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  

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

http://spectra.eng.hawaii.edu/~msmith/ASICS/HTML/ASICS.htm
INTRODUCTION
TO ASICs

An ASIC ("a-sick") is an application-specific integrated circuit

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

History of integration: small-scale integration (SSI, \( \approx 10 \) gates per chip, 60's), medium-scale integration (MSI, \( \approx 100-1000 \) gates per chip, 70's), large-scale integration (LSI, \( \approx 1000-10,000 \) gates per chip, 80's), very large-scale integration (VLSI, \( \approx 10,000-100,000 \) gates per chip, 90's), ultralarge scale integration (ULSI, \( \approx 1M-10M \) 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 \( \lambda \) 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 • An erasable PLD (EPLD) • electrically erasable PROM, or EEPROM • An erasable PLD (EPLD) • electrically erasable PROM, or EEPROM • An erasable PLD (EPLD) • electrically erasable PROM, or EEPROM • 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
ASIC design flow. Steps 1–4 are logical design, and steps 5–9 are physical design.

<table>
<thead>
<tr>
<th>The ASICs in the Sun Microsystems SPARCstation 1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SPARCstation 1 ASIC</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
</tbody>
</table>
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 μm, 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**):

\[
\text{total product cost} = \text{fixed product cost} + \text{variable product cost} \times \text{products sold}
\]

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

\[
\text{total part cost} = \text{fixed part cost} + \text{variable cost per part} \times \text{volume of parts}
\]

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

![Break-even graph](image-url)
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

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

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

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

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

<table>
<thead>
<tr>
<th>Types of ASIC</th>
<th>Family member</th>
<th>Custom mask layers</th>
<th>Custom logic cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full-custom</td>
<td>Analog/digital</td>
<td>All</td>
<td>Some</td>
</tr>
<tr>
<td>Semicustom</td>
<td>Cell-based (CBIC)</td>
<td>All</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Masked gate array (MGA)</td>
<td>Some</td>
<td>None</td>
</tr>
<tr>
<td>Programmable</td>
<td>Field-programmable gate array (FPGA)</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Programmable logic device (PLD)</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

1.7 Problems

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

1.8 Bibliography

1.9 References


CMOS LOGIC

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 λ

• 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

An n-channel transistor • channel • source • drain • depletion region • gate • bulk

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 \( I_{DSn} = \frac{Q}{t_f} \)

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

- \( \mu_n \) is the electron mobility (\( \mu_p \) is the hole mobility)
- \( \mathbf{E} \) is the electric field (units Vm\(^{-1}\))

\[
t_f = \frac{L}{V_x} = \frac{L^2}{\mu_n V_{DS}}
\]
\[ Q = \frac{C}{Q} \left( V_{GC} - V_{tn} \right) = C \left[ (V_{GS} - V_{tn}) - 0.5 \, V_{DS} \right] = WLC_{ox} \left[ (V_{GS} - V_{tn}) - 0.5 \, V_{DS} \right] \]

\[ I_{DSn} = \frac{Q}{t_f} \]
\[ = \frac{(W/L)\mu_nC_{ox} \left[ (V_{GS} - V_{tn}) - 0.5 \, V_{DS} \right] V_{DS}}{(W/L)k_n' \left[ (V_{GS} - V_{tn}) - 0.5 \, V_{DS} \right] V_{DS}} \]

\[ k_n' = \mu_nC_{ox} \] is the process transconductance parameter (or intrinsic transconductance)

\[ \beta_n = k_n'(W/L) \] 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, \( I_{DSn(sat)} \)

\[ I_{DSn(sat)} = \frac{(\beta_n/2)(V_{GS} - V_{tn})^2}{V_{GS} > V_{tn}} \]
2.1.1 P-Channel Transistors

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

\[ I_{DSp(sat)} = -\beta_p/2(V_{GS} - V_{tp})^2; \quad V_{DS} < V_{GS} - V_{tp}. \]

- \( V_{tp} \) is negative
- \( V_{DS} \) and \( V_{GS} \) are normally negative (and \(-3V<-2V\))
2.1.2 Velocity Saturation

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

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

2.1.3 SPICE Models

- $K_P \ (\text{in } \mu \text{A/V}^2) = k'_n (k'_p)$
- $V_{T0}$ and $T_OX = V_{tn} (V_{tp})$ and $T_OX$
- $U_0 \ (\text{in } \text{cm}^2\text{V}^{-1}\text{s}^{-1}) = \mu_n \ (\text{and } \mu_p)$

**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
<table>
<thead>
<tr>
<th>Mask/layer name</th>
<th>Derivation from drawn layers</th>
<th>Alternative names for mask/layer</th>
<th>Mask label</th>
</tr>
</thead>
<tbody>
<tr>
<td>n-well</td>
<td>=nwell</td>
<td>bulk, substrate, tub, n-tub, moat</td>
<td>CWN</td>
</tr>
<tr>
<td>p-well</td>
<td>=pwell</td>
<td>bulk, substrate, tub, p-tub, moat</td>
<td>CWP</td>
</tr>
<tr>
<td>active</td>
<td>=pdiff+ndiff</td>
<td>thin oxide, thinox, island, gate oxide</td>
<td>CAA</td>
</tr>
<tr>
<td>polysilicon</td>
<td>=poly</td>
<td>poly, gate</td>
<td>CPG</td>
</tr>
<tr>
<td>n-diffusion</td>
<td>=grow(ndiff)</td>
<td>ndiff, n-select, nplus, n+</td>
<td>CSN</td>
</tr>
<tr>
<td>implant</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p-diffusion</td>
<td>=grow(pdiff)</td>
<td>pdiff, p-select, pplus, p+</td>
<td>CSP</td>
</tr>
<tr>
<td>implant</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>contact</td>
<td>=contact</td>
<td>contact cut, poly contact, diffusion contact</td>
<td>CCP and CCA</td>
</tr>
<tr>
<td>metal1</td>
<td>=m1</td>
<td>first-level metal</td>
<td>CMF</td>
</tr>
<tr>
<td>metal2</td>
<td>=m2</td>
<td>second-level metal</td>
<td>CMS</td>
</tr>
<tr>
<td>via2</td>
<td>=via2</td>
<td>metal2/metal3 via, m2/m3 via</td>
<td>CVS</td>
</tr>
<tr>
<td>metal3</td>
<td>=m3</td>
<td>third-level metal</td>
<td>CMT</td>
</tr>
<tr>
<td>glass</td>
<td>=glass</td>
<td>passivation, overglass, pad</td>
<td>COG</td>
</tr>
<tr>
<td>(a) nwell</td>
<td>(b) pwell</td>
<td>(c) ndiff</td>
<td>(d) pdiff</td>
</tr>
<tr>
<td>(e) poly</td>
<td>(f) contact</td>
<td>(g) m1</td>
<td>(h) via</td>
</tr>
<tr>
<td>(i) m2</td>
<td>(j) cell</td>
<td>(k) phantom</td>
<td></td>
</tr>
</tbody>
</table>

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

The transistor layers
2.2.1 Sheet Resistance

<table>
<thead>
<tr>
<th>Layer</th>
<th>Sheet resistance (1µm)</th>
<th>Units</th>
<th>Layer</th>
<th>Sheet resistance (0.35µm)</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>n-well</td>
<td>1.15± 0.25</td>
<td>kΩ/square</td>
<td>n-well</td>
<td>1 ± 0.4</td>
<td>kΩ/square</td>
</tr>
<tr>
<td>poly</td>
<td>3.5± 2.0</td>
<td>Ω/square</td>
<td>poly</td>
<td>10± 4.0</td>
<td>Ω/square</td>
</tr>
<tr>
<td>n-diffusion</td>
<td>75± 20</td>
<td>Ω/square</td>
<td>n-diffusion</td>
<td>3.5± 2.0</td>
<td>Ω/square</td>
</tr>
<tr>
<td>p-diffusion</td>
<td>140± 40</td>
<td>Ω/square</td>
<td>p-diffusion</td>
<td>2.5± 1.5</td>
<td>Ω/square</td>
</tr>
<tr>
<td>m1/2</td>
<td>70± 6</td>
<td>mΩ/square</td>
<td>m1/2/3</td>
<td>60± 6</td>
<td>mΩ/square</td>
</tr>
<tr>
<td>m3</td>
<td>30± 3</td>
<td>mΩ/square</td>
<td>metal4</td>
<td>30± 3</td>
<td>mΩ/square</td>
</tr>
</tbody>
</table>

Key words: diffusion • Ω/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

Scalable CMOS design rules
2.4 Combinational Logic Cells

The AOI family of cells with three index numbers or less

<table>
<thead>
<tr>
<th>Cell type$^1$</th>
<th>Cells</th>
<th>Number of unique cells</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xa1</td>
<td>X21, X31</td>
<td>2</td>
</tr>
<tr>
<td>Xa11</td>
<td>X211, X311</td>
<td>2</td>
</tr>
<tr>
<td>Xab</td>
<td>X22, X33, X32</td>
<td>3</td>
</tr>
<tr>
<td>Xab1</td>
<td>X221, X331, X321</td>
<td>3</td>
</tr>
<tr>
<td>Xabc</td>
<td>X222, X333, X332, X322</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>14</td>
</tr>
</tbody>
</table>

$^1$Xabc: 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 $\beta_n = \beta_p$ 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

CMOS latch • **enable** • **transparent** • **static** • **sequential logic cell** • **storage** • **initial value**
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):** \[ \text{SUM} = A \oplus B \oplus \text{CIN} = \text{SUM}(A, B, \text{CIN}) = \text{PARITY}(A, B, \text{CIN}), \]
\[ \text{COUT} = A \cdot B + A \cdot \text{CIN} + B \cdot \text{CIN} = \text{MAJ}(A, B, \text{CIN}). \]

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

\[ S[i] = \text{SUM} (A[i], B[i], \text{CIN}) \]
\[ \text{COUT} = \text{MAJ} (A[i], B[i], \text{CIN}) \]

A datapath adder
- **Ripple-carry adder (RCA)**
- **Data** signals • **control** signals • **datapath** • **datapath cell** or **datapath element**
- Datapath advantages: predictable and equal delay for each bit • built-in interconnect
- Disadvantages of a datapath: overhead • harder design • software is more complex
2.6.1 Datapath Elements
## Binary Arithmetic

<table>
<thead>
<tr>
<th>Operation</th>
<th>Binary Number Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Unsigned</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>no change</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>3=</td>
<td>0011</td>
</tr>
<tr>
<td>-3=</td>
<td>NA</td>
</tr>
<tr>
<td>zero=</td>
<td>0000</td>
</tr>
<tr>
<td>max. positive=</td>
<td>1111=15</td>
</tr>
<tr>
<td>max. negative=</td>
<td>0000=0</td>
</tr>
<tr>
<td>addition=</td>
<td>S=A+B</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SG(A)=sign of A</td>
<td></td>
</tr>
<tr>
<td>addition result:</td>
<td>OR=COUT[MSB]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SG(S)=sign of S</td>
<td>NA</td>
</tr>
<tr>
<td>S= A+B</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>subtraction=</td>
<td>D=A−B</td>
</tr>
<tr>
<td>D=</td>
<td></td>
</tr>
<tr>
<td>=minuend</td>
<td></td>
</tr>
<tr>
<td>=subtrahend</td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
- **COUT** is carry out.
- **OV** is overflow.
- **OR** is out of range.
- **SG** stands for Sign of the result.
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**
  
  \[
  G[i] = A[i] \cdot B[i] \\
  C[i] = G[i] + P[i] \cdot C[i-1] \\
  S[i] = A[i] \oplus B[i] \oplus C[i-1]
  \]

**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**
  
  \[
  C1[i]' = P[i] \cdot C3[i-1] \cdot C4[i-1] \\
  C[i] = C1[i] \cdot C2[i]
  \]

- **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 \( \text{CSA}(A1[i], A2[i], A3[i], CIN, S1[i], S2[i], COUT) \) has three outputs:

- \( S1[i] = \text{CIN} \)
- \( S2[i] = A1[i] \oplus A2[i] \oplus A3[i] = \text{PARITY}(A1[i], A2[i], A3[i]) \)
- \( COUT = A1[i] \cdot A2[i] + [(A1[i] + A2[i]) \cdot A3[i]] = \text{MAJ}(A1[i], A2[i], A3[i]) \)
Carry-propagate adder (CPA)

The carry-save adder (CSA) • pipeline • latency • bit slice

carry-bypass adders (CBA):

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

carry-skip adder:

\[ \text{CSKIP}[i] = (G[i] + P[i] \cdot C[i - 1]) \cdot \text{SKIP}' + C[i - 2] \cdot \text{SKIP} \]
Carry-lookahead adder (CLA, for example the Brent–Kung adder):

\[ = G[1] + P[1] \cdot (G[0] + P[1] \cdot C[-1]) \]
\[ = G[1] + P[1] \cdot G[0] \]


**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.
The conditional-sum adder

- $A[i]$, $B[i]$: carry out (carry in = 0)
- $A[i] \oplus B[i]$: sum (carry in = 0)
- $(A[i] \oplus B[i])'$: sum (carry in = 1)

$S_i$, $C_i$: carry in to the $i$th bit assuming the carry in to the $j$th bit is $k (k = 0$ or $1)$

$S_i$: sum at the $i$th bit assuming the carry in to the $j$th bit is $k (k = 0$ or $1)$
2.6.3 A Simple Example

An 8-bit conditional-sum adder

```verilog
module m8bitCSum (C0, a, b, s, C8); // Verilog conditional-sum adder for an FPGA
  input [7:0] C0, a, b; //2
  output [7:0] s; //2
  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) ; //10
assign C4_2_1 = A3&B3|(A3|B3)&(A2|B2) ; //11
assign S5_4_0 = A5^B5^(A4&B4) ; assign S5_4_1 = A5^B5^(A4|B4) ; //12
assign C6_4_0 = A5&B5|(A5|B5)&(A4&B4) ; assign C6_4_1 = A5&B5|(A5|B5)&(A4|B4) ; //13
assign S2 = A2^B2^C2 ; // start of level 2 //14
assign S3 = A3^B3^(A2&B2|(A2|B2)&C2) ; //15
assign S4 = A4^B4^(C4_2_0|C4_2_1&C2) ; //16
assign S5 = S5_4_0& !(C4_2_0|C4_2_1&C2) | S5_4_1&(C4_2_0|C4_2_1&C2) ; //17
assign C6 = C6_4_0|C6_4_1&(C4_2_0|C4_2_1&C2) ; //18
assign S6 = A6^B6^C6 ; // start of level 3 //19
assign S7 = A7^B7^(A6&B6|(A6|B6)&C6) ; //20
assign C8 = A7&B7|(A7|B7s)&(A6&B6|A6|B6&C6) ; endmodule //21
```

2.6.4 Multipliers

• Mental arithmetic: 15 (multiplicand) × 19 (multiplier) = 15×(20–1) = 15×2\[\bar{T}\]
• Suppose we want to multiply by B=00010111 (decimal 16+4+2+1=23)
• Use the canonical signed-digit vector (CSD vector) D=0010\[\bar{T}\]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:  
\[
D_0 = B_0 + C_0 - 2C_1 = 1 + 0 - 2 = 1, \\
D_1 = B_1 + C_1 - 2C_2 = 1 + 1 - 2 = 0, \\
D_2 = B_2 + C_2 - 2C_3 = 0 + 1 - 0 = 1,
\]
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) ⇒ \( E = \bar{1} \bar{2} \bar{1} \) (decimal –16–8+1=–23)
- \( B = 01011 \) (eleven) ⇒ \( E = 1 \bar{1} \bar{1} \) (16–4–1)
- \( B = 101 \) ⇒ \( E = \bar{1} \bar{1} \)
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
A Wallace-tree multiplier works forward from the multiplier inputs

- Full adder is a 3:2 compressor or (3, 2) counter
- Half adder is a (2, 2) counter
The Dadda multiplier works backward from the final product

- Each stage has a maximum of 2, 3, 4, 6, 9, 13, 19, ... outputs (each successive stage is 3/2 times larger—rounded down to an integer)

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

<table>
<thead>
<tr>
<th>binary</th>
<th>decimal</th>
<th>redundant binary</th>
<th>CSD vector</th>
<th>Addend</th>
<th>Augend</th>
<th>Intermediate sum</th>
<th>Intermediate carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>101011</td>
<td>87</td>
<td>10T0T00T</td>
<td>10T0T00T</td>
<td>1010101</td>
<td>1T10011T</td>
<td>01100101</td>
<td></td>
</tr>
<tr>
<td>+ 1100101</td>
<td>101</td>
<td>+ 1T10011T</td>
<td>01001110</td>
<td>1100110</td>
<td>11000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>= 10111100</td>
<td>= 188</td>
<td>1T1000T00</td>
<td>10T00T100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Redundant binary addition • redundant binary encoding avoids carry propagation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>A[i−1]=0/1 and B[i−1]=0/1</td>
<td>T</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>T</td>
<td>A[i−1]=T or B[i−1]=T</td>
<td>1</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>A[i−1]=0/1 and B[i−1]=0/1</td>
<td>T</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>A[i−1]=T or B[i−1]=T</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- 101 (decimal) is 1100101 (in binary and CSD vector) or 1T100111
- 188 (decimal) is 10111100 (in binary), 1T1000T00, 10T00T100, or 10T000T00 (CSD vector)
- 10T is represented as 010010 (using sign magnitude) — rather wasteful

Residue number system

- 11 (decimal) is represented as [1, 2] residue (5, 3)
- 11R₅=11 mod 5=1 and 11R₃=11 mod 3=2
- The size of this system is 3×5=15
- We can now add, subtract, or multiply without using any carry
The 5, 3 residue number system

<table>
<thead>
<tr>
<th>n</th>
<th>residue 5</th>
<th>residue 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>residue 5</th>
<th>residue 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>n</th>
<th>residue 5</th>
<th>residue 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

4 \[4, 1\] + 7 \[2, 1\] = 11 \[1, 2\]
12 \[2, 0\] - 4 \[4, 1\] = 8 \[3, 2\]
3 \[3, 0\] \times 4 \times \[4, 1\] = 12 \[2, 0\]
2.6.6 Other Datapath Operators

Full subtracter

\[ \text{DIFF} = A \oplus \text{NOT}(B) \oplus \text{NOT}(\text{BIN}) \]
\[ = \text{SUM}(A, \text{NOT}(B), \text{NOT}(\text{BIN})) \]
\[ \text{NOT} (\text{BOUT}) = A \cdot \text{NOT}(B) + A \cdot \text{NOT}(\text{BIN}) + \text{NOT}(B) \cdot \text{NOT}(\text{BIN}) \]
\[ = \text{MAJ} (\text{NOT}(A), B, \text{NOT}(\text{BIN})) \]

Symbols for datapath elements

**Keywords:** adder/subtractor • barrel shifter • normalizer • denormalizer • leading-one detector • priority encoder • exponent correcter • accumulator • multiplier–accumulator (MAC) • incremener • decremener • incremener/decremener • 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 $\lambda$

2.10 Problems

Suggested homework: 2.1, 2.2, 2.38, 2.39 (from ASICs... the book)
ASIC LIBRARY DESIGN

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

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

An output trip point of 0.35 is convenient because \( \ln(1/0.35) = 1.04 \approx 1 \) 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, \( v(\text{in}1) \) and output, \( v(\text{out}1) \)
- 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, \( t_{PDf} \approx R_{pd}(C_p + C_{out}) \)
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
<table>
<thead>
<tr>
<th>NAME</th>
<th>m1</th>
<th>m2</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODEL</td>
<td>CMOSN</td>
<td>CMOSP</td>
</tr>
<tr>
<td>ID</td>
<td>7.49E-11</td>
<td>-7.49E-11</td>
</tr>
<tr>
<td>VGS</td>
<td>0.00E+00</td>
<td>-3.00E+00</td>
</tr>
<tr>
<td>VDS</td>
<td>3.00E+00</td>
<td>-4.40E-08</td>
</tr>
<tr>
<td>VBS</td>
<td>0.00E+00</td>
<td>0.00E+00</td>
</tr>
<tr>
<td>VTH</td>
<td>4.14E-01</td>
<td>-8.96E-01</td>
</tr>
<tr>
<td>VDSAT</td>
<td>3.51E-02</td>
<td>-1.78E+00</td>
</tr>
<tr>
<td>GM</td>
<td>1.75E-09</td>
<td>2.52E-11</td>
</tr>
<tr>
<td>GDS</td>
<td>1.24E-10</td>
<td>1.72E-03</td>
</tr>
<tr>
<td>GMB</td>
<td>6.02E-10</td>
<td>7.02E-12</td>
</tr>
<tr>
<td>CBD</td>
<td>2.06E-15</td>
<td>1.71E-14</td>
</tr>
<tr>
<td>CBS</td>
<td>4.45E-15</td>
<td>1.71E-14</td>
</tr>
<tr>
<td>CGSOV</td>
<td>1.80E-15</td>
<td>2.88E-15</td>
</tr>
<tr>
<td>CGDOV</td>
<td>1.80E-15</td>
<td>2.88E-15</td>
</tr>
<tr>
<td>CGBOV</td>
<td>2.00E-16</td>
<td>2.01E-16</td>
</tr>
<tr>
<td>CGS</td>
<td>0.00E+00</td>
<td>1.10E-14</td>
</tr>
<tr>
<td>CGD</td>
<td>0.00E+00</td>
<td>1.10E-14</td>
</tr>
<tr>
<td>CGB</td>
<td>3.88E-15</td>
<td>0.00E+00</td>
</tr>
</tbody>
</table>

- \( I_D \) (\( I_{DS} \)), \( V_G \), \( V_D \), \( V_B \) (\( V_{th} \)), and \( V_{DS\text{sat}} \) (\( V_{DS\text{(sat)}} \)) are DC parameters
- \( G_M \), \( G_D \), and \( G_{MB} \) are small-signal conductances (corresponding to \( \frac{\partial I_{DS}}{\partial V_{GS}} \), \( \frac{\partial I_{DS}}{\partial V_{DS}} \), and \( \frac{\partial I_{DS}}{\partial V_{BS}} \), respectively)
Calculations of parasitic capacitances for an n-channel MOS transistor.

<table>
<thead>
<tr>
<th>PSpice</th>
<th>Equation</th>
<th>Values for $V_{GS}=0V, V_{DS}=3V, V_{SB}=0V$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CBD</strong></td>
<td>$C_{BD} = C_{BDJ} + C_{BDSW}$</td>
<td>$C_{BD} = 1.855 \times 10^{-13} + 2.04 \times 10^{-16} = 2.06 \times 10^{-13}$ F</td>
</tr>
<tr>
<td></td>
<td>$C_{BDJ} + A_{D} C_{J} (1 + V_{DB}/\phi_{B})^{-m_{J}} (\phi_{B} = P_{B})$</td>
<td>$C_{BDJ} = (4.032 \times 10^{-15})(1 + (3/1))^{-0.56} = 1.86 \times 10^{-15}$ F</td>
</tr>
<tr>
<td></td>
<td>$C_{BDSW} = P_{D} C_{JSW} (1 + V_{DB}/\phi_{B})^{-m_{JSW}} (P_{D}$ may or may not include channel edge)</td>
<td>$C_{BDSW} = (4.2 \times 10^{-16})(1 + (3/1))^{-0.5} = 2.04 \times 10^{-16}$ F</td>
</tr>
<tr>
<td><strong>CBS</strong></td>
<td>$C_{BS} = C_{BSJ} + C_{BSSW}$</td>
<td>$C_{BS} = 4.032 \times 10^{-15} + 4.2 \times 10^{-16} = 4.45 \times 10^{-15}$ F</td>
</tr>
<tr>
<td></td>
<td>$C_{BSJ} + A_{S} C_{J} (1 + V_{SB}/\phi_{B})^{-m_{J}}$</td>
<td>$A_{S} C_{J} = (7.2 \times 10^{-15})(5.6 \times 10^{-4}) = 4.03 \times 10^{-15}$ F</td>
</tr>
<tr>
<td></td>
<td>$C_{BSSW} = P_{S} C_{JSW} (1 + V_{SB}/\phi_{B})^{-m_{JSW}}$</td>
<td>$P_{S} C_{JSW} = (8.4 \times 10^{-6})(5 \times 10^{-11}) = 4.2 \times 10^{-16}$ F</td>
</tr>
<tr>
<td><strong>CGSOV</strong></td>
<td>$C_{GSOV} = W_{EFF} C_{GSO}$</td>
<td>$C_{GSOV} = (6 \times 10^{-6})(3 \times 10^{-10}) = 1.8 \times 10^{-16}$ F</td>
</tr>
<tr>
<td><strong>CGDOV</strong></td>
<td>$C_{GDOV} = W_{EFF} C_{GSO}$</td>
<td>$C_{GDOV} = (6 \times 10^{-6})(3 \times 10^{-10}) = 1.8 \times 10^{-15}$ F</td>
</tr>
<tr>
<td><strong>CGBOV</strong></td>
<td>$C_{GBOV} = L_{EFF} C_{GBO}$</td>
<td>$C_{GDOV} = (0.5 \times 10^{-6})(4 \times 10^{-10}) = 2 \times 10^{-16}$ F</td>
</tr>
<tr>
<td><strong>CGS</strong></td>
<td>$C_{GS}/C_{O} = 0$ (off), 0.5 (lin.), 0.66 (sat.)</td>
<td>$C_{O} = (6 \times 10^{-6})(0.5 \times 10^{-6})(0.00345) = 1.03 \times 10^{-14}$ F</td>
</tr>
<tr>
<td></td>
<td>$C_{O}$ (oxide capacitance) = $W_{EFF} L_{EFF} \varepsilon_{ox} / T_{ox}$</td>
<td>$C_{GS} = 0.0$ F</td>
</tr>
<tr>
<td><strong>CGD</strong></td>
<td>$C_{GD}/C_{O} = 0$ (off), 0.5 (lin.), 0 (sat.)</td>
<td>$C_{GD} = 0.0$ F</td>
</tr>
<tr>
<td><strong>CGB</strong></td>
<td>$C_{GB} = 0$ (on), = $C_{O}$ in series with $C_{GS}$ (off)</td>
<td>$C_{GB} = 3.88 \times 10^{-15}$ F, $C_{S}$ = depletion capacitance</td>
</tr>
</tbody>
</table>

