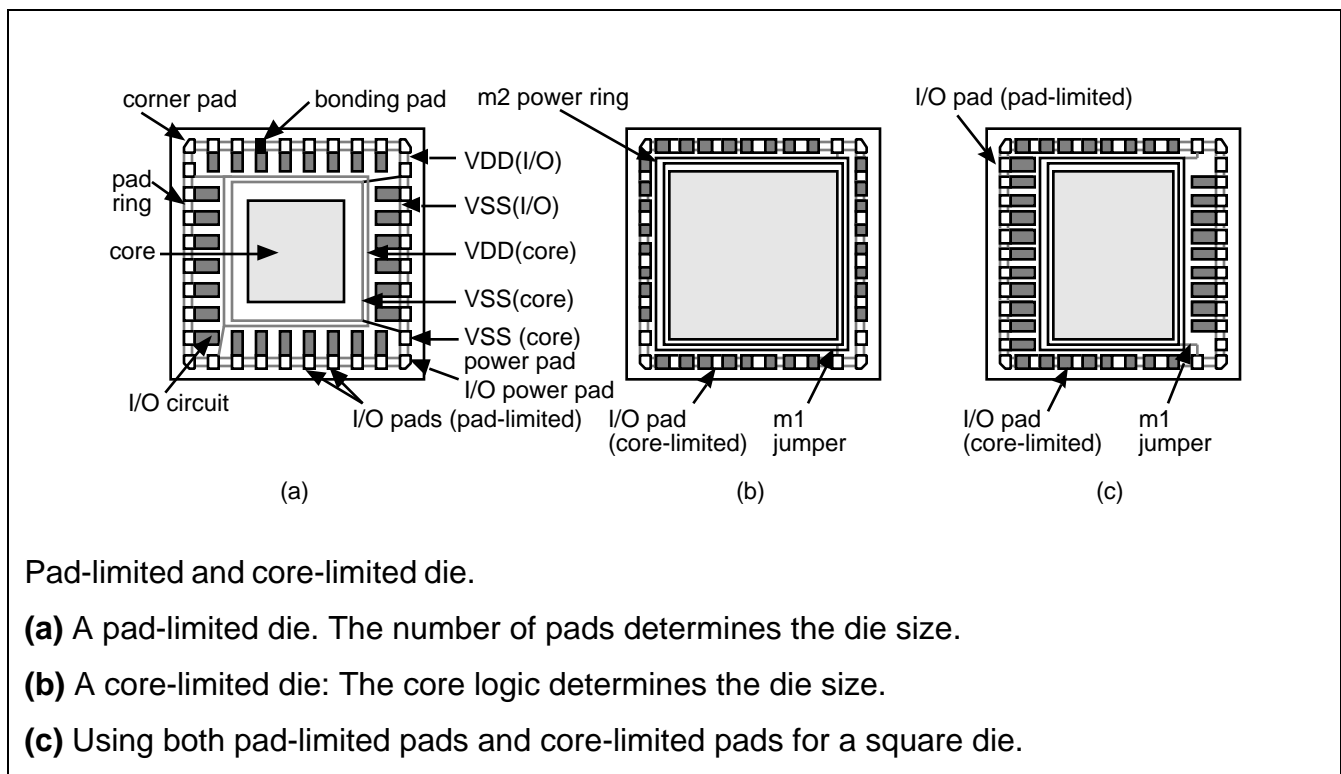
Channel definition and ordering.

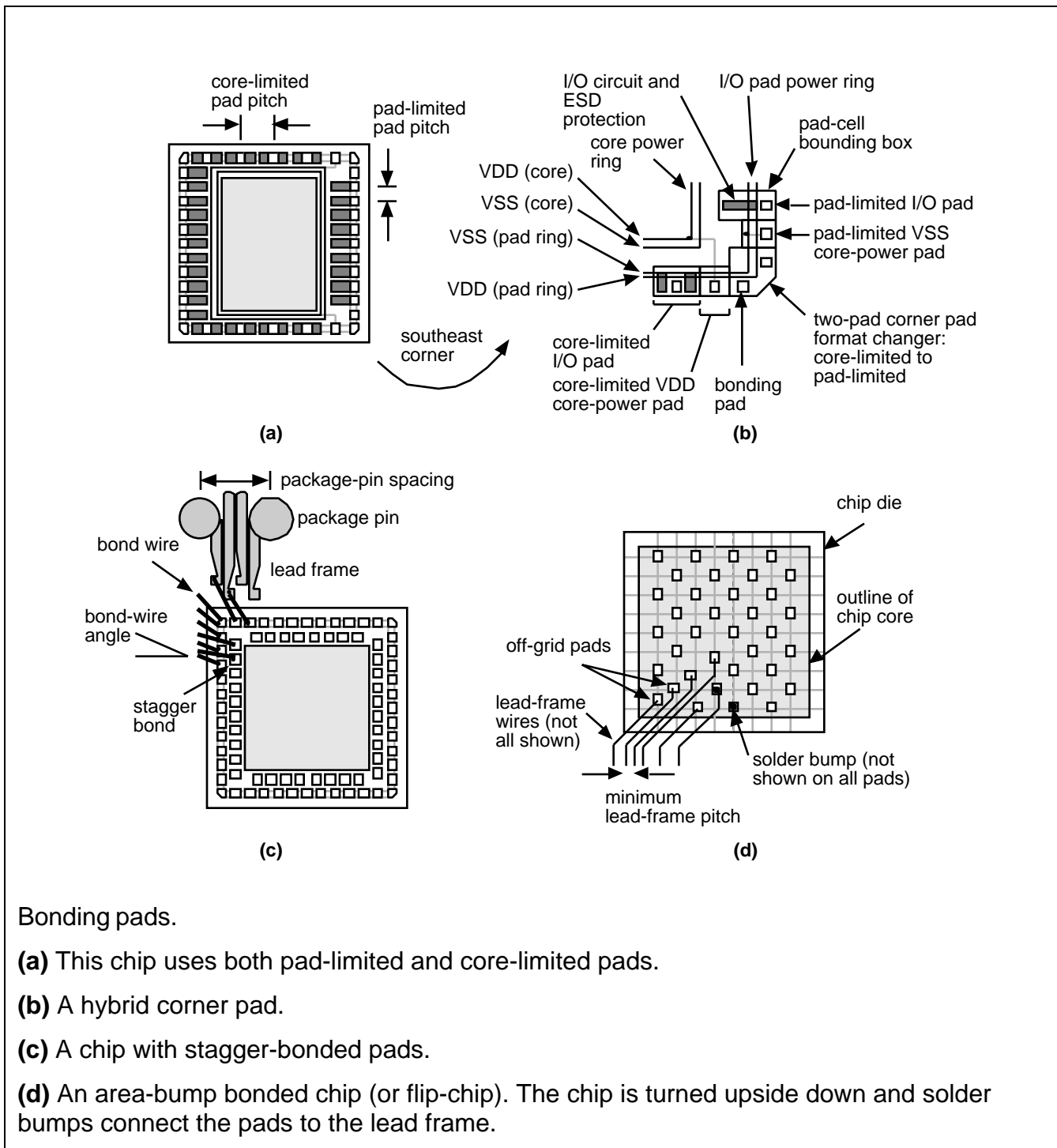
(a) We can eliminate the cyclic constraint by merging the blocks A and C.

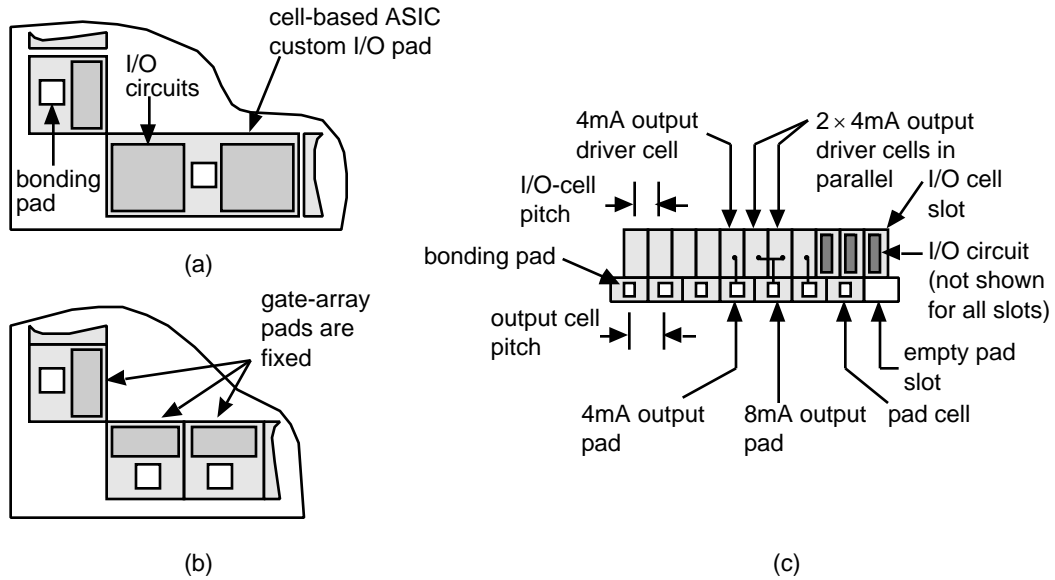
(b) A slicing structure.

16.1.5 I/O and Power Planning

Key terms and concepts: die • chip carrier • package • bonding • pads • lead frame • package pins • core • pad ring • pad-limited die • core-limited die • pad-limited pads • core-limited pads • power pads • power buses (or power rails) • power ring • dirty power • clean power • electrostatic discharge (ESD) • chip cavity • substrate connection • down bond (or drop bond) • pad seed • double bond • multiple-signal pad • oscillator pad • clock pad • corner pad • edge pads • two-pad corner cell • bond-wire angle design rules • simultaneously switching outputs (SSOs) • pad mapping • logical pad • physical pad • pad library • pad-format changer or hybrid corner pad • global power nets • mixed power supplies • multiple power supplies • stagger-bond • area-bump • ball-grid array (BGA) • pad slot (or pad site) • I/O-cell pitch • pad pitch • channel spine • preferred layer • preferred direction





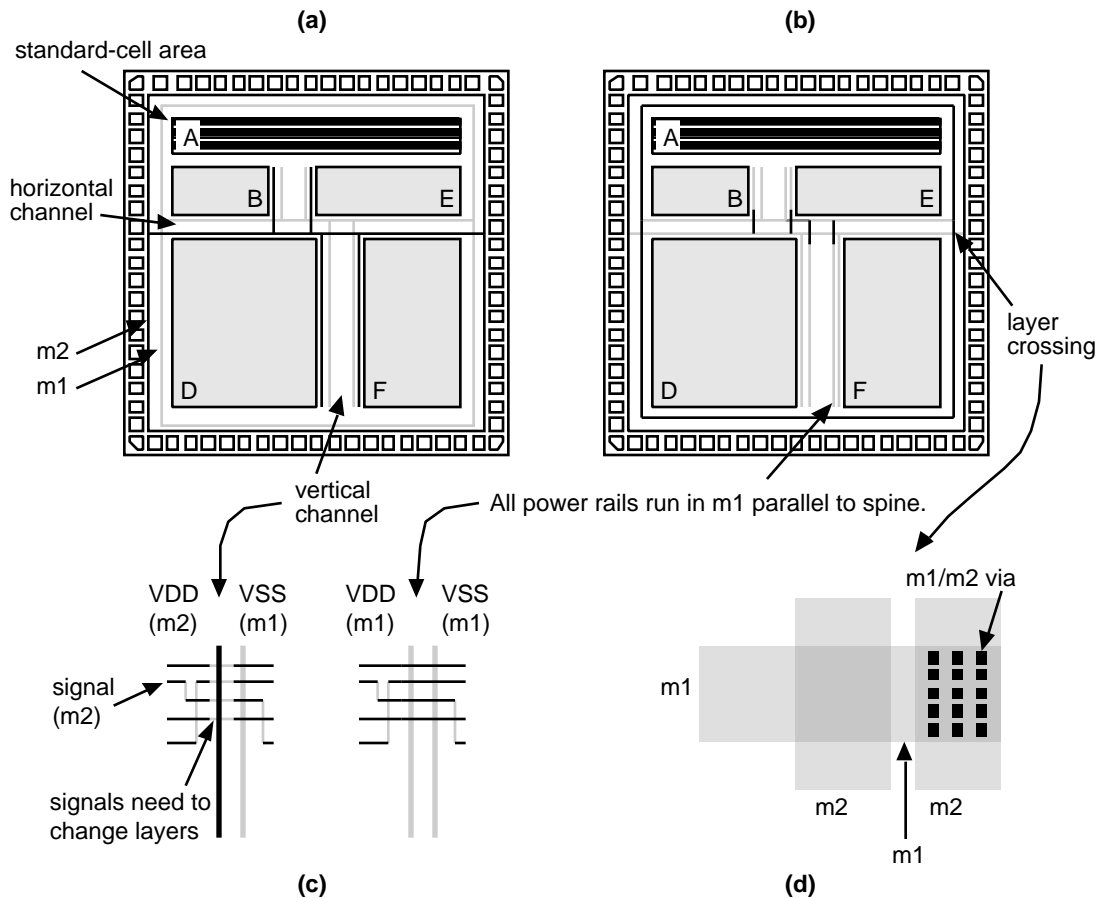


Gate-array I/O pads.

(a) Cell-based ASICs may contain pad cells of different sizes and widths.

(b) A corner of a gate-array base.

(c) A gate-array base with different I/O cell and pad pitches.



Power distribution.

(a) Power distributed using m1 for VSS and m2 for VDD.

This helps minimize the number of vias and layer crossings needed but causes problems in the routing channels.

(b) In this floorplan m1 is run parallel to the longest side of all channels, the channel spine.

This can make automatic routing easier but may increase the number of vias and layer crossings.

(c) An expanded view of part of a channel (interconnect is shown as lines).

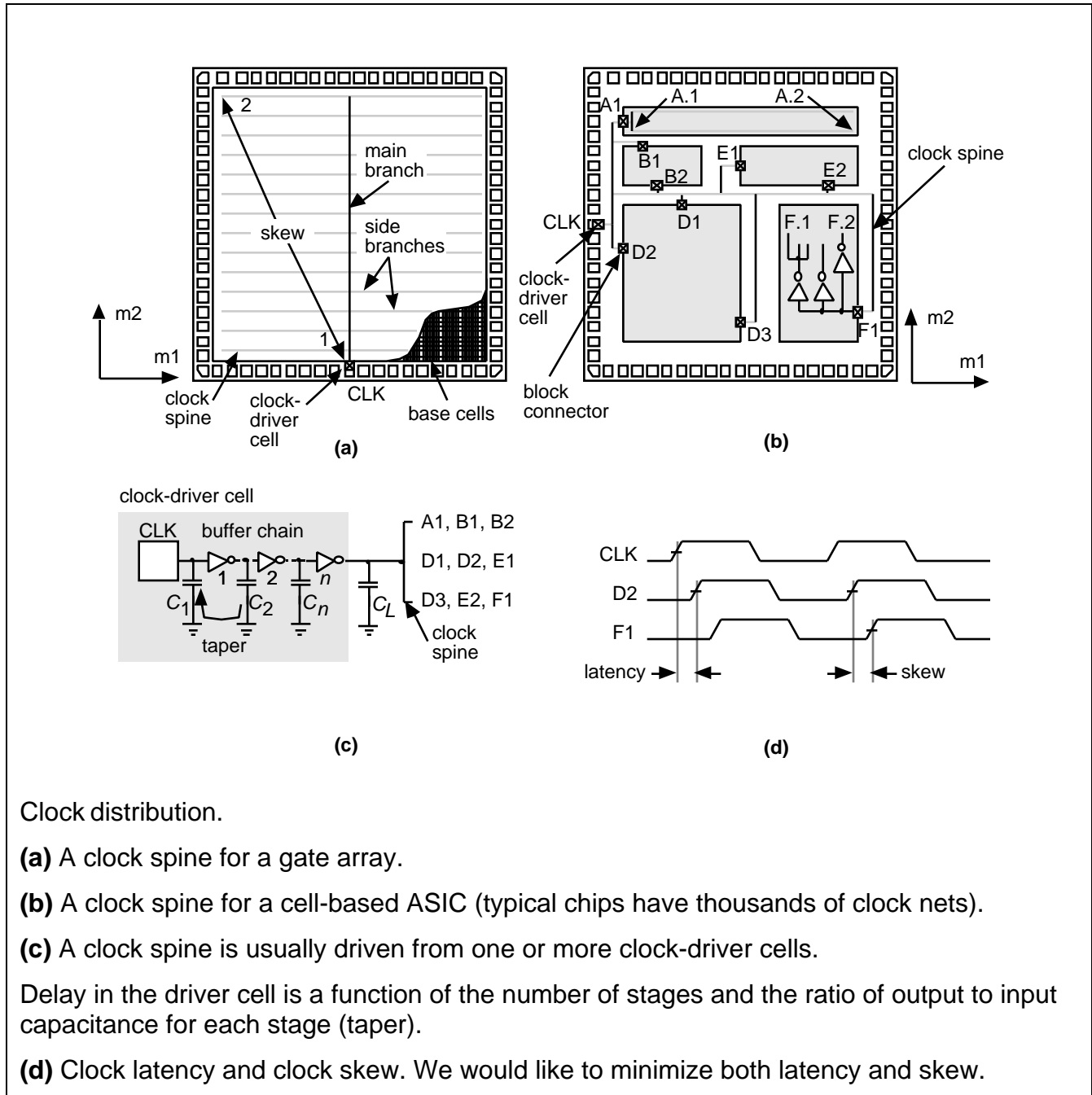
If power runs on different layers along the spine of a channel, this forces signals to change layers.

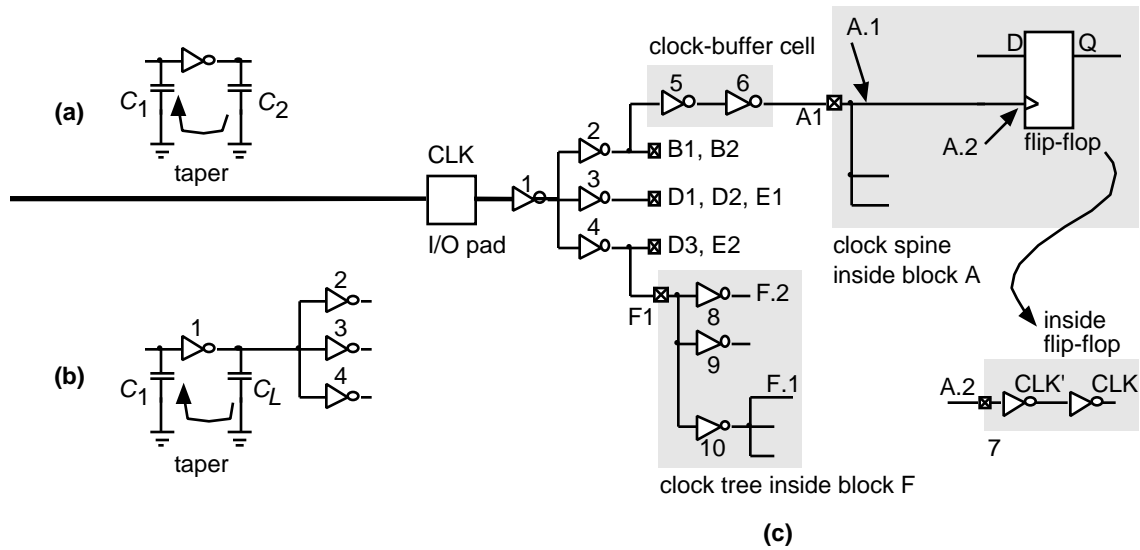
(d) A closeup of VDD and VSS buses as they cross.

Changing layers requires a large number of via contacts to reduce resistance.

16.1.6 Clock Planning

Key terms and concepts: clock spine • clock skew • clock latency • taper • hot-electron wearout • phase-locked loop (PLL) is an electronic flywheel • jitter





A clock tree.

(a) Minimum delay is achieved when the taper of successive stages is about 3.

(b) Using a fanout of three at successive nodes.

(c) A clock tree for a cell-based ASIC

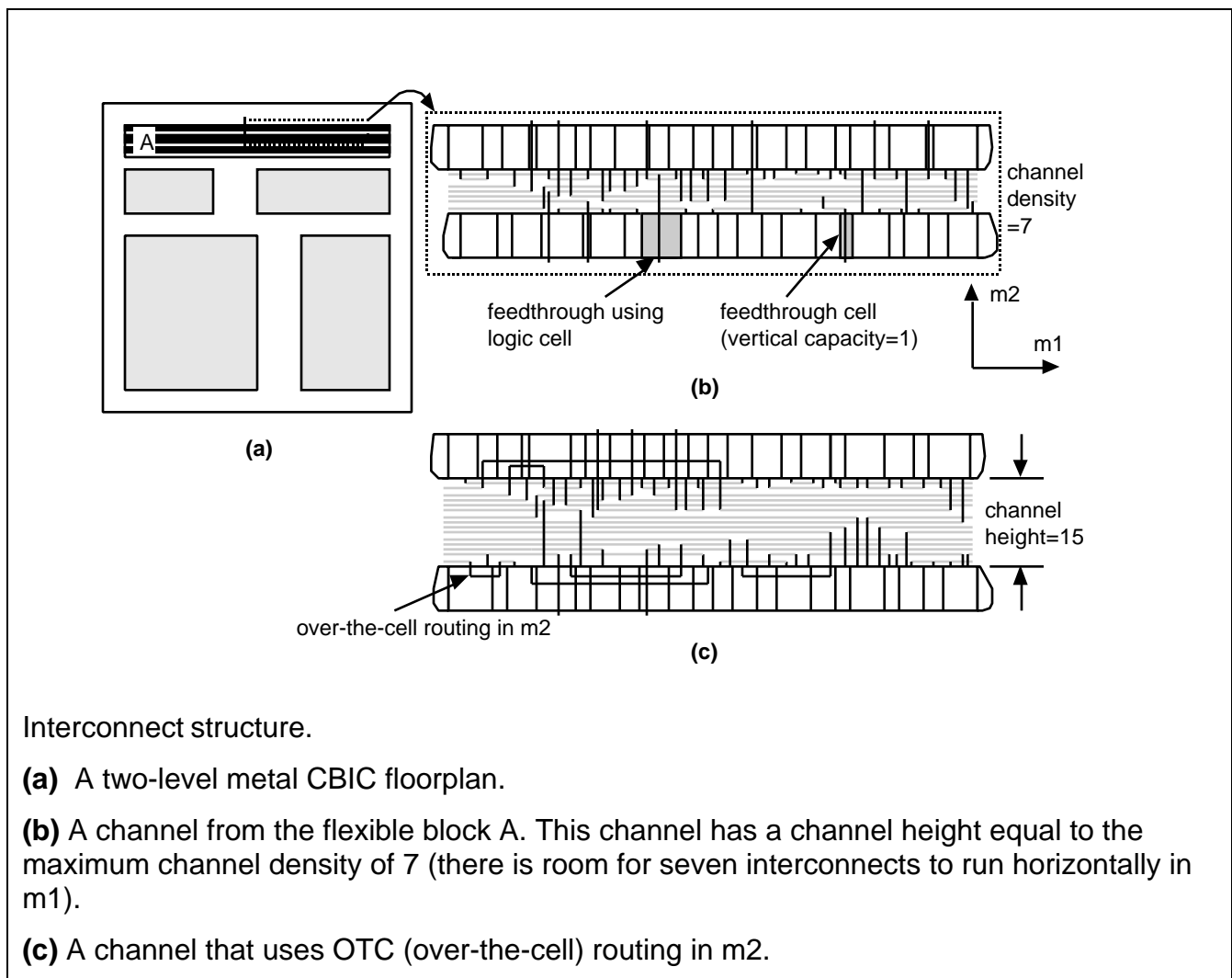
We have to balance the clock arrival times at all of the leaf nodes to minimize clock skew.

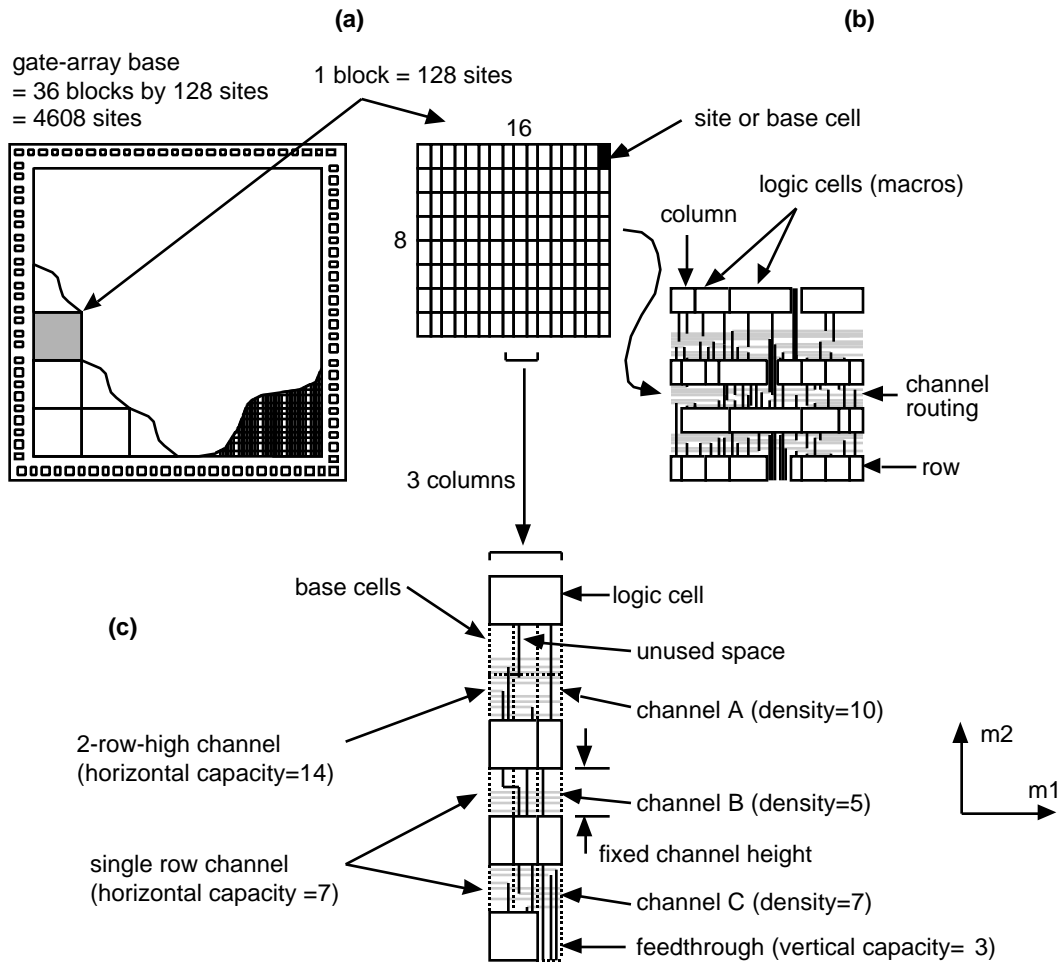
16.2 Placement

Key terms and concepts: Placement is more suited to automation than floorplanning. Thus we need measurement techniques and algorithms.

16.2.1 Placement Terms and Definitions

Key terms and concepts: row-based ASICs • over-the-cell routing (OTC routing) • channel capacity • feedthroughs • vertical track (or just track) • uncommitted feedthrough (also built-in feedthrough, implicit feedthrough, or jumper) • double-entry cells • electrically equivalent connectors (or equipotential connectors) • feedthrough cell (or crosser cell) • feedthrough pin or feedthrough terminal • spacer cell • alternative connectors • must-join connectors • logically equivalent connectors • logically equivalent connector groups • fixed-resource ASICs





Gate-array interconnect.

(a) A small two-level metal gate array (about 4.6k-gate).

(b) Routing in a block.

(c) Channel routing showing channel density and channel capacity.

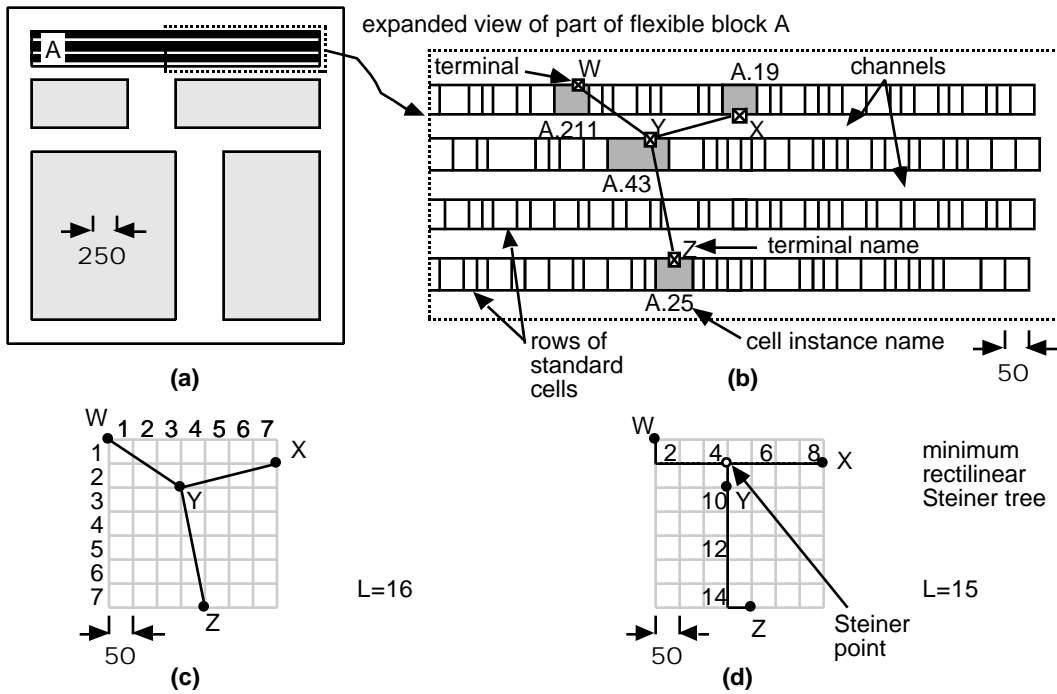
The channel height on a gate array may only be increased in increments of a row. If the interconnect does not use up all of the channel, the rest of the space is wasted. The interconnect in the channel runs in m1 in the horizontal direction with m2 in the vertical direction.

16.2.2 Placement Goals and Objectives

Key terms and concepts: Goals: (1) Guarantee the router can complete the routing step • (2) Minimize all the critical net delays • (3) Make the chip as dense as possible • Objectives: (1) Minimize power dissipation • (2) Minimize crosstalk between signals

16.2.3 Measurement of Placement Goals and Objectives

Key terms and concepts: trees on graphs (or just trees) • Steiner trees • rectilinear routing • Manhattan routing • Euclidean distance • Manhattan distance • minimum rectilinear Steiner tree (MRST) • complete graph • complete-graph measure • bounding box • half-perimeter measure (or bounding-box measure) • meander factor • interconnect congestion • maximum cut line • cut size • timing-driven placement • metal usage



Placement using trees on graphs.

(a) A floorplan.

(b) An expanded view of the flexible block A showing four rows of standard cells for placement (typical blocks may contain thousands or tens of thousands of logic cells).