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, \( v(\text{in1}) \), of an inverter
- Data points are joined by straight lines
- Note that \( CGSOV = CGDOV \)
3.2.4 Input Slew Rate

Measuring the input capacitance of an inverter

(a) Input capacitance is measured by monitoring the input current to the inverter, $i(V_{\text{in}})$

(b) Very fast (non-equilibrium) switching: input current of 40fA = input capacitance of 40fF

(c) Very slow (equilibrium) switching: input capacitance is now equal for both transitions
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\text{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 \( s \):

\[
t_{PD} = \left( \frac{R}{s} \right) (C_{out} + sC_p) + st_q
\]

\[
t_{PD} = \frac{C_{out}}{RC} + RC_p + st_q
\]

Normalizing the delay:

\[
d = \frac{(RC) (C_{out} / C_{in}) + RC_p + st_q}{\tau} = f + p + q
\]

The time constant \( \tau = R_{inv} C_{inv} \), 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 \( C_{inv} \)
Logical effort • For a two-input NAND cell, the logical effort, $g = 4/3$

(a) Find the input capacitance, $C_{\text{inv}}$, 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 $C_{\text{in}}$

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

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

**electrical effort $h = C_{\text{out}} / C_{\text{in}}$**

**parasitic delay $p = RC_{\text{p}} / \tau$** (the parasitic delay of a minimum-size inverter is: $p_{\text{inv}} = C_{\text{p}} / C_{\text{inv}}$)

**nonideal delay $q = st_{\text{q}} / \tau$**

| Cell                | Cell effort (logic ratio=2) | Cell effort (logic ratio=r) | Parasitic delay/$\tau$ | Nonideal delay/$\tau$
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>inverter</td>
<td>1 (by definition)</td>
<td>1 (by definition)</td>
<td>$p_{\text{inv}}$ (by definition)</td>
<td>$q_{\text{inv}}$ (by definition)</td>
</tr>
<tr>
<td>n-input NAND</td>
<td>$(n+2)/3$</td>
<td>$(n+r)/(r+1)$</td>
<td>$np_{\text{inv}}$</td>
<td>$nq_{\text{inv}}$</td>
</tr>
<tr>
<td>n-input NOR</td>
<td>$(2n+1)/3$</td>
<td>$(nr+1)/(r+1)$</td>
<td>$np_{\text{inv}}$</td>
<td>$nq_{\text{inv}}$</td>
</tr>
</tbody>
</table>
### 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)
- \( p = 3p_{\text{inv}} \) and \( q = 3q_{\text{inv}} \) for this cell
- the input gate capacitance of a 1X drive, three-input NOR logic cell is equal to \( gC_{\text{inv}} \)
- for a 2X logic cell, \( C_{\text{in}} = 2gC_{\text{inv}} \)

\[
g h = \frac{C_{\text{out}}}{C_{\text{in}}} = \frac{g \cdot (0.3 \text{ pF})}{2gC_{\text{inv}}} = \frac{(0.3 \text{ pF})}{(2) \cdot (0.036 \text{ pF})}
\]

(Notice \( g \) cancels out in this equation)

The delay of the NOR logic cell, in units of \( \tau \), is thus

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

\[
= 4.1666667 + 3 + 5.1
\]

\[
= 12.266667 \tau \text{ equivalent to an absolute delay, } t_{PD} = 12.3 \times 0.06 \text{ns} = 0.74 \text{ns}
\]

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

With \( C_{\text{out}} = 0.3 \text{pF} \), \( t_{PD} = 0.03 + (0.72) \cdot (0.3) + 0.60 = 0.846 \text{ ns} \) 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

**Path Delay**

\[
D = \sum_{i \in \text{path}} g_i h_i + \sum_{i \in \text{path}} (p_i + q_i)
\]

### 3.3.4 Multistage Cells

**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, \(d_1 = 18.8 + C_L\)

(b) is slightly faster than (a)
3.3.5 Optimum Delay

Path logical effort: \[ G = \prod_{i \in \text{path}} g_i \]

Path electrical effort: \[ H = \prod_{i \in \text{path}} h_i \]

\( C_{out} \) 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: \[ D^* = NF^{1/N} = N(GH)^{1/N} + P + Q \]

\[ P + Q = \sum_{i \in \text{path}} p_i + h_i \]

3.3.6 Optimum Number of Stages

<table>
<thead>
<tr>
<th>Stage effort</th>
<th>delay/(ln H) = h/(ln h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h )</td>
<td>( h/(\ln h) )</td>
</tr>
<tr>
<td>1.5</td>
<td>3.7</td>
</tr>
<tr>
<td>2</td>
<td>2.9</td>
</tr>
<tr>
<td>2.7</td>
<td>2.7</td>
</tr>
<tr>
<td>3</td>
<td>2.7</td>
</tr>
<tr>
<td>4</td>
<td>2.9</td>
</tr>
<tr>
<td>5</td>
<td>3.1</td>
</tr>
<tr>
<td>10</td>
<td>4.3</td>
</tr>
</tbody>
</table>

- 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 $h = e = 2.718$

• 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

Cell library statistics
- 80 percent of an ASIC uses less than 20 percent of the cell library
- **Cell importance**
  - A D flip-flop (with a cell importance of 3.5) contributes 3.5 times as much area on a typical ASIC than does an inverter (with a cell importance of 1)
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
An oxide-isolated gate-array base cell

- 14 tracks high and 4 tracks wide
- VDD (tracks 3 and 4) and GND (tracks 11 and 12) are each 2 tracks wide
- 10 horizontal routing tracks (tracks 1, 2, 5–10, 13, 14)—unusually large number for modern cells
- p-channel and n-channel polysilicon bent gates are tied together in the center of the cell
- The well contacts leave room for a poly cross-under in each base cell.
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 processes)
A simple gate-array base cell
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 1.0μm 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 datapath D flip-flop cell
The schematic of a datapath D flip-flop cell

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

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 FPGAs

<table>
<thead>
<tr>
<th>Device</th>
<th>Antifuses</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1010</td>
<td>112,000</td>
</tr>
<tr>
<td>A1020</td>
<td>186,000</td>
</tr>
<tr>
<td>A1225</td>
<td>250,000</td>
</tr>
<tr>
<td>A1240</td>
<td>400,000</td>
</tr>
<tr>
<td>A1280</td>
<td>750,000</td>
</tr>
</tbody>
</table>

The resistance of blown Actel antifuses
4.1.1 Metal–Metal Antifuse

Metal–metal antifuse

QuickLogic metal–metal antifuse (ViaLink) • alloy of tungsten, titanium, and silicon • bulk resistance of about 500 mΩ cm

Resistance values for the QuickLogic metal–metal antifuse
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 (>12V) 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](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

<table>
<thead>
<tr>
<th>Item Code</th>
<th>Description</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Actel</td>
<td>ATT</td>
<td>AT&amp;T (Lucent)</td>
</tr>
<tr>
<td>XC</td>
<td>Xilinx</td>
<td>isp</td>
<td>Lattice Logic</td>
</tr>
<tr>
<td>EPM</td>
<td>Altera MAX</td>
<td>M5</td>
<td>AMD MACH 5 is on the device</td>
</tr>
<tr>
<td>EPF</td>
<td>Altera FLEX</td>
<td>QL</td>
<td>QuickLogic</td>
</tr>
<tr>
<td>CY7C</td>
<td>Cypress</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Package Type</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL or PC</td>
<td>VQ</td>
<td>very thin quad flatpack, VQFP</td>
</tr>
<tr>
<td>PQ</td>
<td>TQ</td>
<td>thin plastic flatpack, TQFP</td>
</tr>
<tr>
<td>CQ or PB</td>
<td>PP</td>
<td>plastic pin-grid array, PPGA</td>
</tr>
<tr>
<td>PG</td>
<td>WB/PB</td>
<td>ball-grid array, BGA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>B</td>
<td>MIL-STD-883</td>
</tr>
<tr>
<td>I</td>
<td>E</td>
<td>extended</td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Actel part</td>
<td>1H92 base price</td>
<td>Xilinx part</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------</td>
<td>----------------</td>
</tr>
<tr>
<td>A1010A-PL44C</td>
<td>$23.25</td>
<td>XC3020-50PC68C</td>
</tr>
<tr>
<td>A1020A-PL44C</td>
<td>$43.30</td>
<td>XC3030-50PC44C</td>
</tr>
<tr>
<td>A1225-PQ100C</td>
<td>$105.00</td>
<td>XC3042-50PC84C</td>
</tr>
<tr>
<td>A1240-PQ144C</td>
<td>$175.00</td>
<td>XC3064-50PC84C</td>
</tr>
<tr>
<td>A1280-PQ160C</td>
<td>$305.00</td>
<td>XC3090-50PC84C</td>
</tr>
</tbody>
</table>
### 4.7.1 FPGA Pricing

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

<table>
<thead>
<tr>
<th>Actel price adjustment factors</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purchase quantity, all types</strong></td>
</tr>
<tr>
<td>(1–9)</td>
</tr>
<tr>
<td>100%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Purchase time, in (100–999) quantity</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1H92</td>
</tr>
<tr>
<td>100%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Qualification type, same package</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Commercial</td>
</tr>
<tr>
<td>100%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Speed bin</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>ACT 1-Std</td>
</tr>
<tr>
<td>100%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Package type</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A1010:</td>
</tr>
<tr>
<td>100%</td>
</tr>
<tr>
<td>A1020:</td>
</tr>
<tr>
<td>100%</td>
</tr>
<tr>
<td>A1225:</td>
</tr>
<tr>
<td>100%</td>
</tr>
<tr>
<td>A1240:</td>
</tr>
<tr>
<td>100%</td>
</tr>
<tr>
<td>A1280:</td>
</tr>
<tr>
<td>100%</td>
</tr>
</tbody>
</table>

1Actel 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.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Example</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base price</td>
<td>A1020A</td>
<td>$43.30</td>
</tr>
<tr>
<td>Quantity</td>
<td>100–999</td>
<td>84%</td>
</tr>
<tr>
<td>Time</td>
<td>1H92</td>
<td>100%</td>
</tr>
<tr>
<td>Qualification type</td>
<td>Industrial (I)</td>
<td>120%</td>
</tr>
<tr>
<td>Speed bin(^1)</td>
<td>2</td>
<td>140%</td>
</tr>
<tr>
<td>Package</td>
<td>PQ100</td>
<td>125%</td>
</tr>
<tr>
<td>Estimated price (1H92)</td>
<td></td>
<td>$76.38</td>
</tr>
<tr>
<td>Actual Actel price (1H92)</td>
<td></td>
<td>$75.60</td>
</tr>
</tbody>
</table>

\(^1\)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](http://marshall.com), carry Xilinx
- Hamilton-Avnet, at [http://www.hh.avnet.com](http://www.hh.avnet.com), carry Xilinx
- Wyle, at [http://www.wyle.com](http://www.wyle.com) carries Actel and Altera
4.8 Summary

<table>
<thead>
<tr>
<th>Programmable ASIC technologies</th>
<th>Actel</th>
<th>Xilinx LCA&lt;sup&gt;1&lt;/sup&gt;</th>
<th>Altera EPLD</th>
<th>Xilinx EPLD</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Programming technology</strong></td>
<td>Poly–diffusion antifuse, PLICE</td>
<td>Erasable SRAM ISP</td>
<td>UV-erasable EPROM (MAX 5k)</td>
<td>UV-erasable EPROM</td>
</tr>
<tr>
<td><strong>Size of programming element</strong></td>
<td>Small but requires contacts to metal</td>
<td>Two inverters plus pass and switch devices. Largest.</td>
<td>One n-channel EPROM device. Medium.</td>
<td>One n-channel EPROM device. Medium.</td>
</tr>
<tr>
<td><strong>Process</strong></td>
<td>Special: CMOS plus three extra masks.</td>
<td>Standard CMOS and EEPROM</td>
<td>Standard EPROM</td>
<td>Standard EPROM</td>
</tr>
<tr>
<td><strong>Programming method</strong></td>
<td>Special hardware</td>
<td>PC card, PROM, or serial port</td>
<td>ISP (MAX 9k) or EPROM programmer</td>
<td>EPROM programmer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programming technology</th>
<th>QuickLogic</th>
<th>Crosspoint</th>
<th>Atmel</th>
<th>Altera FLEX</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Programming technology</strong></td>
<td>Metal–metal antifuse, ViaLink</td>
<td>Metal–polysilicon antifuse</td>
<td>Erasable SRAM. ISP.</td>
<td>Erasable SRAM. ISP.</td>
</tr>
<tr>
<td><strong>Size of programming element</strong></td>
<td>Smallest</td>
<td>Small</td>
<td>Two inverters plus pass and switch devices. Largest.</td>
<td>Two inverters plus pass and switch devices. Largest.</td>
</tr>
<tr>
<td><strong>Process</strong></td>
<td>Special, CMOS plus ViaLink</td>
<td>Special, CMOS plus antifuse</td>
<td>Standard CMOS</td>
<td>Standard CMOS</td>
</tr>
<tr>
<td><strong>Programming method</strong></td>
<td>Special hardware</td>
<td>Special hardware</td>
<td>PC card, PROM, or serial port</td>
<td>PC card, PROM, or serial port</td>
</tr>
</tbody>
</table>

<sup>1</sup>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 $F$ $wrt$ $B$: $F = B \cdot (A + D) + B' \cdot (C + D) = B \cdot F_2 + B' \cdot F_1$
- $F$ is a 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: $A_0 = D$, $A_1 = '1'$, $B_0 = D$, $B_1 = '1'$, $SA = C$, $SB = A$, $S_0 = '0'$, 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

### Boolean functions using a 2:1 MUX

<table>
<thead>
<tr>
<th>Function, F</th>
<th>F=</th>
<th>Canonical form</th>
<th>Min-terms</th>
<th>Min-term code</th>
<th>Function number</th>
<th>M1 A0 A1 SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 '0'</td>
<td>'0'</td>
<td>'0'</td>
<td>none</td>
<td>0000</td>
<td>0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>2 NOR1-1(A, B)</td>
<td>(A+B')</td>
<td>A'·B</td>
<td>1</td>
<td>0010</td>
<td>2</td>
<td>B 0 A</td>
</tr>
<tr>
<td>3 NOT(A)</td>
<td>A'</td>
<td>A'·B' + A·B</td>
<td>0, 1</td>
<td>0011</td>
<td>3</td>
<td>0 1 A</td>
</tr>
<tr>
<td>4 AND1-1(A, B)</td>
<td>A·B'</td>
<td>A·B'</td>
<td>2</td>
<td>0100</td>
<td>4</td>
<td>A 0 B</td>
</tr>
<tr>
<td>5 NOT(B)</td>
<td>B'</td>
<td>A'·B' + A·B'</td>
<td>0, 2</td>
<td>0101</td>
<td>5</td>
<td>0 1 B</td>
</tr>
<tr>
<td>6 BUF(B)</td>
<td>B</td>
<td>A'·B + A·B</td>
<td>1, 3</td>
<td>1010</td>
<td>6</td>
<td>0 B 1</td>
</tr>
<tr>
<td>7 AND(A, B)</td>
<td>A·B</td>
<td>A·B</td>
<td>3</td>
<td>1000</td>
<td>8</td>
<td>0 B A</td>
</tr>
<tr>
<td>8 BUF(A)</td>
<td>A</td>
<td>A·B' + A·B</td>
<td>2, 3</td>
<td>1100</td>
<td>9</td>
<td>0 A 1</td>
</tr>
<tr>
<td>9 OR(A, B)</td>
<td>A+B</td>
<td>A'·B + A·B' + A·B</td>
<td>1, 2, 3</td>
<td>1110</td>
<td>13</td>
<td>B 1 A</td>
</tr>
<tr>
<td>10 '1'</td>
<td>'1'</td>
<td>A'·B' + A·B + A·B'</td>
<td>0, 1, 2, 3</td>
<td>1111</td>
<td>15</td>
<td>1 1 1</td>
</tr>
</tbody>
</table>

Example of using the WHEEL functions to implement F=NAND(A, B)=(A·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·(B') + A'·('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, S0+S1=A. We can do this using S0=A, S1='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 CLKi
- 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:

$$
t_{SUD} = t_{SUD}' + (t_{PD}' - t_{CLKD}')
$$

- 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°C • Industrial. \( V_{DD} = 5V \pm 10\% \), \( T_A \) (ambient)=−40 to +85°C • Military: \( V_{DD} = 5V \pm 10\% \), \( T_C \) (case)=−55 to +125°C • Military: Standard MIL-STD-883C Class B • Military extended: unmanned spacecraft

### ACT 3 timing parameters

<table>
<thead>
<tr>
<th>Family</th>
<th>Delay</th>
<th>Fanout</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>ACT 3-3 (data book)</td>
<td>( t_{PD} )</td>
<td>2.9</td>
</tr>
<tr>
<td>ACT3-2 (calculated)</td>
<td>( t_{PD}/0.85 )</td>
<td>3.41</td>
</tr>
<tr>
<td>ACT3-1 (calculated)</td>
<td>( t_{PD}/0.75 )</td>
<td>3.87</td>
</tr>
<tr>
<td>ACT3-Std (calculated)</td>
<td>( t_{PD}/0.65 )</td>
<td>4.46</td>
</tr>
</tbody>
</table>

### ACT 3 derating factors

<table>
<thead>
<tr>
<th>( V_{DD}/V )</th>
<th>−55</th>
<th>−40</th>
<th>0</th>
<th>25</th>
<th>70</th>
<th>85</th>
<th>125</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.5</td>
<td>0.72</td>
<td>0.76</td>
<td>0.85</td>
<td>0.90</td>
<td>1.04</td>
<td>1.07</td>
<td>1.17</td>
</tr>
<tr>
<td>4.75</td>
<td>0.70</td>
<td>0.73</td>
<td>0.82</td>
<td>0.87</td>
<td>1.00</td>
<td>1.03</td>
<td>1.12</td>
</tr>
<tr>
<td>5.00</td>
<td>0.68</td>
<td>0.71</td>
<td>0.79</td>
<td>0.84</td>
<td>0.97</td>
<td>1.00</td>
<td>1.09</td>
</tr>
<tr>
<td>5.25</td>
<td>0.66</td>
<td>0.69</td>
<td>0.77</td>
<td>0.82</td>
<td>0.94</td>
<td>0.97</td>
<td>1.06</td>
</tr>
<tr>
<td>5.5</td>
<td>0.63</td>
<td>0.66</td>
<td>0.74</td>
<td>0.79</td>
<td>0.90</td>
<td>0.93</td>
<td>1.01</td>
</tr>
</tbody>
</table>

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=2⁵), 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.)
The Xilinx XC4000 family CLB (configurable logic block). (Source: Xilinx.)
5.2.3 XC5200 Logic Block

The Xilinx XC5200 family **Logic Cell (LC)** and configurable logic block (CLB). *(Source: Xilinx.)*

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 $t_{ilo} = 6.0$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

(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)

Altera MAX timing model (ns for the MAX 9000 series, '15' speed grade) (Source: Altera.)
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
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 470Ω 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

Supply bounce

A substantial current $I_{OL}$ may flow in the resistance, $R_S$, and inductance, $L_S$, that are between the on-chip GND net and the off-chip, external ground connection.

(a) As the pull-down device, M1, switches, it causes the GND net (value $V_{SS}$) to bounce.

(b) The supply bounce is dependent on the output slew rate.

(c) Ground bounce can cause other output buffers to generate a logic glitch.

(d) Bounce can also cause errors on other inputs.

Keywords: simultaneously-switching outputs (SSOs) • quiet I/O • slew-rate control • I/O management • packaging • PCB layout • ground planes • inductance
6.2.2 Transmission Lines

Transmission lines
(a) A printed-circuit board (PCB) trace is a transmission (TX) line \( (Z_0 = 50\,\Omega \text{–} 100\,\Omega) \)
(b) A driver launches an incident wave, which is reflected at the end of the line
(c) A connection starts to look like a TX line when the rise time is about \( 2 \times \) line delay \( (2t_f) \)

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

(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 $V_{OH\text{min}}$ solves the problem
6.3.2 Mixed-Voltage Systems

FPGA logic thresholds

<table>
<thead>
<tr>
<th>I/O options</th>
<th>Input levels</th>
<th>Output levels (high current)</th>
<th>Output levels (low current)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XC3000</td>
<td>TTL</td>
<td>2.0 0.8</td>
<td>3.86 −4.0 0.40 4.0</td>
</tr>
<tr>
<td></td>
<td>CMOS</td>
<td>3.85 0.9</td>
<td>3.86 −4.0 0.40 4.0</td>
</tr>
<tr>
<td>XC3000L</td>
<td>TTL</td>
<td>2.0 0.8</td>
<td>2.40 −4.0 0.40 4.0</td>
</tr>
<tr>
<td></td>
<td>CMOS</td>
<td>3.85 0.9</td>
<td>2.80 −0.1 0.2 0.1</td>
</tr>
<tr>
<td>XC4000</td>
<td>TTL</td>
<td>2.0 0.8</td>
<td>2.40 −4.0 0.40 12.0</td>
</tr>
<tr>
<td></td>
<td>CMOS</td>
<td>3.85 0.9</td>
<td>−1.0 0.50 24.0</td>
</tr>
<tr>
<td>XC4000H</td>
<td>TTL</td>
<td>2.0 0.8</td>
<td>2.40 −4.0 0.50 24.0</td>
</tr>
<tr>
<td></td>
<td>CMOS</td>
<td>3.85 0.9</td>
<td>2.80 −0.1 0.2 0.1</td>
</tr>
<tr>
<td>XC8100</td>
<td>TTL</td>
<td>2.0 0.8</td>
<td>3.86 −4.0 0.50 24.0</td>
</tr>
<tr>
<td></td>
<td>CMOS</td>
<td>3.85 0.9</td>
<td>3.86 −4.0 0.40 4.0</td>
</tr>
<tr>
<td>ACT 2/3</td>
<td></td>
<td>2.0 0.8</td>
<td>2.4 −8.0 0.50 12.0</td>
</tr>
<tr>
<td>FLEX10k</td>
<td>3V/5V</td>
<td>2.0 0.8</td>
<td>2.4 −4.0 0.45 12.0</td>
</tr>
</tbody>
</table>

Mixed-voltage systems

(a) TTL levels

(b) Low-voltage CMOS levels • JEDEC 8 • 3.3±0.3V

(c) Mixed-voltage ASIC • 5V-tolerant I/O • \( V_{DD\text{INT}} \) and \( V_{DD\text{I/O}} \)

(d) A problem when connecting two chips with different supply voltages—caused by the input clamp diodes
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**
The mean time between upsets (MTBU) or MTBF is

\[
\text{MTBU} = \frac{1}{pf_{\text{clock}}f_{\text{data}}} = \frac{\exp t_c/t_c}{f_{\text{clock}} f_{\text{data}}}
\]

where \( f_{\text{clock}} \) is the clock frequency and \( f_{\text{data}} \) is the data frequency.

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

Metastability parameters for FPGA flip-flops (not guaranteed by the vendors)

<table>
<thead>
<tr>
<th>FPGA</th>
<th>( T_0/s )</th>
<th>( \tau_c/s )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actel ACT 1</td>
<td>1.0E–09</td>
<td>2.17E–10</td>
</tr>
<tr>
<td>Xilinx XC3020-70</td>
<td>1.5E–10</td>
<td>2.71E–10</td>
</tr>
<tr>
<td>QuickLogic QL12x16-0</td>
<td>2.94E–11</td>
<td>2.91E–10</td>
</tr>
<tr>
<td>QuickLogic QL12x16-1</td>
<td>8.38E–11</td>
<td>2.09E–10</td>
</tr>
<tr>
<td>QuickLogic QL12x16-2</td>
<td>1.23E–10</td>
<td>1.85E–10</td>
</tr>
<tr>
<td>Altera MAX 7000</td>
<td>2.98E–17</td>
<td>2.00E–10</td>
</tr>
<tr>
<td>Altera FLEX 8000</td>
<td>1.01E–13</td>
<td>7.89E–11</td>
</tr>
</tbody>
</table>
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

Clock input
(a) Timing model (Xilinx XC4005-6)
(b) A simplified view of clock distribution • clock skew • clock latency
(c) Timing diagram
(Xilinx eliminates the variable internal delay $t_{PG}$, by specifying a **pin-to-pin setup time**, $t_{PSUFmin}=2\text{ns}$)
6.5.1 Registered Input

(a) Pin-to-pin timing model (XC4005-6) with pin-to-pin timing parameters

(b) Timing diagrams with and without programmable delay

Notice $t_{PSUFmin} = 2 \text{ ns} \neq t_{PICK} - t_{PGmax} = -1 \text{ ns}$

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

<table>
<thead>
<tr>
<th>Package</th>
<th>Pin count</th>
<th>Max. power $P_{max}$/W</th>
<th>$\theta_{JA}/^\circ CW^{-1}$ (still air)</th>
<th>$\theta_{JA}/^\circ CW^{-1}$ (still air)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPGA</td>
<td>84</td>
<td>33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CQFP</td>
<td>84</td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CQFP</td>
<td>172</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VQFP</td>
<td>80</td>
<td>68</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 XC4000 family IOB (input/output block). (Source: Xilinx.)
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 $0.5V_{DD}$) 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
**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
ACT 1 horizontal and vertical channel architecture. *(Source: Actel.)*

**Features:**
- Input stubs
- Output stubs
- Long vertical tracks (LVT)
- Fully populated interconnect array
7.1.1 Routing Resources

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

7.1.2 Elmore’s Constant

Measuring the delay of a net

(a) An RC tree

(b) The waveforms as a result of closing the switch at $t = 0$

$$V_i(t) = \exp(-t/\tau_{Di})$$  ;  \hspace{1cm} \tau_{Di} = \sum_{k=1}^{n} R_{ki}C_k$$

The time constant $\tau_{Di}$ is often called the **Elmore delay** and is different for each node.

I call $\tau_{Di}$ the **Elmore time constant** as a reminder that, if we approximate $V_i$ by an exponential waveform, the delay of the RC tree using 0.35/0.65 trip points is approximately $\tau_{Di}$ 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.

\[
\tau_D = R_{14}C_1 + R_{24}C_2 + R_{14}C_1 + R_{44}C_4 \\
= (R_1 + R_2 + R_3 + R_4)C_4 + (R_1 + R_2 + R_3)C_3 + (R_1 + R_2)C_2 + R_1C_1
\]

\[
\tau_D = 4RC_4 + 3RC_3 + 2RC_2 + RC_1
\]

- 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 (\(\propto n^2\)) 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
<table>
<thead>
<tr>
<th>Actel interconnect parameters</th>
<th>A1010/A1020</th>
<th>A1010B/A1020B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>2.0 µm, λ=1.0 µm</td>
<td>1.2 µm, λ=0.6 µm</td>
</tr>
<tr>
<td>Die height (A1010)</td>
<td>240mil</td>
<td>144mil</td>
</tr>
<tr>
<td>Die width (A1010)</td>
<td>360mil</td>
<td>216mil</td>
</tr>
<tr>
<td>Die area (A1010)</td>
<td>86,400mil²=56Mλ²</td>
<td>31,104mil²=56Mλ²</td>
</tr>
<tr>
<td>Logic Module (LM) height (Y1)</td>
<td>180µm=180λ</td>
<td>108µm=180λ</td>
</tr>
<tr>
<td>LM width (X)</td>
<td>150µm=150λ</td>
<td>90µm=150λ</td>
</tr>
<tr>
<td>LM area (X×Y1)</td>
<td>27,000µm²=27kλ²</td>
<td>9,720µm²=27kλ²</td>
</tr>
<tr>
<td>Channel height (Y2)</td>
<td>25 tracks=287µm</td>
<td>25 tracks=170µm</td>
</tr>
<tr>
<td>Channel area per LM (X×Y2)</td>
<td>43,050µm²=43kλ²</td>
<td>15,300µm²=43kλ²</td>
</tr>
<tr>
<td>LM and routing area (X×Y1+X×Y2)</td>
<td>70,000µm²=70kλ²</td>
<td>25,000µm²=70kλ²</td>
</tr>
<tr>
<td>Antifuse capacitance</td>
<td>—</td>
<td>10 fF</td>
</tr>
<tr>
<td>Metal capacitance</td>
<td>0.2pFmm⁻¹</td>
<td>0.2pFmm⁻¹</td>
</tr>
<tr>
<td>Output stub length (spans 3 LMs + 4 channels)</td>
<td>4 channels=1688µm</td>
<td>4 channels=1012µm</td>
</tr>
<tr>
<td>Output stub metal capacitance</td>
<td>0.34pF</td>
<td>0.20pF</td>
</tr>
<tr>
<td>Output stub antifuse connections</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>Output stub antifuse capacitance</td>
<td>—</td>
<td>1.0pF</td>
</tr>
<tr>
<td>Horiz. track length</td>
<td>4–44 cols.= 600–6600µm</td>
<td>4–44 cols.= 360–3960µm</td>
</tr>
<tr>
<td>Horiz. track metal capacitance</td>
<td>0.1–1.3pF</td>
<td>0.07–0.8pF</td>
</tr>
<tr>
<td>Horiz. track antifuse connections</td>
<td>52–572 antifuses</td>
<td>52–572 antifuses</td>
</tr>
<tr>
<td>Horiz. track antifuse capacitance</td>
<td>—</td>
<td>0.52–5.72 pF</td>
</tr>
<tr>
<td>Long vertical track (LVT)</td>
<td>8–14 channels=3760–6580 µm</td>
<td>8–14 channels=2240–3920 µm</td>
</tr>
<tr>
<td>LVT metal capacitance</td>
<td>0.08–0.13pF</td>
<td>0.45–0.8pF</td>
</tr>
<tr>
<td>LVT track antifuse connections</td>
<td>200–350 antifuses</td>
<td>200–350 antifuses</td>
</tr>
<tr>
<td>LVT track antifuse capacitance</td>
<td>2–3.5pF</td>
<td></td>
</tr>
<tr>
<td>Antifuse resistance (ACT 1)</td>
<td>0.5kΩ (typ.), 0.7kΩ (max.)</td>
<td></td>
</tr>
</tbody>
</table>
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

Xilinx LCA interconnect

(a) The LCA architecture (notice the matrix element size is larger than a CLB)

(b) A simplified representation of the interconnect resources. Each of the lines is a bus.

- The **vertical lines** and **horizontal lines** run between CLBs.
- The **general-purpose interconnect** joins **switch boxes** (also known as **magic boxes** or **switching matrices**).
- The **long lines** run across the entire chip. It is possible to form internal buses using long lines and the three-state buffers that are next to each CLB.
- The **direct connections** (not used on the XC4000) bypass the switch matrices and directly connect adjacent CLBs.
- The **Programmable Interconnection Points** (PIPs) are programmable pass transistors that connect the CLB inputs and outputs to the routing network.
- The **bidirectional (BIDI) interconnect buffers** restore the logic level and logic strength on long interconnect paths.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>XC3000</th>
<th>XC3020</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>1.0 µm, λ=0.5 µm</td>
<td></td>
</tr>
<tr>
<td>Die height</td>
<td></td>
<td>220 mil</td>
</tr>
<tr>
<td>Die width</td>
<td></td>
<td>180 mil</td>
</tr>
<tr>
<td>Die area</td>
<td></td>
<td>39,600 mil²=102 Mλ²</td>
</tr>
<tr>
<td>CLB matrix height (Y)</td>
<td></td>
<td>480 µm=960λ</td>
</tr>
<tr>
<td>CLB matrix width (X)</td>
<td></td>
<td>370 µm=740λ</td>
</tr>
<tr>
<td>CLB matrix area (X×Y)</td>
<td></td>
<td>17,600 µm²=710kλ²</td>
</tr>
<tr>
<td>Matrix transistor resistance, Rₚ₁</td>
<td></td>
<td>0.5–1 kΩ</td>
</tr>
<tr>
<td>Matrix transistor parasitic capacitance, Cₚ₁</td>
<td></td>
<td>0.01–0.02pF</td>
</tr>
<tr>
<td>PIP transistor resistance, Rₚ₂</td>
<td></td>
<td>0.5–1 kΩ</td>
</tr>
<tr>
<td>PIP transistor parasitic capacitance, Cₚ₂</td>
<td></td>
<td>0.01–0.02pF</td>
</tr>
<tr>
<td>Single-length line (X, Y)</td>
<td>370 µm, 480 µm</td>
<td></td>
</tr>
<tr>
<td>Single-length line capacitance: Cₓ, Cᵧ</td>
<td></td>
<td>0.075pF, 0.1pF</td>
</tr>
<tr>
<td>Horizontal Longline (8X)</td>
<td>8 cols.=2960 µm</td>
<td></td>
</tr>
<tr>
<td>Horizontal Longline metal capacitance, Cₓ</td>
<td></td>
<td>0.6pF</td>
</tr>
</tbody>
</table>
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

The Xilinx EPLD UIM (Universal Interconnection Module)

(a) A simplified block diagram of the UIM. The UIM bus width, \( n \), varies from 68 (XC7236) to 198 (XC73108).

(b) The UIM is actually a large programmable AND array.

(c) The parasitic capacitance of the EPROM cell.
7.4 Altera MAX 5000 and 7000

A simplified block diagram of the Altera MAX interconnect scheme

(a) The PIA (Programmable Interconnect Array) is deterministic—delay is independent of the path length

(b) Each LAB (Logic Array Block) contains a programmable AND array

(c) Interconnect timing within a LAB is also fixed
7.5 Altera MAX 9000

The Altera MAX 9000 interconnect scheme

(a) A 4×5 array of Logic Array Blocks (LABs), the same size as the EMP9400 chip

(b) A simplified block diagram of the interconnect architecture showing the connection of the FastTrack buses to a LAB

---

7.6 Altera FLEX

The Altera FLEX interconnect scheme

(a) The row and column FastTrack interconnect. The chip shown, with 4 rows × 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

<table>
<thead>
<tr>
<th>File types used by Actel design software (an example—these change often)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ADL</strong></td>
</tr>
<tr>
<td><strong>IPF</strong></td>
</tr>
<tr>
<td><strong>CRT</strong></td>
</tr>
<tr>
<td><strong>VALIDATED</strong></td>
</tr>
<tr>
<td><strong>COB</strong></td>
</tr>
<tr>
<td><strong>VLD</strong></td>
</tr>
<tr>
<td><strong>PIN</strong></td>
</tr>
<tr>
<td><strong>DFR</strong></td>
</tr>
<tr>
<td><strong>LOC</strong></td>
</tr>
<tr>
<td><strong>PLI</strong></td>
</tr>
<tr>
<td><strong>SEG</strong></td>
</tr>
<tr>
<td><strong>STF</strong></td>
</tr>
<tr>
<td><strong>RTI</strong></td>
</tr>
<tr>
<td><strong>FUS</strong></td>
</tr>
<tr>
<td><strong>DEL</strong></td>
</tr>
<tr>
<td><strong>AVI</strong></td>
</tr>
</tbody>
</table>
**FPGA state-machine language (an example of “third-party” tools)**

<table>
<thead>
<tr>
<th>LOG/iC state-machine language</th>
<th>PALASM version</th>
</tr>
</thead>
<tbody>
<tr>
<td>*IDENTIFICATION</td>
<td>TITLE sequence detector</td>
</tr>
<tr>
<td>sequence detector</td>
<td>CHIP MEALY USER</td>
</tr>
<tr>
<td>LOG/iC code</td>
<td>CLK Z QQ2 QQ1 X</td>
</tr>
<tr>
<td>*X-NAMES</td>
<td>EQUATIONS</td>
</tr>
<tr>
<td>X; !input</td>
<td>Z = X * QQ2 * QQ1</td>
</tr>
<tr>
<td>*Y-NAMES</td>
<td>QQ2 := X * QQ1 + X * QQ2</td>
</tr>
<tr>
<td>D; !output, D = 1 when three 1's appear on X</td>
<td>QQ1 := X * QQ2 + X * /QQ1</td>
</tr>
<tr>
<td>*FLOW-TABLE</td>
<td>STATE-ASSIGNMENT</td>
</tr>
<tr>
<td>;State, X input, Y output, next state</td>
<td>BINARY;</td>
</tr>
<tr>
<td>S1, X1, Y0, F2;</td>
<td>*RUN-CONTROL</td>
</tr>
<tr>
<td>S1, X0, Y0, F1;</td>
<td>PROGFORMAT = P-EQUATIONS;</td>
</tr>
<tr>
<td>S2, X1, Y0, F3;</td>
<td>*END</td>
</tr>
<tr>
<td>S2, X0, Y0, F1;</td>
<td></td>
</tr>
<tr>
<td>S3, X1, Y0, F4;</td>
<td></td>
</tr>
<tr>
<td>S3, X0, Y0, F1;</td>
<td></td>
</tr>
<tr>
<td>S4, X1, Y1, F4;</td>
<td></td>
</tr>
<tr>
<td>S4, X0, Y0, F1;</td>
<td></td>
</tr>
</tbody>
</table>
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

```vhdl
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 = .
/optimize for area/
/use the TI cell libraries/
link_library = tpc10.db
target_library = tpc10.db
symbol_library = tpc10.sdb
read -f vhdl detector.vhd
compile
max_area 0.0
write -h -f db -o detector_opt.db
report -area -cell -timing > detector.rpt
free -all
/write EDIF netlist/
write -n -f db -hierarchy -0
exit
```
8.3 The Halfgate ASIC
8.3.1 Xilinx

Design flow for the Xilinx implementation of the halfgate ASIC

<table>
<thead>
<tr>
<th>Script (using Compass tools as an example)</th>
<th>Design flow</th>
</tr>
</thead>
<tbody>
<tr>
<td># halfgate.xilinx.inp</td>
<td>1 myOutput = ~myInput</td>
</tr>
</tbody>
</table>
| shell setdef                              | 2 myInput  
| path working xc4000d xblox cmosch000x    | myOutput  |
| quit                                      |             |
|asic                                       |             |
| open [v]halfgate                         | 3           |
| synthesize                                |              |
| save [nls]halfgate_p                      |              |
| quit                                      |              |
|fpga                                       | 4           |
| set tag xc4000                            |              |
| set opt area                              |              |
| optimize [nls]halfgate_p                  |              |
| quit                                      |              |
|qtv                                        | 5           |
| 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                                       |              |
The Xilinx files for the halfgate ASIC

**Verilog file** *(halfgate.v)*

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

**Preroute XNF file** *(halfgate_p.xnf)*

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

`;: halfgate_p.lca (4003PC84-4), makebits 5.2.0, Tue Jul 16 20:09:43 1996
Version 2
Design 4003PC84 4 0
Speed -4
Addnet PAD_myInput PAD61.I2
PAD1.O
Netdelay PAD_myInput PAD1.O
3.1
Program PAD_myInput {65G521}
{65G287} {65G50} {63G50}
{52G50} {45G50}
NProgram PAD_myInput
col.B.long.3:PAD1.O
col.B.long.3:row.G.local.1
col.B.long.3:row.M.local.5-s
40.1.14 MB.40.1.35
row.M.local.5:PAD61.I2

Editblk PAD61
Base IO
Config INFF: I1: I2: I O:
OUT: PAD: TRI:
Endblk
Editblk PAD1
Base IO
Config INFF: I1: I2: O:
OUT:O:NOT PAD: TRI:
Endblk
Nameblk PAD61 myInput
Nameblk PAD1 myOutput
Intnet myOutput PAD
myOutput
Intnet myInput PAD myInput
System FGG 0 VERS 2 !
System FGG 1 GD0 0 !

Postroute XNF file (halfgate_b.xnf)

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

#### The Actel files for the halfgate ASIC

<table>
<thead>
<tr>
<th>ADL file</th>
<th>STF file</th>
</tr>
</thead>
</table>
| ; 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. | ; 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. |
8.3.3 Altera

**EDIF netlist in Altera format for the halfgate ASIC**

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

<table>
<thead>
<tr>
<th>Pin</th>
<th>LC</th>
<th>LAB</th>
<th>Primitive</th>
<th>Code</th>
<th>Total Shared</th>
<th>Fan-In</th>
<th>Fan-Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>-</td>
<td>-</td>
<td>INPUT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

** OUTPUTS **

<table>
<thead>
<tr>
<th>Pin</th>
<th>LC</th>
<th>LAB</th>
<th>Primitive</th>
<th>Code</th>
<th>Total Shared</th>
<th>Fan-In</th>
<th>Fan-Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>17</td>
<td>B</td>
<td>OUTPUT</td>
<td>t</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

** LOGIC CELL INTERCONNECTIONS **

Logic Array Block 'B':

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

| Pin | - * | - * | myInput
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>-</td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

* = 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:

```plaintext
// 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, 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 orl_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

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

<table>
<thead>
<tr>
<th>ANSI sheet</th>
<th>Size (inches)</th>
<th>ISO sheet</th>
<th>Size (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>8.5 × 11</td>
<td>A5</td>
<td>21.0 × 14.8</td>
</tr>
<tr>
<td>B</td>
<td>11 × 17</td>
<td>A4</td>
<td>29.7 × 21.0</td>
</tr>
<tr>
<td>C</td>
<td>17 × 22</td>
<td>A3</td>
<td>42.0 × 29.7</td>
</tr>
<tr>
<td>D</td>
<td>22 × 34</td>
<td>A2</td>
<td>59.4 × 42.0</td>
</tr>
<tr>
<td>E</td>
<td>34 × 44</td>
<td>A1</td>
<td>84.0 × 59.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A0</td>
<td>118.9 × 84.0</td>
</tr>
</tbody>
</table>
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
A 4-bit latch:

(a) drawn as a flat schematic from gate-level primitives
(b) drawn as four instances of the cell symbol DLAT
(c) drawn using a vectored instance of the DLAT cell symbol with cardinality of 4
(d) drawn using a new cell symbol with cell name FourBit
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 desigantor • 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

A 16-bit latch:
(a) drawn as four instances of cell FourBit
(b) drawn as a cell named SixteenBit
(c) drawn as four multiple instances of cell FourBit

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**

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

Defines combinational logic. Two-pass logic

Equals '=' is unlocked assignment.
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);
  P1Y = 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

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF NXT</td>
<td>Conditional next state transition</td>
</tr>
<tr>
<td>IF NXT OUT</td>
<td>Conditional next state transition with synchronous output</td>
</tr>
<tr>
<td>NEXT</td>
<td>Unconditional next state transition</td>
</tr>
<tr>
<td>NEXT OUT</td>
<td>Unconditional next state transition with asynchronous output</td>
</tr>
<tr>
<td>OUT</td>
<td>Unconditional asynchronous output</td>
</tr>
<tr>
<td>IF OUT</td>
<td>Conditional asynchronous output</td>
</tr>
<tr>
<td>DEFAULT NXT</td>
<td>Default next state transition</td>
</tr>
<tr>
<td>DEFAULT OUT</td>
<td>Default asynchronous output</td>
</tr>
<tr>
<td>DEFAULT NXT OUT</td>
<td>Default next state transition with synchronous output</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Extension</th>
<th>Explanation</th>
<th>Extension</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>D input to a D register</td>
<td>DFB</td>
<td>D register feedback of combinational output</td>
</tr>
<tr>
<td>L</td>
<td>L input to a latch</td>
<td>LFB</td>
<td>Latched feedback of combinational output</td>
</tr>
<tr>
<td>J, K</td>
<td>J-K-input to a J-K register</td>
<td>TFB</td>
<td>T register feedback of combinational output</td>
</tr>
<tr>
<td>S, R</td>
<td>S-R input to an S-R register</td>
<td>INT</td>
<td>Internal feedback</td>
</tr>
<tr>
<td>T</td>
<td>T input to a T register</td>
<td>IO</td>
<td>Pin feedback of registered output</td>
</tr>
<tr>
<td>DQ</td>
<td>D output of an input D register</td>
<td>IOD/T</td>
<td>D/T register on pin feedback path selection</td>
</tr>
<tr>
<td>LQ</td>
<td>Q output of an input latch</td>
<td>IOL</td>
<td>Latch on pin feedback path selection</td>
</tr>
<tr>
<td>AP, AR</td>
<td>Asynchronous preset/reset</td>
<td>IOAP, IOAR</td>
<td>Asynchronous preset/reset of register on feedback path</td>
</tr>
<tr>
<td>SP, SR</td>
<td>Synchronous preset/reset</td>
<td>IOSP, IOSR</td>
<td>Synchronous preset/reset of register on feedback path</td>
</tr>
<tr>
<td>CK</td>
<td>Product clock term (async.)</td>
<td>IOCK</td>
<td>Clock for pin feedback register</td>
</tr>
<tr>
<td>OE</td>
<td>Product-term output enable</td>
<td>APMUX, ARMUX</td>
<td>Asynchronous preset/reset multiplexor selection</td>
</tr>
<tr>
<td>CA</td>
<td>Complement array</td>
<td>CKMUX</td>
<td>Clock multiplexor selector</td>
</tr>
<tr>
<td>PR</td>
<td>Programmable preload</td>
<td>LEMUX</td>
<td>Latch enable multiplexor selector</td>
</tr>
<tr>
<td>CE</td>
<td>CE input of a D-CE register</td>
<td>OEMUX</td>
<td>Output enable multiplexor selector</td>
</tr>
<tr>
<td>LE</td>
<td>Product-term latch enable</td>
<td>IMUX</td>
<td>Input multiplexor selector of two pins</td>
</tr>
<tr>
<td>OBS</td>
<td>Programmable observability of buried nodes</td>
<td>TEC</td>
<td>Technology-dependent fuse selection</td>
</tr>
<tr>
<td>BYP</td>
<td>Programmable register bypass</td>
<td>T1</td>
<td>T1 input of 2-T register</td>
</tr>
</tbody>
</table>
### ABEL and CUPL pin declarations for an ATMEL ATV2500B

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

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

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 PLA Tools

**Key terms and concepts:** developed at UC Berkeley • *eqnott* 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): \( F_1 = A | B | \neg C; \ F_2 = \neg B & C; \ F_3 = A & B | C; \)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>eqnott output</th>
<th>espresso output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>.i 3</td>
<td>.i 3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>.o 3</td>
<td>.o 3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>.p 6</td>
<td>.p 6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>--0 100</td>
<td>1-- 100</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>--1 001</td>
<td>11-- 001</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-01 010</td>
<td>-01 011</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-11 100</td>
<td>-11 101</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>.e</td>
<td>.e</td>
</tr>
</tbody>
</table>

Output (5 minterms): \( F_1 = A | \neg C | (B & C); \ F_2 = \neg B & C; \ F_3 = A & B | (\neg B & C) | (B & C); \)
The format of the input and output files used by the PLA design tool **espresso**

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

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

<table>
<thead>
<tr>
<th>Plane</th>
<th>Character</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>I 1</td>
<td>1</td>
<td>The input literal appears in the product term</td>
</tr>
<tr>
<td>I 0</td>
<td>0</td>
<td>The input literal appears complemented in the product term</td>
</tr>
<tr>
<td>I –</td>
<td>–</td>
<td>The input literal does not appear in the product term</td>
</tr>
<tr>
<td>O 1 or 4</td>
<td>1 or 4</td>
<td>This product term appears in the ON set</td>
</tr>
<tr>
<td>O 0</td>
<td>0</td>
<td>This product term appears in the OFF set</td>
</tr>
<tr>
<td>O 2 or –</td>
<td>2 or –</td>
<td>This product term appears in the don’t care set</td>
</tr>
<tr>
<td>O 3 or ~</td>
<td>3 or ~</td>
<td>No meaning for the value of this function</td>
</tr>
</tbody>
</table>
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
(edif halfgate_p
(edifVersion 2 0 0)
(edifLevel 0)
(keywordMap
(keywordLevel 0))
(status
(written
(timeStamp 1996 7 10 22 5 10)
(program "COMPASS Design Automation -- EDIF Interface"
(version "v9r1.2 last updated 26-Mar-96")
(author
"mikes")
(library xc4000d
(edifLevel 0)
(technology
(numberDefinition )
(simulationInfo
(logicValue H)
(logicValue L))
(cell
(rename INV "inv")
(cellType GENERIC)
(view COMPASS_mde_view
(interface
(port I (direction INPUT))
(port O (direction OUTPUT))
(designator "@@Label")))))
(library working
(edifLevel 0)
(technology
(numberDefinition )
(simulationInfo
(logicValue H)
(logicValue L))
(cell
(rename HALFGATE_P "halfgate_p")
(cellType GENERIC)
(view COMPASS_nls_view
(interface
(port myInput
(designator "@@Label"))
(contents
(instance B1_i1)))))
(net myInput
(joined
(portRef myInput)
(portRef I
(instanceRef B1_i1))))
(net myOutput
(joined
(portRef myOutput)
(portRef O
(instanceRef B1_i1))))
(net VDD
(joined )
(net VSS
(joined ))))
(design HALFGATE_P
(cellRef HALFGATE_P
(libraryRef working)))))
```

The EDIF file defines a netlist for a halfgate. It includes information about the netlist, the library, the technology, simulation information, cell definitions, and the netlist. The `port` and `designator` attributes specify input and output ports, respectively. The `contents` attribute references instances of cells, such as `INV` and `HALFGATE_P`, which are connected to nets like `myInput` and `myOutput`, and power rails `VDD` and `VSS`.
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
(figure
connector_FG
(dot
(pt 0 0)))))
(portImplementation
(name A1
(display
connector_FG
(origin
(pt -5 21))))
(connectLocation
(figure
connector_FG
(dot
(pt 0 20)))))
(portImplementation
(name Z
(display
connector_FG
(origin
(pt 60 15))))
(connectLocation
(figure
connector_FG
(dot
(pt 60 10))))
(propertyDisplay
(label
(display
icon_FG
(origin
(pt 20 29))))
(keywordDisplay
(instance
(display
icon_FG
(origin
(pt 60 15))))
(connectLocation
(figure
connector_FG
(dot
(pt 60 10)))))
```

### Compass and corresponding Cadence names

<table>
<thead>
<tr>
<th>Compass name</th>
<th>Cadence name</th>
<th>Compass name</th>
<th>Cadence name</th>
</tr>
</thead>
<tbody>
<tr>
<td>connector_FG</td>
<td>pin</td>
<td>net_FG</td>
<td>wire</td>
</tr>
<tr>
<td>icon_FG</td>
<td>device</td>
<td>bus_FG</td>
<td>not used</td>
</tr>
<tr>
<td>instance_FG</td>
<td>instance</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The bounding box problem

(a) The original bounding box for the an02d1 icon
(b) Problems in Cadence Composer due to overlapping bounding boxes
(c) A “shrink-wrapped” bounding box created using Cadence SKILL

#### 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
**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

```vhdl
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');
  end process;
end Counter_1;
```

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); writeln(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**

```vhdl
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**

```vhdl
entity Adder8 is
port (A, B: in BIT_VECTOR(7 downto 0);
Cin: in BIT; Cout, Sum: out BIT);
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

```vhdl
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;
begins
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;
```

An 8-bit register

```vhdl
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
port (Clr, Clk, D : in BIT; Q, QB : out BIT);
begin
component DFFClr port map (Clr, Clk, D(i), Q(i),
open);
end generate;
end;
```

An 8-bit multiplexer