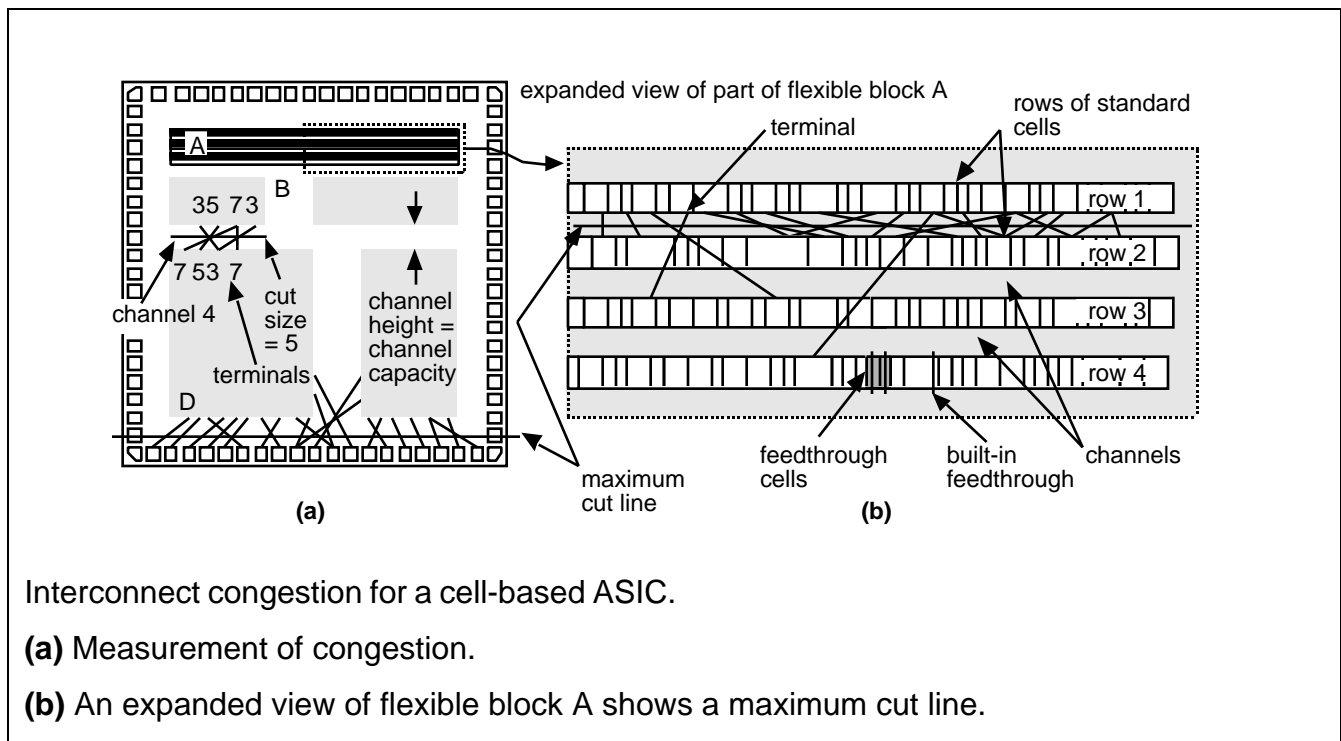
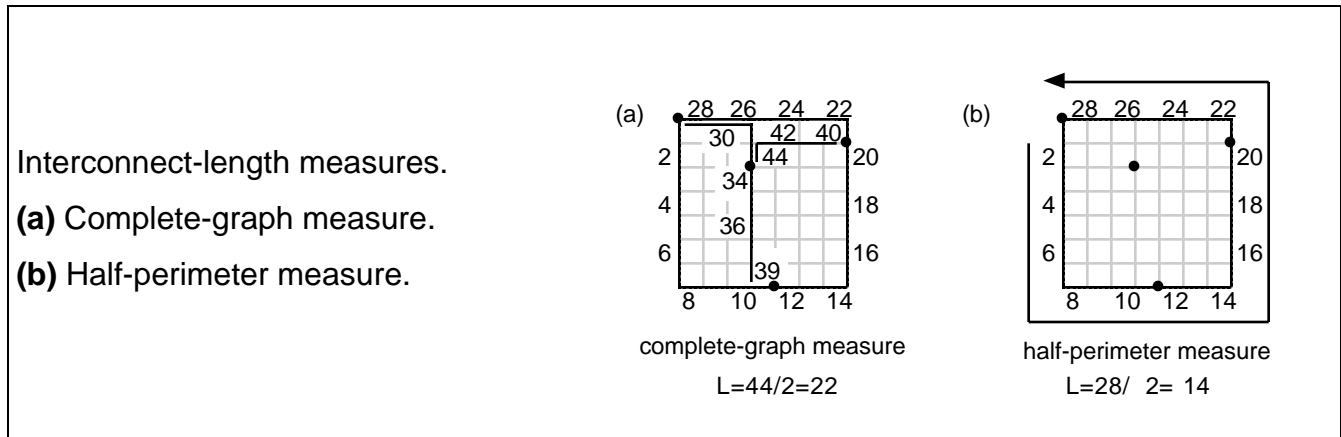
We want to find the length of the net shown with four terminals, W through Z, given the placement of four logic cells (labeled: A.211, A.19, A.43, A.25).

(c) The problem for net (W, X, Y, Z) drawn as a graph.

The shortest connection is the minimum Steiner tree.

(d) The minimum rectilinear Steiner tree using Manhattan routing.

The rectangular (Manhattan) interconnect-length measures are shown for each tree.



Interconnect congestion for a cell-based ASIC.

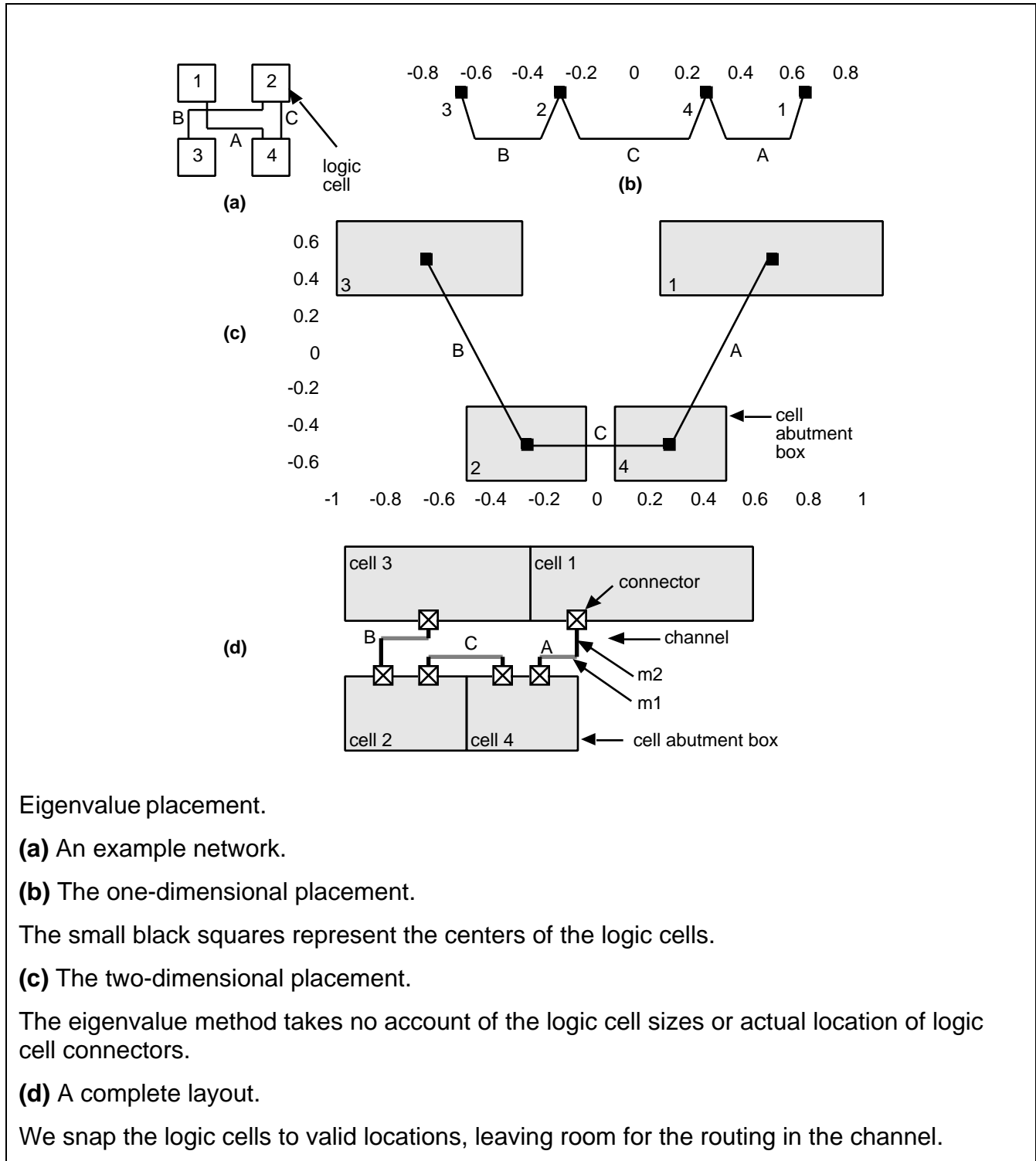
(a) Measurement of congestion.

(b) An expanded view of flexible block A shows a maximum cut line.

16.2.4 Placement Algorithms

Key terms and concepts: constructive placement method • variations on the min-cut algorithm • eigenvalue method • seed placements • min-cut placement • bins • eigenvalue placement algorithm • connectivity matrix (spectral methods) • quadratic placement • disconnection matrix (also called the Laplacian) • characteristic equation • eigenvectors and eigenvalues

16.2.5 Eigenvalue Placement Example



Eigenvalue placement.

(a) An example network.

(b) The one-dimensional placement.

The small black squares represent the centers of the logic cells.

(c) The two-dimensional placement.

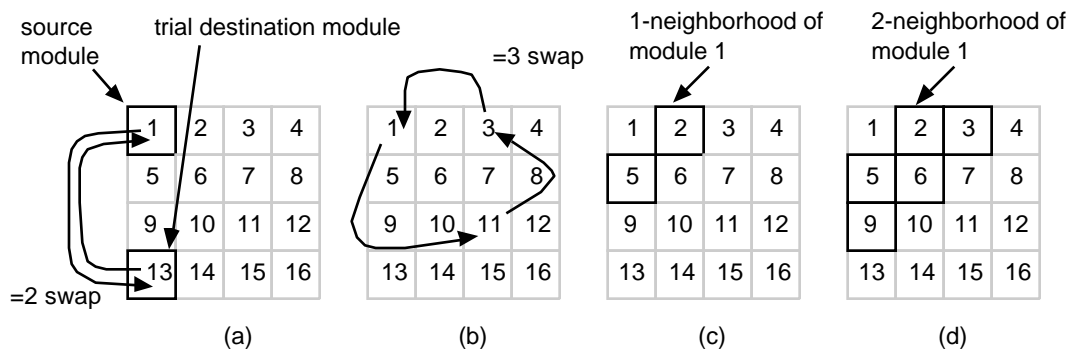
The eigenvalue method takes no account of the logic cell sizes or actual location of logic cell connectors.

(d) A complete layout.

We snap the logic cells to valid locations, leaving room for the routing in the channel.

16.2.6 Iterative Placement Improvement

Key terms and concepts: iterative placement improvement • interchange or iterative exchange • pairwise-interchange algorithm • -optimum • neighborhood exchange algorithm • neighborhood • -neighborhood • force-directed placement methods • Hooke's law • force-directed interchange • force-directed relaxation • force-directed pairwise relaxation



Interchange.

(a) Swapping the source logic cell with a destination logic cell in pairwise interchange.

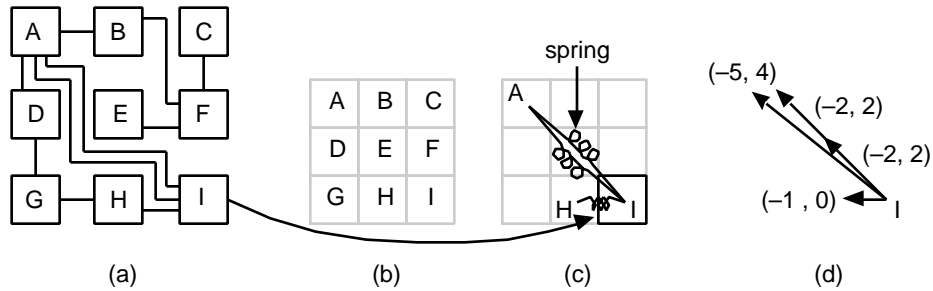
(b) Sometimes we have to swap more than two logic cells at a time to reach an optimum placement, but this is expensive in computation time.

Limiting the search to neighborhoods reduces the search time.

Logic cells within a distance of a logic cell form an -neighborhood.

(c) A one-neighborhood.

(d) A two-neighborhood.



Force-directed placement.

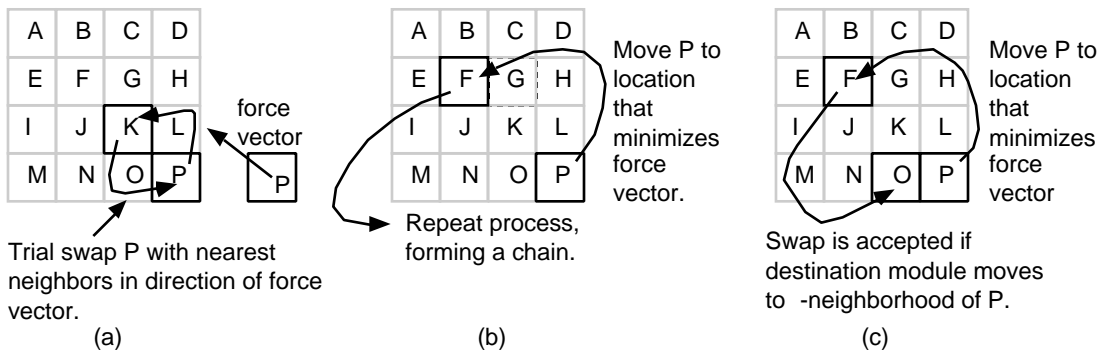
(a) A network with nine logic cells.

(b) We make a grid (one logic cell per bin).

(c) Forces are calculated as if springs were attached to the centers of each logic cell for each connection.

The two nets connecting logic cells A and I correspond to two springs.

(d) The forces are proportional to the spring extensions.



Force-directed iterative placement improvement.

(a) Force-directed interchange.

(b) Force-directed relaxation.

(c) Force-directed pairwise relaxation.

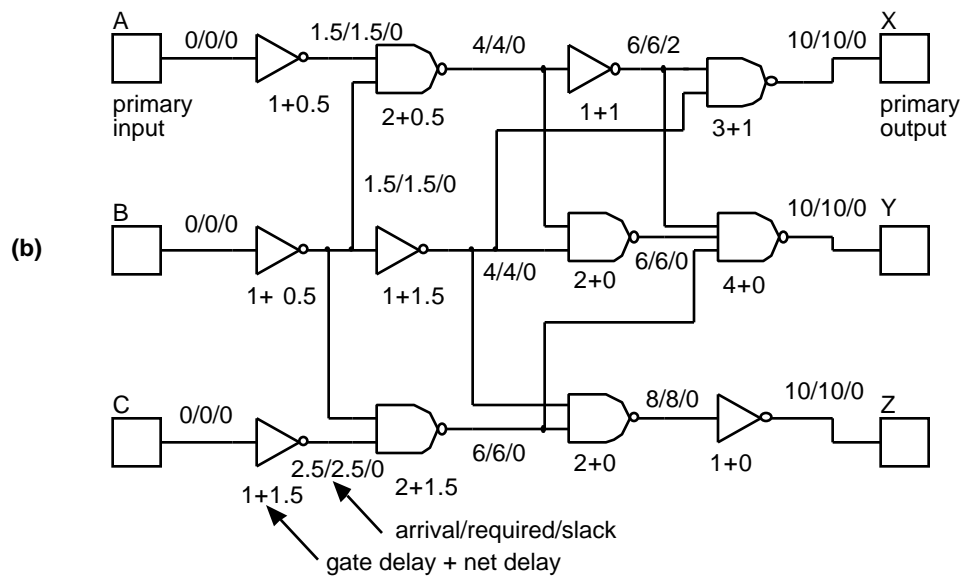
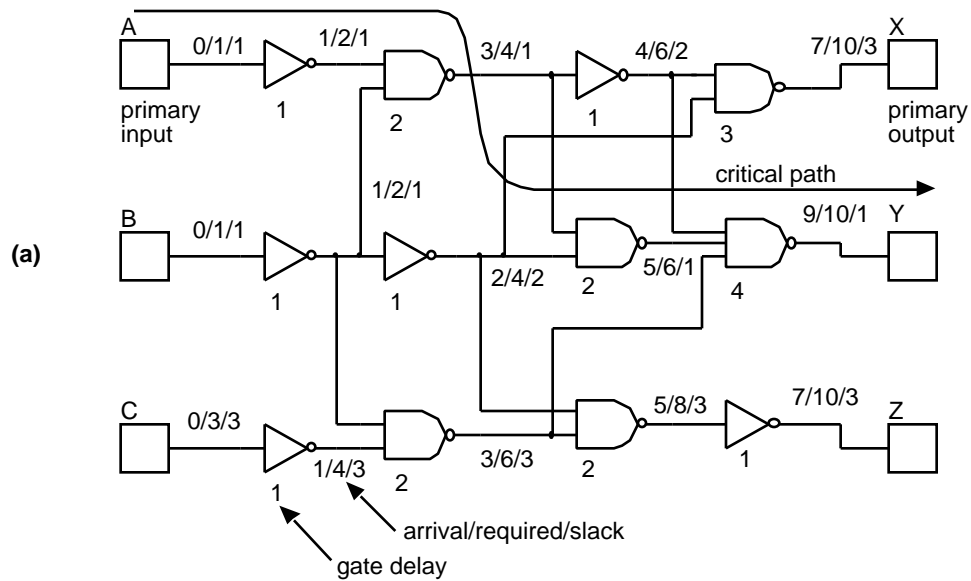
16.2.7 Placement Using Simulated Annealing

Key terms and concepts:

1. Select logic cells for a trial interchange, usually at random.
2. Evaluate the objective function E for the new placement.
3. If E is negative or zero, then exchange the logic cells. If E is positive, then exchange the logic cells with a probability of $\exp(-E/T)$.
4. Go back to step 1 for a fixed number of times, and then lower the temperature T according to a cooling schedule: $T_{n+1} = 0.9 T_n$, for example.

16.2.8 Timing-Driven Placement Methods

Key terms and concepts: zero-slack algorithm primary inputs • arrival times • actual times • required times • primary outputs • slack time



The zero-slack algorithm.

(a) The circuit with no net delays.

(b) The zero-slack algorithm adds net delays (at the outputs of each gate, equivalent to increasing the gate delay) to reduce the slack times to zero.

16.2.9 A Simple Placement Example

(a) An example network showing logic cells A through I connected in a grid-like structure.

(b) A placement layout where cells are arranged in a 3x3 grid. Annotations include: maximum cut line (y) = 4, capacity of each bin edge = 2, wire length = 1, cut line = 2, cut line = 1, and total routing length = 8.

(c) An alternative placement layout with a routing length = 7 and maximum cut (x and y) = 2.

(d) A detailed layout showing channel densities. It includes labels for cell connector, channel density = 2, m1, m2, channel density = 1, and cell abutment box.

Placement example.

(a) An example network.

(b) In this placement, the bin size is equal to the logic cell size and all the logic cells are assumed equal size.

(c) An alternative placement with a lower total routing length.

(d) A layout that might result from the placement shown in b.

The channel densities correspond to the cut-line sizes.

Notice that the logic cells are not all the same size (which means there are errors in the interconnect-length estimates we made during placement).

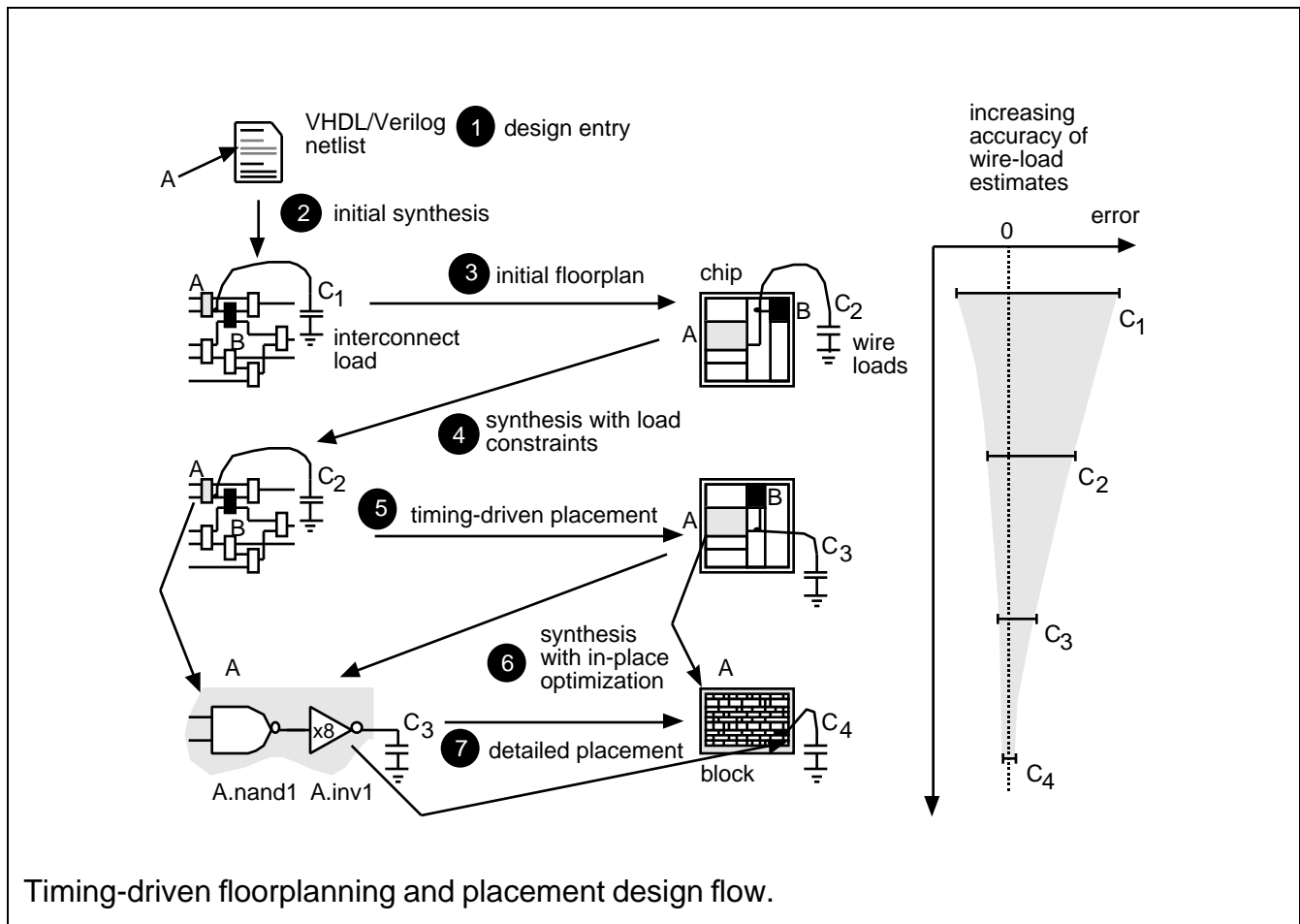
16.3 Physical Design Flow

Key terms and concepts:

Because interconnect delay now dominates gate delay, the trend is to include placement within a floorplanning tool and use a separate router.

1. **Design entry.** The input is a logical description with no physical information.

2. **Initial synthesis.** The initial synthesis contains little or no information on any interconnect loading. The output of the synthesis tool (typically an EDIF netlist) is the input to the floorplanner.
3. **Initial floorplan.** From the initial floorplan interblock capacitances are input to the synthesis tool as load constraints and intrablock capacitances are input as wire-load tables.
4. **Synthesis with load constraints.** At this point the synthesis tool is able to resynthesize the logic based on estimates of the interconnect capacitance each gate is driving. The synthesis tool produces a forward annotation file to constrain path delays in the placement step.
5. **Timing-driven placement.** After placement using constraints from the synthesis tool, the location of every logic cell on the chip is fixed and accurate estimates of interconnect delay can be passed back to the synthesis tool.
6. **Synthesis with in-place optimization (IPO).** The synthesis tool changes the drive strength of gates based on the accurate interconnect delay estimates from the floorplanner without altering the netlist structure.
7. **Detailed placement.** The placement information is ready to be input to the routing step.



16.4 Information Formats

16.4.1 SDF for Floorplanning and Placement

Key terms and concepts: standard delay format (SDF) • back-annotation • forward-annotation • timing constraints

```
(INSTANCE B) (DELAY (ABSOLUTE
  (INTERCONNECT A.INV8.OUT B.DFF1.Q (:0.6:) (:0.6:))))
```

```
(TIMESCALE 100ps) (INSTANCE B) (DELAY (ABSOLUTE
  (NETDELAY net1 (0.6))))
```

```
(TIMESCALE 100ps) (INSTANCE B.DFF1) (DELAY (ABSOLUTE
  (PORT CLR (16:18:22) (17:20:25))))
```

```
(TIMESCALE 100ps) (INSTANCE B) †TIMINGCHECK
  (PATHCONSTRAINT A.AOI22_1.O B.ND02_34.O (0.8) (0.8)))
```

```
(TIMESCALE 100ps) (INSTANCE B) †TIMINGCHECK
  (SUM (AOI22_1.O ND02_34.I1) (ND02_34.O ND02_35.I1) (0.8)))
```

```
(TIMESCALE 100ps) (INSTANCE B) (TIMINGCHECK
  (DIFF (A.I_1.O B.ND02_1.I1) (A.I_1.O.O B.ND02_2.I1) (0.1)))
```

```
(TIMESCALE 100ps) (INSTANCE B) (TIMINGCHECK
  (SKEWCONSTRAINT (posedge clk) (0.1)))
```

16.4.2 PDEF

Key terms and concepts: physical design exchange format (PDEF)

```
(CLUSTERFILE
  (PDEFVERSION "1.0")
  (DESIGN "myDesign")
  (DATE "THU AUG 6 12:00 1995"))
```

```
(VENDOR "ASICS_R_US")
(PROGRAM "PDEF_GEN")
(VERSION "V2.2")
(DIVIDER .)
(CLUSTER (NAME "ROOT")
  (WIRE_LOAD "10mm x 10mm")
  (UTILIZATION 50.0)
  (MAX_UTILIZATION 60.0)
  (X_BOUNDS 100 1000)
  (Y_BOUNDS 100 1000)
  (CLUSTER (NAME "LEAF_1")
    (WIRE_LOAD "50k gates")
    (UTILIZATION 50.0)
    (MAX_UTILIZATION 60.0)
    (X_BOUNDS 100 500)
    (Y_BOUNDS 100 200)
    (CELL (NAME L1.RAM01)
      (CELL (NAME L1.ALU01)
        )
      )
    )
  )
)
```

16.4.3 LEF and DEF

Key terms and concepts: library exchange format (LEF) • design exchange format (DEF)

16.5 Summary

Key terms and concepts: Interconnect delay now dominates gate delay • Floorplanning is a mapping between logical and physical design • Floorplanning is the center of design operations for all types of ASIC • Timing-driven floorplanning is an essential ASIC design tool • Placement is an automated function

ROUTING

17

Key terms and concepts: Routing is usually split into **global routing** followed by **detailed routing**.

Suppose the ASIC is North America and some travelers in California need to drive from Stanford (near San Francisco) to Caltech (near Los Angeles).

The floorplanner decides that California is on the left (west) side of the ASIC and the placement tool has put Stanford in Northern California and Caltech in Southern California.

Floorplanning and placement define the roads and freeways. There are two ways to go: the coastal route (Highway 101) or the inland route (Interstate I5—usually faster).

The global router specifies the coastal route because the travelers are not in a hurry and I5 is congested (the global router knows this because it has already routed onto I5 many other travelers that are in a hurry today).

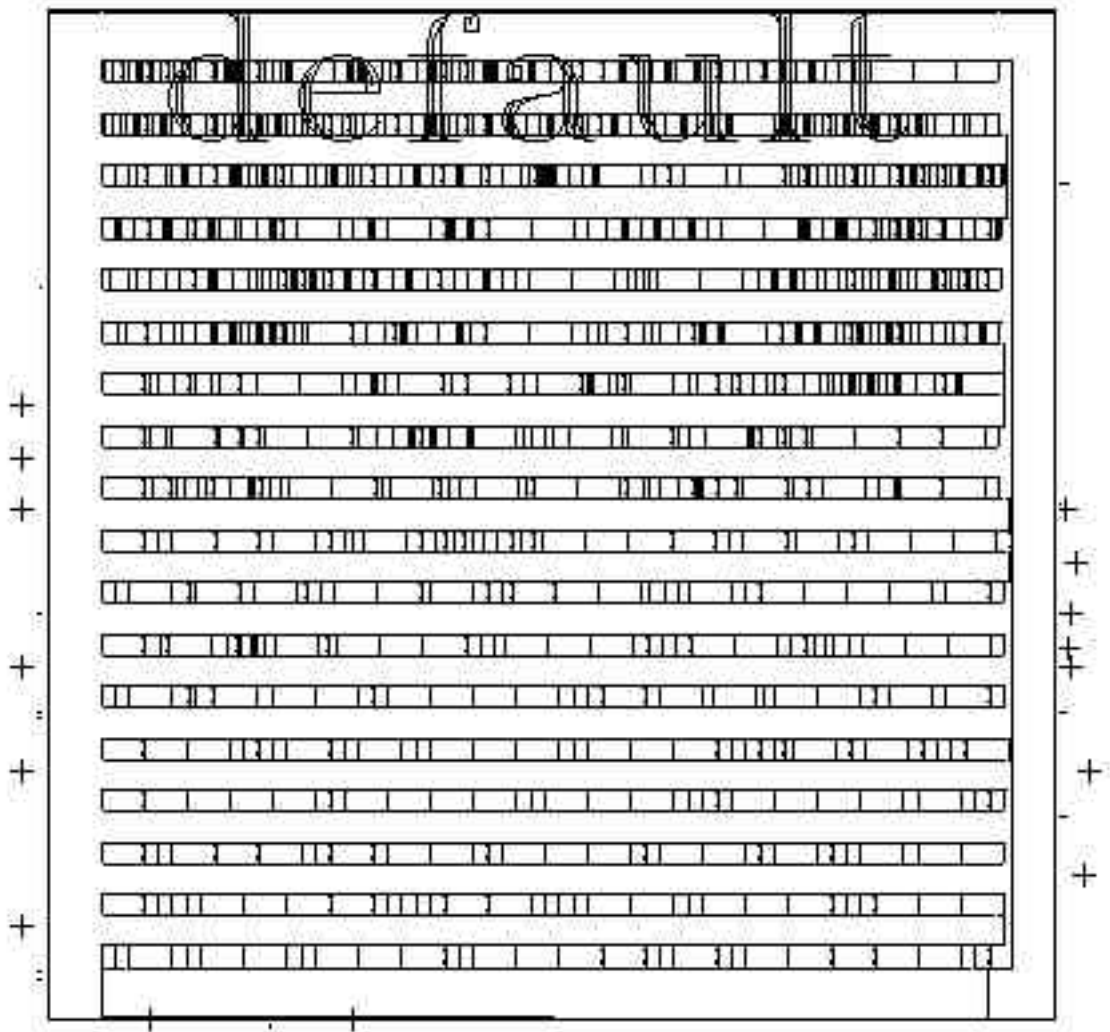
Next, the detailed router looks at a map and gives indications from Stanford onto Highway 101 south through San Jose, Monterey, and Santa Barbara to Los Angeles and then off the freeway to Caltech in Pasadena.

17.1 Global Routing

Key terms and concepts: Global routing differs slightly between CBICs, gate arrays, and FPGAs, but the principles are the same • A global router does not make any connections, it just plans them • We typically global route the whole chip (or large pieces) before detail routing • There are two types of areas to global route: inside the flexible blocks and between blocks

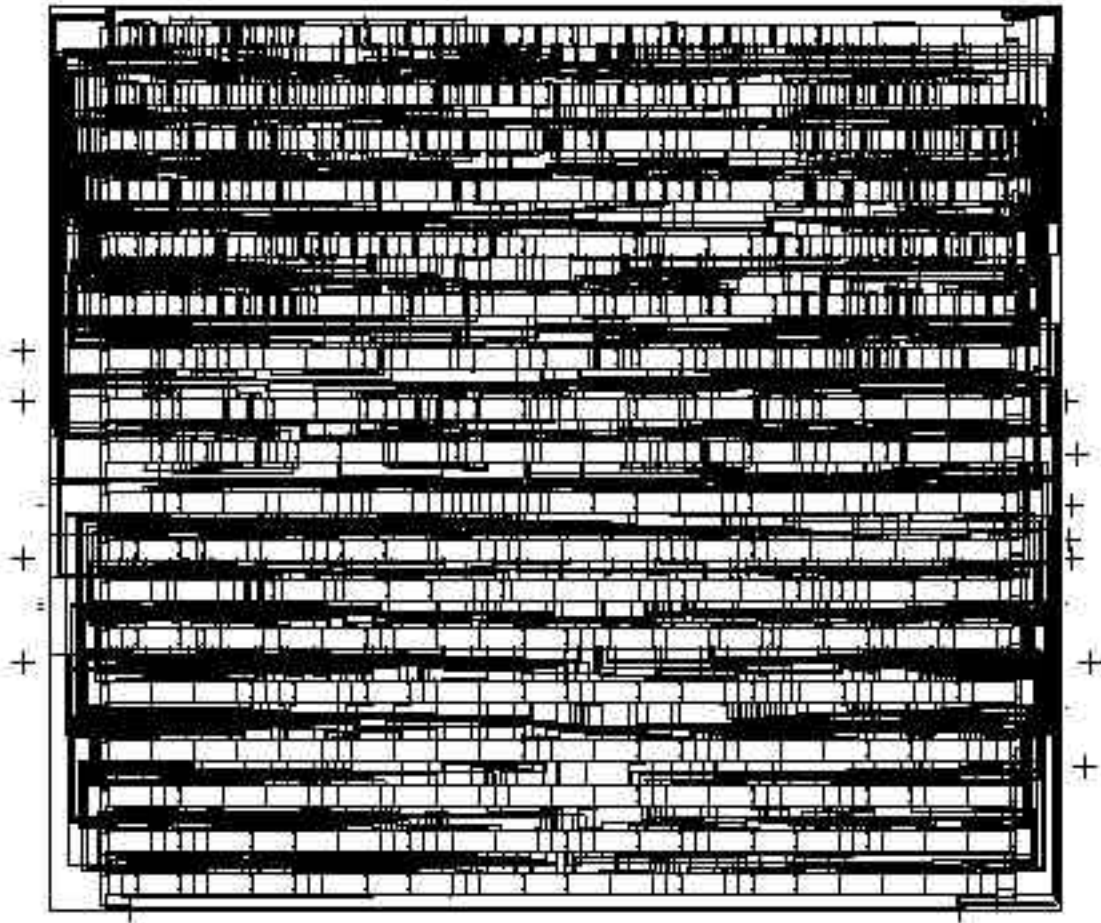
17.1.1 Goals and Objectives

Key terms and concepts: Goal: provide complete instructions to the detailed router • Objectives: Minimize the total interconnect length • Maximize the probability that the detailed router can complete the routing • Minimize the critical path delay



The core of the Viterbi decoder chip after placement.