```vhdl
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;
```

Timing:
- TRQ (CLR to Q/QN) = 2ns
- TCQ (CLK to Q/QN) = 2ns

8-bit register. Uses DFFClr positive edge-triggered flip-flop model.

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) = 1 ns
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 downto 1);
          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;
begins
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=>'0');
  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
  component Mux8 port (A,B:BIT_VECTOR(7 downto 0); Sel:BIT; Y:out BIT_VECTOR(7 downto 0));
  end component;

  component AllZero port (X: BIT_VECTOR; F:out BIT);
  end component;

  component Adder8 port (A,B:BIT_VECTOR(7 downto 0); Cin:BIT; Cout:out BIT; Sum:out BIT_VECTOR(7 downto 0));
  end component;

  component Register8 port (D:BIT_VECTOR(7 downto 0); Clk,Clr:BIT; Q:out BIT_VECTOR);
  end component;

  component ShiftN port (CLK,CLR,LD,SH,DIR:BIT; D:BIT_VECTOR; Q:out BIT_VECTOR);
  end component;

  component SM_1 port (Start,CLK,LSB,Stop,Reset:BIT; Init,Shift,Add,Done:out BIT);
  end component;
end;

Utility code to help test the multiplier:

package Clock_Utils is
  procedure Clock (signal C: out Bit; HT, LT:TIME);
end Clock_Utils;

package body Clock_Utils is
  procedure Clock (signal C: out Bit; HT, LT:TIME)
  begin
    loop C<='1' after LT, '0' after LT + HT; wait for LT + HT;
    end loop;
  end;
end Clock_Utils;

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

package Utils is
  function Convert (N,L: NATURAL) return BIT_VECTOR;
  function Convert (B: BIT_VECTOR) return NATURAL;
end Utils;

package body Utils is
  function Convert (N,L: NATURAL) return BIT_VECTOR is
    variable T:BIT_VECTOR(L-1 downto 0);
    variable V:NATURAL:= N;
    begin for i in T'RIGHT to T'LEFT loop
      T(i) := BIT'VAL(V mod 2); V:= V/2;
    end loop; return T;
  end;

  function Convert (B: BIT_VECTOR) return NATURAL is
    variable T:BIT_VECTOR(B'LENGTH-1 downto 0) := B;
end;
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;
      A<=Convert(i,A'Length);B<=Convert(j,B'Length); --20
      wait until CLK'EVENT and CLK='1'; wait for 1 ns; --21
      Start <= '1', '0' after 20 ns; wait until Done = '1'; --22
      wait until CLK'EVENT and CLK='1'; --23
    end loop; end loop; --24
    for i in 0 to 1 loop for j in 0 to 15 loop --25
      DA <= i; DB <= j;
      A<=Convert(i,A'Length);B<=Convert(j,B'Length); --26
      wait until CLK'EVENT and CLK='1'; wait for 1 ns; --27
      Start <= '1', '0' after 20 ns; wait until Done = '1'; --28
      wait until CLK'EVENT and CLK='1'; --29
    end loop; end loop; --30
    wait; --31
  end process; --32
end; --33

10.3 Syntax and Semanticsof VHDL

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

sentence ::= subject verb object.
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
```

s -- A simple name.
S -- A simple name, the same as s. VHDL is not case 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 for;
end;

Entities, architectures, components, ports, port maps, and configurations
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_1164.all; -- to use the IEEE package

package Part_STD_LOGIC_1164 is
--1
  type STD_ULOGIC is
    ( 'U', -- Uninitialized
      'X', -- Forcing Unknown
      '0', -- Forcing 0
      '1', -- Forcing 1
      'Z', -- High Impedance
      'W', -- Weak Unknown
      'L', -- Weak 0
      'H', -- Weak 1
      '-'  -- Don't Care);
--2
  type STD_ULOGIC_VECTOR is array (NATURAL range <>) of STD_ULOGIC; --3
  function resolved (s : STD_ULOGIC_VECTOR) return STD_ULOGIC; --4
  subtype STD_LOGIC is resolved STD_ULOGIC; --5
  type STD_LOGIC_VECTOR is array (NATURAL range <>) of STD_LOGIC; --6
  subtype X01 is resolved STD_ULOGIC range 'X' to '1'; --7
  subtype X01Z is resolved STD_ULOGIC range 'X' to 'Z'; --8
  subtype UX01 is resolved STD_ULOGIC range 'U' to '1'; --9
  subtype UX01Z is resolved STD_ULOGIC range 'U' to 'Z'; --10
-- Vectorized overloaded logical operators:
  function "and" (L : STD_ULOGIC; R : STD_ULOGIC) return UX01; --11
  -- Logical operators not, and, nand, or, nor, xor, xnor (VHDL-93),--12
  -- overloaded for STD_ULOGIC STD_ULOGIC_VECTOR STD_LOGIC_VECTOR. --13
  -- Strength strippers and type conversion functions:
  -- function To_T (X : F) return T; --14
  -- defined for types, T and F, where
  -- F=BIT VECTOR STD_ULOGIC STD_ULOGIC_VECTOR STD_LOGIC_VECTOR --15
  -- T=types F plus types X01 X01Z UX01 (but not type UX01Z) --16
  -- Exclude _'s in T in name: TO_STDULOGIC not TO_STD_ULOGIC --17
  -- To_X01 : L->0, H->1 others-->X --18


-- To_X01Z: Z->Z, others as To_X01
-- To_UX01: U->U, others as To_X01
-- Edge detection functions:

function rising_edge (signal s: STD_ULOGIC) return BOOLEAN;
function falling_edge (signal s: STD_ULOGIC) return BOOLEAN;

-- Unknown detection (returns true if s = U, X, Z, W):
-- function Is_X (s : T) return BOOLEAN;
-- defined for T = STD_ULOGIC STD_ULOGIC_VECTOR STD_LOGIC_VECTOR.
end Part_STD_LOGIC_1164;

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 |

Means "legal to associate interface object (Outside) of mode X with formal (Inside) of mode Y"
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

<table>
<thead>
<tr>
<th>Properties of ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example entity declaration:</td>
</tr>
<tr>
<td>entity E is port (F_1:BIT; F_2:out BIT; F_3:inout BIT; F_4:buffer BIT); end; -- formals</td>
</tr>
<tr>
<td>Example component declaration:</td>
</tr>
<tr>
<td>component C port (L_1:BIT; L_2:out BIT; L_3:inout BIT; L_4:buffer BIT); -- locals end component;</td>
</tr>
<tr>
<td>Example component instantiation:</td>
</tr>
<tr>
<td>I1 : C port map (L_1 =&gt; A_1,L_2 =&gt; A_2,L_3 =&gt; A_3,L_4 =&gt; A_4); -- locals =&gt; actuals</td>
</tr>
<tr>
<td>Example configuration:</td>
</tr>
<tr>
<td>for I1 : C use entity E(Behave) port map (F_1 =&gt; L_1,F_2 =&gt; L_2,F_3 =&gt; L_3,F_4 =&gt; L_4); -- formals =&gt; locals</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>in</th>
<th>out</th>
<th>inout</th>
<th>buffer</th>
</tr>
</thead>
</table>

| Modes of Outside that Inside may connect to |

<table>
<thead>
<tr>
<th>in</th>
<th>out</th>
<th>inout</th>
<th>buffer</th>
</tr>
</thead>
</table>

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

classification 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)

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 ;
```

```
extent 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 therange 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 decla-
rations • 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

```vhdl
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(15 downto 0);       -- 16-bit bus.
signal GBus : STD_LOGIC_VECTOR(31 downto 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');
begins 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:**

```vhdl
function my_function(Ff) return BIT is -- Formal function parameter, Ff.
procedure my_procedure(Fp); -- Formal procedure parameter, Fp.
```

**Example subprogram calls:**

```vhdl
my_result := my_function(Af); -- Calling a function with an actual parameter, Af.
```

<table>
<thead>
<tr>
<th>Mode of Ff or Fp (formals)</th>
<th>in</th>
<th>out</th>
<th>inout</th>
<th>No mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Permissible classes for Af (function actual parameter)</td>
<td>constant (default)</td>
<td>Not allowed</td>
<td>Not allowed</td>
<td>file</td>
</tr>
<tr>
<td>signal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Permissible classes for Ap (procedure actual parameter)</td>
<td>constant (default)</td>
<td>constant</td>
<td>variable (default)</td>
<td>file</td>
</tr>
<tr>
<td>constant</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>variable</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>signal</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Can you read attributes of Ff or Fp (formals)?**

- Yes, except:
  - 'STABLE
  - 'QUIET
  - 'DELAYED
  - 'TRANSACTION
  - 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
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
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 (23 downto 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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Kind</th>
<th>Parameter</th>
<th>Result type</th>
<th>Result/restrictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>S'DELAYED [(T)]</td>
<td>S</td>
<td>TIME</td>
<td>base(S)</td>
<td>S delayed by time T</td>
</tr>
<tr>
<td>S'ENABLE [(T)]</td>
<td>S</td>
<td>TIME</td>
<td>BOOLEAN</td>
<td>TRUE if no event on S for time T</td>
</tr>
<tr>
<td>S'QUIET [(T)]</td>
<td>S</td>
<td>TIME</td>
<td>BOOLEAN</td>
<td>TRUE if S is quiet for time T</td>
</tr>
<tr>
<td>S'TRANSACTION</td>
<td>S</td>
<td>BIT</td>
<td></td>
<td>Toggles each cycle if S becomes active</td>
</tr>
<tr>
<td>S'EVENET</td>
<td>F</td>
<td>BOOLEAN</td>
<td></td>
<td>TRUE when event occurs on S</td>
</tr>
<tr>
<td>S'ACTIVE</td>
<td>F</td>
<td>BOOLEAN</td>
<td></td>
<td>TRUE if S is active</td>
</tr>
<tr>
<td>S'LAST_EVENT</td>
<td>F</td>
<td>TIME</td>
<td></td>
<td>Elapsed time since the last event on S</td>
</tr>
<tr>
<td>S'LAST_ACTIVE</td>
<td>F</td>
<td>TIME</td>
<td></td>
<td>Elapsed time since S was active</td>
</tr>
<tr>
<td>S'LAST_VALUE</td>
<td>F</td>
<td>base(S)</td>
<td></td>
<td>Previous value of S, before last event</td>
</tr>
<tr>
<td>S'DRIVING</td>
<td>F</td>
<td>BOOLEAN</td>
<td></td>
<td>TRUE if every element of S is driven</td>
</tr>
<tr>
<td>S'DRIVING_VALUE</td>
<td>F</td>
<td>base(S)</td>
<td></td>
<td>Value of the driver for S in the current process</td>
</tr>
</tbody>
</table>

1 F=function, S=signal.

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

3 base(S)=base type of S.

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

5 VHDL-93 only.
## Predefined attributes for scalar and array types

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

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

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

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

Or 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] ; 

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

declaration DFF is port (CLK, D : BIT; Q : out BIT); end;
architecture Behave of DFF is
process begin
    wait until Clk = '1'; Q <= D;
end process;

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
        -- 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, δ • 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;
entity sm_mealy is
  port (reset, clock, i1, i2 : STD_LOGIC; o1, o2 :out STD_LOGIC);
end sm_mealy;
architecture Behave of sm_mealy is
type STATES is (s0, s1, s2, s3); signal current, new : STATES;
begin
synchronous : process (clock, reset) begin
  if To_X01(reset) = '0' then current <= s0;
  elsif rising_edge(clock) then current <= new; end if;
end process;
combinational : process (current, i1, i2) begin
case current is
when s0 =>
  if To_X01(i1) = '1' then o2 <= '0'; o1 <= '0'; new <= s2;
  else o2 <= '1'; o1 <= '1'; new <= s1;end if;
when s1 =>
  if To_X01(i2) = '1' then o2 <= '1'; o1 <= '0'; new <= s1;
  else o2 <= '0'; o1 <= '1'; new <= s3;end if;
when s2 =>
  if To_X01(i2) = '1' then o2 <= '0'; o1 <= '1'; new <= s2;
    else o2 <= '1'; o1 <= '0'; new <= s0; end if;
    when s3 => o2 <= '0'; o1 <= '0'; new <= s0;
    when others => o2 <= '0'; o1 <= '0'; new <= s0;
end case;
end process;
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 operators: `and | or | nand | nor | xor | xnor`
- Relational operators: `= | /= | < | <= | > | >=`
- Shift operators: `sll | srl | sla | sra | rol | ror`
- Adding operators: `+ | - | &`
- Sign: `+ | -`
- Multiplying operators: `* | / | mod | rem`
- Miscellaneous operators: `** | abs | not`

```
entity Operator_1 is end; architecture Behave of Operator_1 is
begin

variable b : BOOLEAN; variable bt : BIT := '1'; variable i : INTEGER;
variable pi : REAL := 3.14; variable epsilon : REAL := 0.01;
variable bv4 : BIT_VECTOR (3 downto 0) := "0001";
variable bv8 : BIT_VECTOR (0 to 7);
begin
  b := "0000" < bv4; -- b is TRUE, "0000" treated as BIT_VECTOR.
  b := 'f' > 'g'; -- b is FALSE, 'dictionary' comparison.
  bt := '0' and bt; -- bt is '0', analyzer knows '0' is BIT.
  bv4 := not bv4; -- bv4 is now "1110".
  i := 1 + 2; -- Addition, must be compatible types.
  i := 2 ** 3; -- Exponentiation, exponent must be integer.
  i := 7/3; -- Division, L/R rounded towards zero, i=2.
  i := 12 rem 7; -- Remainder, i=5. In general:
                   -- L rem R = L-(L/R)*R).
  i := 12 mod 7; -- Modulus, i=5. In general:
                  -- L mod R = L-(R*N) for an integer N.
  -- Shift := sll | srl | sla | sra | rol | ror (VHDL-93 only)
  bv4 := "1001" srl 2; -- Shift right logical, now bv4="0100".
  -- Logical shift fills with T'LEFT.
  bv4 := "1001" sra 2; -- Shift right arithmetic, now bv4="0111".
  -- Arithmetic shift fills with element at end being vacated.
  bv4 := "1001" rol 2; -- Rotate right, now bv4="0110".
  -- Rotate wraps around.
  -- Integer argument to any shift operator may be negative or zero.
```
if (pi*2.718)/2.718 = 3.14 then wait; end if; -- This is unreliable.
if (abs((pi*2.718)/2.718)-3.14)<epsilon) then wait; end if; -- Better.

bv8 := bv8(1 to 7) & bv8(0); -- Concatenation, a left rotation.
wait; end process;
end;

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

entity Arithmetic_2 is end; architecture Behave of Arithmetic_2 is

begin
    t1 : TC := 25; variable t2 : TF := 32;
    st1 : STC := 25; variable st2 : STF := 32;
begin
    -- t1 := t2; -- Illegal, different types.
    -- t1 := st1; -- Illegal, different types and subtypes.
section 10  vhdl

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;

tenity 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;

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.

dend 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_LEFT downto 0) is L; -- Descending alias,
alias XR : UNSIGNED(L_LEFT downto 0) is R; -- aligns left ends.
variable RESULT : UNSIGNED(L_LEFT downto 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_LEFT downto 0) is ARG; -- Descending range.
variable RESULT : UNSIGNED(NEW_SIZE-1 downto 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'LEFT downto 0) := XARG(RESULT'LEFT downto 0);
else -- Need to pad the result with some '0's.
RESULT(RESULT'LEFT downto XARG'LEFT + 1) := (others => '0');
RESULT(XARG'LEFT downto 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;
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;

library STD; use STD.TEXTIO.all;
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                 --21
        write(L, now);                                     --22
        write(L, STRING'(" Count="); write(L, TO_INTEGER(Count)); --23
        writeln(output, L);                                --24
    end process;                                           --25
end;
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].

```vhdl
selected_signal_assignment ::= 
with expression select
  name|aggregate <= [guarded]
  [transport][reject time_expression] inertial
  waveform when choice { | choice}
  {, waveform when choice { | choice} }
; 
```

```vhdl
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;
```

```vhdl
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;
```

```vhdl
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;
begn 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;
begn 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

\[
\text{generate_statement} ::= \\
\text{generate_label: for generate_parameter_specification} \\
\quad \text{| if boolean_expression} \\
\quad \text{generate [(block_declarative_item) begin]} \\
\quad \{\text{block_declarative_item} \} \begin{align*} \\
\text{end generate [generate_label]} ; \\
\end{align*}
\]

\[
\text{entity Full_Adder is port (X, Y, Cin : BIT; Cout, Sum: out BIT); end;} \\
\text{architecture Behave of Full_Adder is begin Sum <= X xor Y xor Cin; Cout <= (X and Y) or (X and Cin) or (Y and Cin); end;}
\]

\[
\text{entity Adder_1 is} \\
\text{port (A, B : in BIT_VECTOR (7 downto 0) := (others => '0'); Cin : in BIT := '0'; Sum : out BIT_VECTOR (7 downto 0); Cout : out BIT);} \\
\text{end;}
\]

\[
\text{architecture Structure of Adder_1 is use work.all;}
\]

\[
\text{component Full_Adder port (X, Y, Cin: BIT; Cout, Sum:out BIT);} \\
\text{end component;}
\]

\[
\text{signal C : BIT_VECTOR(7 downto 0);} \\
\text{begin AllBits : for i in 7 downto 0 generate} \\
\text{\quad LowBit : if i = 0 generate} \\
\text{\quad \quad FA : Full_Adder port map (A(0), B(0), Cin, C(0), Sum(0));} \\
\text{\quad end generate;}
\]

\[
\text{\quad OtherBits : if i /= 0 generate} \\
\text{\quad \quad FA : Full_Adder port map (A(i), B(i), C(i-1), C(i), Sum(i));} \\
\text{\quad end generate;}
\]

\[
\text{\quad end generate;}
\]

\[
\text{Cout <= C(7);} \\
\text{end;}
\]

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**

<table>
<thead>
<tr>
<th>Variables</th>
<th>Signals</th>
</tr>
</thead>
<tbody>
<tr>
<td>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;</td>
<td>entity Execute_2 is end; architecture Behave of Execute_2 is signal s1 : INTEGER := 1; signal s2 : INTEGER := 2; begin process begin s1 &lt;= s2; -- before: s1 = 1, s2 = 2 s2 &lt;= s1; -- after: s1 = 2, s2 = 1 wait; end process; end;</td>
</tr>
</tbody>
</table>

**Concurrent and sequential statements in VHDL**

<table>
<thead>
<tr>
<th>Concurrent [VHDL LRM9]</th>
<th>Sequential [VHDL LRM8]</th>
</tr>
</thead>
<tbody>
<tr>
<td>block</td>
<td>wait</td>
</tr>
<tr>
<td>process</td>
<td>assertion</td>
</tr>
<tr>
<td>concurrent_procedure_call</td>
<td>signal_assignment</td>
</tr>
<tr>
<td>concurrent_assertion</td>
<td>variable_assignment</td>
</tr>
<tr>
<td>concurrent_signal_assignment</td>
<td>procedure_call</td>
</tr>
<tr>
<td>component_instantiation</td>
<td></td>
</tr>
<tr>
<td>generate</td>
<td></td>
</tr>
<tr>
<td></td>
<td>case</td>
</tr>
<tr>
<td></td>
<td>loop</td>
</tr>
<tr>
<td></td>
<td>next</td>
</tr>
<tr>
<td></td>
<td>exit</td>
</tr>
<tr>
<td></td>
<td>return</td>
</tr>
<tr>
<td></td>
<td>null</td>
</tr>
</tbody>
</table>

_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

```vhdl
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;
```

```vhdl
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;
```

```vhdl
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**

```vhdl
configuration identifier of entity_name is
    {use_clause|attribute_specification|group_declaration}
    block_configuration
end configuration [configuration_identifier];
```

**block configuration**

```vhdl
for architecture_name
    |block_statement_label
    |generate_statement_label [(index_specification)]
    {use selected_name {, selected_name};}
    (block_configuration|component_configuration)
end for;
```

**configuration specification**

```vhdl
for instantiation_label {, instantiation_label}:component_name
    |others:component_name
    |all:component_name
    [use
        entity entity_name [(architecture_identifier)]
        configuration configuration_name
        |open]
    [generic map (generic_association_list)]
    [port map (port_association_list)];
end for;
```

**component declaration**

```vhdl
component identifier [is]
    [generic (local_generic_interface_list)];
    [port (local_port_interface_list)];
end component [component_identifier];
```

**component configuration**

```vhdl
for instantiation_label {, instantiation_label}:component_name
    |others:component_name
    |all:component_name
    [use
        entity entity_name [(architecture_identifier)]
        configuration configuration_name
        |open]
    [generic map (generic_association_list)]
    [port map (port_association_list)];
[block_configuration]
end for;
```
### 10.16 An Engine Controller

**A temperature converter**

```vhdl
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;
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T_in</td>
<td>Temperature in °C</td>
</tr>
<tr>
<td>T_out</td>
<td>Temperature in °F</td>
</tr>
</tbody>
</table>

The conversion formula from Centigrade to Fahrenheit is:

$$T(°F) = \left(\frac{9}{5}\right) \times T(°C) + 32$$

This converter uses the approximation:

$$\frac{9}{5} \approx 1.75 = 1 + 0.5 + 0.25$$
A digital filter

```vhdl
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";
begind
  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

\[ \frac{i(0) + i(1) + i(2) + i(3)}{T4} \]

is 12 bits wide.

All delays are generic TPD.
The input register

```vhdl
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 = '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;
    else if(rising_edge(clk)) then
      if (push = '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 process;
end;
A FIFO controller

```vhdl
library IEEE; use IEEE.STD_LOGIC_1164.all; use IEEE.NUMERIC_STD.all;

dentity 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
(r, 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;
end case;
when others => D <= "ZZZZZZZZZZZ" after TPD;
end case;
elsif (read = '0') then
D <= "ZZZZZZZZZZZ" after TPD; w12 <= '1' after TPD;
else D <= "ZZZZZZZZZZZ" 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
- 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

<table>
<thead>
<tr>
<th>VHDL feature</th>
<th>Example</th>
<th>93LRM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments</td>
<td>-- this is a comment</td>
<td>13.8</td>
</tr>
<tr>
<td>Literals (fixed-value items)</td>
<td>12 1.0E6 '1' &quot;110&quot; 'Z' 2#1111_1111# &quot;Hello world&quot; STRING(&quot;110&quot;)</td>
<td>13.4</td>
</tr>
<tr>
<td>Identifiers (case-insensitive, start with letter)</td>
<td>a_good_name Same same 2_Bad bad_ _bad very__bad</td>
<td>13.3</td>
</tr>
<tr>
<td>Several basic units of code</td>
<td>entity architecture configuration</td>
<td>1.1–1.3</td>
</tr>
<tr>
<td>Connections made through ports</td>
<td>port (signal in i : BIT; out o : BIT);</td>
<td>4.3</td>
</tr>
<tr>
<td>Default expression</td>
<td>port (i : BIT := '1'); -- i='1' if left open</td>
<td>4.3</td>
</tr>
<tr>
<td>No built-in logic-value system. BIT and BIT_VECTOR (STD).</td>
<td>type BIT is ('0', '1'); -- predefined signal myArray: BIT_VECTOR (7 downto 0);</td>
<td>14.2</td>
</tr>
<tr>
<td>Arrays</td>
<td>myArray(1 downto 0) &lt;= ('0', '1');</td>
<td>3.2.1</td>
</tr>
<tr>
<td>Two basic types of logic signals</td>
<td>a signal corresponds to a real wire a variable is a memory location in RAM</td>
<td>4.3.1.2 4.3.1.3</td>
</tr>
<tr>
<td>Types and explicit initial/default value</td>
<td>signal ONE : BIT := '1';</td>
<td>4.3.2</td>
</tr>
<tr>
<td>Implicit initial/default value</td>
<td>BIT'LEFT = '0'</td>
<td>4.3.2</td>
</tr>
<tr>
<td>Predefined attributes</td>
<td>clk'EVENT, clk'STABLE</td>
<td>14.1</td>
</tr>
<tr>
<td>Sequential statements inside processes model things that happen one after another and repeat</td>
<td>process begin wait until alarm = ring; eat; work; sleep; end process;</td>
<td>8</td>
</tr>
<tr>
<td>Timing with wait statement</td>
<td>wait for 1 ns; -- not wait 1 ns wait on light until light = green;</td>
<td>8.1</td>
</tr>
<tr>
<td>Update to signals occurs at the end of a simulation cycle</td>
<td>signal &lt;= 1; -- delta time delay signal &lt;= variable1 after 2 ns;</td>
<td>8.3</td>
</tr>
<tr>
<td>Update to variables is immediate</td>
<td>variable := 1; -- immediate update</td>
<td>8.4</td>
</tr>
<tr>
<td>Processes and concurrent statements model things that happen at the same time</td>
<td>process begin rain; end process; process begin sing; end process; process begin dance; end process;</td>
<td>9.2</td>
</tr>
<tr>
<td>IEEE Std_Logic_1164 (defines logic operators on 1164 types)</td>
<td>STD_ULOGIC,STD_LOGIC, STD_ULOGIC_VECTOR,STD_LOGIC_VECTOR type STD_ULOGIC is ('U','X','0','1','Z','W','L','H','-');</td>
<td></td>
</tr>
<tr>
<td>IEEE Numeric_Bit and Numeric_Std (defines arithmetic operators on BIT and 1164 types)</td>
<td>UNSIGNED and SIGNED X &lt;= &quot;10&quot; * &quot;01&quot; -- OK with numeric pkgs.</td>
<td></td>
</tr>
</tbody>
</table>


**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.
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
dendmodule //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:
assign pwr_stable = 1'b1; assign pwr_on = 1; // 1 or 1'b1 //7
assign pwr_good = pwr_on & pwr_stable; //8
initial begin //9
i = 123.456; // There must be a digit on either side //10
r = 123456e-3; // of the decimal point if it is present. //11
t = 123456e-3; // Time is rounded to 1 second by default. //12
end //13
$display("i=%0g",i," t=%6.2f",t," r=%f",r); //14
#2 $display("TIME=%0d",$time," ON="pwr_on,
  " STABLE="pwr_stable," GOOD="pwr_good);
$finish; end
endmodule

module declarations_2; //1
reg Q, Clk; wire D; //2
// Drive the wire (D):
assign D = 1; //3
// At a +ve clock edge assign the value of wire D to the reg Q:
always @(posedge Clk) Q = D; //4
initial Clk = 0; always #10 Clk = ~ Clk; //5
initial begin #50; $finish; end //6
always begin //7
  $display("T=%2g", $time," D="D," Clk="Clk," Q="Q); #10; //8
end
endmodule

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
always begin //6
  @(a or b or c or d) e = (a|b)&(c|d); //7
  $display("T=%0g",$time," e="e); //8
end
endmodule

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
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 **tior** 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'b3 and 1'b2 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