You can see the rows of standard cells; the widest cells are the D flip-flops.



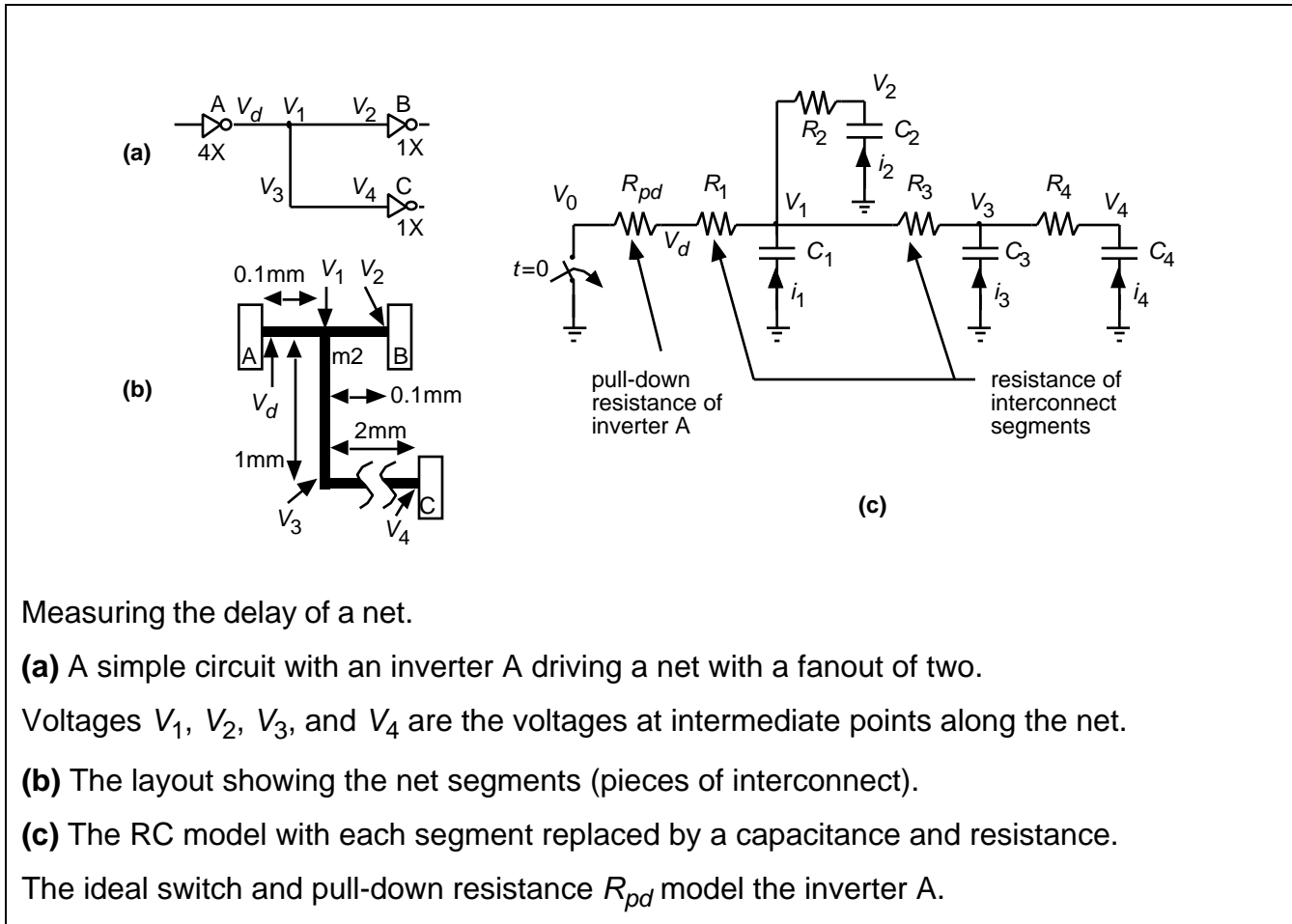
The core of the Viterbi decoder chip after the completion of global and detailed routing.

This chip uses two-level metal.

Although you cannot see the difference, m1 runs in the horizontal direction and m2 in the vertical direction.

17.1.2 Measurement of Interconnect Delay

Key terms and concepts: **lumped-delay model** • **lumped capacitance** • as interconnect delay becomes more important other, more complex models, are used

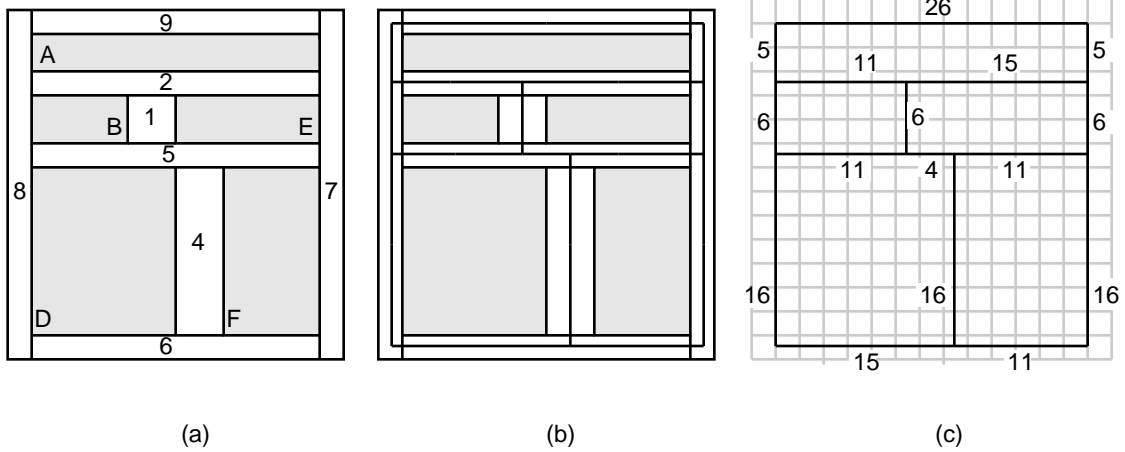


17.1.3 Global Routing Methods

Key terms and concepts: sequential routing • order-independent routing • order dependent routing • hierarchical routing (top-down or bottom-up)

17.1.4 Global Routing Between Blocks

Key terms and concepts: use of the channel-intersection graph



Global routing for a cell-based ASIC formulated as a graph problem.

(a) A cell-based ASIC with numbered channels.

(b) The channels form the edges of a graph.

(c) The channel-intersection graph. Each channel corresponds to an edge on a graph whose weight corresponds to the channel length.

17.1.5 Global Routing Inside Flexible Blocks

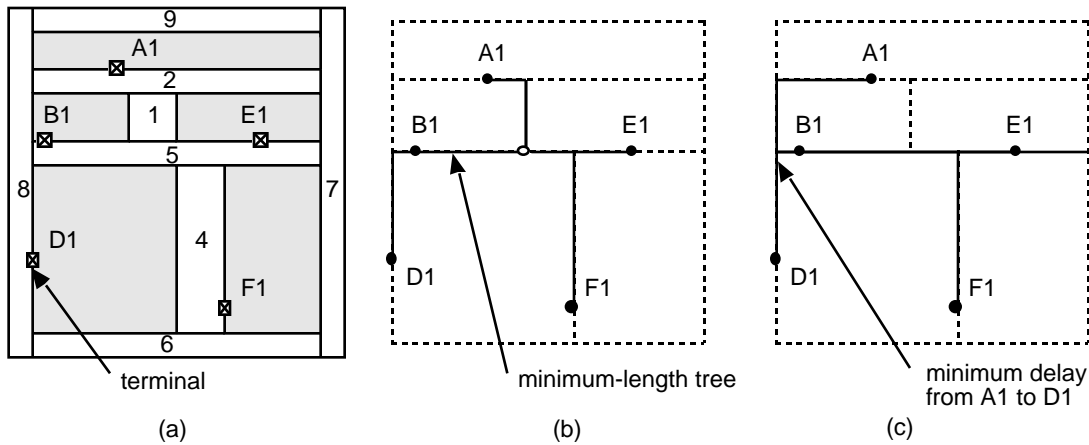
Key terms and concepts: track • landing pad • pick-up point, connector, terminal, pin, or port • area pick-up point • horizontal tracks • routing bins (or just bins, also called global routing cells or GRCs)

17.1.6 Timing-Driven Methods

Key terms and concepts: use of timing engine • path or node based

17.1.7 Back-annotation

Key terms and concepts: RC information • huge files • database problem

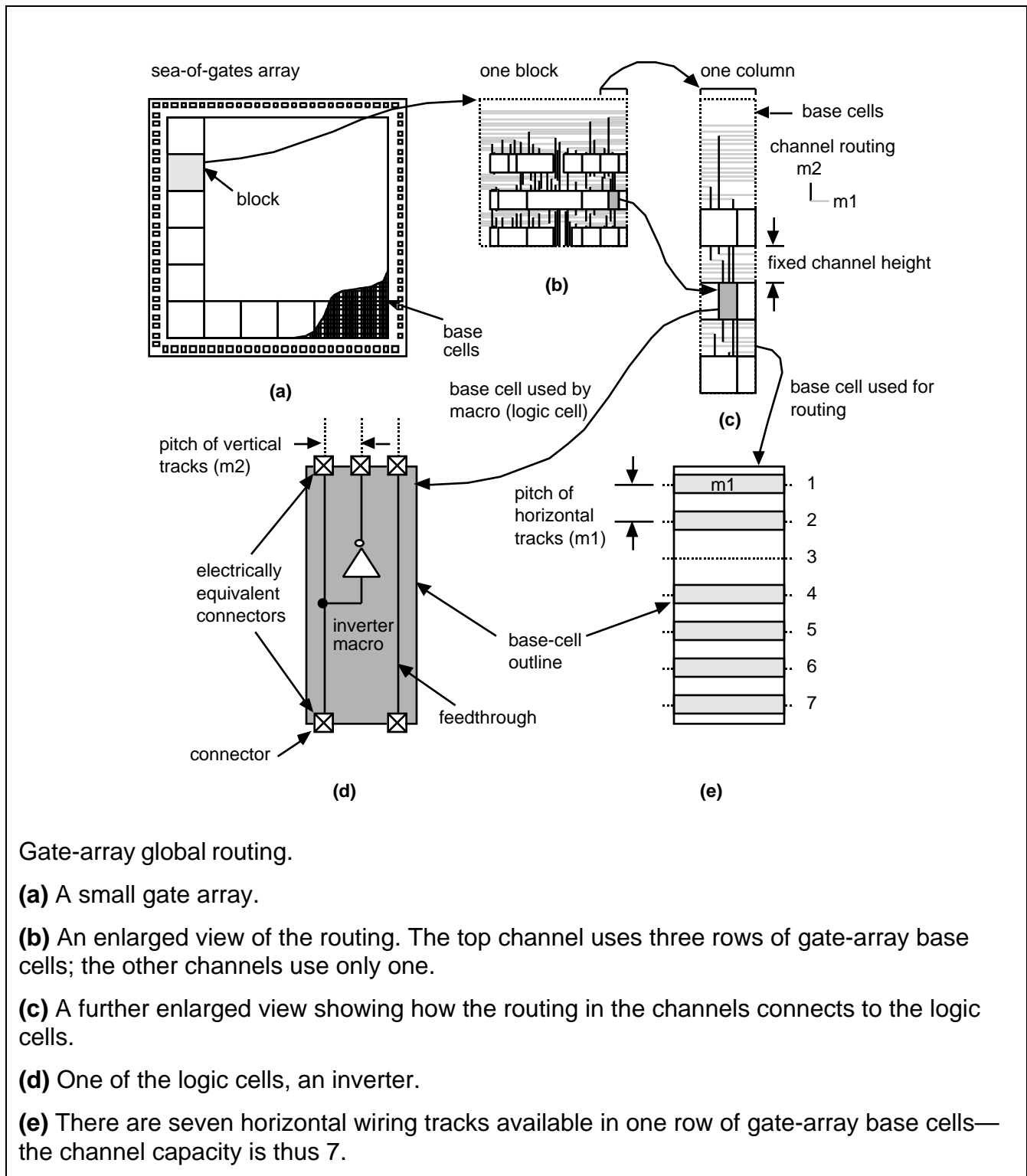


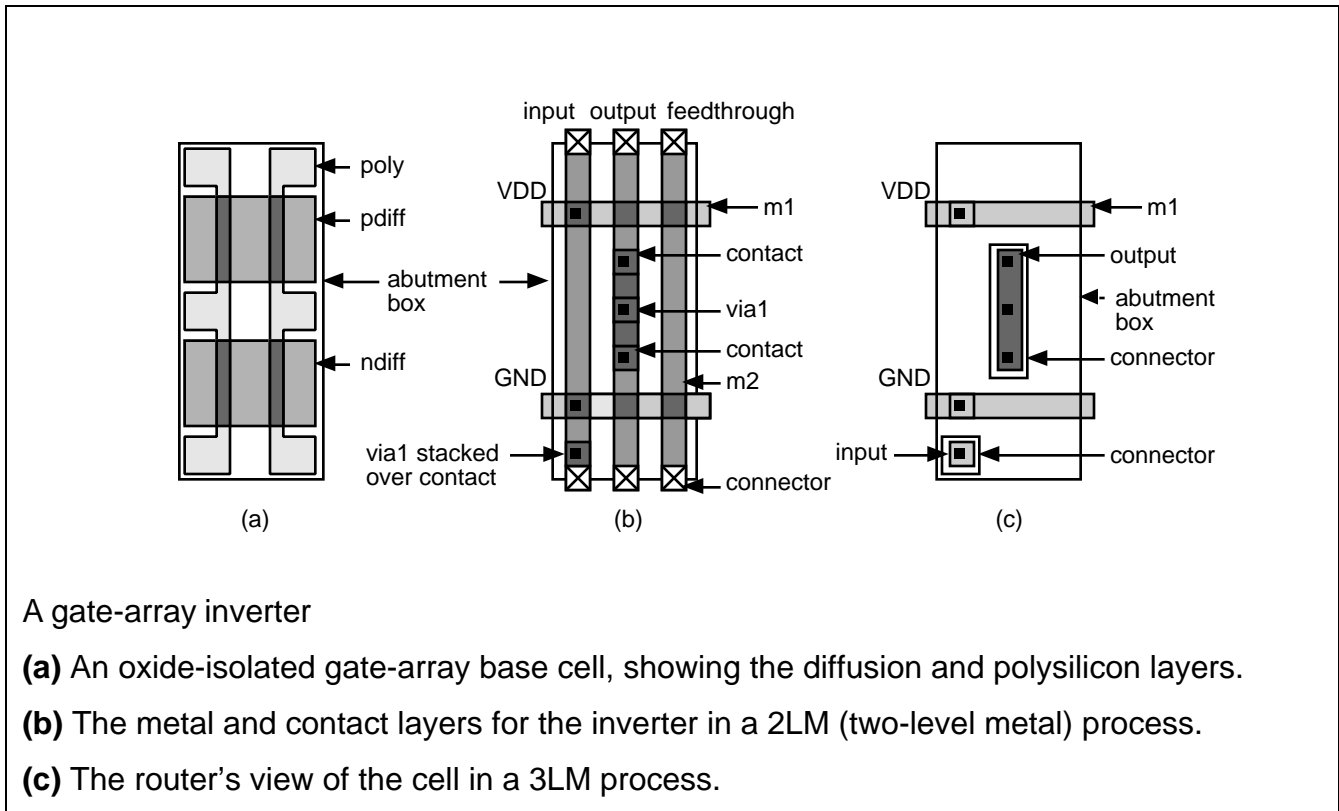
Finding paths in global routing.

(a) A cell-based ASIC showing a single net with a fanout of four (five terminals). We have to order the numbered channels to complete the interconnect path for terminals A1 through F1.

(b) The terminals are projected to the center of the nearest channel, forming a graph. A minimum-length tree for the net that uses the channels and takes into account the channel capacities.

(c) The minimum-length tree does not necessarily correspond to minimum delay. If we wish to minimize the delay from terminal A1 to D1, a different tree might be better.