```verilog
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; //15
real R1, R2, R3; integer I1, I3, I_3;
initial begin //16
B0011 = 4'b11; Bxxx1 = 4'bx1; Bzzz1 = 4'bz1; // Left padded. //17
R1 = 0.1e1; R2 = 2.0; R3 = 30E-01; // Real numbers. //18
I1 = 1.1; I3 = 2.5; I_3 = -2.5; // IEEE rounds away from 0. //19
end //20
initial begin #1; //21
$display //22
("H12_UNSIZED, H12_SIZED (hex) = %h, %h", H12_UNSIZED, H12_SIZED); //23
$display("D42 (bin) = %b", D42, " (dec) = %d", D42); //24
$display("D123 (hex) = %h", D123, " (dec) = %d", D123); //25
$display("D63 (oct) = %o", D63); //26
$display("A (hex) = %h", A, " B (hex) = %h", B); //27
$display("C (hex) = %h", C, " D (hex) = %h", D, " E (hex) = %h", E); //28
$display("BXZ (bin) = %b", BXZ, " (hex) = %h", BXZ); //29
$display("B0011, Bxxx1, Bzzz1 (bin) = %b, %b, %b", B0011, Bxxx1, Bzzz1); //30
$display("R1, R2, R3 (e, f, g) = %e, %f, %g", R1, R2, R3); //31
$display("I1, I3, I_3 (d) = %d, %d, %d", I1, I3, I_3); //32
end //33
endmodule //34
```
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

```verilog
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
endmodule //16
```

<table>
<thead>
<tr>
<th>Value</th>
<th>Integer</th>
<th>Reg 31:0</th>
</tr>
</thead>
<tbody>
<tr>
<td>-12</td>
<td>-12</td>
<td>4294967284</td>
</tr>
<tr>
<td>'-'d12</td>
<td>4294967284</td>
<td>-12 4294967284</td>
</tr>
<tr>
<td>-32'd12</td>
<td>4294967284</td>
<td>-12 4294967284</td>
</tr>
<tr>
<td>-4'd12</td>
<td>4</td>
<td>-12 4294967284</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 4294967284</td>
</tr>
</tbody>
</table>
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)

```verilog
module characters; /*
  " is ASCII 34 (hex 22), double quote.
  ' is ASCII 39 (hex 27), tick or apostrophe.
  / is ASCII 47 (hex 2F), forward slash.
  \ is ASCII 92 (hex 5C), back slash.
  ` is ASCII 96 (hex 60), accent grave.
  | is ASCII 124 (hex 7C), vertical bar.
There are no standards for the graphic symbols for codes above 128. */
  ' is 171 (hex AB), accent acute in almost all fonts.
  " is 210 (hex D2), open double quote, like 66 (in some fonts).
  " is 211 (hex D3), close double quote, like 99 (in some fonts).
  ' is 212 (hex D4), open single quote, like 6 (in some fonts).
  ' is 213 (hex D5), close single quote, like 9 (in some fonts).
*/
endmodule

module text; //
parameter A_String = "abc"; // string constant, must be on one line
parameter Say = "Say \"Hey!\"";
// use escape quote \" for an embedded quote
parameter Tab = "\t"; // tab character
parameter NewLine = "\n"; // newline character
parameter BackSlash = "\"; // back slash
parameter Tick = "\047"; // ASCII code for tick in octal
// parameter Illegal = "\500"; // illegal - no such ASCII code
initial begin
$display("A_String(str) = %s ",A_String," (hex) = %h ",A_String);
$display("Say = %s ",Say," Say \"Hey!\" ");
$display("NewLine(str) = %s ",NewLine," (hex) = %h ",NewLine);
$display("\n(str) = %s ",BackSlash," (hex) = %h ",BackSlash);
$display("\n Tab(str) = %s ",Tab," (hex) = %h ",Tab,"1 newline... ");
$display("\n ");
$display("Tick(str) = %s ",Tick," (hex) = %h ",Tick);
#1.23; $display("Time is %t", $time);
```

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
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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>logical negation</td>
<td>!123 is 'b0 [0, 1, or x for ambiguous; legal for real]</td>
</tr>
<tr>
<td>~</td>
<td>bitwise unary negation</td>
<td>~1'b10xz is 1'b01xx</td>
</tr>
<tr>
<td>&amp;</td>
<td>unary reduction and</td>
<td>&amp; 4'b1111 is 1'b1, &amp; 2'b01 is 1'bx, &amp; 2'b01 is 1'bx</td>
</tr>
<tr>
<td>~&amp;</td>
<td>unary reduction nand</td>
<td>~&amp; 4'b1111 is 1'b0, ~&amp; 2'b01 is 1'bx</td>
</tr>
<tr>
<td></td>
<td>unary reduction or</td>
<td>Note:</td>
</tr>
<tr>
<td>~</td>
<td>unary reduction nor</td>
<td>Reduction is performed left (first bit) to right</td>
</tr>
<tr>
<td>^</td>
<td>unary reduction xor</td>
<td>Beware of the non-associative reduction operators</td>
</tr>
<tr>
<td>^</td>
<td>unary reduction xnor</td>
<td>z is treated as x for all unary operators</td>
</tr>
<tr>
<td>+</td>
<td>unary plus</td>
<td>+2'b01 is +2'b01 [+m is the same as m; legal for real]</td>
</tr>
<tr>
<td>-</td>
<td>unary minus</td>
<td>-2'b01 is x [-m is unary minus m; legal for real]</td>
</tr>
</tbody>
</table>

module operators; //1
parameter A10xz = {1'b1,1'b0,1'b0,1'b0}; // 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:
// ~ (negation), & (and), | (inclusive or),  //24
// ^ (exclusive or), ~ or ^~ (equivalence)  //25
parameter A00 = 2'b01 & 2'b10;  //26
// Unary logical reduction operators:
// & (and), ~& (nand), | (or), ~| (nor),  //27
// ^ (xor), ~^ or ^~ (xnor)  //28
parameter G1= & 4'b1111;  //29
// Conditional expression f = a ? b : c [if (a) then f=b else f=c]//30
// if a=(x or z), then (bitwise) f=0 if b=c=0, f=1 if b=c=1, else f=x //33
reg H0, a, b, c; initial begin a=1; b=0; c=1; H0=a?b:c; end //34
reg[2:0] J0lx, Jxxx, J0lz, J0ll; //35
initial begin Jxxx = 3'bxxx; J0lz = 3'b01z; J0ll = 3'b011; //36
J0lx = Jxxx ? J0lz : J0ll; end // A bitwise result. //37
initial begin #1; //38
display("A10xz=%b",A10xz," A01010101=%b",A01010101); //39
display("A1=%0d",A1," A2=%d",A2," A4=%d",A4); //40
display("B1=%b",B1," B0=%b",B0," A00x=%b",A00x); //41
display("Cl=%b",Cl," 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," G0=%b",G0," H0=%b",H0); //45
display("J0lx=%b",J0lx); 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

\begin{tabular}{ll}
hex & default \\
IA = -4'd12 & = fffffff4 -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 & = fffffffc -4 \\
IE = IA / 3 & = fffffffc -4 \\
\end{tabular}

\subsection*{11.4 Hierarchy}

\textit{Key terms and concepts:} \textit{module} • the \textit{module interface} interconnects two Verilog modules using \textit{ports} • ports must be explicitly declared as \textit{input}, \textit{output}, or \textit{inout} • a \textit{reg} cannot be \textit{input} or \textit{inout} port (to connection of a \textit{reg} to another \textit{reg}) • \textit{instantiation} • ports are linked using \textit{named association} or \textit{positional association} • \textit{hierarchical name} (\textit{ml.weekend}) • The compiler will first search downward (or inward) then upward (outward)

\begin{center}
\begin{tabular}{llll}
\textit{Verilog ports.} & \textit{Verilog port Characteristics} & \textit{input} & \textit{output} & \textit{inout} \\
\textit{wire} (or other net) & \textit{reg} or \textit{wire} (or other net) & \textit{wire} (or other net) & \\
\textit{We can read an output port inside a module} & \\
\end{tabular}
\end{center}

\begin{verbatim}
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
\end{verbatim}
$display("Day=",n,"  hours=%0d ",($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.
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 //6
Ok = 0; Fire = 0; #1 Ok = 1; #5 Fire = 1;
end
initial begin //7
$pmonitor("TIME=%0d",$time," ON="pwr_on, " STABLE=", pwr_stable," OK="Ok," FIRE="Fire," GOOD="pwr_good);
#10 $finish; end //8
endmodule //9

module assignment_2; reg Enable; wire [31:0] Data; //1
/* The following single statement is equivalent to a declaration and 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);
Enable = 0; #1; Enable = 1; #1;end //6
endmodule //7

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;
always // Statements in an always statement execute repeatedly:
begin: my_block // Start of sequential block.
  @(posedge Clk) #5 Y = 1; // At +ve edge set Y=1,
  @(posedge Clk) #5 Y = 0; // at the NEXT +ve edge set Y=0.
end // End of sequential block.
always #10 Clk = ~ Clk; // We need a clock.
initial Y = 0; // These initial statements execute
initial Clk = 0; // only once, but first.
initial $monitor("T=%2g","Clk","Y");
initial #70 $finish;
endmodule

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;
always @(A)
  Y = A; // Procedural assignment.
initial begin A=0; #5; A=1; #5; A=0; #5; $finish;
end
initial $monitor("T=%2g","A","Y");
endmodule

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

```verilog
x = #1 y; // intra-assignment delay
#1 x = y; // delayed assignment
```

```verilog
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
```

```verilog
begin // Equivalent to delayed assignment.
    #1; // Delay.
    x = y; // Assign y to x. Overall same as #1 x = y.
end
```

```verilog
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:
always begin #25 X=1;#10 X=0;#5; end //4
// An event control:
always @(posedge Clk) Y=X; // Wait for +ve clock edge. //6
always #10 Clk = !Clk; // The real clock. //7
initial begin Clk = 0; //8
 endlmodule //11

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 begin $display("Strike 1!!"); -> event_2; end // Trigger event_2. //6
always @ event_2 begin $display("Strike 2!!"); $finish; end // Stop on detection of event_2. //8
always #10 clock = ~ clock; // We need a clock. //9
initial clock = 0;
endlmodule //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"); $monitor("%3g",$time,,Clk,,,D,,Q1,,,Q2); end //7
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 $finish;end //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; //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
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"); $monitor("%g",$time,a,b,c,d,e,f,g,bds,,bsd); end //15
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 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.

<table>
<thead>
<tr>
<th>Type of Verilog assignment</th>
<th>Continuous assignment statement</th>
<th>Procedural assignment statement</th>
<th>Nonblocking procedural assignment statement</th>
<th>Procedural continuous assignment statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Where it can occur</td>
<td>outside an always or initial statement, task, or function</td>
<td>inside an always or initial statement, task, or function</td>
<td>inside an always or initial statement, task, or function</td>
<td>always or initial statement, task, or function</td>
</tr>
</tbody>
</table>

#### Example

```verilog
wire [31:0] DataBus;
assign DataBus = Enable ? Data : 32'bz

reg Y;
always @(posedge clock) Y = 1;

reg Y;
always Y <= 1;

always @(Enable)
if(Enable)
assign Q = D;
else deassign Q;
```

<table>
<thead>
<tr>
<th>Valid LHS of assignment</th>
<th>net</th>
<th>register or memory element</th>
<th>register or memory element</th>
<th>net</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid RHS of assignment</td>
<td>&lt;expression&gt; net, reg or memory element</td>
<td>&lt;expression&gt; net, reg or memory element</td>
<td>&lt;expression&gt; net, reg or memory element</td>
<td>&lt;expression&gt; net, reg or memory element</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Book</th>
<th>11.5.1</th>
<th>11.5.3</th>
<th>11.6.4</th>
<th>11.6.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Verilog LRM</td>
<td>6.1</td>
<td>9.2</td>
<td>9.2.2</td>
<td>9.3</td>
</tr>
</tbody>
</table>

```verilog
module dff_procedural_assign;
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;
#20; d = 0; #20; pre_ = 0; #20; pre_ = 1; #20; clr_ = 0;
#20; clr_ = 1; #20; d = 1; #20; $finish;end //6
initial begin
$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);
output q; input d,clear_,preset_,clock; reg q;
always @(clear_ or preset_) //3
```
if (!clear_) assign q = 0; // active-low clear
else if (!preset_) assign q = 1; // active-low preset
else deassign q;
always @(posedge clock) q = d;
endmodule

module all_assignments
//... continuous assignments.
always // beginning of procedure
begin // beginning of sequential block
//... blocking procedural assignments.
//... nonblocking procedural assignments.
//... procedural continuous assignments.
end
endmodule

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;
initial begin A = 1; B = 0; D = 2; E = 3;
C = subset_decode(A, B); F = subset_decode(D,E);
$display("A B C D E F"); $display(A,,B,,C,,D,,E,,F)
endfunction

begin if (a <= b) subset_decode = a; else subset_decode = b; end
endfunction
endmodule

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

```verilog
if (switch) Y = 1; else Y = 0;

module test_mux; reg a, b, select; wire out;
mux mux_1(a, b, out, select);
initial begin
  select = 0; a = 0; b = 1;
  #2; select = 1'bx; #2; select = 1'bz; #2; select = 1;end
initial $monitor("T=%2g",$time,"  Select=",select,"  Out=",out);
initial #10 $finish;
endmodule

module mux(a, b, mux_output, mux_select);
input a, b, mux_select;
output mux_output;
reg mux_output;
always begin
  case(mux_select)
    0: mux_output = a;
    1: mux_output = b;
    default mux_output = 1'bx; // If select = x or z set output to x.
  endcase
#1; // Need some delay, otherwise we'll spin forever.
end
endmodule

casex (instruction_register[31:29])
  3b'??1 : add;
  3b'?1? : subtract;
  3b'??? : branch;
endcase
```
11.8.2 Loop Statement

*Key terms and concepts:* A loop statement is a for, while, repeat, or forever statement •

```vhdl
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

```verilog
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

```verilog
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

<table>
<thead>
<tr>
<th>'and'</th>
<th>0</th>
<th>1</th>
<th>x</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

module primitive;
  nand (strong0, strong1) #2.2
    Nand_1(n001, n004, n005),
    Nand_2(n003, n001, n005, n002);
  nand (n006, n005, n002);
endmodule

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);
  output Sum; input Ina, InB;
  table
    // inputs : output
    00 : 0;
    01 : 1;
    10 : 1;
    11 : 0;
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 #(1:1.1:1.3:1.7) a_delay; // min:typ:max
wire #(1:1.1:1.3:1.7) 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 => y specifies a parallel connection (or parallel path) • x and y must have the same number of bits • x *> 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
DFlipFlop(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
```verilog
initial begin
A1=0; A2=1; B=1; #5; B=0; #5; A1=1; A2=0; B=1; #5; B=0;
end

initial
endmodule

`timescale 100 ps / 10 fs
module M(Z, A1, A2, B);
input A1, A2, B;
output Z;
or (Z1, A1, A2);
nand (Z, Z1, B);

/*A1 A2 B Z  Delay=10*100 ps unless indicated in the table below.*/
0 0 0 1
0 0 1 1
0 1 0 1 B:0->1 Z:1->0 delay=t2
0 1 1 0 B:1->0 Z:0->1 delay=t1
1 0 0 1 B:0->1 Z:1->0 delay=t4
1 0 1 0 B:1->0 Z:0->1 delay=t3
1 1 0 1
1 1 1 0 */
specify specparam t1 = 11, t2 = 12; specparam t3 = 13, t4 = 14;
(A1 => Z) = 10; (A2 => Z) = 10;
if (~A1) (B => Z) = (t1, t2);
if (A1) (B => Z) = (t3, t4);
endspecify
endmodule

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);
parameter CARDINALITY = 1;
input [CARDINALITY-1:0] A, B;
output [CARDINALITY-1:0] Z;
wire [CARDINALITY-1:0] Z = A & B;
endmodule

module Four_And_Gates(OutBus, InBusA, InBusB);
Vector_And #(4) My_AND(OutBus, InBusA, InBusB); // 4 AND gates
endmodule

module And_Gates(OutBus, InBusA, InBusB);
parameter WIDTH = 1;
input [WIDTH-1:0] InBusA, InBusB; output [WIDTH-1:0] OutBus;
```
### 11.12 A Viterbi Decoder

#### 11.12.1 Viterbi Encoder

```
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

iscriminated

/* 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

```
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.  

```verilog
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;
always @(Y2N or Y1N or Y0N) begin
  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

```verilog
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 // 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-si||**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 s0 to
s7.
Original by Christeen Gray, University of Hawaii. Heavily modified
by MJSS; any errors are mine. Use freely. */
/***************************************************************************/
/* module viterbi */
/***************************************************************************/
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. */

text module subset_decode
   (in0,in1,in2,in3,in4,in5,in6,in7,
    s0,s1,s2,s3,
    sout0,sout1,sout2,sout3,
    clk,reset);

text input [2:0] in0,in1,in2,in3,in4,in5,in6,in7;

text output [2:0] s0,s1,s2,s3;

text output sout0,sout1,sout2,sout3;

text input clk,reset;

text 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

_SHARED_WITHDRAWAL_AUTHORITY/>
/* module compute_metric */
_SHARED_WITHDRAWAL_AUTHORITY/>
/* 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=strength %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 point */
   // $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] xxxxxxxxx
# 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: $printtimescale 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

<table>
<thead>
<tr>
<th>Timing task argument</th>
<th>Description of argument</th>
<th>Type of argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>reference_event</td>
<td>to establish reference time</td>
<td>module input or inout (scalar or vector net)</td>
</tr>
<tr>
<td>data_event</td>
<td>signal to check against reference_event</td>
<td>module input or inout (scalar or vector net)</td>
</tr>
<tr>
<td>limit</td>
<td>time limit to detect timing violation on data_event</td>
<td>constant expression or specparam</td>
</tr>
<tr>
<td>threshold</td>
<td>largest pulse width ignored by timing check $width</td>
<td>constant expression or specparam</td>
</tr>
<tr>
<td>notifier</td>
<td>flags a timing violation (before -&gt; after): x-&gt;0, 0-&gt;1, 1-&gt;0, z-&gt;z</td>
<td>register</td>
</tr>
</tbody>
</table>

edge_control_specifier ::= edge [edge_descriptor {, edge_descriptor}]
edge_descriptor ::= 01 | 0x | 10 | 1x | x0 | x1

// timescale tasks:
module a; initial $printtimescale(b.cl); endmodule
module b; c cl (); 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:
/* $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 = (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 = 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  ;
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
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;
initialize
$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
initialize $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;
initialize
$async$and$plane(mem, {a[1], a[2], a[3]}, {b[1], b[2], b[3], b[4]});
#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.

<table>
<thead>
<tr>
<th>Status value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>1</td>
<td>queue full, cannot add</td>
</tr>
<tr>
<td>2</td>
<td>undefined q_id</td>
</tr>
<tr>
<td>3</td>
<td>queue empty, cannot remove</td>
</tr>
<tr>
<td>4</td>
<td>unsupported q_type, cannot create queue</td>
</tr>
<tr>
<td>5</td>
<td>max_length &lt;= 0, cannot create queue</td>
</tr>
<tr>
<td>6</td>
<td>duplicate q_id, cannot create queue</td>
</tr>
<tr>
<td>7</td>
<td>not enough memory, cannot create queue</td>
</tr>
</tbody>
</table>

$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) ; //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

<table>
<thead>
<tr>
<th>Verilog feature</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Comments</strong></td>
<td><code>a = 0; // comment ends with newline</code> &lt;br&gt;<code>/* This is a multilive or block</code> **comment */`</td>
</tr>
<tr>
<td><strong>Constants: string and numeric</strong></td>
<td><code>parameter BW = 32 // local, BW</code> &lt;br&gt;``define G_BUS 32 // global, <code>G_BUS</code> &lt;br&gt;<code>4'b2</code> &lt;br&gt;<code>1'b4</code></td>
</tr>
<tr>
<td><strong>Names</strong> (case-sensitive, start with letter or <code>'_</code>):</td>
<td><code>_12name A_name $BAD NotSame notsame</code></td>
</tr>
<tr>
<td><strong>Two basic types of logic signals: wire and reg</strong></td>
<td><code>wire myWire; reg myReg;</code></td>
</tr>
<tr>
<td><strong>Use continuous assignment statement with wire</strong></td>
<td><code>assign myWire = 1;</code></td>
</tr>
<tr>
<td><strong>Use procedural assignment statement with reg</strong></td>
<td><code>always myReg = myWire;</code></td>
</tr>
<tr>
<td><strong>Buses and vectors use square brackets</strong></td>
<td><code>reg [31:0] DBus; DBus[12] = 1'b4</code></td>
</tr>
<tr>
<td><strong>We can perform arithmetic on bit vectors</strong></td>
<td><code>reg [31:0] DBus; DBus = DBus + 2;</code></td>
</tr>
<tr>
<td><strong>Arithmetic is performed modulo 2^n</strong></td>
<td><code>reg [2:0] R; R = 7 + 1; // now R = 0</code></td>
</tr>
<tr>
<td><strong>Operators: as in C (but not ++ or --)</strong></td>
<td><code>1, 0, x (unknown), z (high-impedance)</code></td>
</tr>
<tr>
<td><strong>Fixed logic-value system</strong></td>
<td><code>module bake (chips, dough, cookies);</code> &lt;br&gt;<code>input chips, dough;</code> &lt;br&gt;<code>output cookies;</code> &lt;br&gt;<code>assign cookies = chips &amp; dough;</code> &lt;br&gt;<code>endmodule</code></td>
</tr>
<tr>
<td><strong>Basic unit of code is the module</strong></td>
<td><code>input or input/output ports are wire</code> &lt;br&gt;<code>output ports are wire or reg</code></td>
</tr>
<tr>
<td><strong>Procedures happen at the same time</strong></td>
<td><code>always @rain sing; always @rain dance;</code></td>
</tr>
<tr>
<td>and may be sensitive to an edge, <strong>posedge</strong>, <strong>negedge</strong>, or to a level.</td>
<td><code>always @[posedge clock] D = Q; // flop</code> &lt;br&gt;<code>always @(a or b) c = a &amp; b; // and gate</code></td>
</tr>
<tr>
<td><strong>Sequential blocks model repeating things:</strong></td>
<td><code>initial born;</code> &lt;br&gt;<code>always @alarm_clock begin : a_day</code> &lt;br&gt;<code>metro=commute; bulot=work; dodo=sleep;</code> &lt;br&gt;<code>end</code></td>
</tr>
<tr>
<td><strong>Functions and tasks</strong></td>
<td><code>function ... endfunction</code> &lt;br&gt;<code>task ... endtask</code></td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td><code>$display(&quot;a=%f&quot;,a);$dumpvars;$monitor (a)</code></td>
</tr>
<tr>
<td><strong>Control simulation</strong></td>
<td><code>$stop; $finish // sudden/gentle halt</code></td>
</tr>
<tr>
<td><strong>Compiler directives</strong></td>
<td><code>\</code>timescale 1ns/1ps // units/resolution`</td>
</tr>
<tr>
<td><strong>Delay</strong></td>
<td><code>#1 a = b; // delay then sample b</code> &lt;br&gt;<code>a = #1 b; // sample b then delay</code></td>
</tr>
</tbody>
</table>
LOGIC SYNTHESIS

*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)

<table>
<thead>
<tr>
<th></th>
<th>Path delay/ ns</th>
<th>No. of standard cells</th>
<th>No. of transistors</th>
<th>Chip area/ mils²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hand design</td>
<td>41.6</td>
<td>1,359</td>
<td>16,545</td>
<td>21,877</td>
</tr>
<tr>
<td>Synthesized design</td>
<td>36.3</td>
<td>1,493</td>
<td>11,946</td>
<td>18,322</td>
</tr>
</tbody>
</table>

```
// 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

<table>
<thead>
<tr>
<th></th>
<th>Delay /ns</th>
<th>No. of standard cells</th>
<th>No. of transistors</th>
<th>Area /mils²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hand design</td>
<td>4.3</td>
<td>12</td>
<td>116</td>
<td>68.68</td>
</tr>
<tr>
<td>Synthesized</td>
<td>2.9</td>
<td>15</td>
<td>66</td>
<td>46.43</td>
</tr>
</tbody>
</table>

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)
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
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

```verilog

`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.adl_e.v** 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

```vhdl
// 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 \( sel = \) '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

```verilog
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
module dff(D, Q, Clk, Rst); // new flip-flop for Viterbi decoder
  parameter width = 1, reset_value = 0;
  input [width - 1 : 0] D;
  output [width - 1 : 0] Q;
  reg [width - 1 : 0] Q;
  input Clk, Rst;
  initial Q <= {width{1'bx}};
  always @(posedge Clk or negedge Rst ) //5
    if ( Rst == 0 ) Q <= #1 reset_value;else Q <= #1 D;
endmodule
```

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

```verilog
/* This is the top-level module, viterbi ASIC.v */
module viterbi ASIC
  (padin0, padin1, padin2, padin3, padin4, padin5, padin6, padin7, padOut, padClk, padRes, padError);
  input [2:0] padin0, padin1, padin2, padin3,
            padin4, padin5, padin6, padin7;
  input padRes, padClk;
  output padError; output [2:0] padOut;
  wire Error, Clk, Res; wire [2:0] Out; // core
  wire padError, padClk, padRes;wire [2:0] padOut;
  wire [2:0] in0,in1,in2,in3,in4,in5,in6,in7; // core
  wire [2:0]
    padin0, padin1,padin2,padin3,padin4,padin5,padin6,padin7;
  // Do not let the software mess with the pads.
  //compass dontTouch u*
  asPadIn #(3,"1,2,3") u0 (in0, padin0);
  asPadIn #(3,"4,5,6") u1 (in1, padin1);
  asPadIn #(3,"7,8,9") u2 (in2, padin2);
  asPadIn #(3,"10,11,12") u3 (in3, padin3);
  asPadIn #(3,"13,14,15") u4 (in4, padin4);
```
The core logic of the Viterbi decoder. Bus names are abbreviated (label m_out0-3 denotes four buses: m_out0, m_out1, m_out2, and m_out3)

```vhdl
asPadIn #3,"16,17,18") u5 (in5, padin5);
asPadIn #3,"19,20,21") u6 (in6, padin6);
asPadIn #3,"22,23,24") u7 (in7, padin7);
asPadVdd #"25","both") u25 (vddb);
asPadVss #"26","both") u26 (vssb);
asPadClk #"27") u27 (Clk, padClk);
asPadOut #1,"28") u28 (padError, Error);
asPadin #1,"29") u29 (Res, padRes);
asPadOut #3,"30,31,32") u30 (padOut, Out);
// Here is the core module:
viterbi v_1
```

(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

```verilog
class MyChip_ASMIC()
    // 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

```verilog
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_ASGN(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]
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
        if (OE == 1) OutBit = InBus[sel]; else OutBit = 1'bz; //6
    end
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
endmodule //7

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;
    oneHot[1]: z <= b;
    oneHot[2]: z <= c;
    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'

```verilog
module Decoder_4To16(enable, In_4, Out_16); // 4-to-16 decoder
input enable; input [3:0] In_4; output [15:0] Out_16;
reg [15:0] Out_16;
always @(enable or In_4)
    begin Out_16 = 16'hzzzz;
        if (enable == 1)
            begin Out_16 = 16'h0000; Out_16[In_4] = 1; end
    end
endmodule

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.

```verilog
module Pri_Encoder32 (InBus, Clk, OE, OutBus); //1
input [31:0]InBus; input OE, Clk; output [4:0]OutBus;
reg j; reg [4:0]OutBus;
always@(posedge Clk)
    begin
        if (OE == 0) OutBus = 5'bzzz;
        else
            begin
                OutBus = 0;
                for (j = 31; j >= 0; j = j - 1)
                    begin
                        if (InBus[j] == 1) OutBus = j; end
            end
    end
endmodule
```

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

```verilog
module Adder_8 (A, B, Z, Cin, Cout); //1
input [7:0] A, B; input Cin; output [7:0] Z; output Cout;
assign {Cout, Z} = A + B + Cin;
endmodule

module Adder_16 (A, B, Sum, Cout); //1
input [15:0] A, B; output [15:0] Sum; output Cout;
```
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?

```verilog
sum                  //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
```
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

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
dendmodule

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
dendmodule

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

module DP_sub_B (A, B, CarryIn, Z) ;                         //1
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
endmodule                                                  //7

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.

```vhdl
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.

```vhdl
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; use IEEE.STD_LOGIC_1164 all; use IEEE.NUMERIC_STD all;

entity Decoder is port (enable : in BIT;
  Din: STD_LOGIC_VECTOR (2 downto 0);
  Dout: out STD_LOGIC_VECTOR (7 downto 0));
end Decoder;
12.6 VHDL and Logic Synthesis

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.

```vhdl
architecture Synthesis_1 of Concurrent_Decoder is --8
begin process (Din, enable) --9
variable T : STD_LOGIC_VECTOR(7 downto 0); --10
begin if (enable = '1') then --11
T := "00000000"; T( TO_INTEGER (UNSIGNED(Din))) := '1'; --12
Dout <= T; --13
else Dout <= (others => 'Z'); --14
end if; --15
end process; --16
end Synthesis_1; --17

library IEEE; --1
use IEEE.NUMERIC_STD.all; use IEEE.STD_LOGIC_1164 all; --2

database Synthesis_1 of 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 Adder_1 is --8
begin process (Din, enable) --9
variable T : STD_LOGIC_VECTOR(7 downto 0); --10
begin if (enable = '1') then --11
T := "00000000"; T( TO_INTEGER (UNSIGNED(Din))) := '1'; --12
Dout <= T; --13
else Dout <= (others => 'Z'); --14
end if; --15
end process; --16
end Synthesis_1; --17

library IEEE; --1
use IEEE.NUMERIC_STD.all; use IEEE.STD_LOGIC_1164 all; --2

database Synthesis_1 of Adder_1 is --3
end Adder_1; --4

architecture Synthesis_1 of Adder_1 is --5
end Synthesis_1; --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

```vhdl
'timescale 1ns/1ns  //1
module halfgate (myInput, myOutput);  //2
input myInput; output myOutput; wire myOutput;
   assign myOutput = ~myInput;
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;
begin  //18
   u2: in01d0 port map ( I => myInput, ZN => myOutput );  //19
end halfgate_u;  //20

--compass compile_off -- synopsys etc.
library cb60hd230d;  //22
configuration halfgate_u_CON of halfgate_u is  //23
   for halfgate_u
      for u2 : in01d0 use configuration cb60hd230d.in01d0_CON  //25
   generic map (  //26
```
ZN_cap => 0.0100 + myOutput_cap, 
INSTACE_NAME => INSTACE_NAME&"/u2" )
port map ( I => I, ZN => ZN);
end for;
end for;
end halfgate_u_CON;
--compass compile_on -- synopsys etc.

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;
use IEEE.STD_LOGIC_1164.all; use COMPASS_LIB.STDCOMP all;
entity Ripple_4 is
  port (Trig, Reset: STD_LOGIC; QN0_5x:out STD_LOGIC;
    Q : inout STD_LOGIC_VECTOR(0 to 3));
end Ripple_4;
architecture structure of Ripple_4 is
  signal QN : STD_LOGIC_VECTOR(0 to 3);
component in01d1
  port ( I : in Std_Logic; ZN : out Std_Logic );end component;
component in01d5
  port ( I : in Std_Logic; ZN : out Std_Logic );end component;
begi
--compass dontTouch inv5x -- synopsys dont_touch etc.
-- Named association for hand-instantiated library cells:
--                           Q          D        Clk   Rst
inv5x: IN01D5 port map( I=>Q(0), ZN=>QN0_5x );
inv0 : IN01D1 port map ( I=>Q(0), ZN=>QN(0) );
inv1 : IN01D1 port map ( I=>Q(1), ZN=>QN(1) );
inv2 : IN01D1 port map ( I=>Q(2), ZN=>QN(2) );
inv3 : IN01D1 port map ( I=>Q(3), ZN=>QN(3) );
-- Positional association for standard components:

--
  Q  D  Clk  Rst
d0: asdff port map(Q (0 to 0), QN(0 to 0), Trig, Reset);
d1: asdff port map(Q (1 to 1), QN(1 to 1), Q(0), Reset);
d2: asdff port map(Q (2 to 2), QN(2 to 2), Q(1), Reset);
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 [3:0] trig; input reset; output qn0_5x; inout [3:0] q; //3
wire [3:0] qn; supply1 VDD; supply0 VSS; //4
in0d5 inv5x (.I(q[0]),.ZN(qn0_5x)); //5
in0d1 inv0 (.I(q[0]),.ZN(qn[0])); //6
in0d1 inv1 (.I(q[1]),.ZN(qn[1])); //7
in0d1 inv2 (.I(q[2]),.ZN(qn[2])); //8
in0d1 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 (   clk : in STD_LOGIC ; res : in STD_LOGIC ;   SI : in STD_LOGIC ; PO : out STD_LOGIC_VECTOR(3 downto 0)); end;

architecture Synthesis_1 of SIPO_R is   signal PO_t : STD_LOGIC_VECTOR(3 downto 0);   begin   process (PO_t) begin PO <= PO_t; end process;   process (clk, res) begin   if (res = '0') then PO_t <= (others => '0');   elsif (rising_edge(clk)) then PO_t <= SI & PO_t(3 downto 1);   end if;   end process; end Synthesis_1;

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; use IEEE.STD_LOGIC_1164 all; use IEEE.NUMERIC_STD all;

entity Adder4 is port (   in1, in2 : in BIT_VECTOR(3 downto 0) ;   mySum : out BIT_VECTOR(3 downto 0) ) ; end Adder4;

architecture Behave_A of Adder4 is function DIY(L,R: BIT_VECTOR(3 downto 0)) return BIT_VECTOR is variable sum:BIT_VECTOR(3 downto 0);variable lt,rt,st,cry: BIT;   begin cry := '0';   for i in L'REVERSE_RANGE loop   lt := L(i); rt := R(i); st := lt xor rt;   sum(i):= st xor cry; cry:= (lt and rt) or (st and cry);   end loop;   return sum; end;

begin mySum <= DIY (in1, in2); -- do it yourself (DIY) add end Behave_A;

library IEEE; use IEEE.STD_LOGIC_1164 all; use IEEE.NUMERIC_STD all;
entity Adder4 is port (  
in1, in2 : in UNSIGNED(3 downto 0);  
mySum : out UNSIGNED(3 downto 0) );  
end Adder4;

architecture Behave_B of Adder4 is  
begin mySum <= in1 + in2; -- This uses an overloaded '+'.  
end Behave_B;

12.6.10 Adder/Subtracter 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;
use IEEE.STD_LOGIC_1164.all; use IEEE.NUMERIC_STD.all;
entity Adder_Subtracter is port (  
xin : in UNSIGNED(15 downto 0);  
clk, addsub, clr: in STD_LOGIC;  
result : out UNSIGNED(15 downto 0));  
end Adder_Subtracter;

architecture Behave_A of Adder_Subtracter is  
signal addout, result_t: UNSIGNED(15 downto 0);  
begin  
result <= result_t;  
with addsub select  
addout <= (xin + result_t) when '1',  
(xin - result_t) when '0',  
(others => '-') when others;  
process (clr, clk) begin  
if (clr = '0') then result_t <= (others => '0');  
elif rising_edge(clk) then result_t <= addout;  
end if;  
end process;  
end Behave_A;

architecture Behave_B of Adder_Subtracter is  
signal result_t: UNSIGNED(15 downto 0);  
begin  
result <= result_t;  
process (clr, clk) begin  
if (clr = '0') then result_t <= (others => '0');  
elif rising_edge(clk) then  
case addsub is  
when '1' => result_t <= (xin + result_t);  
when '0' => result_t <= (xin - result_t);

end process;  
end Behave_B;
when others => result_t <= others => '‐');  --11
end case;
end if;
end process;
end Behave_B;  --15

12.7 Finite-State Machine Synthesis

Key terms and concepts: synthesis of a finite-state machine (FSM) • let the logic synthesizer operate on the state machine as random logic • use directives to guide the logic synthesis tool to improve or modify state assignment • use a special state-machine compiler • FSM encoding options • Adjacent encoding (Gray codes) • One-hot encoding • Random encoding • User-specified encoding (keep explicit state assignment) • Moore encoding (useful for FSMs that require fast outputs)

12.7.1 FSM Synthesis in Verilog

Key terms and concepts: FSM paired processes • one process synthesizes to sequential logic and the second process synthesizes to combinational logic • pseudocomments to define the states and state vector

```verilog
`define resSt 0 //1
`define S1 1 //2
`define S2 2 //3
`define S3 3 //4
module StateMachine_1 (reset, clk, yOutReg); //5
input reset, clk; output yOutReg; //6
reg yOutReg, yOut; reg [1:0] curSt, nextSt; //7
always @(posedge clk or posedge reset) //8
begin:Seq //Compass statemachine oneHot curSt //9
if (reset == 1) //10
begin yOut = 0; yOutReg = yOut; curSt = `resSt;end //11
else begin //12
case (curSt) //13
`resSt:yOut = 0;`S1:yOut = 1;`S2:yOut = 1;`S3:yOut = 1; //14
default:yOut = 0;
endcase //16
yOutReg = yOut; curSt = nextSt; // ... update the state. //17
end //18
```

```
always @ (curSt or yOut) // Assign the next state:
begin:Comb
    case (curSt)
        `resSt:nextSt = `S3; `S1:nextSt = `S2;
        `S2:nextSt = `S1; `S3:nextSt = `S1;
    default:nextSt = `resSt;
endcase
end
endmodule

module StateMachine_2 (reset, clk, yOutReg);
    input reset, clk;
    output yOutReg, yOut;
    reg yOutReg, yOut;
    parameter [1:0] //synopsys enum states
        resSt = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11;
    reg [1:0] /* synopsys enum states */ curSt, nextSt;
    //synopsys state_vector curSt
    always @(posedge clk or posedge reset) begin
        if (reset == 1)
            begin
                yOut = 0; yOutReg = yOut; curSt = resSt;
            end
        else begin
            case (curSt)
                resSt: yOut = 0; S1: yOut = 1; S2: yOut = 1; S3: yOut = 1;
                default: yOut = 0;
            endcase
            yOutReg = yOut; curSt = nextSt;
        end
    end
always @(curSt or yOut) begin
    case (curSt)
        resSt:nextSt = S3; S1:nextSt = S2; S2:nextSt = S1; S3:nextSt = S1;
    default:nextSt = S1;
endcase
end
endmodule

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;
entity SM1 is
  port (aIn, clk : in Std_logic; yOut: out Std_logic);
end SM1;
architecture Moore of SM1 is
  type state is (s1, s2, s3, s4);
signal pS, nS : state;
begin
  process (aIn, pS) begin
    case pS is
      when s1 => yOut <= '0'; nS <= s4;
      when s2 => yOut <= '1'; nS <= s3;
      when s3 => yOut <= '1'; nS <= s1;
      when s4 => yOut <= '1'; nS <= s2;
    end case;
  end process;
  process begin
    -- synopsys etc.
    --compass Statemachine adj pS
    wait until clk = '1'; pS <= nS;
  end process;
end Moore;

library IEEE; use IEEE.STD_LOGIC_1164 all;
entity SM2 is
  port (aIn, clk : in Std_logic; yOut: out Std_logic);
end SM2;
architecture Mealy of SM2 is
  type state is (s1, s2, s3, s4);
signal pS, nS : state;
begin
  process(aIn, pS) begin
    case pS is
      when s1 => if (aIn = '1')
        then yOut <= '0'; nS <= s4;
        else yOut <= '1'; nS <= s3;
      end if;
      when s2 => yOut <= '1'; nS <= s3;
      when s3 => yOut <= '1'; nS <= s1;
    end case;
  end process;
end Mealy;
when s4 => if (aIn = '1')
  then yOut <= '1'; nS <= s2;
  else yOut <= '0'; nS <= s1;
  end if;
end case;
end process;

process begin
  wait until clk = '1' ;
  --Compass Statemachine oneHot pS
  pS <= nS;
end process;
end Mealy;

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

```verilog
reg [31:0] MyMemory [3:0]; // a 4 x 32-bit register

module RAM_1(A, CEB, WEB, OEB, INN, OUTT);
  input [6:0] A; input CEB,WEB,OEB; input [4:0]INN;
  output [4:0] OUTT;
  reg [4:0] OUTT; reg [4:0] int_bus; reg [4:0] memory [127:0];
always@ (negedge CEB) begin
  if (CEB == 0) begin
    if (WEB == 1) int_bus = memory[A];
    else if (WEB == 0) begin memory[A] = INN; int_bus = INN; end
    else int_bus = 5'bxxxxx;
  end
end
always@ (OEB or int_bus) begin
  case (OEB) 0 : OUTT = int_bus;
  default : OUTT = 5'bzzzzz; endcase
```

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

```vhdl
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;
```

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.all;
package RAM_package is
constant numOut : INTEGER := 8; -- So is this.
constant wordDepth: INTEGER := 8;
constant numAddr : INTEGER := 3;
subtype MEMV is STD_LOGIC_VECTOR(numOut-1 downto 0);
type MEM is array (wordDepth-1 downto 0) of MEMV;
end RAM_package;
library IEEE;
use IEEE.STD_LOGIC_1164.all; use IEEE.NUMERIC_STD.all;
use work.RAM_package.all;
entity RAM_1 is
  port (signal A : in STD_LOGIC_VECTOR(numAddr-1 downto 0));
signal CEB, WEB, OEB : in STD_LOGIC;
signal INN : in MEMV;
signal OUTT : out MEMV);
end RAM_1;
architecture Synthesis_1 of RAM_1 is
  signal i_bus : MEMV; -- RAM internal data latch
  signal mem : MEM; -- RAM data
begin
  process begin
    wait until CEB = '0';
end process begin;
```

```vhdl
end
endmodule
```
if WEB = '1' then i_bus <= mem(TO_INTEGER(UNSIGNED(A))); --25
elsif WEB = '0' then
  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
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;
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

```vhdl
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
The comparator/MUX example after logic optimization with timing constraints

The structural netlist, `comp_mux_o2.v`, and its derived schematic
SIMULATION

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

```vhdl
// comp_mux.v //1
module comp_mux(a, b, outp);
    input [2:0] a, b;
    output [2:0] outp;
endmodule

function [2:0] compare;
    input [2:0] ina, inb;
begin
    if (ina <= inb) compare = ina;
    else compare = inb;
endfunction

assign outp = compare(a, b);
endmodule

// testbench.v //1
module comp_mux_testbench;
    integer i, j;
    reg [2:0] x, y, smaller;
    wire [2:0] z;
always @(x) $display("t
x y actual calculated");
initial $monitor("%4g",$time,,x,,y,,z,,,,,,smaller);
initial $dumpvars; initial #1000 $finish;
initial
```

1
begin
for (i = 0; i <= 7; i = i + 1)
begin
  for (j = 0; j <= 7; j = j + 1)
begin
    x = i; y = j; smaller = (x <= y) ? x : y;
    #1 if (z  != smaller) $display("error");
  end
end
comp_mux v_1 (x, y, z);
endmodule

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)
(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
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]

\[\begin{align*}
    &a = 'D6 [0] \text{ (input)(display)} \\
    &b = 'D7 [0] \text{ (input)(display)} \\
    &\text{outp} = 'Buuu ('Du) [0] \text{ (display)} \\
    &\text{outp} \rightarrow 'Bluu ('Du) [.47] \\
    &\text{outp} \rightarrow 'B11u ('Du) [.97] \\
    &\text{outp} \rightarrow 'D6 [4.08] \\
    &a \rightarrow 'D7 [10] \\
    &b \rightarrow 'D6 [10] \\
    &\text{outp} \rightarrow 'D7 [10.97] \\
    &\text{outp} \rightarrow 'D6 [14.15] \\
\end{align*}\]

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] \\

@data
\[\begin{align*}
    &.00 \quad a \rightarrow 'D6 \\
    &.00 \quad b \rightarrow 'D7 \\
    &.00 \quad \text{outp} \rightarrow 'Du \\
    &.53 \quad \text{outp} \rightarrow 'Du \\
    &.93 \quad \text{outp} \rightarrow 'Du \\
    &4.42 \quad \text{outp} \rightarrow 'D6 \\
    &10.00 \quad a \rightarrow 'D7 \\
    &10.00 \quad b \rightarrow 'D6 \\
    &11.03 \quad \text{outp} \rightarrow 'D7 \\
    &14.43 \quad \text{outp} \rightarrow 'D6 \\
\end{align*}\]

### 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

<table>
<thead>
<tr>
<th>Logic state</th>
<th>Logic level</th>
<th>Logic value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>zero</td>
<td>zero</td>
</tr>
<tr>
<td>1</td>
<td>one</td>
<td>one</td>
</tr>
<tr>
<td>x</td>
<td>zero or one</td>
<td>unknown</td>
</tr>
<tr>
<td>z</td>
<td>zero, one, or neither</td>
<td>high impedance</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$R{A, B}$</th>
<th>$B = 0$</th>
<th>$B = 1$</th>
<th>$B = X$</th>
<th>$B = Z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A = 0$</td>
<td>0</td>
<td>$X$</td>
<td>$X$</td>
<td>0</td>
</tr>
<tr>
<td>$A = 1$</td>
<td>$X$</td>
<td>1</td>
<td>$X$</td>
<td>1</td>
</tr>
<tr>
<td>$A = X$</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
</tr>
<tr>
<td>$A = Z$</td>
<td>0</td>
<td>1</td>
<td>$X$</td>
<td>$Z$</td>
</tr>
</tbody>
</table>

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 \textit{'one'}, a \textit{resistive \textit{'one}}, with \textit{resistive strength \textbullet{} high impedance \textbullet{} Verilog logic system \textbullet{} VHDL signal resolution} using \textit{VHDL signal-resolution functions}

A 12-state logic system

<table>
<thead>
<tr>
<th>Logic strength</th>
<th>Logic level</th>
<th>Logic value</th>
</tr>
</thead>
<tbody>
<tr>
<td>strong</td>
<td>zero</td>
<td>S0</td>
</tr>
<tr>
<td>weak</td>
<td>unknown</td>
<td>WX</td>
</tr>
<tr>
<td>high impedance</td>
<td>one</td>
<td>S1</td>
</tr>
<tr>
<td>unknown</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Verilog logic strengths

<table>
<thead>
<tr>
<th>Logic strength</th>
<th>Strength number</th>
<th>Models</th>
<th>Abbreviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>supply drive</td>
<td>7</td>
<td>power supply</td>
<td>supply Su</td>
</tr>
<tr>
<td>strong drive</td>
<td>6</td>
<td>default gate and assign output strength</td>
<td>strong St</td>
</tr>
<tr>
<td>pull drive</td>
<td>5</td>
<td>gate and assign output strength</td>
<td>pull Pu</td>
</tr>
<tr>
<td>large capacitor</td>
<td>4</td>
<td>size of trireg net capacitor</td>
<td>large La</td>
</tr>
<tr>
<td>weak drive</td>
<td>3</td>
<td>gate and assign output strength</td>
<td>weak We</td>
</tr>
<tr>
<td>medium capacitor</td>
<td>2</td>
<td>size of trireg net capacitor</td>
<td>medium Me</td>
</tr>
<tr>
<td>small capacitor</td>
<td>1</td>
<td>size of trireg net capacitor</td>
<td>small Sm</td>
</tr>
<tr>
<td>high impedance</td>
<td>0</td>
<td>not applicable</td>
<td>highz Hi</td>
</tr>
</tbody>
</table>

The nine-value logic system, IEEE Std 1164-1993.

<table>
<thead>
<tr>
<th>Logic state</th>
<th>Logic value</th>
<th>Logic state</th>
<th>Logic value</th>
</tr>
</thead>
<tbody>
<tr>
<td>'0'</td>
<td>strong low</td>
<td>'X'</td>
<td>strong unknown</td>
</tr>
<tr>
<td>'1'</td>
<td>strong high</td>
<td>'W'</td>
<td>weak unknown</td>
</tr>
<tr>
<td>'L'</td>
<td>weak low</td>
<td>'Z'</td>
<td>high impedance</td>
</tr>
<tr>
<td>'H'</td>
<td>weak high</td>
<td>'-'</td>
<td>don't care</td>
</tr>
<tr>
<td>'U'</td>
<td>uninitialized</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\texttt{function}\texttt{ "and"}(l, r : std_ulogic_vector)\texttt{return}\texttt{ std_ulogic_vector} \texttt{is}\texttt{ --1}
\texttt{alias lv : std_ulogic_vector (1to l'LENGTH ) is l;} \texttt{--2}
\texttt{alias rv : std_ulogic_vector (1to r'LENGTH ) is r;} \texttt{--3}
\texttt{variable result : std_ulogic_vector (1to l'LENGTH );} \texttt{--4}
### 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

```plaintext
model nd01d1 (a, b, zn)
function (a, b) !(a & b); function end
model end
define 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:
Op <= transport Ip after 10 ns;            --1
Op <= transport Ip after 10 ns, not Ip after 20 ns; --3
```
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 = "stdcell135";  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

-- Their equivalent assignments: --4
Op <= reject 0 ns inertial Ip after 10 ns;
Op <= reject 0 ns inertial Ip after 10 ns, not Ip after 10 ns; --6
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
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

```verilog
'celldefine          //1
'delay_mode_path     //2
'suppress_faults    //3
'enable_portfaults  //4
'timescale 1 ns / 1 ps  //5
module in0ld1 (zn, i); input i; output zn; not G2(zn, i);  //6
specify specparam
    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
celldefine          //17

timescale 1 ns / 1 ps  //1
module SDF_b; reg A; in0ld1 il (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; use IEEE.VITAL_timing all; use IEEE.VITAL_primitives all;
entity IN01D1 is
  generic ( 
    tipd_I : VitalDelayType01 := (0 ns, 0 ns); 
    tpd_I_ZN : VitalDelayType01 := (0 ns, 0 ns) ); 
  port ( 
    I : in STD_LOGIC := 'U'; 
    ZN : out STD_LOGIC := 'U' );
attribute VITAL_LEVEL0 of IN01D1 : entity is TRUE;
end IN01D1;
architecture IN01D1 of IN01D1 is
attribute VITAL_LEVEL1 of IN01D1 : architecture is TRUE;
signal I_ipd : STD_LOGIC := 'X';
begin
  WIREDELAY: block 
  begin VitalWireDelay(I_ipd, I, tipd_I); end block;
VITALbehavior : process (I_ipd) --18
variable ZN_zd : STD_LOGIC; --19
variable ZN_GlitchData : VitalGlitchDataType; --20
begin
ZN_zd := VitalINV(I_ipd); --21
VitalPathDelay01(
    OutSignal => ZN,
    OutSignalName => "ZN",
    OutTemp => ZN_zd,
    Paths => (0 => (I_ipd'LAST_EVENT, tpd_I_ZN, TRUE)),
    GlitchData => ZN_GlitchData,
    DefaultDelay => VitalZeroDelay01,
    Mode => OnEvent,
    MsgOn => FALSE,
    XOn => TRUE,
    MsgSeverity => ERROR);
end process;
end IN01D1; --35

library IEEE; use IEEE.STD_LOGIC_1164.all; --1
entity SDF is port ( A : in STD_LOGIC; B : out STD_LOGIC ); --2
end SDF;
architecture SDF of SDF is --4
component in01d1 port ( I : in STD_LOGIC; ZN : out STD_LOGIC ); --5
end component;
begin i1: in01d1 port map ( I => A, ZN => B); --7
end SDF; --8

library STD; use STD.TEXTIO.all; --1
library IEEE; use IEEE.STD_LOGIC_1164.all; --2
entity SDF_testbench is end SDF_testbench; --3
architecture SDF_testbench of SDF_testbench is --4
component SDF port ( A : in STD_LOGIC; B : out STD_LOGIC ); --5
end component; --6
signal A, B : STD_LOGIC := '0'; --7
begin
SDF_b : SDF port map ( A => A, B => B); --9
process begin
A <= '0'; wait for 5 ns; A <= '1';
wait for 5 ns; A <= '0'; wait;
end process;
process (A, B) variable L: LINE; begin --14
write(L, now, right, 10, TIME'(ps)); --15
write(L, STRING'(" A=")); write(L, TO_BIT(A)); --16
write(L, STRING'(" B=")); write(L, TO_BIT(B)); --17
writeline(output, L); --18
end process; --19
end SDF_testbench; --20

(DelayFile
(DelayVersion "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
(DelayVersion "1.0")
(Design "halfgateASIC_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
(Design "MYDESIGN")
(Date "26 AUG 1996")
(Vendor "ASICS INC")
(Program "SDF_GEN")
(Version "3.0")
(Divider .)