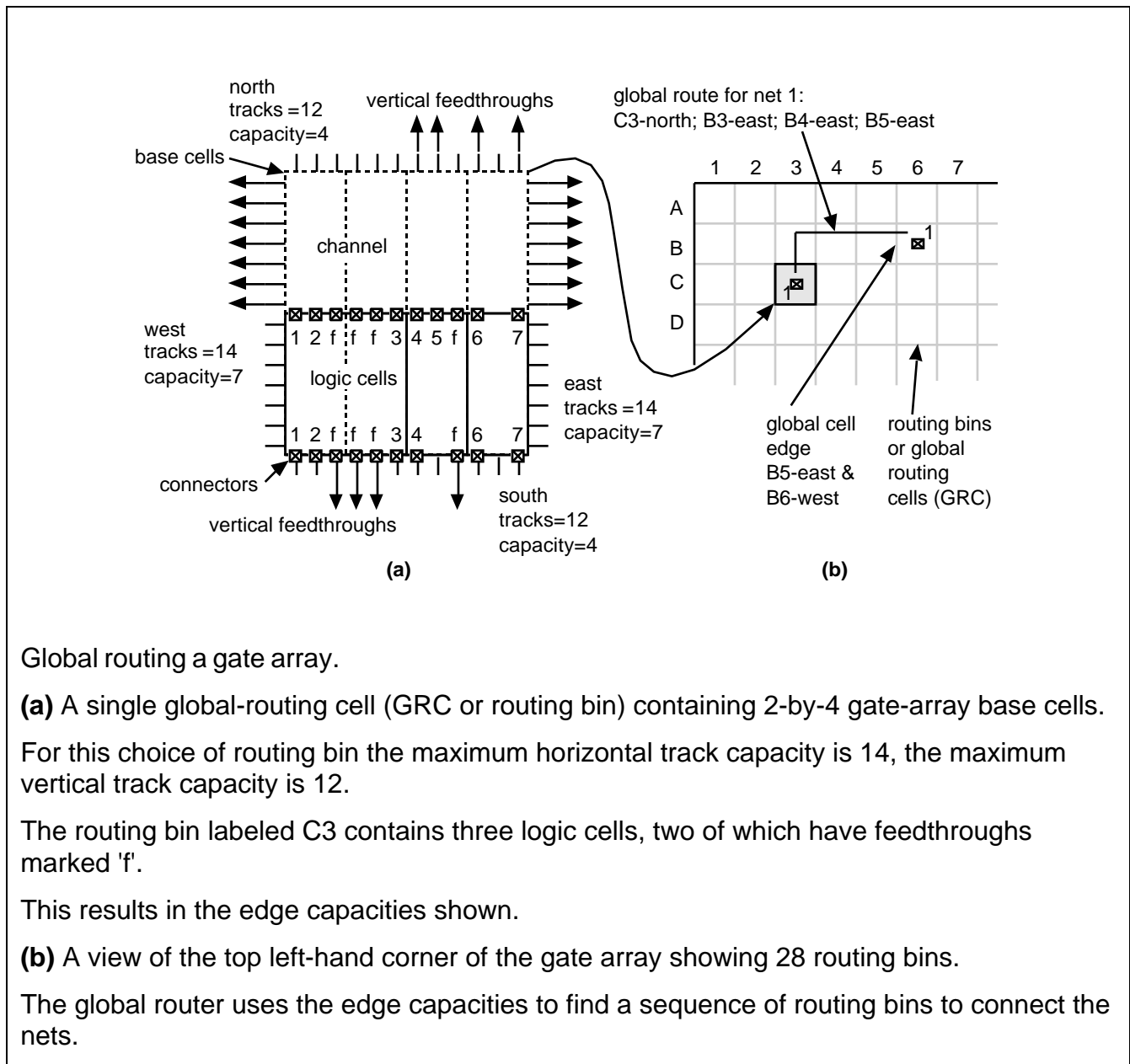


A gate-array inverter

(a) An oxide-isolated gate-array base cell, showing the diffusion and polysilicon layers.

(b) The metal and contact layers for the inverter in a 2LM (two-level metal) process.

(c) The router's view of the cell in a 3LM process.



Global routing a gate array.

(a) A single global-routing cell (GRC or routing bin) containing 2-by-4 gate-array base cells.

For this choice of routing bin the maximum horizontal track capacity is 14, the maximum vertical track capacity is 12.

The routing bin labeled C3 contains three logic cells, two of which have feedthroughs marked 'f'.

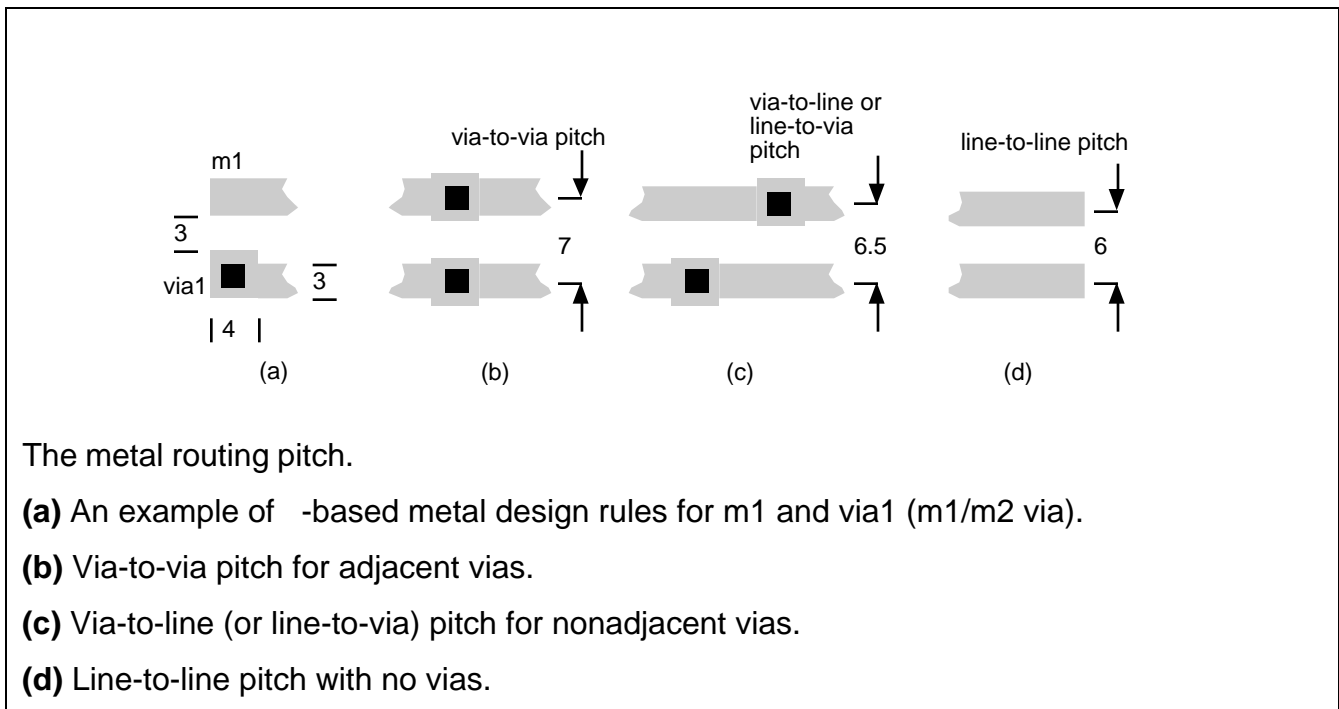
This results in the edge capacities shown.

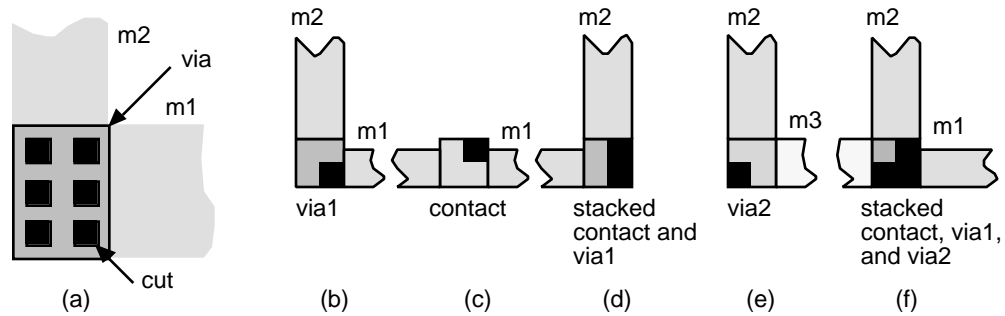
(b) A view of the top left-hand corner of the gate array showing 28 routing bins.

The global router uses the edge capacities to find a sequence of routing bins to connect the nets.

17.2 Detailed Routing

Key terms and concepts: routing pitch (track pitch, track spacing, or just pitch) • via-to-via (VTV) pitch (or spacing) • via-to-line (VTL or line-to-via) pitch • line-to-line (LTL) pitch. • stitch • waffle via • stacked via • Manhattan routing • preferred direction • preferred metal layer • phantom • blockage map • on-grid • off-grid • trunks • branches • doglegs • pseudoterminals • tracks (like railway tracks) • horizontal track spacing • track spacing • column • column spacing (or vertical track spacing)





Vias

(a) A large m1 to m2 via. The black squares represent the holes (or cuts) that are etched in the insulating material between the m1 and 2 layers.

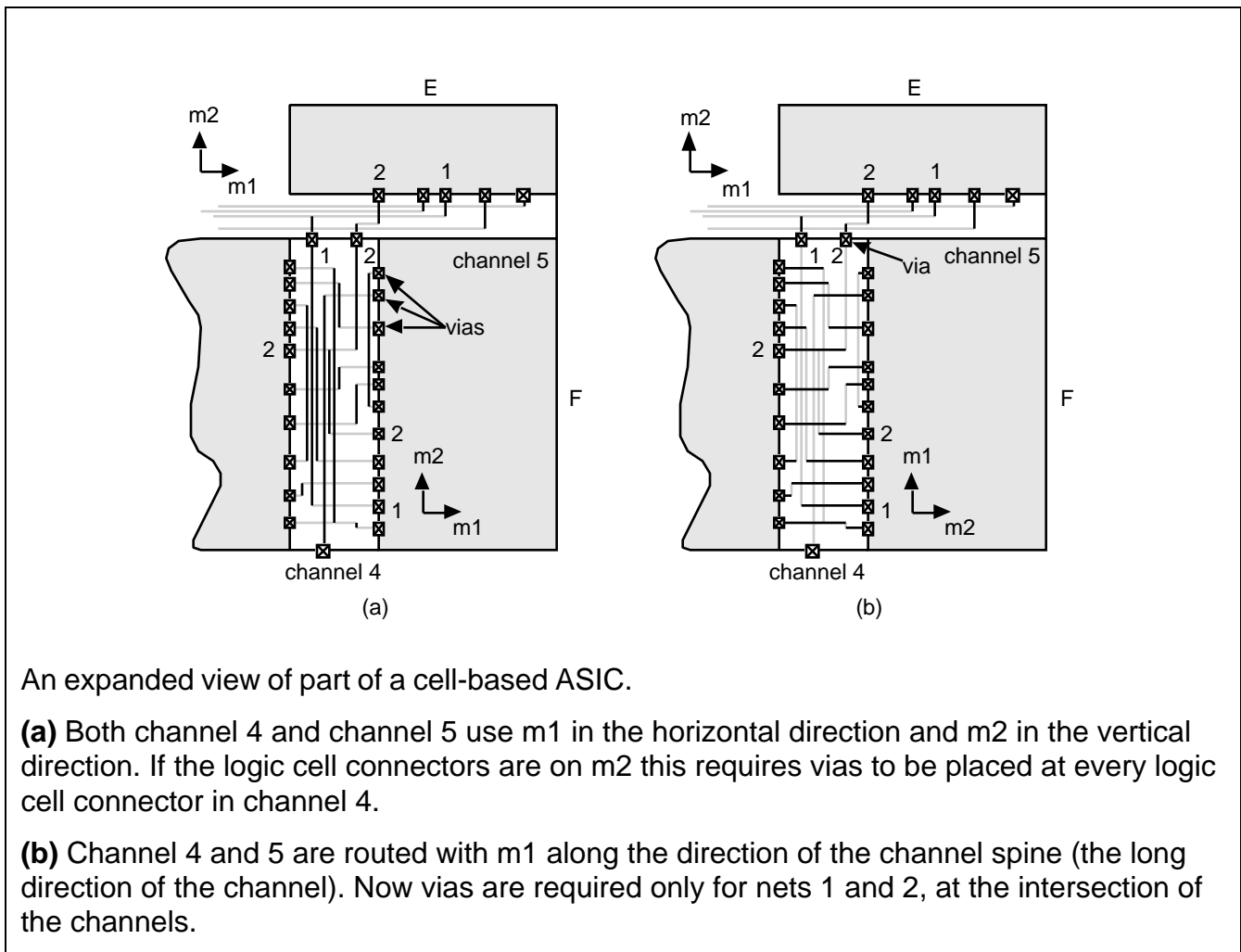
(b) A m1 to m2 via (a via1).

(c) A contact from m1 to diffusion or polysilicon (a contact).

(d) A via1 placed over (or stacked over) a contact.

(e) A m2 to m3 via (a via2).

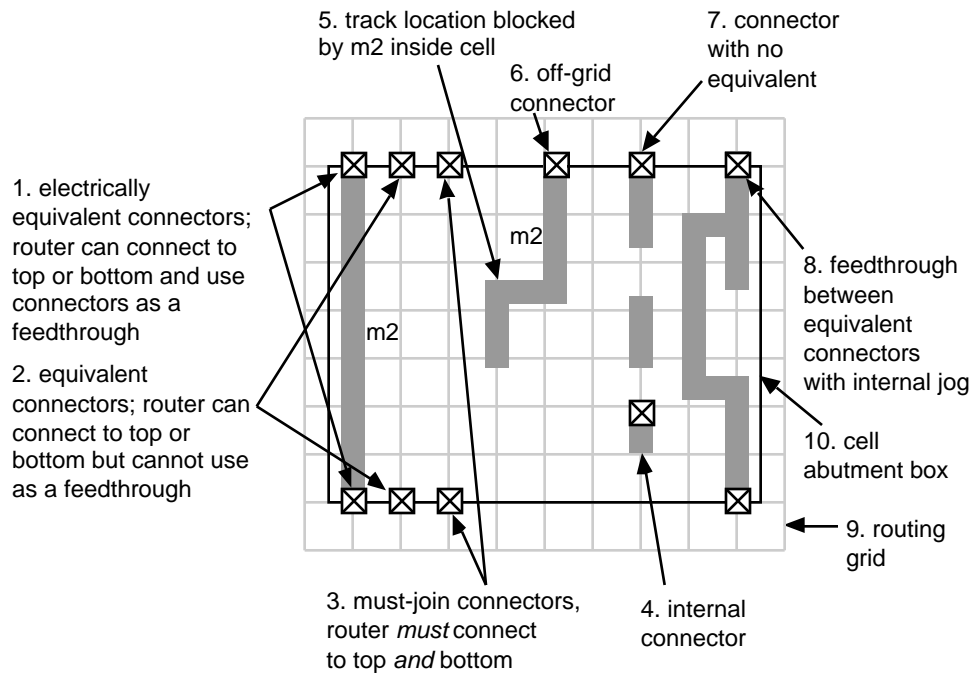
(f) A via2 stacked over a via1 stacked over a contact. Notice that the black square in parts b–c do *not* represent the actual location of the cuts. The black squares are offset so you can recognize stacked vias and contacts.



An expanded view of part of a cell-based ASIC.

(a) Both channel 4 and channel 5 use m1 in the horizontal direction and m2 in the vertical direction. If the logic cell connectors are on m2 this requires vias to be placed at every logic cell connector in channel 4.

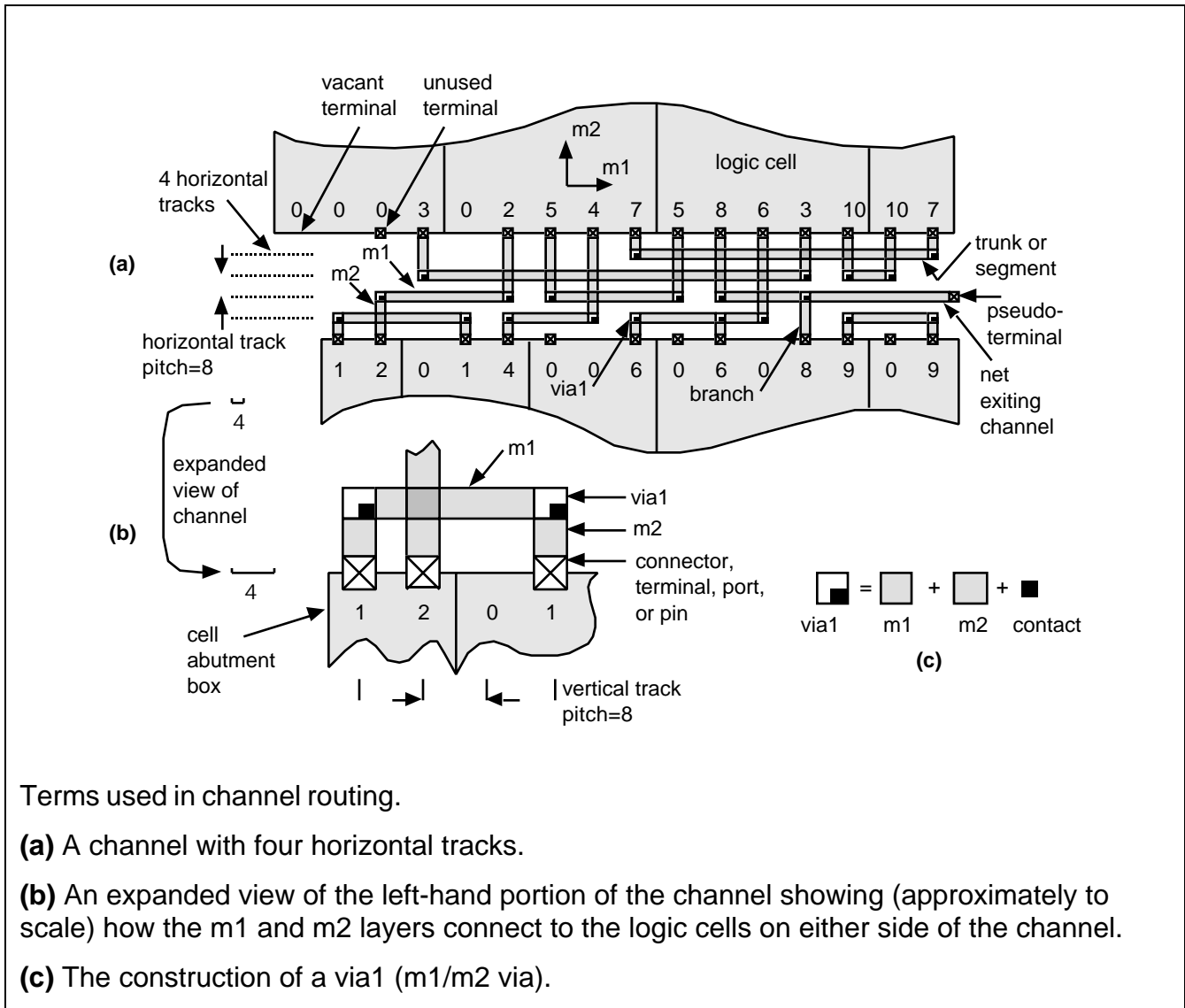
(b) Channel 4 and 5 are routed with m1 along the direction of the channel spine (the long direction of the channel). Now vias are required only for nets 1 and 2, at the intersection of the channels.



The different types of connections that can be made to a cell.

This cell has connectors at the top and bottom of the cell (normal for cells intended for use with a two-level metal process) and internal connectors (normal for logic cells intended for use with a three-level metal process).

The interconnect and connections are drawn to scale.



Terms used in channel routing.

(a) A channel with four horizontal tracks.

(b) An expanded view of the left-hand portion of the channel showing (approximately to scale) how the m1 and m2 layers connect to the logic cells on either side of the channel.

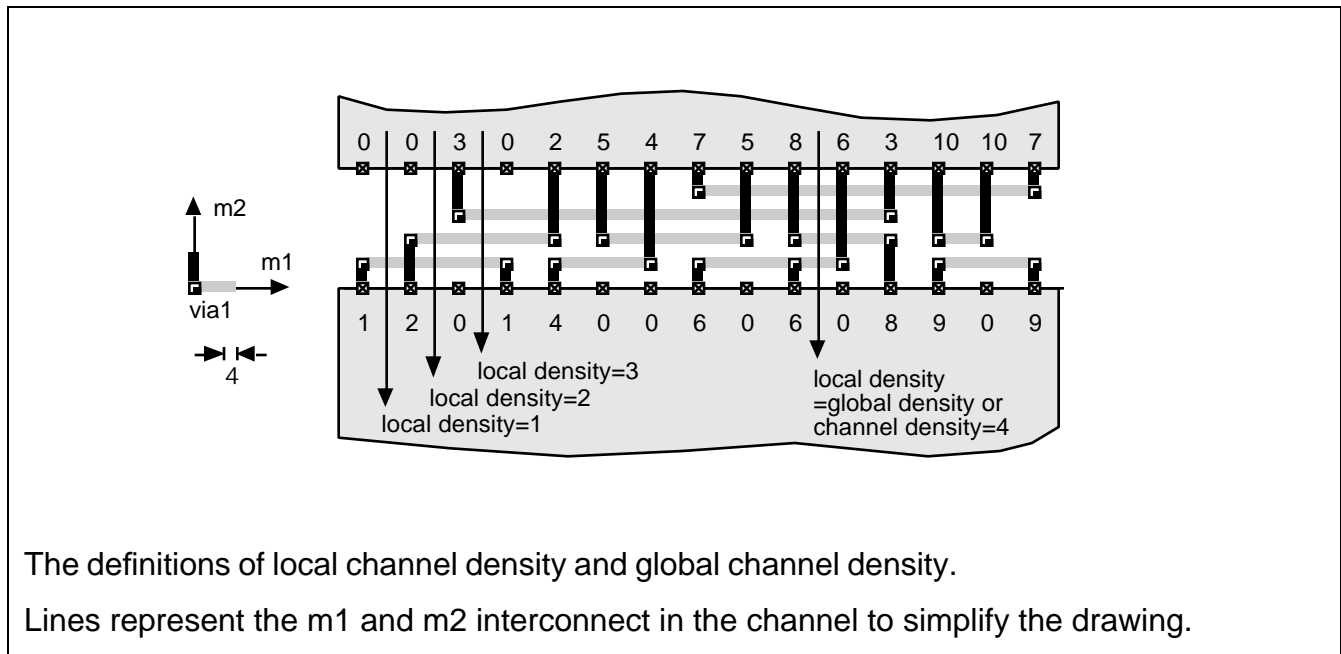
(c) The construction of a via1 (m1/m2 via).

17.2.1 Goals and Objectives

Key terms and concepts: Goal: to complete all the connections between logic cells • Objectives: The total interconnect length and area • The number of layer changes that the connections have to make • The delay of critical paths

17.2.2 Measurement of Channel Density

Key terms and concepts: local density • global density • channel density



17.2.3 Algorithms

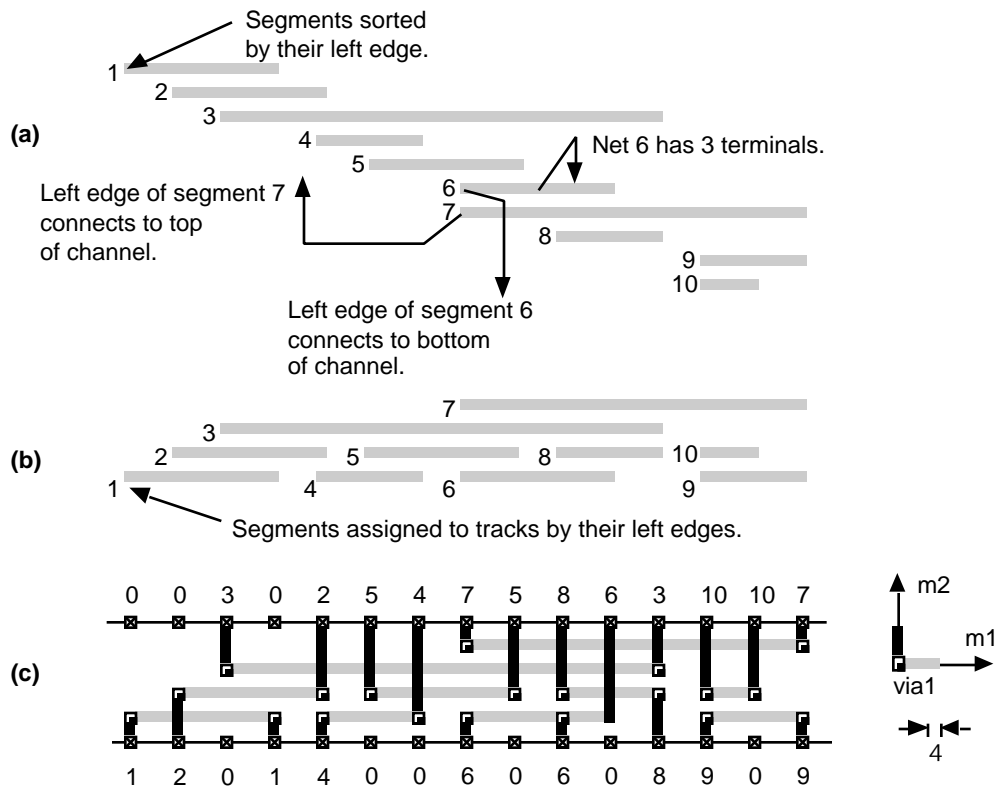
Key terms and concepts: restricted channel-routing problem

17.2.4 Left-Edge Algorithm

Key terms and concepts: left-edge algorithm (LEA)

17.2.5 Constraints and Routing Graphs

Key terms and concepts: vertical constraint • vertical-constraint graph • directed graph • horizontal constraint • horizontal-constraint graph • vertical-constraint cycle (or cyclic constraint) • dogleg router • overlap • overlap capacitance • coupling capacitance • overlap capacitance • channel-routing compaction

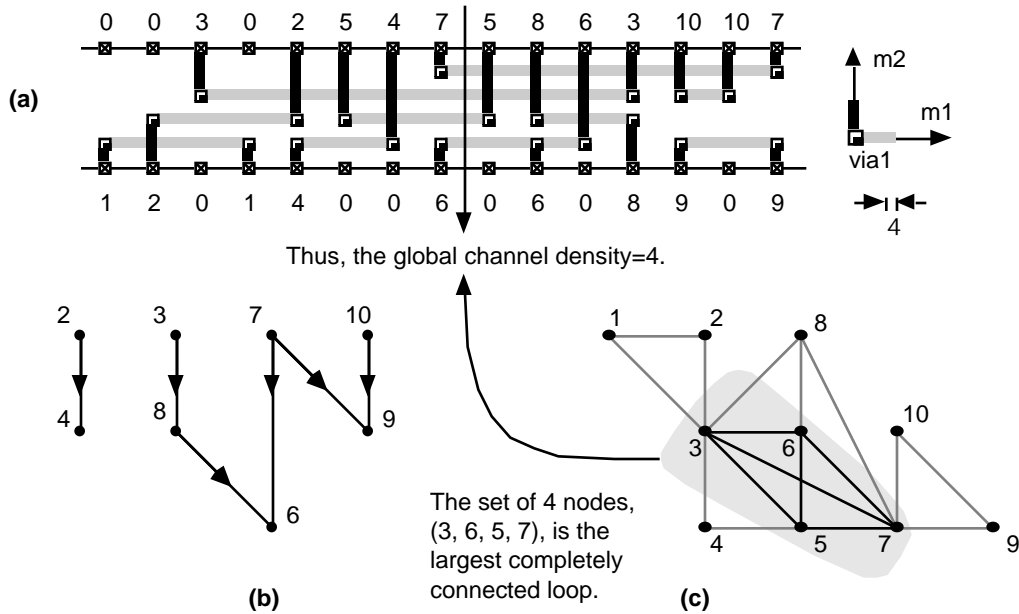


Left-edge algorithm.

(a) Sorted list of segments.

(b) Assignment to tracks.

(c) Completed channel route (with m1 and m2 interconnect represented by lines).



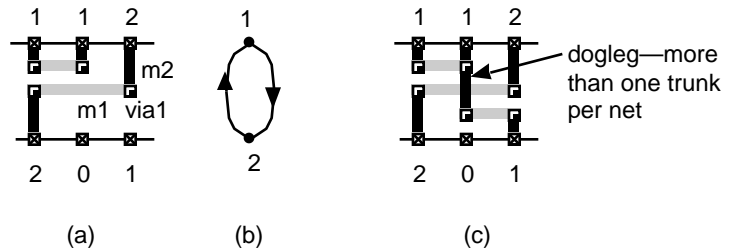
Routing graphs.

(a) Channel with a global density of 4.

(b) The vertical constraint graph. If two nets occupy the same column, the net at the top of the channel imposes a vertical constraint on the net at the bottom. For example, net 2 imposes a vertical constraint on net 4. Thus the interconnect for net 4 must use a track above net 2.

(c) Horizontal-constraint graph. If the segments of two nets overlap, they are connected in the horizontal-constraint graph. This graph determines the global channel density.

The addition of a dogleg, an extra trunk, in the wiring of a net can resolve cyclic vertical constraints.



17.2.6 Area-Routing Algorithms

Key terms and concepts: grid-expansion • maze-running • line-search • **Lee maze-running algorithm** • wave propagation • **Hightower algorithm** • line-search algorithm (or line-probe algorithm) • escape line • escape point

The Lee maze-running algorithm.

The algorithm finds a path from source (X) to target (Y) by emitting a wave from both the source and the target at the same time.

Successive outward moves are marked in each bin.

Once the target is reached, the path is found by backtracking (if there is a choice of bins with equal labeled values, we choose the bin that avoids changing direction).

(The original form of the Lee algorithm uses a single wave.)

Hightower area-routing algorithm.

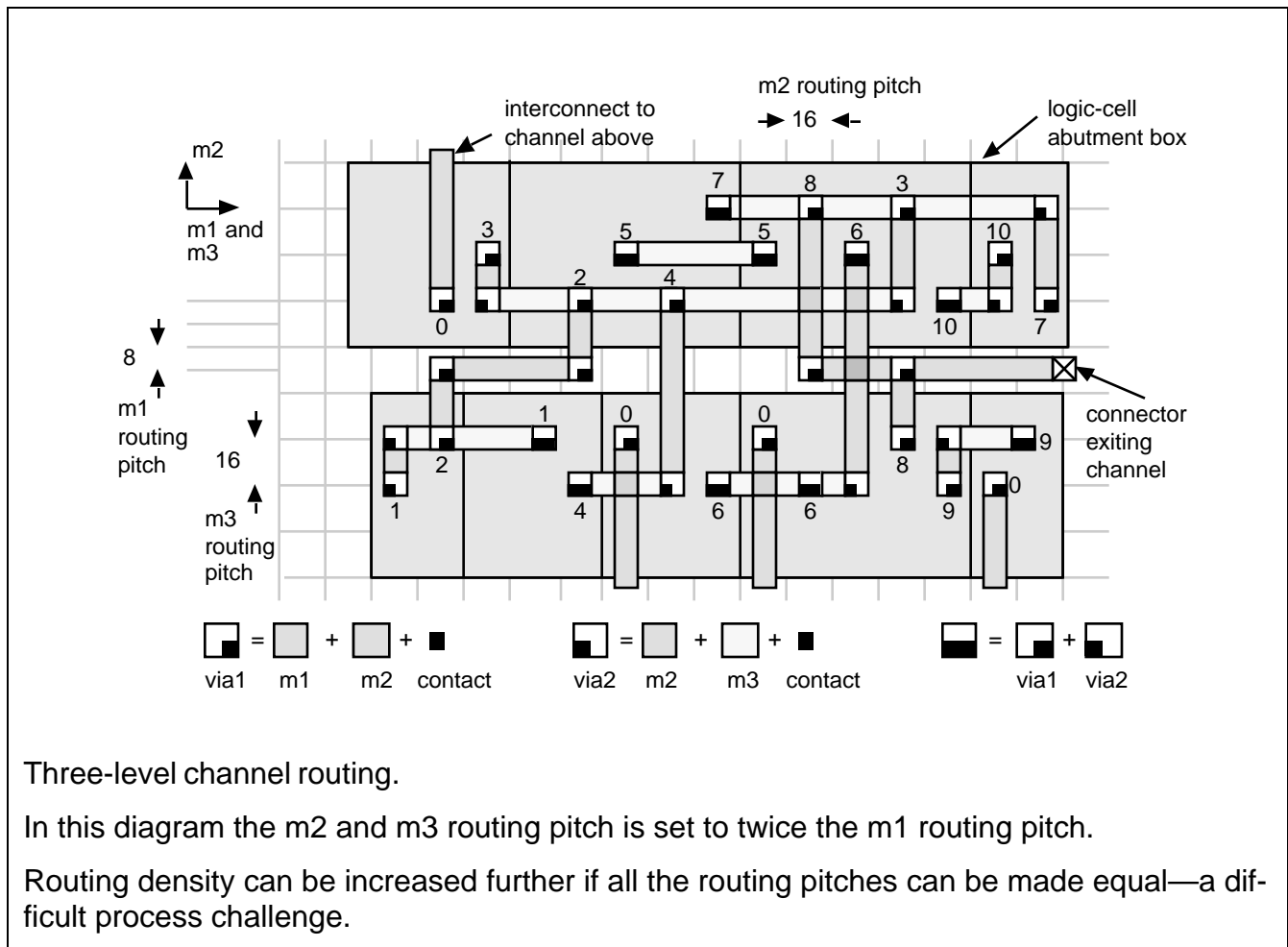
(a) Escape lines are constructed from source (X) and target (Y) toward each other until they hit obstacles.