(TIME SCALE )

(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
  InCap$i = 0.060, OutCap$zn = 0.038, MaxLoad$zn = 1.538,
  R_Ramp$i$zn = 0.542:0.980:1.750, F_Ramp$i$zn = 0.605:1.092:1.950;
  specparam cell_count = 1.000000; specparam Transistors = 4 ;
  specparam Power = 1.400000; specparam MaxLoadedRamp = 3 ;
  (i=>zn)=(0.031:0.056:0.100, 0.028:0.050:0.090);

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)

<table>
<thead>
<tr>
<th>Library</th>
<th>inv1</th>
<th>invh</th>
<th>invs</th>
<th>inv8</th>
<th>inv12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
<td>0.034</td>
<td>0.067</td>
<td>0.133</td>
<td>0.265</td>
<td>0.397</td>
</tr>
<tr>
<td>Performance</td>
<td>0.145</td>
<td>0.292</td>
<td>0.584</td>
<td>1.169</td>
<td>1.753</td>
</tr>
</tbody>
</table>
Delay information for a 2:1 MUX

<table>
<thead>
<tr>
<th>From input</th>
<th>To output</th>
<th>Propagation delay</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Area</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extrinsic / nspF^{-1}</td>
</tr>
<tr>
<td>D0\</td>
<td>Z\</td>
<td>2.10</td>
</tr>
<tr>
<td>D0/</td>
<td>Z/</td>
<td>3.66</td>
</tr>
<tr>
<td>D1\</td>
<td>Z\</td>
<td>2.10</td>
</tr>
<tr>
<td>D1/</td>
<td>Z/</td>
<td>3.66</td>
</tr>
<tr>
<td>SD\</td>
<td>Z\</td>
<td>2.10</td>
</tr>
<tr>
<td>SD/</td>
<td>Z/</td>
<td>3.66</td>
</tr>
<tr>
<td>SD/</td>
<td>Z/</td>
<td>2.10</td>
</tr>
<tr>
<td>SD/</td>
<td>Z/</td>
<td>3.66</td>
</tr>
</tbody>
</table>

Process derating factors

<table>
<thead>
<tr>
<th>Process</th>
<th>Derating factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow</td>
<td>1.31</td>
</tr>
<tr>
<td>Nominal</td>
<td>1.0</td>
</tr>
<tr>
<td>Fast</td>
<td>0.75</td>
</tr>
</tbody>
</table>

Temperature and voltage derating factors

<table>
<thead>
<tr>
<th>Temperature/°C</th>
<th>4.5V</th>
<th>4.75V</th>
<th>5.00V</th>
<th>5.25V</th>
<th>5.50V</th>
</tr>
</thead>
<tbody>
<tr>
<td>-40</td>
<td>0.77</td>
<td>0.73</td>
<td>0.68</td>
<td>0.64</td>
<td>0.61</td>
</tr>
<tr>
<td>0</td>
<td>1.00</td>
<td>0.93</td>
<td>0.87</td>
<td>0.82</td>
<td>0.78</td>
</tr>
<tr>
<td>25</td>
<td>1.14</td>
<td>1.07</td>
<td>1.00</td>
<td>0.94</td>
<td>0.90</td>
</tr>
<tr>
<td>85</td>
<td>1.50</td>
<td>1.40</td>
<td>1.33</td>
<td>1.26</td>
<td>1.20</td>
</tr>
<tr>
<td>100</td>
<td>1.60</td>
<td>1.49</td>
<td>1.41</td>
<td>1.34</td>
<td>1.28</td>
</tr>
<tr>
<td>125</td>
<td>1.76</td>
<td>1.65</td>
<td>1.56</td>
<td>1.47</td>
<td>1.41</td>
</tr>
</tbody>
</table>
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, $D_{ISM}(50 \%, \text{FR})$, as follows (0.5 trip points):

$$D_{ISM} (50\%, \text{FR}) = A_0 + D_0 C_L + 0.5 O_R = A_0 + D_0 C_L + d_A / 2 + d_D C_L / 2$$

$$= 0.0015 + 0.5 \times 0.0789 + (-0.2828 + 0.5 \times 4.6642) C_L$$

$$= 0.041 + 2.05 C_L$$

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

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Fanout</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FO = 0 /ns</td>
<td>FO = 1 /ns</td>
</tr>
<tr>
<td>$t_{PLH}$</td>
<td>Propagation delay, A to X</td>
<td>0.25</td>
</tr>
<tr>
<td>$t_{PHL}$</td>
<td>Propagation delay, B to X</td>
<td>0.17</td>
</tr>
<tr>
<td>$t_r$</td>
<td>Output rise time, X</td>
<td>1.01</td>
</tr>
<tr>
<td>$t_f$</td>
<td>Output fall time, X</td>
<td>0.54</td>
</tr>
</tbody>
</table>
13.7 Static Timing Analysis

Key terms and concepts: static timing analysis • pipelining • critical path

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>FO = 0</th>
<th>FO = 1</th>
<th>FO = 2</th>
<th>FO = 4</th>
<th>FO = 8</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>/ns</td>
<td>/ns</td>
<td>/ns</td>
<td>/ns</td>
<td>/ns</td>
<td>/nspF−1</td>
</tr>
<tr>
<td>( t_{PLH} )</td>
<td>Delay, A to S (B = '0')</td>
<td>0.58</td>
<td>0.68</td>
<td>0.78</td>
<td>0.98</td>
<td>1.38</td>
<td>1.25</td>
</tr>
<tr>
<td>( t_{PHL} )</td>
<td>Delay, A to S (B = '1')</td>
<td>0.93</td>
<td>0.97</td>
<td>1.00</td>
<td>1.08</td>
<td>1.24</td>
<td>0.48</td>
</tr>
<tr>
<td>( t_{PLH} )</td>
<td>Delay, B to S (B = '0')</td>
<td>0.89</td>
<td>0.99</td>
<td>1.09</td>
<td>1.29</td>
<td>1.69</td>
<td>1.25</td>
</tr>
<tr>
<td>( t_{PHL} )</td>
<td>Delay, B to S (B = '1')</td>
<td>1.00</td>
<td>1.04</td>
<td>1.08</td>
<td>1.15</td>
<td>1.31</td>
<td>0.48</td>
</tr>
<tr>
<td>( t_{PLH} )</td>
<td>Delay, A to CO</td>
<td>0.43</td>
<td>0.53</td>
<td>0.63</td>
<td>0.83</td>
<td>1.23</td>
<td>1.25</td>
</tr>
<tr>
<td>( t_{PHL} )</td>
<td>Delay, A to CO</td>
<td>0.59</td>
<td>0.63</td>
<td>0.67</td>
<td>0.75</td>
<td>0.90</td>
<td>0.48</td>
</tr>
<tr>
<td>( t_r )</td>
<td>Output rise time, X</td>
<td>1.01</td>
<td>1.28</td>
<td>1.56</td>
<td>2.10</td>
<td>3.19</td>
<td>3.40</td>
</tr>
<tr>
<td>( t_f )</td>
<td>Output fall time, X</td>
<td>0.54</td>
<td>0.69</td>
<td>0.84</td>
<td>1.13</td>
<td>1.71</td>
<td>1.83</td>
</tr>
</tbody>
</table>

Switching characteristics of a half adder

Fanout

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>FO = 0</th>
<th>FO = 1</th>
<th>FO = 2</th>
<th>FO = 4</th>
<th>FO = 8</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FO = 0</td>
<td>FO = 1</td>
<td>FO = 2</td>
<td>FO = 4</td>
<td>FO = 8</td>
<td>K</td>
<td></td>
</tr>
<tr>
<td></td>
<td>/ns</td>
<td>/ns</td>
<td>/ns</td>
<td>/ns</td>
<td>/ns</td>
<td>/nspF−1</td>
<td></td>
</tr>
<tr>
<td>( t_{PLH} )</td>
<td>Delay, A to S (B = '0')</td>
<td>0.58</td>
<td>0.68</td>
<td>0.78</td>
<td>0.98</td>
<td>1.38</td>
<td>1.25</td>
</tr>
<tr>
<td>( t_{PHL} )</td>
<td>Delay, A to S (B = '1')</td>
<td>0.93</td>
<td>0.97</td>
<td>1.00</td>
<td>1.08</td>
<td>1.24</td>
<td>0.48</td>
</tr>
<tr>
<td>( t_{PLH} )</td>
<td>Delay, B to S (B = '0')</td>
<td>0.89</td>
<td>0.99</td>
<td>1.09</td>
<td>1.29</td>
<td>1.69</td>
<td>1.25</td>
</tr>
<tr>
<td>( t_{PHL} )</td>
<td>Delay, B to S (B = '1')</td>
<td>1.00</td>
<td>1.04</td>
<td>1.08</td>
<td>1.15</td>
<td>1.31</td>
<td>0.48</td>
</tr>
<tr>
<td>( t_{PLH} )</td>
<td>Delay, A to CO</td>
<td>0.43</td>
<td>0.53</td>
<td>0.63</td>
<td>0.83</td>
<td>1.23</td>
<td>1.25</td>
</tr>
<tr>
<td>( t_{PHL} )</td>
<td>Delay, A to CO</td>
<td>0.59</td>
<td>0.63</td>
<td>0.67</td>
<td>0.75</td>
<td>0.90</td>
<td>0.48</td>
</tr>
</tbody>
</table>

\( t_{r} \) 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

Rise delay, Worst case

<table>
<thead>
<tr>
<th>Instance name</th>
<th>in pin--&gt;out pin</th>
<th>( tr )</th>
<th>total</th>
<th>incr</th>
<th>cell</th>
</tr>
</thead>
<tbody>
<tr>
<td>END_OF_PATH</td>
<td>outp_2_</td>
<td>R</td>
<td>27.26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT1</td>
<td>: D----&gt;PAD</td>
<td>R</td>
<td>27.26</td>
<td>7.55</td>
<td>OUTBUF</td>
</tr>
<tr>
<td>I_1_CM8</td>
<td>: S11----&gt;Y</td>
<td>R</td>
<td>19.71</td>
<td>4.40</td>
<td>CM8</td>
</tr>
<tr>
<td>I_2_CM8</td>
<td>: S11----&gt;Y</td>
<td>R</td>
<td>15.31</td>
<td>5.20</td>
<td>CM8</td>
</tr>
<tr>
<td>I_3_CM8</td>
<td>: S11----&gt;Y</td>
<td>R</td>
<td>10.11</td>
<td>4.80</td>
<td>CM8</td>
</tr>
<tr>
<td>IN1</td>
<td>: PAD----&gt;Y</td>
<td>R</td>
<td>5.32</td>
<td>5.32</td>
<td>INBUF</td>
</tr>
<tr>
<td>a_2_</td>
<td></td>
<td>R</td>
<td>0.00</td>
<td>0.00</td>
<td></td>
</tr>
</tbody>
</table>

BEGIN_OF_PATH

// comp_mux_rrr.v
module comp_mux_rrr(a, b, clock, outp);
input [2:0] a, b; output [2:0] outp; input clock;
reg [2:0] a_r, a_rr, b_r, b_rr, outp; reg sel_r;
wire sel = ( a_r <= b_r ) ? 0 : 1;
always @ (posedge clock) begin a_r <= a; b_r <= b; end
always @ (posedge clock) begin a_rr <= a_r; b_rr <= b_r; end
always @ (posedge clock) outp <= sel_r ? b_rr : a_rr;
always @ (posedge clock) sel_r <= sel;
endmodule
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.
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:

\[ t_{SU}^{(external)} > t_{SU}^{(internal)} - (\text{clock delay}) + (\text{data delay}) \]

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
  port(Clock, Key, Trip : in bit; Ring : out bit);
end Alarm;

architecture RTL of Alarm is
  type States is (Armed, Off, Ringing);
signal State : States;
begin
  process (Clock) begin
    if Clock = '1' and Clock'EVENT then
      case State is
        when Off => if Key = '1' then State <= Armed; end if;
        when Armed => if Key = '0' then State <= Off;
        elsif Trip = '1' then State <= Ringing;
        end if;
  end process
end RTL;
```
ASICs... THE COURSE 13.8 Formal Verification

when Ringing => if Key = '0' then State <= Off; end if;
end case;
end if;
end process;
Ring <= '1' when State = Ringing else '0';
end RTL;

library cells; use cells.all; // ...contains logic cell models
architecture Gates of Alarm is
component Inverter port(i : in BIT; z : out BIT) ; end component;
component NAnd2 port(a,b : in BIT; z : out BIT) ; end component;
component NAnd3 port(a,b,c : in BIT; z : out BIT) ; end component;
component DFF port(d,c : in BIT; q,qn : out BIT) ; end component;
signal State, NextState : BIT_VECTOR(Idownto 0);
signal s0, s1, s2, s3 : BIT;
begin
  g2: Inverter port map ( i => State(0), z => s1 );
g3: NAnd2 port map ( a => s1, b => State(1), z => s2 );
g4: Inverter port map ( i => s2, z => Ring );
g5: NAnd2 port map ( a => State(1), b => Key, z => s0 );
g6: NAnd3 port map ( a => Trip, b => s1, c => Key, z => s3 );
g7: NAnd2 port map ( a => s0, b => s3, z => NextState(1) );
g8: Inverter port map ( i => Key, z => NextState(0) );
state_ff_b0: DFF port map
  ( d => NextState(0), c => Clock, q => State(0), qn =>open );
state_ff_b1: DFF port map
  ( d => NextState(1), c => Clock, q => State(1), qn =>open );
end Gates;

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**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

**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
  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;
  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

...

```vhdl
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
```
A TSPC (true single-phase clock) flip-flop
(a) The schematic (all devices are W/L=3/2)
(b) The switch-level simulation results

The parameter chargeDecayTime sets the time after which the simulator sets an undriven node to an invalid logic level (shown shaded).

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)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>n-ch. value</th>
<th>p-ch. value</th>
<th>Units</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CGBO</td>
<td>4.0E-10</td>
<td>3.8E-10</td>
<td>Fm⁻¹</td>
<td>Gate–bulk overlap capacitance (CGBoh, not CGBzero)</td>
</tr>
<tr>
<td>CGDO</td>
<td>3.0E-10</td>
<td>2.4E-10</td>
<td>Fm⁻¹</td>
<td>Gate–drain overlap capacitance (CGDoh, not CGDzero)</td>
</tr>
<tr>
<td>CGSO</td>
<td>3.0E-10</td>
<td>2.4E-10</td>
<td>Fm⁻¹</td>
<td>Gate–source overlap capacitance (CGSo, not CGSzero)</td>
</tr>
<tr>
<td>CJ</td>
<td>5.6E-4</td>
<td>9.3E-4</td>
<td>Fm⁻²</td>
<td>Junction area capacitance</td>
</tr>
<tr>
<td>CJSW</td>
<td>5E-11</td>
<td>2.9E-10</td>
<td>Fm⁻¹</td>
<td>Junction sidewall capacitance</td>
</tr>
<tr>
<td>DELTA</td>
<td>0.7</td>
<td>0.29</td>
<td>m</td>
<td>Narrow-width factor for adjusting threshold voltage</td>
</tr>
<tr>
<td>ETA</td>
<td>3.7E-2</td>
<td>2.45E-2</td>
<td>1</td>
<td>Static-feedback factor for adjusting threshold voltage</td>
</tr>
<tr>
<td>GAMMA</td>
<td>0.6</td>
<td>0.47</td>
<td>V⁰.⁵</td>
<td>Body-effect factor</td>
</tr>
<tr>
<td>KAPPA</td>
<td>2.9E-2</td>
<td>8</td>
<td>V⁻¹</td>
<td>Saturation-field factor (channel-length modulation)</td>
</tr>
<tr>
<td>KP</td>
<td>2E-4</td>
<td>4.9E-5</td>
<td>AV⁻²</td>
<td>Intrinsic transconductance (µCox, not 0.5µCox)</td>
</tr>
<tr>
<td>LD</td>
<td>5E-8</td>
<td>3.5E-8</td>
<td>m</td>
<td>Lateral diffusion into channel</td>
</tr>
<tr>
<td>LEVEL</td>
<td>3</td>
<td>none</td>
<td>Empirical model</td>
<td></td>
</tr>
<tr>
<td>MJ</td>
<td>0.56</td>
<td>0.47</td>
<td>1</td>
<td>Junction area exponent</td>
</tr>
<tr>
<td>MJSW</td>
<td>0.52</td>
<td>0.50</td>
<td>1</td>
<td>Junction sidewall exponent</td>
</tr>
<tr>
<td>NFS</td>
<td>6E11</td>
<td>6.5E11</td>
<td>cm⁻²V⁻¹</td>
<td>Fast surface-state density</td>
</tr>
<tr>
<td>NSUB</td>
<td>1.4E17</td>
<td>8.5E16</td>
<td>cm⁻³</td>
<td>Bulk surface doping</td>
</tr>
<tr>
<td>PB</td>
<td>1</td>
<td>1</td>
<td>V</td>
<td>Junction area contact potential</td>
</tr>
<tr>
<td>PHI</td>
<td>0.7</td>
<td></td>
<td>V</td>
<td>Surface inversion potential</td>
</tr>
<tr>
<td>RSH</td>
<td>2</td>
<td>Ω/square</td>
<td>Sheet resistance of source and drain</td>
<td></td>
</tr>
<tr>
<td>THETA</td>
<td>0.27</td>
<td>0.29</td>
<td>V⁻¹</td>
<td>Mobility-degradation factor</td>
</tr>
<tr>
<td>TOX</td>
<td>1E⁻⁸</td>
<td>m</td>
<td>Gate-oxide thickness</td>
<td></td>
</tr>
<tr>
<td>TPG</td>
<td>1</td>
<td>none</td>
<td>Type of polysilicon gate</td>
<td></td>
</tr>
<tr>
<td>U0</td>
<td>550</td>
<td>135</td>
<td>cm²V⁻¹s⁻¹</td>
<td>Low-field bulk carrier mobility (Uzero, not Uoh)</td>
</tr>
<tr>
<td>XJ</td>
<td>0.2E⁻⁶</td>
<td>m</td>
<td>Junction depth</td>
<td></td>
</tr>
<tr>
<td>VMAX</td>
<td>2E5</td>
<td>2.5E5</td>
<td>ms⁻¹</td>
<td>Saturated carrier velocity</td>
</tr>
<tr>
<td>VTO</td>
<td>0.65</td>
<td>-0.92</td>
<td>V</td>
<td>Zero-bias threshold voltage (VTo, not VToh)</td>
</tr>
</tbody>
</table>
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
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)

<table>
<thead>
<tr>
<th>ASIC defect level</th>
<th>Defective ASICs</th>
<th>Total PCB repair cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>5%</td>
<td>5000</td>
<td>$1 million</td>
</tr>
<tr>
<td>1%</td>
<td>1000</td>
<td>$200,000</td>
</tr>
<tr>
<td>0.1%</td>
<td>100</td>
<td>$20,000</td>
</tr>
<tr>
<td>0.01%</td>
<td>10</td>
<td>$2,000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ASIC defect level</th>
<th>Defective ASICs</th>
<th>Defective boards</th>
<th>Total repair cost at system level</th>
</tr>
</thead>
<tbody>
<tr>
<td>5%</td>
<td>5000</td>
<td>500</td>
<td>$5 million</td>
</tr>
<tr>
<td>1%</td>
<td>1000</td>
<td>100</td>
<td>$1 million</td>
</tr>
<tr>
<td>0.1%</td>
<td>100</td>
<td>10</td>
<td>$100,000</td>
</tr>
<tr>
<td>0.01%</td>
<td>10</td>
<td>1</td>
<td>$10,000</td>
</tr>
</tbody>
</table>
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)

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Meaning</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BR</td>
<td>Bypass register</td>
<td>A TDR, directly connects TDI and TDO, bypassing BSR</td>
</tr>
<tr>
<td>BSC</td>
<td>Boundary-scan cell</td>
<td>Each I/O pad has a BSC to monitor signals</td>
</tr>
<tr>
<td>BSR</td>
<td>Boundary-scan register</td>
<td>A TDR, a shift register formed from a chain of BSCs</td>
</tr>
<tr>
<td>BST</td>
<td>Boundary-scan test</td>
<td>Not to be confused with BIST (built-in self-test)</td>
</tr>
<tr>
<td>IDCODE</td>
<td>Device-identification register</td>
<td>Optional TDR, contains manufacturer and part number</td>
</tr>
<tr>
<td>IR</td>
<td>Instruction register</td>
<td>Holds a BST instruction, provides control signals</td>
</tr>
<tr>
<td>JTAG</td>
<td>Joint Test Action Group</td>
<td>The organization that developed boundary scan</td>
</tr>
<tr>
<td>TAP</td>
<td>Test-access port</td>
<td>Four- (or five-)wire test interface to an ASIC</td>
</tr>
<tr>
<td>TCK</td>
<td>Test clock</td>
<td>A TAP wire, the clock that controls BST operation</td>
</tr>
<tr>
<td>TDI</td>
<td>Test-data input</td>
<td>A TAP wire, the input to the IR and TDRs</td>
</tr>
<tr>
<td>TDO</td>
<td>Test-data output</td>
<td>A TAP wire, the output from the IR and TDRs</td>
</tr>
<tr>
<td>TDR</td>
<td>Test-data register</td>
<td>Group of BST registers: IDCODE, BR, BSR</td>
</tr>
<tr>
<td>TMS</td>
<td>Test-mode select</td>
<td>A TAP wire, together with TCK controls the BST state</td>
</tr>
<tr>
<td>TRST* or nTRST</td>
<td>Test-reset input signal</td>
<td>Optional TAP wire, resets the TAP controller (active-low)</td>
</tr>
</tbody>
</table>
IEEE 1149.1 boundary scan

(a) Boundary scan is intended to check for shorts or opens between ICs mounted on a board

(b) Shorts and opens may also occur inside the IC package

(c) The boundary-scan architecture is a long chain of shift registers allowing data to be sent over all the connections between the ICs on a board
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)
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, updatedDR : 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;
begins
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;
  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

![TAP Controller State Machine Diagram]

The TAP (test-access port) controller state machine

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)
14.3 Faults

Key terms and concepts: defect • fault • defect mechanisms • bridge or short circuit (shorts) • breaks or open circuits (opens) • rework
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

<table>
<thead>
<tr>
<th>Fault level</th>
<th>Physical fault</th>
<th>Degradation fault</th>
<th>Open-circuit fault</th>
<th>Short-circuit fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chip</td>
<td>Leakage or short between package leads</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Broken, misaligned, or poor wire bonding</td>
<td>•</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Surface contamination, moisture</td>
<td>•</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Metal migration, stress, peeling</td>
<td></td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Metallization (open or short)</td>
<td></td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Gate</td>
<td>Contact opens</td>
<td></td>
<td>•</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gate to S/D junction short</td>
<td></td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Field-oxide parasitic device</td>
<td></td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Gate-oxide imperfection, spiking</td>
<td></td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td></td>
<td>Mask misalignment</td>
<td></td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>

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

Fault models

(a) Physical faults at the layout level (problems during fabrication) translate to electrical problems on the detailed circuit schematic. The location and effect of fault F1 is shown. The locations of the other faults are shown, but not their effect.

(b) We can translate some of these faults to the simplified transistor schematic.

(c) Only a few of the physical faults still remain in a gate-level fault model of the logic cell.

(d) Finally at the functional-level fault model of a logic cell, we abandon the connection between physical and logical faults and model all faults by stuck-at faults. This is a very poor model of the physical reality, but it works well in practice.

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
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

<table>
<thead>
<tr>
<th>Fault coverage</th>
<th>Average defect level</th>
<th>Average quality level (AQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>7%</td>
<td>93%</td>
</tr>
<tr>
<td>90%</td>
<td>3%</td>
<td>97%</td>
</tr>
<tr>
<td>95%</td>
<td>1%</td>
<td>99%</td>
</tr>
<tr>
<td>99%</td>
<td>0.1%</td>
<td>99.9%</td>
</tr>
<tr>
<td>99.9%</td>
<td>0.01%</td>
<td>99.99%</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Good circuit</th>
<th>0</th>
<th>1</th>
<th>Z</th>
<th>L</th>
<th>H</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>U</td>
<td>D</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>1</td>
<td>D</td>
<td>U</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>Z</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>L</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>H</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>X</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>U</td>
</tr>
</tbody>
</table>

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 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) 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

A basic ATPG (automatic test-pattern generation) algorithm for A'B+BC

(a) We activate a fault, U2.ZN stuck at 1, by setting the pin or node to '0', the opposite value of the fault

(b) We work backward from the fault origin to the PIs (primary inputs) by recursively justifying signals at the output of logic cells

(c) We then work forward from the fault origin to a PO (primary output), setting inputs to gates on a sensitized path to their enabling values. We propagate the fault until the D-frontier reaches a PO

(d) We then work backward from the PO to the PIs recursively justifying outputs to generate the sensitized path. This simple algorithm always works, providing signals do not branch out and then rejoin again.
14.5.2 A Basic ATPG Algorithm

Key terms and concepts: activating (or exciting the fault) • sensitize • observed • D-frontier, • reconvergent fanout • multipath sensitization

Reconvergent fanout

(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 • backtrace • implication • D-frontier • X-path check • backtrack • FAN (fanout-oriented test generation)
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 • combinational one-controllability • combinational observability

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
Observability measures

(a) The combinational observability, OC(X₁), of an input, X₁, 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 Scan Test

*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

<table>
<thead>
<tr>
<th>Clock tick, t=</th>
<th>Q0_{t+1}=Q1\oplus Q2_t</th>
<th>Q1_{t+1}=Q0_t</th>
<th>Q2_{t+1}=Q1_t</th>
<th>Q0Q1Q2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

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 Q1Q2Q3='000' using the common RES (reset) signal

The signature, Q1Q2Q3, is formed from shift-and-add operations on the sequence of input bits (IN)
14.7.3 A Simple BIST Example

<table>
<thead>
<tr>
<th>$Q_{0_{t+1}} = Q1_t \oplus Q2_t$</th>
<th>$Q1_{t+1} = Q0_t$</th>
<th>$Q2_{t+1} = Q1_t$</th>
<th>$Z = Q0' \cdot Q1 + Q1 \cdot Q2$</th>
<th>$R0_{t+1} = Z \oplus R0_t \oplus R2_t$</th>
<th>$R1_{t+1} = R0_t$</th>
<th>$R2_{t+1} = R1_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

BIST example. (a) A simple BIST structure showing bit sequences for both good and bad circuits. (b) Bit sequence calculations for the good circuit. The signature appears on the eighth clock cycle (after seven positive clock edges) and is $R0=0$, $R1=1$, $R2=1$; with $R2$ as the MSB this is '011' or hex 3.
The waveforms of the BIST example

(a) The good-circuit response. The waveforms Q1 and Q2, as well as R1 and R2, are delayed by one clock cycle as they move through each stage of the shift registers.

(b) The same good-circuit response with the register outputs Q0–Q2 and R0–R2 grouped and their values displayed in hexadecimal (Q0 and R0 are the MSBs). The signature hex 3 or '011' (R0=0, R1=1, R2=1) in R appears seven positive clock edges after the reset signal is taken high. This is one clock cycle after the generator completes its first sequence (hex pattern 4, 2, 5, 6, 7, 3, 1).

(c) The response of the bad circuit with fault F1 and fault signature hex 0 (circled).
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

<table>
<thead>
<tr>
<th>n</th>
<th>s</th>
<th>Octal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0, 1</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>0, 1, 2</td>
<td>7</td>
<td>111</td>
</tr>
<tr>
<td>3</td>
<td>0, 1, 3</td>
<td>13</td>
<td>1011</td>
</tr>
<tr>
<td>4</td>
<td>0, 1, 4</td>
<td>3</td>
<td>10011</td>
</tr>
<tr>
<td>5</td>
<td>0, 2, 5</td>
<td>45</td>
<td>100101</td>
</tr>
<tr>
<td>6</td>
<td>0, 1, 6</td>
<td>103</td>
<td>100011</td>
</tr>
<tr>
<td>7</td>
<td>0, 1, 7</td>
<td>211</td>
<td>1000101</td>
</tr>
<tr>
<td>8</td>
<td>0, 1, 5, 6, 8</td>
<td>435</td>
<td>100011101</td>
</tr>
<tr>
<td>9</td>
<td>0, 4, 9</td>
<td>1021</td>
<td>1000010001</td>
</tr>
<tr>
<td>10</td>
<td>0, 3, 10</td>
<td>2011</td>
<td>10000001001</td>
</tr>
</tbody>
</table>

For $n=3$ and $s=0, 1, 3$: $c_0=1, c_1=1, c_2=0, c_3=1$

$P(x) = 1 \oplus c_1x \oplus ... \oplus c_{n-1}x^{n-1} \oplus x^n$

or $P^*(x) = 1 \oplus c_{n-1}x \oplus ... \oplus c_1x^{n-1} \oplus x^n$

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 $P(x)=1 \oplus x \oplus x^3$ and $P^*(x)=1 \oplus x^2 \oplus x^3$.

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 \( P^*(x) = 1 \oplus x^2 \oplus x^3 \)