(b) An escape point is found on the escape line so that the next escape line perpendicular to the original misses the next obstacle.

The path is complete when escape lines from source and target meet.

17.2.7 Multilevel Routing

Key terms and concepts: **two-layer routing** • **2.5-layer routing** • **three-layer routing** • **reserved-layer routing** • **unreserved-layer routing** • **HVH routing** • **VHV routing** • **multilevel routing** • **cell porosity**



Three-level channel routing.

In this diagram the m2 and m3 routing pitch is set to twice the m1 routing pitch.

Routing density can be increased further if all the routing pitches can be made equal—a difficult process challenge.

17.2.8 Timing-Driven Detailed Routing

Key terms and concepts: the global router has already set the path the interconnect will follow and little can be done to improve timing • reduce the number of vias • alter the interconnect width to optimize delay • minimize overlap capacitance • gains are small • high-frequency clock nets are **chamfered** (rounded) to match impedances at branches and control reflections at corners.

17.2.9 Final Routing Steps

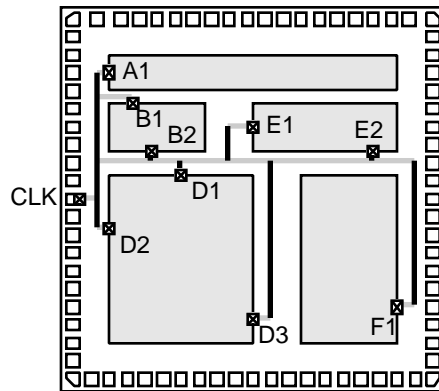
Key terms and concepts: unroutes • rip-up and reroute • engineering change orders (ECO) • via removal • routing compaction

17.3 Special Routing

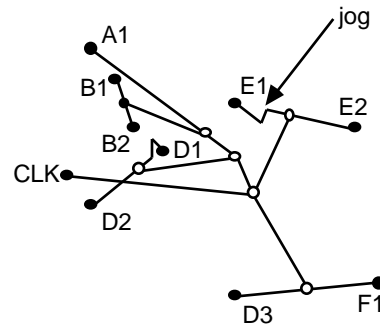
Key terms and concepts: clock and power nets

17.3.1 Clock Routing

Key terms and concepts: **clock-tree synthesis • clock-buffer insertion • activity-induced clock skew**



(a)



(b)

Clock routing.

(a) A clock network for a cell-based ASIC.

(b) Equalizing the interconnect segments between CLK and all destinations (by including jogs if necessary) minimizes clock skew.

17.3.2 Power Routing

Key terms and concepts: power-bus sizing • metal electromigration • power simulation • mean time to failure (MTTF) • metallization reliability rules • maximum metal-width rules (fat-metal rules) • die attach • power grid • end-cap cells • routing bias • flip and abut

Metallization reliability rules for a typical 0.5 micron ($\lambda=0.25\mu\text{m}$) CMOS process.

Layer/contact/via	Current limit	Metal thickness	Resistance
m1	$1\text{mA } \mu\text{m}^{-1}$	7000Å	95m /square
m2	$1\text{mA } \mu\text{m}^{-1}$	7000Å	95m /square
m3	$2\text{mA } \mu\text{m}^{-1}$	12,000Å	48m /square
0.8 μm square m1 contact to diffusion	0.7 mA		11
0.8 μm square m1 contact to poly	0.7mA		16
0.8 μm square m1/m2 via (via1)	0.7mA		3.6
0.8 μm square m2/m3 via (via2)	0.7mA		3.6

17.4 Circuit Extraction and DRC

Key terms and concepts: circuit-extraction • design-rule check • Dracula deck • design rule violations

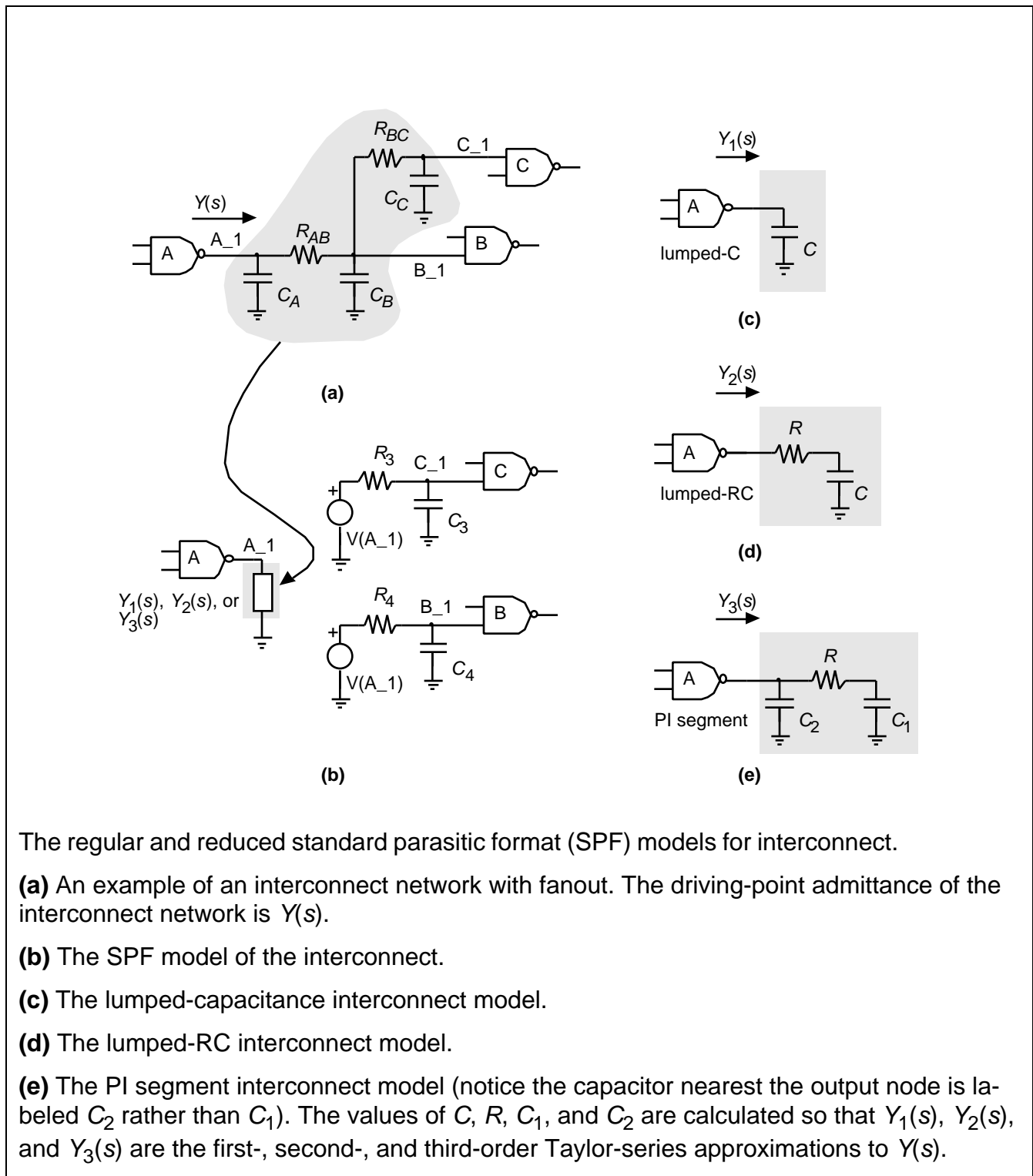
17.4.1 SPF, RSPF, and DSPF

Key terms and concepts: standard parasitic format (SPF) • regular SPF • reduced SPF • detailed SPF

Parasitic capacitances for a typical 1 μm (=0.5 μm) three-level metal CMOS process.

Element	Area/fF μm^{-2}	Fringing/fF μm^{-1}
poly (over gate oxide) to substrate	1.73	NA
poly (over field oxide) to substrate	0.058	0.043
m1 to diffusion or poly	0.055	0.049
m1 to substrate	0.031	0.044
m2 to diffusion	0.019	0.038
m2 to substrate	0.015	0.035
m2 to poly	0.022	0.040
m2 to m1	0.035	0.046
m3 to diffusion	0.011	0.034
m3 to substrate	0.010	0.033
m3 to poly	0.012	0.034
m3 to m1	0.016	0.039
m3 to m2	0.035	0.049
<i>n</i> + junction (at 0V bias)	0.36	NA
<i>p</i> + junction (at 0V bias)	0.46	NA

```
#Design Name : EXAMPLE1
#Date : 6 August 1995
#Time : 12:00:00
#Resistance Units : 1 ohms
#Capacitance Units : 1 pico farads
#Syntax :
#N <netName>
#C <capVal>
# F <from CompName> <fromPinName>
# GC <conductance>
# |
# REQ <res>
# GRC <conductance>
# T <toCompName> <toPinName> RC <rcConstant> A <value>
# |
```



The regular and reduced standard parasitic format (SPF) models for interconnect.

(a) An example of an interconnect network with fanout. The driving-point admittance of the interconnect network is $Y(s)$.

(b) The SPF model of the interconnect.

(c) The lumped-capacitance interconnect model.

(d) The lumped-RC interconnect model.

(e) The PI segment interconnect model (notice the capacitor nearest the output node is labeled C_2 rather than C_1). The values of C , R , C_1 , and C_2 are calculated so that $Y_1(s)$, $Y_2(s)$, and $Y_3(s)$ are the first-, second-, and third-order Taylor-series approximations to $Y(s)$.

```
# RPI <res>
# C1 <cap>
# C2 <cap>
```



```

# GPI <conductance>
# T <toCompName> <toPinName> RC <rcConstant> A <value>
# TIMING.ADMITTANCE.MODEL = PI
# TIMING.CAPACITANCE.MODEL = PP
N CLOCK
C 3.66
  F ROOT Z
  RPI 8.85
  C1 2.49
  C2 1.17
  GPI = 0.0
  T DF1 G RC 22.20
  T DF2 G RC 13.05

* Design Name : EXAMPLE1
* Date : 6 August 1995
* Time : 12:00:00
* Resistance Units : 1 ohms
* Capacitance Units : 1 pico farads
* | RSPF 1.0
* | DELIMITER "_"
.SUBCKT EXAMPLE1 OUT IN
* | GROUND_NET VSS
* TIMING.CAPACITANCE.MODEL = PP
* | NET CLOCK 3.66PF
* | DRIVER ROOT_Z ROOT Z
* | S (ROOT_Z_OUTP1 0.0 0.0)
R2 ROOT_Z ROOT_Z_OUTP1 8.85
C1 ROOT_Z_OUTP1 VSS 2.49PF
C2 ROOT_Z VSS 1.17PF
* | LOAD DF2_G DF1 G
* | S (DF1_G_INP1 0.0 0.0)
E1 DF1_G_INP1 VSS ROOT_Z VSS 1.0
R3 DF1_G_INP1 DF1_G 22.20
C3 DF1_G VSS 1.0PF
* | LOAD DF2_G DF2 G
* | S (DF2_G_INP1 0.0 0.0)
E2 DF2_G_INP1 VSS ROOT_Z VSS 1.0
R4 DF2_G_INP1 DF2_G 13.05
C4 DF2_G VSS 1.0PF
*Instance Section
XDF1 DF1_Q DF1_QN DF1_D DF1_G DF1_CD DF1_VDD DF1_VSS DFF3
XDF2 DF2_Q DF2_QN DF2_D DF2_G DF2_CD DF2_VDD DF2_VSS DFF3
XROOT ROOT_Z ROOT_A ROOT_VDD ROOT_VSS BUF

```

```

.ENDS
.END

.SUBCKT BUFFER OUT IN
* Net Section
* |GROUND_NET VSS
* |NET IN 3.8E-01PF
* |P (IN I 0.0 0.0 5.0)
* |I (INV1:A INV A I 0.0 10.0 5.0)
C1 IN VSS 1.1E-01PF
C2 INV1:A VSS 2.7E-01PF
R1 IN INV1:A 1.7E00
* |NET OUT 1.54E-01PF
* |S (OUT:1 30.0 10.0)
* |P (OUT O 0.0 30.0 0.0)
* |I (INV:OUT INV1 OUT O 0.0 20.0 10.0)
C3 INV1:OUT VSS 1.4E-01PF
C4 OUT:1 VSS 6.3E-03PF
C5 OUT VSS 7.7E-03PF
R2 INV1:OUT OUT:1 3.11E00
R3 OUT:1 OUT 3.03E00
*Instance Section
XINV1 INV:A INV1:OUT INV
.ENDS

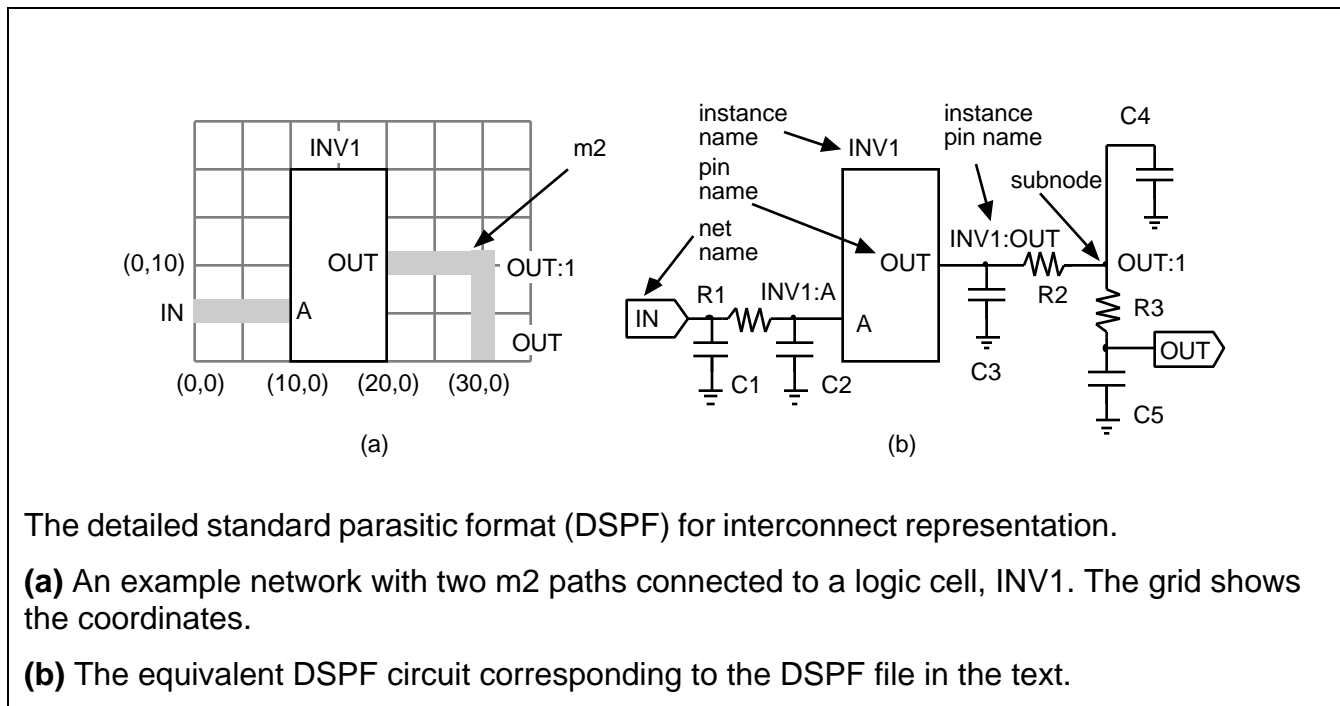
```

17.4.2 Design Checks

Key terms and concepts: design-rule check (DRC) • phantom-level DRC • hard layout • Dracula deck • layout versus schematic (LVS)

17.4.3 Mask Preparation

Key terms and concepts: maskwork symbol (M inside a circle) • copyright symbol (C inside a circle) • kerf • scribe lines • edge-seal structures • Caltech Intermediate Format (CIF, a public domain text format) • GDSII Stream (Calma Stream, Cadence Stream) • fab • mask shop • grace value • sizing or mask tooling • tooling specification • mask bias • bird's beak effect • glass masks or reticles • spot size • critical layers • optical proximity correction (OPC)



17.5 Summary

Key terms and concepts:

- Routing is divided into global and detailed routing.
- Routing algorithms should match the placement algorithms.
- Routing is not complete if there are unroutes.
- Clock and power nets are handled as special cases.
- Clock-net widths and power-bus widths must usually be set by hand.
- DRC and LVS checks are needed before a design is complete.