```verilog
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));
dfcntb FF1 (.D(XOR0_Z), .CP(u4_Z), .CDN(u2_Z), .Q(FF1_Q), .QN(FF1_QN));
dfcntb 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));
oxo2d1 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 \( P^*(x) = 1 \oplus x^2 \oplus x^3 \)) 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].
14.8 A Simple Test Example

14.8.1 Test-Logic Insertion

Key terms and concepts: \( \text{outp} = \text{a}_r[0]' \cdot \text{a}_r[1] + \text{a}_r[1] \cdot \text{a}_r[2] \cdot \text{test-logic insertion} \)

![A Simple Test Example Diagram]

The core of the Threegates ASIC after 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.
The top level of the Threegates ASIC after test-logic insertion.
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:
# Aborted                  0                   0
# Detected                 89                  43
# Untested                 58                  20
#                     ------    ------
# Total of detectable      147                 63
# Reduced                  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

```bash
# 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

<table>
<thead>
<tr>
<th>Slack(ns)</th>
<th>Num Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3.3826</td>
<td>1 *</td>
</tr>
<tr>
<td>-1.7536</td>
<td>18 ******</td>
</tr>
<tr>
<td>-.1245</td>
<td>4 **</td>
</tr>
<tr>
<td>1.5045</td>
<td>1 *</td>
</tr>
<tr>
<td>3.1336</td>
<td>0 *</td>
</tr>
<tr>
<td>4.7626</td>
<td>0 *</td>
</tr>
<tr>
<td>6.3916</td>
<td>134 *******</td>
</tr>
<tr>
<td>8.0207</td>
<td>6 ***</td>
</tr>
<tr>
<td>9.6497</td>
<td>3 **</td>
</tr>
<tr>
<td>11.2787</td>
<td>0 *</td>
</tr>
<tr>
<td>12.9078</td>
<td>24 ******</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_1.u100.u1.subout6.Q_ff_b0</td>
</tr>
<tr>
<td>CP --&gt; QN</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>v_1.u100.u2.metric0.Q_ff_b4</td>
</tr>
<tr>
<td>setup: D --&gt; CP</td>
</tr>
</tbody>
</table>

### After test-logic insertion

<table>
<thead>
<tr>
<th>Slack(ns)</th>
<th>Num Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>-4.0034</td>
<td>1 *</td>
</tr>
<tr>
<td>-1.9835</td>
<td>18 ******</td>
</tr>
<tr>
<td>.0365</td>
<td>4 **</td>
</tr>
<tr>
<td>2.0565</td>
<td>1 *</td>
</tr>
<tr>
<td>4.0764</td>
<td>0 *</td>
</tr>
<tr>
<td>6.0964</td>
<td>138 *******</td>
</tr>
<tr>
<td>8.1164</td>
<td>2 *</td>
</tr>
<tr>
<td>10.1363</td>
<td>3 **</td>
</tr>
<tr>
<td>12.1563</td>
<td>24 ******</td>
</tr>
<tr>
<td>14.1763</td>
<td>0 *</td>
</tr>
<tr>
<td>16.1963</td>
<td>187 *******</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>instance name</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_1.u100.u1.subout7.Q_ff_b1</td>
</tr>
<tr>
<td>CP --&gt; Q</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>v_1.u100.u2.metric0.Q_ff_b4</td>
</tr>
<tr>
<td>setup: DB --&gt; CP</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th># VECTORS</th>
<th>FAULTS</th>
<th>FAULT COVER</th>
</tr>
</thead>
<tbody>
<tr>
<td>processed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>7515</td>
<td>82.92%</td>
</tr>
<tr>
<td>40</td>
<td>8087</td>
<td>89.39%</td>
</tr>
<tr>
<td>60</td>
<td>8313</td>
<td>91.74%</td>
</tr>
<tr>
<td>80</td>
<td>8632</td>
<td>95.29%</td>
</tr>
<tr>
<td>87</td>
<td>8846</td>
<td>96.06%</td>
</tr>
</tbody>
</table>

# Total number of backtracks: 3000
# Highest backtrack : 30
# Total number of vectors : 87

# STAR RESULTS summary

<table>
<thead>
<tr>
<th>Fault counts:</th>
<th>Noncollapsed</th>
<th>Collapsed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aborted</td>
<td>178</td>
<td>85</td>
</tr>
<tr>
<td>Detected</td>
<td>8427</td>
<td>3680</td>
</tr>
<tr>
<td>Untested</td>
<td>168</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>Total of detectable</td>
<td>8773</td>
<td>3825</td>
</tr>
<tr>
<td>Redundant</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>Tied</td>
<td>63</td>
<td>38</td>
</tr>
</tbody>
</table>

# 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

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 CAD Tools

*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?”

<table>
<thead>
<tr>
<th>SPARCstation 1 ASIC</th>
<th>Gates /k-gate</th>
<th>Pins</th>
<th>Package</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 SPARC IU (integer unit)</td>
<td>20</td>
<td>179</td>
<td>PGA</td>
<td>CBIC</td>
</tr>
<tr>
<td>2 SPARC FPU (floating-point unit)</td>
<td>50</td>
<td>144</td>
<td>PGA</td>
<td>FC</td>
</tr>
<tr>
<td>3 Cache controller</td>
<td>9</td>
<td>160</td>
<td>PQFP</td>
<td>GA</td>
</tr>
<tr>
<td>4 MMU (memory-management unit)</td>
<td>5</td>
<td>120</td>
<td>PQFP</td>
<td>GA</td>
</tr>
<tr>
<td>5 Data buffer</td>
<td>3</td>
<td>120</td>
<td>PQFP</td>
<td>GA</td>
</tr>
<tr>
<td>6 DMA (direct memory access) controller</td>
<td>9</td>
<td>120</td>
<td>PQFP</td>
<td>GA</td>
</tr>
<tr>
<td>7 Video controller/data buffer</td>
<td>4</td>
<td>120</td>
<td>PQFP</td>
<td>GA</td>
</tr>
<tr>
<td>8 RAM controller</td>
<td>1</td>
<td>100</td>
<td>PQFP</td>
<td>GA</td>
</tr>
<tr>
<td>9 Clock generator</td>
<td>1</td>
<td>44</td>
<td>PLCC</td>
<td>GA</td>
</tr>
</tbody>
</table>
### 15.4 Estimating ASIC Size

<table>
<thead>
<tr>
<th>SPARCstation 10 ASIC</th>
<th>Gates</th>
<th>Pins</th>
<th>Package</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 SuperSPARC Superscalar SPARC</td>
<td>3M-transistors</td>
<td>293</td>
<td>PGA</td>
<td>FC</td>
</tr>
<tr>
<td>2 SuperCache cache controller</td>
<td>2M-transistors</td>
<td>369</td>
<td>PGA</td>
<td>FC</td>
</tr>
<tr>
<td>3 EMC memory control</td>
<td>40k-gate</td>
<td>299</td>
<td>PGA</td>
<td>GA</td>
</tr>
<tr>
<td>4 MSI MBus–SBus interface</td>
<td>40k-gate</td>
<td>223</td>
<td>PGA</td>
<td>GA</td>
</tr>
<tr>
<td>5 DMA2 Ethernet, SCSI, parallel port</td>
<td>30k-gate</td>
<td>160</td>
<td>PQFP</td>
<td>GA</td>
</tr>
<tr>
<td>6 SEC SBus to 8-bit bus</td>
<td>20k-gate</td>
<td>160</td>
<td>PQFP</td>
<td>GA</td>
</tr>
<tr>
<td>7 DBRI dual ISDN interface</td>
<td>72k-gate</td>
<td>132</td>
<td>PQFP</td>
<td>GA</td>
</tr>
<tr>
<td>8 MMCodec stereo codec</td>
<td>32k-gate</td>
<td>44</td>
<td>PLCC</td>
<td>FC</td>
</tr>
</tbody>
</table>
**Some useful numbers for ASIC estimates, normalized to a 1 µm technology**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Typical value</th>
<th>Comment</th>
<th>Scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lambda, ( \lambda )</td>
<td>0.5 µm = 0.5 (minimum feature size)</td>
<td>In a 1 µm technology, ( \lambda \approx 0.5 ) µm.</td>
<td>NA</td>
</tr>
<tr>
<td>Effective gate length</td>
<td>0.25 to 1.0 µm</td>
<td>Less than drawn gate length, usually by about 10 percent.</td>
<td>( \lambda )</td>
</tr>
<tr>
<td>I/O-pad width (pitch)</td>
<td>5 to 10 mil</td>
<td>For a 1 µm technology, 2LM ( (\lambda = 0.5 ) µm). Scales less than linearly with ( \lambda ).</td>
<td>( \lambda )</td>
</tr>
<tr>
<td>I/O-pad height</td>
<td>15 to 20 mil</td>
<td>For a 1 µm technology, 2LM ( (\lambda = 0.5 ) µm). Scales approximately linearly with ( \lambda ).</td>
<td>( \lambda )</td>
</tr>
<tr>
<td>Large die</td>
<td>1000 mil/side, ( 10^6 ) mil²</td>
<td>Approximately constant</td>
<td>1</td>
</tr>
<tr>
<td>Small die</td>
<td>100 mil/side, ( 10^4 ) mil²</td>
<td>Approximately constant</td>
<td>1</td>
</tr>
<tr>
<td>Standard-cell density</td>
<td>( 1.5 \times 10^{-3} ) gate/µm²</td>
<td>For 1 µm, 2LM, library ( = 4 \times 10^{-4} ) gate/( \lambda )^2 (independent of scaling).</td>
<td>( 1/\lambda^2 )</td>
</tr>
<tr>
<td>Standard-cell density</td>
<td>( 8 \times 10^{-3} ) gate/µm²</td>
<td>For 0.5 µm, 3LM, library ( = 5 \times 10^{-4} ) gate/( \lambda )^2 (independent of scaling).</td>
<td>( 1/\lambda^2 )</td>
</tr>
<tr>
<td>Gate-array utilization</td>
<td>60 to 80%</td>
<td>For 2LM, approximately constant</td>
<td>1</td>
</tr>
<tr>
<td>Gate-array density</td>
<td>(0.8 to 0.9) × standard cell density</td>
<td>For the same process as standard cells</td>
<td>1</td>
</tr>
<tr>
<td>Standard-cell routing factor</td>
<td>1.5 to 2.5 (2LM)</td>
<td>Approximately constant</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1.0 to 2.0 (3LM)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Package cost</td>
<td>$0.01/pin, “penny per pin”</td>
<td>Varies widely, figure is for low-cost plastic package, approximately constant</td>
<td>1</td>
</tr>
<tr>
<td>Wafer cost</td>
<td>$1k to $5k average $2k</td>
<td>Varies widely, figure is for a mature, 2LM CMOS process, approximately constant</td>
<td>1</td>
</tr>
</tbody>
</table>
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 $= 0.5 \cdot CV_{DD}^2 = IV = CV(dV/dt)$ for one-half the period of the input, $t = 1/(2f)$ • total power $= P_1 = fCV^2_{DD}$ • estimate power by counting nodes that toggle

15.5.2 Short-Circuit Current

Key terms and concepts: $P_2 = (1/12)\beta f t_n(V_{DD} - 2V_{th})$ • short-circuit current is typically less than 20 percent of the switching current

Estimating circuit size

(a) ASIC memory size. These figures are for static RAM constructed using compilers in a 2LM ASIC process, but with no special memory design rules.

The actual area of a RAM will depend on the speed and number of read–write ports.

(b) Multiplier size for a 2LM process.

The actual area will depend on the multiplier architecture and speed.
15.5.3 Subthreshold and Leakage Current

**Key terms and concepts:** subthreshold current is normally less than 5pA\(\mu\)m\(^{-1}\) of gate width • subthreshold current for 10 million transistors (each 10\(\mu\)m wide) is 0.1mA • subthreshold current does not scale • it takes about 120mV to reduce subthreshold current by a factor of 10 • if \(V_t = 0.36V\), at \(V_{GS}=0 V\) 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

<table>
<thead>
<tr>
<th>Chip #</th>
<th>Size</th>
<th>Chip #</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>42 GLBs</td>
<td>7</td>
<td>36 GLBs</td>
</tr>
<tr>
<td>2</td>
<td>64k-bit × 8 SRAM</td>
<td>8</td>
<td>22 GLBs</td>
</tr>
<tr>
<td>3</td>
<td>38 GLBs</td>
<td>9</td>
<td>256k-bit × 16 SRAM</td>
</tr>
<tr>
<td>4</td>
<td>38 GLBs</td>
<td>10</td>
<td>43 GLBs</td>
</tr>
<tr>
<td>5</td>
<td>42 GLBs</td>
<td>11</td>
<td>40 GLBs</td>
</tr>
<tr>
<td>6</td>
<td>64k-bit × 16 SRAM</td>
<td>12</td>
<td>30 GLBs</td>
</tr>
</tbody>
</table>

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
Partitioning example.

(a) We wish to partition this network into three ASICs with no more than four logic cells per ASIC.

(b) A partitioning with five external connections (nets 2, 4, 5, 6, and 8)—the minimum number.

(c) A constructed partition using logic cell C as a seed. It is difficult to get from this local minimum, with seven external connections (2, 3, 5, 7, 9, 11, 12), to the optimum solution of b.
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 \( \exp(-\Delta E/T) \) • \( \Delta E \) is an increase in energy function • \( T \) corresponds to temperature • we hill climb to get out of a local minimum • cooling schedule • \( T_{i+1} = \alpha T_i \)
• good results at the expense of long run times • Xilinx used simulated annealing in one verion 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 $2\text{pFcm}^{-1}$ for a minimum-width wire, but gate delay continues to decrease • Floorplanning predicts interconnect delay by estimating interconnect length

Interconnect and gate delays.
As feature sizes decrease, both average interconnect delay and average gate delay decrease—but at different rates.
This is because interconnect capacitance tends to a limit that is independent of scaling.
Interconnect delay now dominates gate delay.

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 takes 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).

<table>
<thead>
<tr>
<th>Array (available gates)</th>
<th>Chip size (mm)</th>
<th>1</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3k</td>
<td>3.45</td>
<td>0.56</td>
<td>0.85</td>
<td>1.46</td>
</tr>
<tr>
<td>11k</td>
<td>5.11</td>
<td>0.84</td>
<td>1.34</td>
<td>2.25</td>
</tr>
<tr>
<td>105k</td>
<td>12.50</td>
<td>1.75</td>
<td>2.70</td>
<td>4.92</td>
</tr>
</tbody>
</table>

Worst-case interconnect delay.

As we scale circuits, but avoid scaling the chip size, the worst-case interconnect delay increases.

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
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

Defining the channel routing order for a slicing floorplan using a slicing tree.

(a) Make a cut all the way across the chip between circuit blocks. Continue slicing until each piece contains just one circuit block. Each cut divides a piece into two without cutting through a circuit block.

(b) A sequence of cuts: 1, 2, 3, and 4 that successively slices the chip until only circuit blocks are left.

(c) The slicing tree corresponding to the sequence of cuts gives the order in which to route the channels: 4, 3, 2, and finally 1.
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

(a) A pad-limited die. The number of pads determines the die size.
(b) A core-limited die: The core logic determines the die size.
(c) Using both pad-limited pads and core-limited pads for a square die.
Bonding pads.

(a) This chip uses both pad-limited and core-limited pads.
(b) A hybrid corner pad.
(c) A chip with stagger-bonded pads.
(d) An area-bump bonded chip (or flip-chip). The chip is turned upside down and solder bumps connect the pads to the lead frame.
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

Clock distribution.

(a) A clock spine for a gate array.

(b) A clock spine for a cell-based ASIC (typical chips have thousands of clock nets).

(c) A clock spine is usually driven from one or more clock-driver cells.

Delay in the driver cell is a function of the number of stages and the ratio of output to input capacitance for each stage (taper).

(d) Clock latency and clock skew. We would like to minimize both latency and skew.
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

Interconnect structure.

(a) A two-level metal CBIC floorplan.

(b) A channel from the flexible block A. This channel has a channel height equal to the maximum channel density of 7 (there is room for seven interconnects to run horizontally in m1).

(c) A channel that uses OTC (over-the-cell) routing in m2.
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-length measures.
(a) Complete-graph measure.
(b) Half-perimeter measure.

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 • \( \lambda \)-optimum • neighborhood exchange algorithm • neighborhood • \( \epsilon \)-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 \( \epsilon \) of a logic cell form an \( \epsilon \)-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 $\Delta E$ is negative or zero, then exchange the logic cells. If $\Delta E$ is positive, then exchange the logic cells with a probability of $\exp(-\Delta E/T)$.
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

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.

Timing-driven floorplanning and placement design flow.
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")
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**

**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

Measuring the delay of a net.

(a) A simple circuit with an inverter A driving a net with a fanout of two. Voltages $V_1$, $V_2$, $V_3$, and $V_4$ are the voltages at intermediate points along the net.

(b) The layout showing the net segments (pieces of interconnect).

(c) The RC model with each segment replaced by a capacitance and resistance. The ideal switch and pull-down resistance $R_{pd}$ model the inverter A.

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
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.
Gate-array global routing.

(a) A small gate array.

(b) An enlarged view of the routing. The top channel uses three rows of gate-array base cells; the other channels use only one.

(c) A further enlarged view showing how the routing in the channels connects to the logic cells.

(d) One of the logic cells, an inverter.

(e) There are seven horizontal wiring tracks available in one row of gate-array base cells—the channel capacity is thus 7.
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)

![Diagram](image)

The metal routing pitch.

(a) An example of $\lambda$-based metal design rules for m1 and via1 (m1/m2 via).

(b) Via-to-via pitch for adjacent vias.

(c) Via-to-line (or line-to-via) pitch for nonadjacent vias.

(d) Line-to-line pitch with no vias.
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

The definitions of local channel density and global channel density.

The definitions of local channel density and global channel density.

Lines represent the m1 and m2 interconnect in the channel to simplify the drawing.

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

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 ($\lambda=0.25\mu$m) CMOS process.**

<table>
<thead>
<tr>
<th>Layer/contact/via</th>
<th>Current limit</th>
<th>Metal thickness</th>
<th>Resistance</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>1mA $\mu$m$^{-1}$</td>
<td>7000Å</td>
<td>95Ω/square</td>
</tr>
<tr>
<td>m2</td>
<td>1mA $\mu$m$^{-1}$</td>
<td>7000Å</td>
<td>95Ω/square</td>
</tr>
<tr>
<td>m3</td>
<td>2mA $\mu$m$^{-1}$</td>
<td>12,000Å</td>
<td>48Ω/square</td>
</tr>
<tr>
<td>0.8 $\mu$m square m1 contact to diffusion</td>
<td>0.7 mA</td>
<td></td>
<td>11Ω</td>
</tr>
<tr>
<td>0.8 $\mu$m square m1 contact to poly</td>
<td>0.7 mA</td>
<td></td>
<td>16Ω</td>
</tr>
<tr>
<td>0.8 $\mu$m square m1/m2 via (via1)</td>
<td>0.7 mA</td>
<td></td>
<td>3.6Ω</td>
</tr>
<tr>
<td>0.8 $\mu$m square m2/m3 via (via2)</td>
<td>0.7 mA</td>
<td></td>
<td>3.6Ω</td>
</tr>
</tbody>
</table>

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 (λ=0.5 µm) three-level metal CMOS process.

<table>
<thead>
<tr>
<th>Element</th>
<th>Area/fFµm⁻²</th>
<th>Fringing/fFµm⁻¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>poly (over gate oxide) to substrate</td>
<td>1.73</td>
<td>NA</td>
</tr>
<tr>
<td>poly (over field oxide) to substrate</td>
<td>0.058</td>
<td>0.043</td>
</tr>
<tr>
<td>m1 to diffusion or poly</td>
<td>0.055</td>
<td>0.049</td>
</tr>
<tr>
<td>m1 to substrate</td>
<td>0.031</td>
<td>0.044</td>
</tr>
<tr>
<td>m2 to diffusion</td>
<td>0.019</td>
<td>0.038</td>
</tr>
<tr>
<td>m2 to substrate</td>
<td>0.015</td>
<td>0.035</td>
</tr>
<tr>
<td>m2 to poly</td>
<td>0.022</td>
<td>0.040</td>
</tr>
<tr>
<td>m2 to m1</td>
<td>0.035</td>
<td>0.046</td>
</tr>
<tr>
<td>m3 to diffusion</td>
<td>0.011</td>
<td>0.034</td>
</tr>
<tr>
<td>m3 to substrate</td>
<td>0.010</td>
<td>0.033</td>
</tr>
<tr>
<td>m3 to poly</td>
<td>0.012</td>
<td>0.034</td>
</tr>
<tr>
<td>m3 to m1</td>
<td>0.016</td>
<td>0.039</td>
</tr>
<tr>
<td>m3 to m2</td>
<td>0.035</td>
<td>0.049</td>
</tr>
<tr>
<td>n+ junction (at 0V bias)</td>
<td>0.36</td>
<td>NA</td>
</tr>
<tr>
<td>p+ junction (at 0V bias)</td>
<td>0.46</td>
<td>NA</td>
</tr>
</tbody>
</table>

#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)
C2 ROOT_Z VSS 1.17PF
C1 ROOT_Z_OUTP1 VSS 2.49PF
R2 ROOT_Z ROOT_Z_OUTP1 8.85
S (DF1_G_INP1 0.0 0.0)
E2 DF1_G_INP1 0.0 0.0
R3 DF2_G_INP1 VSS ROOT_Z VSS 1.0
C3 DF1_G VSS 1.0PF
* LOAD DF2_G DF2 G
* S (DF2_G_INP1 0.0 0.0)
E3 DF2_G_INP1 0.0 0.0
R4 DF2_G_INP1 DF2_G 13.05
C4 DF2_G VSS 1.0PF
* Instance Section
XDF1 DF1_D DF1_Q DF1_QN 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.